MCUXpresso SDK API Reference Manual

NXP Semiconductors

Document Number: MCUXSDKAPIRM
Rev. 0
Jan 2021

Contents

Contents

Contents

Contents

Contents

Section Number	Title	Page Number

Contents

Contents

Section Number	Title	Page Number

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Section Number	Title	Page Number

Chapter **PDB: Programmable Delay Block**

Contents

Contents

Section Number	Title	Page Number

Contents

Contents

Contents

Contents

Section Number	Title	Page Number

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Contents

Section Number	Title	Page Number

Contents

Contents

Contents

Section Number	Title	Page Number

Contents

Section Number	Title	Page Number

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Section Number	Title	Page Number

Contents

Contents

Chapter 1
Introduction

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

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

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

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

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

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

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

Table 2: MCUXpresso SDK Folder Structure

Chapter 2
Trademarks

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

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

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

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

Chapter 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

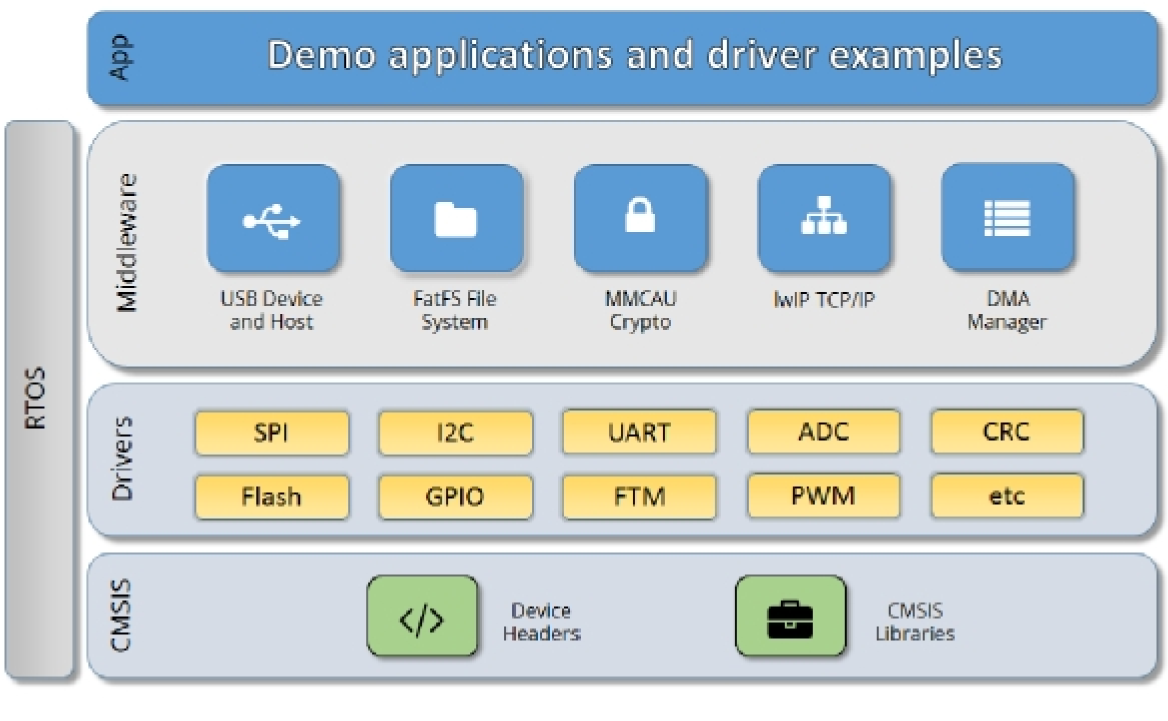

Figure 1: MCUXpresso SDK Block Diagram

MCU header files

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

CMSIS Support

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

MCUXpresso SDK Peripheral Drivers

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

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

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

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

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

Interrupt handling for transactional APIs

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

```
        PUBWEAK SPI0_IRQHandler
        PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

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

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

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

- kStatus_DSPI_Error = 601
- kStatus_EDMA_QueueFull = 5100
- kStatus_EDMA_Busy = 5101
- kStatus_ENET_RxFrameError = 4000
- kStatus_ENET_RxFrameFail = 4001
- kStatus_ENET_RxFrameEmpty = 4002
- kStatus_ENET_TxFrameOverLen = 4003
- kStatus_ENET_TxFrameBusy = 4004
- kStatus_ENET_TxFrameFail = 4005
- kStatus_FLEXCAN_TxBusy = 5300
- kStatus_FLEXCAN_TxIdle = 5301
- kStatus_FLEXCAN_TxSwitchToRx = 5302
- kStatus_FLEXCAN_RxBusy = 5303
- kStatus_FLEXCAN_RxIdle = 5304
- kStatus_FLEXCAN_RxOverflow = 5305
- kStatus_FLEXCAN_RxFifoBusy = 5306
- kStatus_FLEXCAN_RxFifoIdle = 5307
- kStatus_FLEXCAN_RxFifoOverflow = 5308
- kStatus_FLEXCAN_RxFifoWarning = 5309
- kStatus_FLEXCAN_ErrorStatus = 5310
- kStatus_FLEXCAN_UnHandled = 5311
- kStatus_I2C_Busy = 1100
- kStatus_I2C_Idle = 1101
- kStatus_I2C_Nak = 1102
- kStatus_I2C_ArbitrationLost = 1103
- kStatus_I2C_Timeout = 1104
- kStatus_I2C_Addr_Nak = 1105
- kStatus_LPUART_TxBusy = 1300
- kStatus_LPUART_RxBusy = 1301
- kStatus_LPUART_TxIdle = 1302
- kStatus_LPUART_RxIdle = 1303
- kStatus_LPUART_TxWatermarkTooLarge = 1304
- kStatus_LPUART_RxWatermarkTooLarge = 1305
- kStatus_LPUART_FlagCannotClearManually = 1306
- kStatus_LPUART_Error = 1307
- kStatus_LPUART_RxRingBufferOverrun = 1308
- kStatus_LPUART_RxHardwareOverrun = 1309
- kStatus_LPUART_NoiseError = 1310

- kStatus_LPUART_FramingError = 1311
- kStatus_LPUART_ParityError = 1312
- kStatus_LPUART_BaudrateNotSupport = 1313
- kStatus_LPUART_IdleLineDetected = 1314
- kStatus_SAI_TxBusy = 1900
- kStatus_SAI_RxBusy = 1901
- kStatus_SAI_TxError = 1902
- kStatus_SAI_RxError = 1903
- kStatus_SAI_QueueFull = 1904
- kStatus_SAI_TxIdle = 1905
- kStatus_SAI_RxIdle = 1906
- kStatus_SMC_StopAbort = 3900
- kStatus_UART_TxBusy = 1000
- kStatus_UART_RxBusy = 1001
- kStatus_UART_TxIdle = 1002
- kStatus_UART_RxIdle = 1003
- kStatus_UART_TxWatermarkTooLarge = 1004
- kStatus_UART_RxWatermarkTooLarge = 1005
- kStatus_UART_FlagCannotClearManually = 1006
- kStatus_UART_Error = 1007
- kStatus_UART_RxRingBufferOverrun = 1008
- kStatus_UART_RxHardwareOverrun = 1009
- kStatus_UART_NoiseError = 1010
- kStatus_UART_FramingError = 1011
- kStatus_UART_ParityError = 1012
- kStatus_UART_BaudrateNotSupport = 1013
- kStatus_UART_IdleLineDetected = 1014
- kStatus_NOTIFIER_ErrorNotificationBefore = 9800
- kStatus_NOTIFIER_ErrorNotificationAfter = 9801

Chapter 5
Deprecated List

Global CS42888_SetFuncMode (cs42888_handle_t *handle, cs42888_func_mode mode)

api, Do not use it anymore. It has been superceded by CS42888_SelectFunctionalMode.

Global ENET_StartExtC45SMIRead (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg)

Do not use this function. It has been superceded by ENET_StartExtC45SMIWriteReg and ENET_-StartExtC45SMIReadData.

Global ENET_StartExtC45SMIWrite (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg, uint32_t data)

Do not use this function. It has been superceded by ENET_StartExtC45SMIWriteReg and ENET_-StartExtC45SMIWriteData.

Global flexcan_clock_source_t

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

Global MMC_PowerOffCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)

Do not use this function. It has been superceded by MMC_SetCardPower.

Global MMC_PowerOnCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)

Do not use this function. It has been superceded by MMC_SetCardPower.

Global SAI_RxGetDefaultConfig (sai_config_t *config)

Do not use this function. It has been superceded by SAI_GetClassicI2SConfig, SAI_GetLeftJustified-Config , SAI_GetRightJustifiedConfig, SAI_GetDSPConfig, SAI_GetTDMConfig

Global SAI_RxInit (I2S_Type *base, const sai_config_t *config)

Do not use this function. It has been superceded by SAI_Init

Global SAI_RxSetFormat (I2S_Type *base, sai_transfer_format_t *format, uint32_t mclkSource-ClockHz, uint32_t bclkSourceClockHz)

Do not use this function. It has been superceded by SAI_RxSetConfig

Global SAI_TransferRxSetFormat (I2S_Type *base, sai_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Do not use this function. It has been superceded by SAI_TransferRxSetConfig

Global SAI_TransferTxSetFormat (I2S_Type *base, sai_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Do not use this function. It has been superceded by SAI_TransferTxSetConfig

Global SAI_TxGetDefaultConfig (sai_config_t *config)

Do not use this function. It has been superceded by SAI_GetClassicI2SConfig, SAI_GetLeftJustifiedConfig , SAI_GetRightJustifiedConfig, SAI_GetDSPConfig, SAI_GetTDMConfig

Global SAI_TxInit (I2S_Type *base, const sai_config_t *config)

Do not use this function. It has been superceded by SAI_Init

Global SAI_TxSetFormat (I2S_Type *base, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Do not use this function. It has been superceded by SAI_TxSetConfig

Global SD_Deinit (sd_card_t *card)

Do not use this function. It has been superceded by SD_HostDeinit,SD_CardDeinit. This function deinitializes the specific card and host.

Global SD_HostReset (SDMMCHOST_CONFIG *host)

Do not use this function. It has been superceded by SD_HostDoReset.

Global SD_Init (sd_card_t *card)

Do not use this function. It has been superceded by SD_HostInit,SD_CardInit.

Global SD_PowerOffCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)

Do not use this function. It has been superceded by SD_SetCardPower.

Global SD_PowerOnCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)

Do not use this function. It has been superceded by SD_SetCardPower.

Global SD_WaitCardDetectStatus (SDMMCHOST_TYPE *hostBase, const sdmmchost_detect_card_t *cd, bool waitCardStatus)

Do not use this function. It has been superceded by SD_PollingCardInsert.

Global SDIO_HostReset (SDMMCHOST_CONFIG *host)

Do not use this function. It has been superceded by SDIO_HostDoReset.

Global SDIO_PowerOffCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)

Do not use this function. It has been superceded by SDIO_SetCardPower.

Global SDIO_PowerOnCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)

Do not use this function. It has been superceded by SDIO_SetCardPower.

Global SDIO_WaitCardDetectStatus (SDMMCHOST_TYPE *hostBase, const sdmmchost_detect_card_t *cd, bool waitCardStatus)

Do not use this function. It has been superceded by SDIO_PollingCardInsert.

Global SDMMCHOST_PowerOffCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)

Do not use this function.It has been superceded by SDMMCHOST_SetCardPower..

Global SDMMCHOST_PowerOnCard (SDMMCHOST_TYPE ∗base, const sdmmchost_pwr_card_t ∗pwr)

Do not use this function.It has been superceded by SDMMCHOST_SetCardPower..

Class sdmmchost_pwr_card_t

Do not use this structure anymore.

Global SDMMCHOST_Reset (sdmmchost_t ∗host)

Do not use this function. Application should not call this function, driver is responsible for the host reset..

Global SDMMCHOST_WaitCardDetectStatus (SDMMCHOST_TYPE ∗hostBase, const sdmmchost_detect_card_t ∗cd, bool waitCardStatus)

Do not use this function.It has been superceded by SDMMCHOST_PollingCardDetectStatus..

Global WM8904_SetMasterSlave (wm8904_handle_t ∗handle, bool master)

DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY WM8904_SeMasterClock

Chapter 6
Clock Driver

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 MCG_INTERNAL_IRC_48M 48000000U
  IRC48M clock frequency in Hz.
- #define DMAMUX_CLOCKS
  Clock ip name array for DMAMUX.
- #define RTC_CLOCKS
  Clock ip name array for RTC.
- #define ENET_CLOCKS

Clock ip name array for ENET.
- #define PORT_CLOCKS
  Clock ip name array for PORT.
- #define SAI_CLOCKS
  Clock ip name array for SAI.
- #define FLEXBUS_CLOCKS
  Clock ip name array for FLEXBUS.
- #define TSI_CLOCKS
  Clock ip name array for TSI.
- #define LPUART_CLOCKS
  Clock ip name array for LPUART.
- #define EWM_CLOCKS
  Clock ip name array for EWM.
- #define PIT_CLOCKS
  Clock ip name array for PIT.
- #define DSPI_CLOCKS
  Clock ip name array for DSPI.
- #define LPTMR_CLOCKS
  Clock ip name array for LPTMR.
- #define SDHC_CLOCKS
  Clock ip name array for SDHC.
- #define FTM_CLOCKS
  Clock ip name array for FTM.
- #define EDMA_CLOCKS
  Clock ip name array for EDMA.
- #define FLEXCAN_CLOCKS
  Clock ip name array for FLEXCAN.
- #define DAC_CLOCKS
  Clock ip name array for DAC.
- #define ADC16_CLOCKS
  Clock ip name array for ADC16.
- #define SDRAM_CLOCKS
  Clock ip name array for SDRAM.
- #define SYSMPU_CLOCKS
  Clock ip name array for MPU.
- #define VREF_CLOCKS
  Clock ip name array for VREF.
- #define CMT_CLOCKS
  Clock ip name array for CMT.
- #define UART_CLOCKS
  Clock ip name array for UART.
- #define TPM_CLOCKS
  Clock ip name array for TPM.
- #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 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_FlexBusClk,
  kCLOCK_FlashClk,
  kCLOCK_FastPeriphClk,
  kCLOCK_PllFllSelClk,
  kCLOCK_Er32kClk,
  kCLOCK_Osc0ErClk,
  kCLOCK_Osc1ErClk,
  kCLOCK_Osc0ErClkUndiv,
  kCLOCK_McgFixedFreqClk,
  kCLOCK_McgInternalRefClk,
  kCLOCK_McgFllClk,
  kCLOCK_McgPll0Clk,
  kCLOCK_McgPll1Clk,
  kCLOCK_McgExtPllClk,
  kCLOCK_McgPeriphClk,
  kCLOCK_McgIrc48MClk,
  kCLOCK_LpoClk }
  Clock name used to get clock frequency.
- enum clock_usb_src_t {
  kCLOCK_UsbSrcPll0 = SIM_SOPT2_USBSRC(1U) | SIM_SOPT2_PLLFLLSEL(1U),
  kCLOCK_UsbSrcUsbPfd = SIM_SOPT2_USBSRC(1U) | SIM_SOPT2_PLLFLLSEL(2U),
  kCLOCK_UsbSrcIrc48M = SIM_SOPT2_USBSRC(1U) | SIM_SOPT2_PLLFLLSEL(3U),
  kCLOCK_UsbSrcExt = SIM_SOPT2_USBSRC(0U),
  kCLOCK_UsbSrcUnused = (int)0xFFFFFFFFU }
  USB clock source definition.
- enum clock_usb_phy_src_t { kCLOCK_UsbPhySrcExt = 0U }
  Source of the USB HS PHY.
- enum clock_usb_pfd_src_t {
  kCLOCK_UsbPfdSrcExt = 0U,
  kCLOCK_UsbPfdSrcFracDivBy4 = 1U,
  kCLOCK_UsbPfdSrcFracDivBy2 = 2U,
  kCLOCK_UsbPfdSrcFrac = 3U }

Source of the USB HS PFD clock (USB1PFDCLK)
- 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_PllRefOsc0,
  kMCG_PllRefOsc1 }
  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,
kMCG_OscselIrc }
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),
kMCG_ExtPllLostFlag = (1U << 9U) }
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_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_SetSdhc0Clock (uint32_t src)
  Set SDHC0 clock source.
- static void CLOCK_SetEnetTime0Clock (uint32_t src)
  Set enet timestamp clock source.
- static void CLOCK_SetRmii0Clock (uint32_t src)
  Set RMII clock source.
- static void CLOCK_SetLpuartClock (uint32_t src)
  Set LPUART clock source.
- static void CLOCK_SetTpmClock (uint32_t src)
  Set TPM clock source.
- static void CLOCK_SetTraceClock (uint32_t src, uint32_t divValue, uint32_t fracValue)
  Set debug trace clock source.
- static void CLOCK_SetPllFllSelClock (uint32_t src, uint32_t divValue, uint32_t fracValue)
  Set PLLFLLSEL clock source.
- static void CLOCK_SetClkOutClock (uint32_t src)
  Set CLKOUT source.
- static void CLOCK_SetRtcClkOutClock (uint32_t src)
  Set RTC_CLKOUT source.
- bool CLOCK_EnableUsbhs0Clock (clock_usb_src_t src, uint32_t freq)
  Enable USB HS clock.
- void CLOCK_DisableUsbhs0Clock (void)
  Disable USB HS clock.
- bool CLOCK_EnableUsbhs0PhyPllClock (clock_usb_phy_src_t src, uint32_t freq)
  Enable USB HS PHY PLL clock.
- void CLOCK_DisableUsbhs0PhyPllClock (void)
  Disable USB HS PHY PLL clock.
- void CLOCK_EnableUsbhs0PfdClock (uint8_t frac, clock_usb_pfd_src_t src)
  Enable USB HS PFD clock.
- void CLOCK_DisableUsbhs0PfdClock (void)
  Disable USB HS PFD clock.
- bool CLOCK_EnableUsbfs0Clock (clock_usb_src_t src, uint32_t freq)

Enable USB FS clock.
- static void CLOCK_DisableUsbfs0Clock (void)
  Disable USB FS clock.
- static void CLOCK_SetOutDiv (uint32_t outdiv1, uint32_t outdiv2, uint32_t outdiv3, uint32_t outdiv4)
  System clock divider.
- 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_GetFlexBusClkFreq (void)
  Get the flexbus clock frequency.
- uint32_t CLOCK_GetFlashClkFreq (void)
  Get the flash clock frequency.
- uint32_t CLOCK_GetPllFllSelClkFreq (void)
  Get the output clock frequency selected by SIM[PLLFLLSEL].
- 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).
- uint32_t CLOCK_GetOsc0ErClkDivFreq (void)
  Get the OSC0 external reference divided clock frequency.
- uint32_t CLOCK_GetOsc0ErClkUndivFreq (void)
  Get the OSC0 external reference undivided clock frequency (OSC0ERCLK_UNDIV).
- 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, 5, 2))
  CLOCK driver version 2.5.2.

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.
- uint32_t CLOCK_GetExtPllFreq (void)
  Gets the MCG external PLL frequency.
- void CLOCK_SetExtPllFreq (uint32_t freq)
  Sets the MCG external PLL 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.
- uint32_t CLOCK_CalcPllDiv (uint32_t refFreq, uint32_t desireFreq, uint8_t *prdiv, uint8_t *vdiv)
  Calculates the PLL divider setting for a desired output frequency.
- void CLOCK_SetPllClkSel (mcg_pll_clk_select_t pllcs)
  Set the PLL selection.

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.
- void CLOCK_SetExtPllMonitorMode (mcg_monitor_mode_t mode)
  Sets the external PLL 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_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_SetMcgConfig (mcg_config_t const *config)
  Sets the MCG to a target mode.

Data Structure Documentation

6.2.1 struct sim_clock_config_t

Data Fields

- uint8_t pllFllSel
  PLL/FLL/IRC48M selection.
- uint8_t pllFllDiv
  PLLFLLSEL clock divider divisor.
- uint8_t pllFllFrac
  PLLFLLSEL clock divider fraction.
- uint8_t er32kSrc
  ERCLK32K source selection.
- uint32_t clkdiv1
  SIM_CLKDIV1.

6.2.1.0.0.1 Field Documentation

6.2.1.0.0.1.1 uint8_t sim_clock_config_t::pllFllSel

6.2.1.0.0.1.2 uint8_t sim_clock_config_t::pllFllDiv

6.2.1.0.0.1.3 uint8_t sim_clock_config_t::pllFllFrac

6.2.1.0.0.1.4 uint8_t sim_clock_config_t::er32kSrc

6.2.1.0.0.1.5 uint32_t sim_clock_config_t::clkdiv1

6.2.2 struct oscer_config_t

Data Fields

- uint8_t enableMode
  OSCERCLK enable mode.
- uint8_t erclkDiv
  Divider for OSCERCLK.

6.2.2.0.0.2 Field Documentation

6.2.2.0.0.2.1 uint8_t oscer_config_t::enableMode

OR'ed value of _oscer_enable_mode.

6.2.2.0.0.2.2 uint8_t oscer_config_t::erclkDiv

6.2.3 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.

6.2.3.0.0.3 Field Documentation

6.2.3.0.0.3.1 uint32_t osc_config_t::freq

6.2.3.0.0.3.2 uint8_t osc_config_t::capLoad

6.2.3.0.0.3.3 osc_mode_t osc_config_t::workMode

6.2.3.0.0.3.4 oscer_config_t osc_config_t::oscerConfig

6.2.4 struct mcg_pll_config_t

Data Fields

- uint8_t enableMode
  Enable mode.
- uint8_t prdiv
  Reference divider PRDIV.
- uint8_t vdiv
  VCO divider VDIV.

6.2.4.0.0.4 Field Documentation

6.2.4.0.0.4.1 uint8_t mcg_pll_config_t::enableMode

OR'ed value of enumeration _mcg_pll_enable_mode.

6.2.4.0.0.4.2 uint8_t mcg_pll_config_t::prdiv

6.2.4.0.0.4.3 uint8_t mcg_pll_config_t::vdiv

6.2.5 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.
- mcg_pll_clk_select_t pllcs
  PLL select as output, PLLCS.

6.2.5.0.0.5 Field Documentation

6.2.5.0.0.5.1 mcg_mode_t mcg_config_t::mcgMode

6.2.5.0.0.5.2 uint8_t mcg_config_t::irclkEnableMode

6.2.5.0.0.5.3 mcg_irc_mode_t mcg_config_t::ircs

6.2.5.0.0.5.4 uint8_t mcg_config_t::fcrdiv

6.2.5.0.0.5.5 uint8_t mcg_config_t::frdiv

6.2.5.0.0.5.6 mcg_drs_t mcg_config_t::drs

6.2.5.0.0.5.7 mcg_dmx32_t mcg_config_t::dmx32

6.2.5.0.0.5.8 mcg_oscsel_t mcg_config_t::oscsel

6.2.5.0.0.5.9 mcg_pll_config_t mcg_config_t::pll0Config

6.2.5.0.0.5.10 mcg_pll_clk_select_t mcg_config_t::pllcs

Macro Definition Documentation

6.3.1 #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 MCGIRCLK 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.

6.3.2 #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.

6.3.3 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 5, 2))

6.3.4 #define MCG_INTERNAL_IRC_48M 48000000U

6.3.5 #define DMAMUX_CLOCKS

Value:

```
{                  \
        kCLOCK_Dmamux0 \
    }
```

6.3.6 #define RTC_CLOCKS

Value:

```
{               \
        kCLOCK_Rtc0 \
    }
```

6.3.7 #define ENET_CLOCKS

Value:

```
{                \
        kCLOCK_Enet0 \
    }
```

6.3.8 #define PORT_CLOCKS

Value:

```
{                                                                         \
        kCLOCK_PortA, kCLOCK_PortB, kCLOCK_PortC, kCLOCK_PortD, kCLOCK_PortE \
    }
```

6.3.9 #define SAI_CLOCKS

Value:

```
{               \
        kCLOCK_Sai0 \
    }
```

6.3.10 #define FLEXBUS_CLOCKS

Value:

```
{                   \
        kCLOCK_Flexbus0 \
    }
```

6.3.11 #define TSI_CLOCKS

Value:

```
{               \
        kCLOCK_Tsi0 \
    }
```

6.3.12 #define LPUART_CLOCKS

Value:

```
{                  \
        kCLOCK_Lpuart0 \
    }
```

6.3.13 #define EWM_CLOCKS

Value:

```
{               \
        kCLOCK_Ewm0 \
    }
```

6.3.14 #define PIT_CLOCKS

Value:

```
{               \
        kCLOCK_Pit0 \
    }
```

6.3.15 #define DSPI_CLOCKS

Value:

```
{                                         \
        kCLOCK_Spi0, kCLOCK_Spi1, kCLOCK_Spi2 \
    }
```

6.3.16 #define LPTMR_CLOCKS

Value:

```
{                 \
        kCLOCK_Lptmr0 \
    }
```

6.3.17 #define SDHC_CLOCKS

Value:

```
{                \
        kCLOCK_Sdhc0 \
    }
```

6.3.18 #define FTM_CLOCKS

Value:

```
{                                                           \
        kCLOCK_Ftm0, kCLOCK_Ftm1, kCLOCK_Ftm2, kCLOCK_Ftm3 \
    }
```

6.3.19 #define EDMA_CLOCKS

Value:

```
{               \
        kCLOCK_Dma0 \
    }
```

6.3.20 #define FLEXCAN_CLOCKS

Value:

```
{                                         \
        kCLOCK_Flexcan0, kCLOCK_Flexcan1 \
    }
```

6.3.21 #define DAC_CLOCKS

Value:

```
{                             \
        kCLOCK_Dac0, kCLOCK_Dac1 \
    }
```

6.3.22 #define ADC16_CLOCKS

Value:

```
{                             \
        kCLOCK_Adc0, kCLOCK_Adc1 \
    }
```

6.3.23 #define SDRAM_CLOCKS

Value:

```
{                  \
        kCLOCK_Sdramc0 \
    }
```

6.3.24 #define SYSMPU_CLOCKS

Value:

```
{                  \
        kCLOCK_Sysmpu0 \
    }
```

6.3.25 #define VREF_CLOCKS

Value:

```
{                \
        kCLOCK_Vref0 \
    }
```

6.3.26 #define CMT_CLOCKS

Value:

```
{               \
        kCLOCK_Cmt0 \
    }
```

6.3.27 #define UART_CLOCKS

Value:

```
{                                                                        \
        kCLOCK_Uart0, kCLOCK_Uart1, kCLOCK_Uart2, kCLOCK_Uart3, kCLOCK_Uart4 \
    }
```

6.3.28 #define TPM_CLOCKS

Value:

```
{                                               \
        kCLOCK_IpInvalid, kCLOCK_Tpm1, kCLOCK_Tpm2 \
    }
```

6.3.29 #define RNGA_CLOCKS

Value:

```
{                \
        kCLOCK_Rnga0 \
    }
```

6.3.30 #define CRC_CLOCKS

Value:

```
{               \
        kCLOCK_Crc0 \
    }
```

6.3.31 #define I2C_CLOCKS

Value:

```
{                                                       \
        kCLOCK_I2c0, kCLOCK_I2c1, kCLOCK_I2c2, kCLOCK_I2c3 \
    }
```

6.3.32 #define PDB_CLOCKS

Value:

```
{               \
        kCLOCK_Pdb0 \
    }
```

6.3.33 #define FTF_CLOCKS

Value:

```
{                \
        kCLOCK_Ftf0 \
    }
```

6.3.34 #define CMP_CLOCKS

Value:

```
{                                                      \
        kCLOCK_Cmp0, kCLOCK_Cmp1, kCLOCK_Cmp2, kCLOCK_Cmp3 \
    }
```

6.3.35 #define SYS_CLK kCLOCK_CoreSysClk

Enumeration Type Documentation

6.4.1 enum clock_name_t

Enumerator

kCLOCK_CoreSysClk Core/system clock.
kCLOCK_PlatClk Platform clock.
kCLOCK_BusClk Bus clock.
kCLOCK_FlexBusClk FlexBus clock.
kCLOCK_FlashClk Flash clock.
kCLOCK_FastPeriphClk Fast peripheral clock.
kCLOCK_PllFllSelClk The clock after SIM[PLLFLLSEL].
kCLOCK_Er32kClk External reference 32K clock (ERCLK32K)
kCLOCK_Osc0ErClk OSC0 external reference clock (OSC0ERCLK)
kCLOCK_Osc1ErClk OSC1 external reference clock (OSC1ERCLK)
kCLOCK_Osc0ErClkUndiv OSC0 external reference undivided clock(OSC0ERCLK_UNDIV).
kCLOCK_McgFixedFreqClk MCG fixed frequency clock (MCGFFCLK)
kCLOCK_McgInternalRefClk MCG internal reference clock (MCGIRCLK)
kCLOCK_McgFllClk MCGFLLCLK.
kCLOCK_McgPll0Clk MCGPLL0CLK.
kCLOCK_McgPll1Clk MCGPLL1CLK.
kCLOCK_McgExtPllClk EXT_PLLCLK.
kCLOCK_McgPeriphClk MCG peripheral clock (MCGPCLK)
kCLOCK_McgIrc48MClk MCG IRC48M clock.
kCLOCK_LpoClk LPO clock.

6.4.2 enum clock_usb_src_t

Enumerator

kCLOCK_UsbSrcPll0 Use PLL0.
kCLOCK_UsbSrcUsbPfd Use USBPFDCLK.
kCLOCK_UsbSrcIrc48M Use IRC48M.
kCLOCK_UsbSrcExt Use USB_CLKIN.
kCLOCK_UsbSrcUnused Used when the function does not care the clock source.

6.4.3 enum clock_usb_phy_src_t

Enumerator

kCLOCK_UsbPhySrcExt Use external crystal.

6.4.4 enum clock_usb_pfd_src_t

Enumerator

kCLOCK_UsbPfdSrcExt Use external crystal.
kCLOCK_UsbPfdSrcFracDivBy4 Use PFD_FRAC output divided by 4.
kCLOCK_UsbPfdSrcFracDivBy2 Use PFD_FRAC output divided by 2.
kCLOCK_UsbPfdSrcFrac Use PFD_FRAC output.

6.4.5 enum clock_ip_name_t

6.4.6 enum osc_mode_t

Enumerator

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

6.4.7 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

6.4.8 enum _oscer_enable_mode

Enumerator

kOSC_ErClkEnable Enable.
kOSC_ErClkEnableInStop Enable in stop mode.

6.4.9 enum mcg_fll_src_t

Enumerator

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

6.4.10 enum mcg_irc_mode_t

Enumerator

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

6.4.11 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.

6.4.12 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.

6.4.13 enum mcg_pll_ref_src_t

Enumerator

kMCG_PllRefOsc0 Selects OSC0 as PLL reference clock.
kMCG_PllRefOsc1 Selects OSC1 as PLL reference clock.

6.4.14 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.

6.4.15 enum mcg_atm_select_t

Enumerator

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

6.4.16 enum mcg_oscsel_t

Enumerator

kMCG_OscselOsc Selects System Oscillator (OSCCLK)
kMCG_OscselRtc Selects 32 kHz RTC Oscillator.
kMCG_OscselIrc Selects 48 MHz IRC Oscillator.

6.4.17 enum mcg_pll_clk_select_t

Enumerator

kMCG_PllClkSelPll0 PLL0 output clock is selected.

6.4.18 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.

6.4.19 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.

6.4.20 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.
kMCG_ExtPllLostFlag External PLL lost.

6.4.21 anonymous enum

Enumeration _mcg_irclk_enable_mode

Enumerator

kMCG_IrclkEnable MCGIRCLK enable.
kMCG_IrclkEnableInStop MCGIRCLK enable in stop mode.

6.4.22 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.

6.4.23 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_ModeError Unknown mode.

Function Documentation

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

Parameters

name	Which clock to enable, see clock_ip_name_t.

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

Parameters

name	Which clock to disable, see clock_ip_name_t.

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

Parameters

src	The value to set ERCLK32K clock source.

6.5.4 static void CLOCK_SetSdhc0Clock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set SDHC0 clock source.

6.5.5 static void CLOCK_SetEnetTime0Clock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set enet timestamp clock source.

6.5.6 static void CLOCK_SetRmii0Clock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set RMII clock source.

6.5.7 static void CLOCK_SetLpuartClock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set LPUART clock source.

6.5.8 static void CLOCK_SetTpmClock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set TPM clock source.

6.5.9 static void CLOCK_SetTraceClock (uint32_t *src*, uint32_t *divValue*, uint32_t *fracValue*) [inline], [static]

Parameters

src	The value to set debug trace clock source.
divValue	PLLFLL clock divider divisor.
fracValue	PLLFLL clock divider fraction.

6.5.10 static void CLOCK_SetPllFllSelClock (uint32_t *src*, uint32_t *divValue*, uint32_t *fracValue*) [inline], [static]

Parameters

src	The value to set PLLFLLSEL clock source.
divValue	PLLFLL clock divider divisor.
fracValue	PLLFLL clock divider fraction.

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

Parameters

src	The value to set CLKOUT source.

6.5.12 static void CLOCK_SetRtcClkOutClock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set RTC_CLKOUT source.

6.5.13 bool CLOCK_EnableUsbhs0Clock (clock_usb_src_t *src*, uint32_t *freq*)

This function only enables the access to USB HS prepheral, upper layer should first call the CLOCK_EnableUsbhs0PhyPllClock to enable the PHY clock to use USB HS.

Parameters

src	USB HS does not care about the clock source, here must be kCLOCK_UsbSrcUnused.
freq	USB HS does not care about the clock source, so this parameter is ignored.

Return values

true	The clock is set successfully.
false	The clock source is invalid to get proper USB HS clock.

6.5.14 void CLOCK_DisableUsbhs0Clock (void)

Disable USB HS clock, this function should not be called after CLOCK_DisableUsbhs0PhyPllClock.

6.5.15 bool CLOCK_EnableUsbhs0PhyPllClock (clock_usb_phy_src_t *src*, uint32_t *freq*)

This function enables the internal 480MHz USB PHY PLL clock.

Parameters

src	USB HS PHY PLL clock source.
freq	The frequency specified by src.

Return values

true	The clock is set successfully.
false	The clock source is invalid to get proper USB HS clock.

6.5.16 void CLOCK_DisableUsbhs0PhyPllClock (void)

This function disables USB HS PHY PLL clock.

6.5.17 void CLOCK_EnableUsbhs0PfdClock (uint8_t *frac*, clock_usb_pfd_src_t *src*)

This function enables USB HS PFD clock. It should be called after function CLOCK_EnableUsbhs0PhyPllClock. The PFD output clock is selected by the parameter `src`. When the `src` is kCLOCK_UsbPfdSrcExt, then the PFD outout is from external crystal directly, in this case, the `frac` is not used. In other cases, the PFD_FRAC output clock frequency is 480MHz∗18/frac, the PFD output frequency is based on the PFD_FRAC output.

Parameters

frac	The value set to PFD_FRAC, it must be in the range of 18 to 35.
src	Source of the USB HS PFD clock (USB1PFDCLK).

6.5.18 void CLOCK_DisableUsbhs0PfdClock (void)

This function disables USB HS PFD clock. It should be called before function CLOCK_DisableUsbhs0PhyPllClock.

6.5.19 bool CLOCK_EnableUsbfs0Clock (clock_usb_src_t *src*, uint32_t *freq*)

Parameters

src	USB FS clock source.
freq	The frequency specified by src.

Return values

true	The clock is set successfully.
false	The clock source is invalid to get proper USB FS clock.

6.5.20 static void CLOCK_DisableUsbfs0Clock (void) [inline], [static]

Disable USB FS clock.

6.5.21 static void CLOCK_SetOutDiv (uint32_t *outdiv1,* uint32_t *outdiv2,* uint32_t *outdiv3,* uint32_t *outdiv4*) [inline], [static]

Set the SIM_CLKDIV1[OUTDIV1], SIM_CLKDIV1[OUTDIV2], SIM_CLKDIV1[OUTDIV3], SIM_-CLKDIV1[OUTDIV4].

Parameters

outdiv1	Clock 1 output divider value.
outdiv2	Clock 2 output divider value.
outdiv3	Clock 3 output divider value.
outdiv4	Clock 4 output divider value.

6.5.22 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

6.5.23 uint32_t CLOCK_GetCoreSysClkFreq (void)

Returns

Clock frequency in Hz.

6.5.24 uint32_t CLOCK_GetPlatClkFreq (void)

Returns

Clock frequency in Hz.

6.5.25 uint32_t CLOCK_GetBusClkFreq (void)

Returns

Clock frequency in Hz.

6.5.26 uint32_t CLOCK_GetFlexBusClkFreq (void)

Returns

Clock frequency in Hz.

6.5.27 uint32_t CLOCK_GetFlashClkFreq (void)

Returns

Clock frequency in Hz.

6.5.28 uint32_t CLOCK_GetPllFllSelClkFreq (void)

Returns

Clock frequency in Hz.

6.5.29 uint32_t CLOCK_GetEr32kClkFreq (void)

Returns

Clock frequency in Hz.

6.5.30 uint32_t CLOCK_GetOsc0ErClkFreq (void)

Returns

Clock frequency in Hz.

6.5.31 uint32_t CLOCK_GetOsc0ErClkDivFreq (void)

Returns

Clock frequency in Hz.

6.5.32 uint32_t CLOCK_GetOsc0ErClkUndivFreq (void)

Returns

Clock frequency in Hz.

6.5.33 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.

6.5.34 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.

6.5.35 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.

6.5.36 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.

6.5.37 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.

6.5.38 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.

6.5.39 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.

6.5.40 uint32_t CLOCK_GetExtPllFreq (void)

This function gets the MCG external PLL frequency in Hz.

Returns

The frequency of the MCG external PLL.

6.5.41 void CLOCK_SetExtPllFreq (uint32_t *freq*)

This function sets the MCG external PLL frequency in Hz. The MCG external PLL frequency is passed to the MCG driver using this function. Call this function after the external PLL frequency is changed. Otherwise, the APIs, which are used to get the frequency, may return an incorrect value.

Parameters

freq	The frequency of MCG external PLL.

6.5.42 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.

6.5.43 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.

6.5.44 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_Source-Used	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.

6.5.45 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].

6.5.46 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.

6.5.47 static void CLOCK_DisablePll0 (void) [inline], [static]

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

6.5.48 uint32_t CLOCK_CalcPllDiv (uint32_t *refFreq*, uint32_t *desireFreq*, uint8_t * *prdiv*, uint8_t * *vdiv*)

This function calculates the correct reference clock divider (`PRDIV`) and VCO divider (`VDIV`) to generate a desired PLL output frequency. It returns the closest frequency match with the corresponding `PRDIV/VDIV` returned from parameters. If a desired frequency is not valid, this function returns 0.

Parameters

refFreq	PLL reference clock frequency.
desireFreq	Desired PLL output frequency.
prdiv	PRDIV value to generate desired PLL frequency.
vdiv	VDIV value to generate desired PLL frequency.

Returns

Closest frequency match that the PLL was able generate.

6.5.49 void CLOCK_SetPllClkSel (mcg_pll_clk_select_t *pllcs*)

This function sets the PLL selection between PLL0/PLL1/EXTPLL, and waits for change finished.

Parameters

pllcs	The PLL to select.

6.5.50 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.

6.5.51 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.

6.5.52 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.

6.5.53 void CLOCK_SetExtPllMonitorMode (mcg_monitor_mode_t *mode*)

This function ets the external PLL clock monitor mode. See mcg_monitor_mode_t for details.

Parameters

mode	Monitor mode to set.

6.5.54 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.

6.5.55 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.

6.5.56 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.

6.5.57 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);
```

6.5.58 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.

6.5.59 void CLOCK_DeinitOsc0 (void)

This function deinitializes the OSC0.

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

Parameters

freq	The XTAL0/EXTAL0 input clock frequency in Hz.

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

Parameters

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

6.5.62 void CLOCK_SetSlowIrcFreq (uint32_t *freq*)

Parameters

freq	The Slow IRC frequency input clock frequency in Hz.

6.5.63 void CLOCK_SetFastIrcFreq (uint32_t *freq*)

Parameters

freq	The Fast IRC frequency input clock frequency in Hz.

6.5.64 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_Atm-HardwareFail	Hardware fails while trimming.

6.5.65 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.

6.5.66 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_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 a frequency above 32768 Hz.

6.5.67 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.

6.5.68 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.

6.5.69 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.

6.5.70 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.

6.5.71 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.

6.5.72 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_PllClkSelExtPll, NULL);

6.5.73 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.

6.5.74 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.

6.5.75 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.

6.5.76 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.

6.5.77 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.

6.5.78 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_Source-Used	Could not change MCGIRCLK setting.
kStatus_Success	Switched to the target mode successfully.

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

6.5.80 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.

6.5.81 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.

Variable Documentation

6.6.1 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.

6.6.2 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.

Multipurpose Clock Generator (MCG)

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

6.7.1 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.

6.7.1.1 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 C-LOCK_GetOutClkFreq(), CLOCK_GetInternalRefClkFreq(), CLOCK_GetFixedFreqClkFreq(), CLOC-K_GetFllFreq(), CLOCK_GetPll0Freq(), CLOCK_GetPll1Freq(), and CLOCK_GetExtPllFreq(). These functions get the clock frequency based on the current MCG registers.

6.7.1.2 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 M-CGPLLCLK 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.

6.7.1.3 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.

6.7.1.4 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.

6.7.1.5 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.

6.7.1.6 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.

6.7.2 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.

6.7.2.1 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)

6.7.2.2 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)

6.7.2.3 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)

6.7.2.4 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)

6.7.2.5 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)

6.7.2.6 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)

6.7.3 Code Configuration Option

6.7.3.1 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 his 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 7
ADC16: 16-bit SAR Analog-to-Digital Converter Driver

Overview

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

Typical use case

7.2.1 Polling Configuration

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

7.2.2 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_COC-O_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_channel_mux_mode_t {
  kADC16_ChannelMuxA = 0U,
  kADC16_ChannelMuxB = 1U }
  Channel multiplexer mode for each channel.

- 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_ResolutionDF9Bit = kADC16_Resolution8or9Bit,
  kADC16_ResolutionDF13Bit = kADC16_Resolution12or13Bit,
  kADC16_ResolutionDF11Bit = kADC16_Resolution10or11Bit,
  kADC16_Resolution16Bit = 3U,
  kADC16_ResolutionSE16Bit = kADC16_Resolution16Bit,
  kADC16_ResolutionDF16Bit = 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, 2, 0))
  ADC16 driver version 2.2.0.

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_SetChannelMuxMode (ADC_Type ∗base, adc16_channel_mux_mode_t mode)
  Sets the channel mux 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.

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.

Data Structure Documentation

7.3.1 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.

7.3.1.0.0.6 Field Documentation

7.3.1.0.0.6.1 adc16_reference_voltage_source_t adc16_config_t::referenceVoltageSource

7.3.1.0.0.6.2 adc16_clock_source_t adc16_config_t::clockSource

7.3.1.0.0.6.3 bool adc16_config_t::enableAsynchronousClock

7.3.1.0.0.6.4 adc16_clock_divider_t adc16_config_t::clockDivider

7.3.1.0.0.6.5 adc16_resolution_t adc16_config_t::resolution

7.3.1.0.0.6.6 adc16_long_sample_mode_t adc16_config_t::longSampleMode

7.3.1.0.0.6.7 bool adc16_config_t::enableHighSpeed

7.3.1.0.0.6.8 bool adc16_config_t::enableLowPower

7.3.1.0.0.6.9 bool adc16_config_t::enableContinuousConversion

7.3.1.0.0.6.10 adc16_hardware_average_mode_t adc16_config_t::hardwareAverageMode

7.3.2 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.

7.3.2.0.0.7 Field Documentation

7.3.2.0.0.7.1 adc16_hardware_compare_mode_t adc16_hardware_compare_config_t::hardwareCompareMode

See "adc16_hardware_compare_mode_t".

7.3.2.0.0.7.2 int16_t adc16_hardware_compare_config_t::value1

7.3.2.0.0.7.3 int16_t adc16_hardware_compare_config_t::value2

7.3.3 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.
- bool enableDifferentialConversion
  Using Differential sample mode.

7.3.3.0.0.8 Field Documentation

7.3.3.0.0.8.1 uint32_t adc16_channel_config_t::channelNumber

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

7.3.3.0.0.8.2 bool adc16_channel_config_t::enableInterruptOnConversionCompleted

7.3.3.0.0.8.3 bool adc16_channel_config_t::enableDifferentialConversion

Macro Definition Documentation

7.4.1 #define FSL_ADC16_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

Enumeration Type Documentation

7.5.1 enum _adc16_channel_status_flags

Enumerator

kADC16_ChannelConversionDoneFlag Conversion done.

7.5.2 enum _adc16_status_flags

Enumerator

kADC16_ActiveFlag Converter is active.
kADC16_CalibrationFailedFlag Calibration is failed.

7.5.3 enum adc16_channel_mux_mode_t

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

Enumerator

kADC16_ChannelMuxA For channel with channel mux a.
kADC16_ChannelMuxB For channel with channel mux b.

7.5.4 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.

7.5.5 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_ResolutionDF9Bit Differential Sample 9-bit.
kADC16_ResolutionDF13Bit Differential Sample 13-bit.
kADC16_ResolutionDF11Bit Differential Sample 11-bit.
kADC16_Resolution16Bit Single End 16-bit or Differential Sample 16-bit.
kADC16_ResolutionSE16Bit Single End 16-bit.
kADC16_ResolutionDF16Bit Differential Sample 16-bit.

7.5.6 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.

7.5.7 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.

7.5.8 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.

7.5.9 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.

7.5.10 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.

Function Documentation

7.6.1 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".

7.6.2 void ADC16_Deinit (ADC_Type * *base*)

Parameters

base	ADC16 peripheral base address.

7.6.3 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    = true;
*   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.

7.6.4 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.

7.6.5 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.

7.6.6 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.

7.6.7 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.

7.6.8 void ADC16_SetChannelMuxMode (ADC_Type ∗ *base,* adc16_channel_mux_mode_t *mode*)

Some sample pins share the same channel index. The channel mux mode decides which pin is used for an indicated channel.

Parameters

base	ADC16 peripheral base address.
mode	Setting channel mux mode. See "adc16_channel_mux_mode_t".

7.6.9 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.

7.6.10 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".

7.6.11 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".

7.6.12 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".

7.6.13 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.

7.6.14 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.

7.6.15 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 8
CACHE: LMEM CACHE Memory Controller

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

The CACHE driver is created to help the user more easily operate the cache memory. The APIs for basic operations are including the following three levels: 1L. The L1 cache driver API. This level provides the level 1 caches controller drivers. The L1 caches in this arch is the previous the local memory controller (LMEM).

2L. The unified cache driver API. This level provides many APIs for unified cache driver APIs for combined L1 and L2 cache maintain operations. This is provided for SDK drivers (DMA, ENET, USDHC, etc) which should do the cache maintenance in their transactional APIs. Because in this arch, there is no L2 cache so the unified cache driver API directly calls only L1 driver APIs.

Function groups

8.1.1 L1 CACHE Operation

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

Chapter 9
CMP: Analog Comparator Driver

Overview

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

Typical use case

9.2.1 Polling Configuration

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

9.2.2 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.
- static void CMP_EnablePassThroughMode (CMP_Type ∗base, bool enable)
    Enables/disables the pass through 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.

Data Structure Documentation

9.3.1 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.

9.3.1.0.0.9 Field Documentation

9.3.1.0.0.9.1 bool cmp_config_t::enableCmp

9.3.1.0.0.9.2 cmp_hysteresis_mode_t cmp_config_t::hysteresisMode

9.3.1.0.0.9.3 bool cmp_config_t::enableHighSpeed

9.3.1.0.0.9.4 bool cmp_config_t::enableInvertOutput

9.3.1.0.0.9.5 bool cmp_config_t::useUnfilteredOutput

9.3.1.0.0.9.6 bool cmp_config_t::enablePinOut

9.3.1.0.0.9.7 bool cmp_config_t::enableTriggerMode

9.3.2 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.

9.3.2.0.0.10 Field Documentation

9.3.2.0.0.10.1 bool cmp_filter_config_t::enableSample

9.3.2.0.0.10.2 uint8_t cmp_filter_config_t::filterCount

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

9.3.2.0.0.10.3 uint8_t cmp_filter_config_t::filterPeriod

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

9.3.3 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.

9.3.3.0.0.11 Field Documentation

9.3.3.0.0.11.1 cmp_reference_voltage_source_t cmp_dac_config_t::referenceVoltageSource

9.3.3.0.0.11.2 uint8_t cmp_dac_config_t::DACValue

Available range is 0-63.

Macro Definition Documentation

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

Enumeration Type Documentation

9.5.1 enum _cmp_interrupt_enable

Enumerator

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

9.5.2 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.

9.5.3 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.

9.5.4 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.

Function Documentation

9.6.1 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.

9.6.2 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.

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

Parameters

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

9.6.4 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.

9.6.5 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.

9.6.6 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.

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

Parameters

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

9.6.8 static void CMP_EnablePassThroughMode (CMP_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

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

9.6.9 void CMP_SetFilterConfig (CMP_Type ∗ *base*, const cmp_filter_config_t ∗ *config*)

Parameters

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

9.6.10 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.

9.6.11 void CMP_EnableInterrupts (CMP_Type ∗ *base*, uint32_t *mask*)

Parameters

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

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

Parameters

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

9.6.13 uint32_t CMP_GetStatusFlags (CMP_Type * *base*)

Parameters

base	CMP peripheral base address.

Returns

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

9.6.14 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 10
CMT: Carrier Modulator Transmitter Driver

Overview

The carrier modulator transmitter (CMT) module provides the means to generate the protocol timing and carrier signals for a side variety of encoding schemes. The CMT incorporates hardware to off-load the critical and/or lengthy timing requirements associated with signal generation from the CPU. The MCUXpresso SDK provides a driver for the CMT module of the MCUXpresso SDK devices.

Clock formulas

The CMT module has internal clock dividers. It was originally designed to be based on an 8 MHz bus clock that can be divided by 1, 2, 4, or 8 according to the specification. To be compatible with a higher bus frequency, the primary prescaler (PPS) was developed to receive a higher frequency and generate a clock enable signal called an intermediate frequency (IF). The IF must be approximately equal to 8 MHz and works as a clock enable to the secondary prescaler. For the PPS, the prescaler is selected according to the bus clock to generate an intermediate clock approximate to 8 MHz and is selected as (bus_clock_hz/8000000). The secondary prescaler is the "cmtDivider". The clocks for the CMT module are listed below.

1. CMT clock frequency = bus_clock_Hz / (bus_clock_Hz / 8000000) / cmtDivider
2. CMT carrier and generator frequency = CMT clock frequency / (highCount1 + lowCount1)
   (In FSK mode, the second frequency = CMT clock frequency / (highCount2 + lowCount2))
3. CMT infrared output signal frequency
   a. In Time and Baseband mode
   CMT IRO signal mark time = (markCount + 1) / (CMT clock frequency / 8)
   CMT IRO signal space time = spaceCount / (CMT clock frequency / 8)
   b. In FSK mode
   CMT IRO signal mark time = (markCount + 1) / CMT carrier and generator frequency
   CMT IRO signal space time = spaceCount / CMT carrier and generator frequency

Typical use case

This is an example code to initialize data.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/cmt
This is an example IRQ handler to change the mark and space count to complete data modulation.

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

Data Structures

- struct cmt_modulate_config_t
  CMT carrier generator and modulator configuration structure. More...

- struct cmt_config_t
  CMT basic configuration structure. More...

Enumerations

- enum cmt_mode_t {
  kCMT_DirectIROCtl = 0x00U,
  kCMT_TimeMode = 0x01U,
  kCMT_FSKMode = 0x05U,
  kCMT_BasebandMode = 0x09U }
  The modes of CMT.
- enum cmt_primary_clkdiv_t {
  kCMT_PrimaryClkDiv1 = 0U,
  kCMT_PrimaryClkDiv2 = 1U,
  kCMT_PrimaryClkDiv3 = 2U,
  kCMT_PrimaryClkDiv4 = 3U,
  kCMT_PrimaryClkDiv5 = 4U,
  kCMT_PrimaryClkDiv6 = 5U,
  kCMT_PrimaryClkDiv7 = 6U,
  kCMT_PrimaryClkDiv8 = 7U,
  kCMT_PrimaryClkDiv9 = 8U,
  kCMT_PrimaryClkDiv10 = 9U,
  kCMT_PrimaryClkDiv11 = 10U,
  kCMT_PrimaryClkDiv12 = 11U,
  kCMT_PrimaryClkDiv13 = 12U,
  kCMT_PrimaryClkDiv14 = 13U,
  kCMT_PrimaryClkDiv15 = 14U,
  kCMT_PrimaryClkDiv16 = 15U }
  The CMT clock divide primary prescaler.
- enum cmt_second_clkdiv_t {
  kCMT_SecondClkDiv1 = 0U,
  kCMT_SecondClkDiv2 = 1U,
  kCMT_SecondClkDiv4 = 2U,
  kCMT_SecondClkDiv8 = 3U }
  The CMT clock divide secondary prescaler.
- enum cmt_infrared_output_polarity_t {
  kCMT_IROActiveLow = 0U,
  kCMT_IROActiveHigh = 1U }
  The CMT infrared output polarity.
- enum cmt_infrared_output_state_t {
  kCMT_IROCtlLow = 0U,
  kCMT_IROCtlHigh = 1U }
  The CMT infrared output signal state control.
- enum _cmt_interrupt_enable { kCMT_EndOfCycleInterruptEnable = CMT_MSC_EOCIE_MASK
  }
  CMT interrupt configuration structure, default settings all disabled.

Driver version

- #define FSL_CMT_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  CMT driver version 2.0.3.

Initialization and deinitialization

- void CMT_GetDefaultConfig (cmt_config_t ∗config)
  Gets the CMT default configuration structure.
- void CMT_Init (CMT_Type ∗base, const cmt_config_t ∗config, uint32_t busClock_Hz)
  Initializes the CMT module.
- void CMT_Deinit (CMT_Type ∗base)
  Disables the CMT module and gate control.

Basic Control Operations

- void CMT_SetMode (CMT_Type ∗base, cmt_mode_t mode, cmt_modulate_config_t ∗modulateConfig)
  Selects the mode for CMT.
- cmt_mode_t CMT_GetMode (CMT_Type ∗base)
  Gets the mode of the CMT module.
- uint32_t CMT_GetCMTFrequency (CMT_Type ∗base, uint32_t busClock_Hz)
  Gets the actual CMT clock frequency.
- static void CMT_SetCarrirGenerateCountOne (CMT_Type ∗base, uint32_t highCount, uint32_t lowCount)
  Sets the primary data set for the CMT carrier generator counter.
- static void CMT_SetCarrirGenerateCountTwo (CMT_Type ∗base, uint32_t highCount, uint32_t lowCount)
  Sets the secondary data set for the CMT carrier generator counter.
- void CMT_SetModulateMarkSpace (CMT_Type ∗base, uint32_t markCount, uint32_t spaceCount)
  Sets the modulation mark and space time period for the CMT modulator.
- static void CMT_EnableExtendedSpace (CMT_Type ∗base, bool enable)
  Enables or disables the extended space operation.
- void CMT_SetIroState (CMT_Type ∗base, cmt_infrared_output_state_t state)
  Sets the IRO (infrared output) signal state.
- static void CMT_EnableInterrupts (CMT_Type ∗base, uint32_t mask)
  Enables the CMT interrupt.
- static void CMT_DisableInterrupts (CMT_Type ∗base, uint32_t mask)
  Disables the CMT interrupt.
- static uint32_t CMT_GetStatusFlags (CMT_Type ∗base)
  Gets the end of the cycle status flag.

Data Structure Documentation

10.4.1 struct cmt_modulate_config_t

Data Fields

- uint8_t highCount1
  The high-time for carrier generator first register.

- uint8_t lowCount1
  The low-time for carrier generator first register.
- uint8_t highCount2
  The high-time for carrier generator second register for FSK mode.
- uint8_t lowCount2
  The low-time for carrier generator second register for FSK mode.
- uint16_t markCount
  The mark time for the modulator gate.
- uint16_t spaceCount
  The space time for the modulator gate.

10.4.1.0.0.12 Field Documentation

10.4.1.0.0.12.1 uint8_t cmt_modulate_config_t::highCount1

10.4.1.0.0.12.2 uint8_t cmt_modulate_config_t::lowCount1

10.4.1.0.0.12.3 uint8_t cmt_modulate_config_t::highCount2

10.4.1.0.0.12.4 uint8_t cmt_modulate_config_t::lowCount2

10.4.1.0.0.12.5 uint16_t cmt_modulate_config_t::markCount

10.4.1.0.0.12.6 uint16_t cmt_modulate_config_t::spaceCount

10.4.2 struct cmt_config_t

Data Fields

- bool isInterruptEnabled
  Timer interrupt 0-disable, 1-enable.
- bool isIroEnabled
  The IRO output 0-disabled, 1-enabled.
- cmt_infrared_output_polarity_t iroPolarity
  The IRO polarity.
- cmt_second_clkdiv_t divider
  The CMT clock divide prescaler.

10.4.2.0.0.13 Field Documentation

10.4.2.0.0.13.1 bool cmt_config_t::isInterruptEnabled

10.4.2.0.0.13.2 bool cmt_config_t::isIroEnabled

10.4.2.0.0.13.3 cmt_infrared_output_polarity_t cmt_config_t::iroPolarity

10.4.2.0.0.13.4 cmt_second_clkdiv_t cmt_config_t::divider

Macro Definition Documentation

10.5.1 #define FSL_CMT_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

Enumeration Type Documentation

10.6.1 enum cmt_mode_t

Enumerator

kCMT_DirectIROCtl Carrier modulator is disabled and the IRO signal is directly in software control.
kCMT_TimeMode Carrier modulator is enabled in time mode.
kCMT_FSKMode Carrier modulator is enabled in FSK mode.
kCMT_BasebandMode Carrier modulator is enabled in baseband mode.

10.6.2 enum cmt_primary_clkdiv_t

The primary clock divider is used to divider the bus clock to get the intermediate frequency to approximately equal to 8 MHZ. When the bus clock is 8 MHZ, set primary prescaler to "kCMT_PrimaryClkDiv1".

Enumerator

kCMT_PrimaryClkDiv1 The intermediate frequency is the bus clock divided by 1.
kCMT_PrimaryClkDiv2 The intermediate frequency is the bus clock divided by 2.
kCMT_PrimaryClkDiv3 The intermediate frequency is the bus clock divided by 3.
kCMT_PrimaryClkDiv4 The intermediate frequency is the bus clock divided by 4.
kCMT_PrimaryClkDiv5 The intermediate frequency is the bus clock divided by 5.
kCMT_PrimaryClkDiv6 The intermediate frequency is the bus clock divided by 6.
kCMT_PrimaryClkDiv7 The intermediate frequency is the bus clock divided by 7.
kCMT_PrimaryClkDiv8 The intermediate frequency is the bus clock divided by 8.
kCMT_PrimaryClkDiv9 The intermediate frequency is the bus clock divided by 9.
kCMT_PrimaryClkDiv10 The intermediate frequency is the bus clock divided by 10.
kCMT_PrimaryClkDiv11 The intermediate frequency is the bus clock divided by 11.
kCMT_PrimaryClkDiv12 The intermediate frequency is the bus clock divided by 12.
kCMT_PrimaryClkDiv13 The intermediate frequency is the bus clock divided by 13.
kCMT_PrimaryClkDiv14 The intermediate frequency is the bus clock divided by 14.
kCMT_PrimaryClkDiv15 The intermediate frequency is the bus clock divided by 15.
kCMT_PrimaryClkDiv16 The intermediate frequency is the bus clock divided by 16.

10.6.3 enum cmt_second_clkdiv_t

The second prescaler can be used to divide the 8 MHZ CMT clock by 1, 2, 4, or 8 according to the specification.

Enumerator

kCMT_SecondClkDiv1 The CMT clock is the intermediate frequency frequency divided by 1.
kCMT_SecondClkDiv2 The CMT clock is the intermediate frequency frequency divided by 2.
kCMT_SecondClkDiv4 The CMT clock is the intermediate frequency frequency divided by 4.
kCMT_SecondClkDiv8 The CMT clock is the intermediate frequency frequency divided by 8.

10.6.4 enum cmt_infrared_output_polarity_t

Enumerator

kCMT_IROActiveLow The CMT infrared output signal polarity is active-low.
kCMT_IROActiveHigh The CMT infrared output signal polarity is active-high.

10.6.5 enum cmt_infrared_output_state_t

Enumerator

kCMT_IROCtlLow The CMT Infrared output signal state is controlled to low.
kCMT_IROCtlHigh The CMT Infrared output signal state is controlled to high.

10.6.6 enum _cmt_interrupt_enable

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

Enumerator

kCMT_EndOfCycleInterruptEnable CMT end of cycle interrupt.

Function Documentation

10.7.1 void CMT_GetDefaultConfig (cmt_config_t * *config*)

This API gets the default configuration structure for the CMT_Init(). Use the initialized structure unchanged in CMT_Init() or modify fields of the structure before calling the CMT_Init().

Parameters

config	The CMT configuration structure pointer.

10.7.2 void CMT_Init (CMT_Type * *base*, const cmt_config_t * *config*, uint32_t *busClock_Hz*)

This function ungates the module clock and sets the CMT internal clock, interrupt, and infrared output signal for the CMT module.

Parameters

base	CMT peripheral base address.
config	The CMT basic configuration structure.
busClock_Hz	The CMT module input clock - bus clock frequency.

10.7.3 void CMT_Deinit (CMT_Type * *base*)

This function disables CMT modulator, interrupts, and gates the CMT clock control. CMT_Init must be called to use the CMT again.

Parameters

base	CMT peripheral base address.

10.7.4 void CMT_SetMode (CMT_Type * *base*, cmt_mode_t *mode*, cmt_modulate_config_t * *modulateConfig*)

Parameters

base	CMT peripheral base address.
mode	The CMT feature mode enumeration. See "cmt_mode_t".
modulateConfig	The carrier generation and modulator configuration.

10.7.5 cmt_mode_t CMT_GetMode (CMT_Type * *base*)

Parameters

base	CMT peripheral base address.

Returns

The CMT mode. kCMT_DirectIROCtl Carrier modulator is disabled; the IRO signal is directly in software control. kCMT_TimeMode Carrier modulator is enabled in time mode. kCMT_FSKMode Carrier modulator is enabled in FSK mode. kCMT_BasebandMode Carrier modulator is enabled in baseband mode.

10.7.6 uint32_t CMT_GetCMTFrequency (CMT_Type ∗ *base*, uint32_t *busClock_Hz*)

Parameters

base	CMT peripheral base address.
busClock_Hz	CMT module input clock - bus clock frequency.

Returns

The CMT clock frequency.

10.7.7 static void CMT_SetCarrirGenerateCountOne (CMT_Type ∗ *base*, uint32_t *highCount*, uint32_t *lowCount*) [inline], [static]

This function sets the high-time and low-time of the primary data set for the CMT carrier generator counter to control the period and the duty cycle of the output carrier signal. If the CMT clock period is Tcmt, the period of the carrier generator signal equals (highCount + lowCount) ∗ Tcmt. The duty cycle equals to highCount / (highCount + lowCount).

Parameters

base	CMT peripheral base address.
highCount	The number of CMT clocks for carrier generator signal high time, integer in the range of 1 ∼ 0xFF.

lowCount	The number of CMT clocks for carrier generator signal low time, integer in the range of 1 ∼ 0xFF.

10.7.8 static void CMT_SetCarrirGenerateCountTwo (CMT_Type ∗ *base,* uint32_t *highCount,* uint32_t *lowCount*) [inline], [static]

This function is used for FSK mode setting the high-time and low-time of the secondary data set CMT carrier generator counter to control the period and the duty cycle of the output carrier signal. If the CMT clock period is Tcmt, the period of the carrier generator signal equals (highCount + lowCount) ∗ Tcmt. The duty cycle equals highCount / (highCount + lowCount).

Parameters

base	CMT peripheral base address.
highCount	The number of CMT clocks for carrier generator signal high time, integer in the range of 1 ∼ 0xFF.
lowCount	The number of CMT clocks for carrier generator signal low time, integer in the range of 1 ∼ 0xFF.

10.7.9 void CMT_SetModulateMarkSpace (CMT_Type ∗ *base,* uint32_t *markCount,* uint32_t *spaceCount*)

This function sets the mark time period of the CMT modulator counter to control the mark time of the output modulated signal from the carrier generator output signal. If the CMT clock frequency is Fcmt and the carrier out signal frequency is fcg:

- In Time and Baseband mode: The mark period of the generated signal equals (markCount + 1) / (Fcmt/8). The space period of the generated signal equals spaceCount / (Fcmt/8).
- In FSK mode: The mark period of the generated signal equals (markCount + 1)/fcg. The space period of the generated signal equals spaceCount / fcg.

Parameters

base	Base address for current CMT instance.
markCount	The number of clock period for CMT modulator signal mark period, in the range of 0 ∼ 0xFFFF.
spaceCount	The number of clock period for CMT modulator signal space period, in the range of the 0 ∼ 0xFFFF.

10.7.10 static void CMT_EnableExtendedSpace (CMT_Type * *base*, bool *enable*) [inline], [static]

This function is used to make the space period longer for time, baseband, and FSK modes.

Parameters

base	CMT peripheral base address.
enable	True enable the extended space, false disable the extended space.

10.7.11 void CMT_SetIroState (CMT_Type * *base*, cmt_infrared_output_state_t *state*)

Changes the states of the IRO signal when the kCMT_DirectIROMode mode is set and the IRO signal is enabled.

Parameters

base	CMT peripheral base address.
state	The control of the IRO signal. See "cmt_infrared_output_state_t"

10.7.12 static void CMT_EnableInterrupts (CMT_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the CMT interrupts according to the provided mask if enabled. The CMT only has the end of the cycle interrupt - an interrupt occurs at the end of the modulator cycle. This interrupt provides a means for the user to reload the new mark/space values into the CMT modulator data registers and verify the modulator mark and space. For example, to enable the end of cycle, do the following.

```
*      CMT_EnableInterrupts(CMT,
       kCMT_EndOfCycleInterruptEnable);
*
```

Parameters

base	CMT peripheral base address.
mask	The interrupts to enable. Logical OR of _cmt_interrupt_enable.

10.7.13 static void CMT_DisableInterrupts (CMT_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the CMT interrupts according to the provided maskIf enabled. The CMT only has the end of the cycle interrupt. For example, to disable the end of cycle, do the following.

```
*      CMT_DisableInterrupts(CMT,
       kCMT_EndOfCycleInterruptEnable);
*
```

Parameters

base	CMT peripheral base address.
mask	The interrupts to enable. Logical OR of _cmt_interrupt_enable.

10.7.14 static uint32_t CMT_GetStatusFlags (CMT_Type * *base*) [inline], [static]

The flag is set:

- When the modulator is not currently active and carrier and modulator are set to start the initial CMT transmission.
- At the end of each modulation cycle when the counter is reloaded and the carrier and modulator are enabled.

  Parameters

base	CMT peripheral base address.

  Returns

  Current status of the end of cycle status flag
  - non-zero: End-of-cycle has occurred.
  - zero: End-of-cycle has not yet occurred since the flag last cleared.

Chapter 11
Common Driver

Overview

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

Macros

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

Typedefs

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

Enumerations

- enum _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_LOG = 154 }
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 = MAKE_STATUS(kStatusGroup_Generic, 6) }
Generic status return codes.

Functions

- static status_t EnableIRQ (IRQn_Type interrupt)
  Enable specific interrupt.
- static status_t DisableIRQ (IRQn_Type interrupt)
  Disable specific interrupt.
- static uint32_t DisableGlobalIRQ (void)
  Disable the global IRQ.
- static void EnableGlobalIRQ (uint32_t primask)
  Enable the global IRQ.
- void * 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, 2, 9))
  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)

Timer utilities

- #define USEC_TO_COUNT(us, clockFreqInHz) (uint64_t)(((uint64_t)(us) * (clockFreqInHz)) / 1000000U)
  Macro to convert a microsecond period to raw count value.

- #define COUNT_TO_USEC(count, clockFreqInHz) (uint64_t)((uint64_t)(count) * 1000000U / (clockFreqInHz))

  Macro to convert a raw count value to microsecond.
- #define MSEC_TO_COUNT(ms, clockFreqInHz) (uint64_t)((uint64_t)(ms) * (clockFreqInHz) / 1000U)

  Macro to convert a millisecond period to raw count value.
- #define COUNT_TO_MSEC(count, clockFreqInHz) (uint64_t)((uint64_t)(count) * 1000U / (clockFreqInHz))

  Macro to convert a raw count value to millisecond.

Alignment variable definition macros

- #define **SDK_ALIGN**(var, alignbytes) var
- #define **SDK_L1DCACHE_ALIGN**(var) var
- #define SDK_SIZEALIGN(var, alignbytes) ((unsigned int)((var) + ((alignbytes)-1U)) & (unsigned int)(~(unsigned int)((alignbytes)-1U)))

  Macro to change a value to a given size aligned value.

Non-cacheable region definition macros

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

Suppress fallthrough warning macro

- #define **SUPPRESS_FALL_THROUGH_WARNING**()

Atomic modification

These macros are used for atomic access, such as read-modify-write to the peripheral registers.

- SDK_ATOMIC_LOCAL_ADD
- SDK_ATOMIC_LOCAL_SET
- SDK_ATOMIC_LOCAL_CLEAR
- SDK_ATOMIC_LOCAL_TOGGLE
- SDK_ATOMIC_LOCAL_CLEAR_AND_SET

Take SDK_ATOMIC_LOCAL_CLEAR_AND_SET as an example: the parameter `addr` means the address of the peripheral register or variable you want to modify atomically, the parameter `clearBits` is the bits to clear, the parameter `setBits` it the bits to set. For example, to set a 32-bit register bit1:bit0 to 0b10, use like this:

```
volatile uint32_t * reg = (volatile uint32_t *)REG_ADDR;

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(reg, 0x03, 0x02);
```

In this example, the register bit1:bit0 are cleared and bit1 is set, as a result, register bit1:bit0 = 0b10.

Note

For the platforms don't support exclusive load and store, these macros disable the global interrupt to pretect the modification.
These macros only guarantee the local processor atomic operations. For the multi-processor devices, use hardware semaphore such as SEMA42 to guarantee exclusive access if necessary.

- #define **SDK_ATOMIC_LOCAL_ADD**(addr, val)
- #define **SDK_ATOMIC_LOCAL_SET**(addr, bits)
- #define **SDK_ATOMIC_LOCAL_CLEAR**(addr, bits)
- #define **SDK_ATOMIC_LOCAL_TOGGLE**(addr, bits)
- #define **SDK_ATOMIC_LOCAL_CLEAR_AND_SET**(addr, clearBits, setBits)

Macro Definition Documentation

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

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

11.2.3 #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 2, 9))

11.2.4 #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U

11.2.5 #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U

11.2.6 #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U

11.2.7 #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U

11.2.8 #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U

11.2.9 #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U

11.2.10 #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U

11.2.11 #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U

11.2.12 #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U

11.2.13 #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U

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

Typedef Documentation

11.3.1 typedef int32_t status_t

Enumeration Type Documentation

11.4.1 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_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.

11.4.2 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.

Function Documentation

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

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

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

Parameters

interrupt	The IRQ number.

Return values

kStatus_Success	Interrupt enabled successfully
kStatus_Fail	Failed to enable the interrupt

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

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

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

Parameters

interrupt	The IRQ number.

Return values

kStatus_Success	Interrupt disabled successfully
kStatus_Fail	Failed to disable the interrupt

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

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

Returns

Current primask value.

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

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

Parameters

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

11.5.5 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.

11.5.6 void SDK_Free (void * *ptr*)

Parameters

ptr	The memory to be release.

11.5.7 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 12
CRC: Cyclic Redundancy Check Driver

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.

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.

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.

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.

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.

Data Structure Documentation

12.6.1 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()

12.6.1.0.0.14 Field Documentation

12.6.1.0.0.14.1 uint32_t crc_config_t::polynomial

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

12.6.1.0.0.14.2 bool crc_config_t::reflectIn

12.6.1.0.0.14.3 bool crc_config_t::reflectOut

12.6.1.0.0.14.4 bool crc_config_t::complementChecksum

12.6.1.0.0.14.5 crc_bits_t crc_config_t::crcBits

Macro Definition Documentation

12.7.1 #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

12.7.2 #define CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT 1

Use CRC16-CCIT-FALSE as defeault.

Enumeration Type Documentation

12.8.1 enum crc_bits_t

Enumerator

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

12.8.2 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.

Function Documentation

12.9.1 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.

12.9.2 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.

12.9.3 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.

12.9.4 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.

12.9.5 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.

12.9.6 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 13
DAC: Digital-to-Analog Converter Driver

Overview

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

The DAC driver includes a basic DAC module (converter) and a DAC buffer.

The basic DAC module supports operations unique to the DAC converter in each DAC instance. The APIs in this section are used in the initialization phase, which enables the DAC module in the application. The APIs enable/disable the clock, enable/disable the module, and configure the converter. Call the initial APIs to prepare the DAC module for the application.

The DAC buffer operates the DAC hardware buffer. The DAC module supports a hardware buffer to keep a group of DAC values to be converted. This feature supports updating the DAC output value automatically by triggering the buffer read pointer to move in the buffer. Use the APIs to configure the hardware buffer's trigger mode, watermark, work mode, and use size. Additionally, the APIs operate the DMA, interrupts, flags, the pointer (the index of the buffer), item values, and so on.

Note that the most functional features are designed for the DAC hardware buffer.

Typical use case

13.2.1 Working as a basic DAC without the hardware buffer feature

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

13.2.2 Working with the hardware buffer

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

Data Structures

- struct dac_config_t
  DAC module configuration. More...
- struct dac_buffer_config_t
  DAC buffer configuration. More...

Enumerations

- enum _dac_buffer_status_flags {
  kDAC_BufferWatermarkFlag = DAC_SR_DACBFWMF_MASK,
  kDAC_BufferReadPointerTopPositionFlag = DAC_SR_DACBFRPTF_MASK,

kDAC_BufferReadPointerBottomPositionFlag = DAC_SR_DACBFRPBF_MASK }
DAC buffer flags.
- enum _dac_buffer_interrupt_enable {
kDAC_BufferWatermarkInterruptEnable = DAC_C0_DACBWIEN_MASK,
kDAC_BufferReadPointerTopInterruptEnable = DAC_C0_DACBTIEN_MASK,
kDAC_BufferReadPointerBottomInterruptEnable = DAC_C0_DACBBIEN_MASK }
DAC buffer interrupts.
- enum dac_reference_voltage_source_t {
kDAC_ReferenceVoltageSourceVref1 = 0U,
kDAC_ReferenceVoltageSourceVref2 = 1U }
DAC reference voltage source.
- enum dac_buffer_trigger_mode_t {
kDAC_BufferTriggerByHardwareMode = 0U,
kDAC_BufferTriggerBySoftwareMode = 1U }
DAC buffer trigger mode.
- enum dac_buffer_watermark_t {
kDAC_BufferWatermark1Word = 0U,
kDAC_BufferWatermark2Word = 1U,
kDAC_BufferWatermark3Word = 2U,
kDAC_BufferWatermark4Word = 3U }
DAC buffer watermark.
- enum dac_buffer_work_mode_t {
kDAC_BufferWorkAsNormalMode = 0U,
kDAC_BufferWorkAsSwingMode,
kDAC_BufferWorkAsOneTimeScanMode }
DAC buffer work mode.

Driver version

- #define FSL_DAC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
DAC driver version 2.0.2.

Initialization

- void DAC_Init (DAC_Type ∗base, const dac_config_t ∗config)
Initializes the DAC module.
- void DAC_Deinit (DAC_Type ∗base)
De-initializes the DAC module.
- void DAC_GetDefaultConfig (dac_config_t ∗config)
Initializes the DAC user configuration structure.
- static void DAC_Enable (DAC_Type ∗base, bool enable)
Enables the DAC module.

Buffer

- static void DAC_EnableBuffer (DAC_Type ∗base, bool enable)
Enables the DAC buffer.
- void DAC_SetBufferConfig (DAC_Type ∗base, const dac_buffer_config_t ∗config)
Configures the CMP buffer.

- void DAC_GetDefaultBufferConfig (dac_buffer_config_t *config)
  Initializes the DAC buffer configuration structure.
- static void DAC_EnableBufferDMA (DAC_Type *base, bool enable)
  Enables the DMA for DAC buffer.
- void DAC_SetBufferValue (DAC_Type *base, uint8_t index, uint16_t value)
  Sets the value for items in the buffer.
- static void DAC_DoSoftwareTriggerBuffer (DAC_Type *base)
  Triggers the buffer using software and updates the read pointer of the DAC buffer.
- static uint8_t DAC_GetBufferReadPointer (DAC_Type *base)
  Gets the current read pointer of the DAC buffer.
- void DAC_SetBufferReadPointer (DAC_Type *base, uint8_t index)
  Sets the current read pointer of the DAC buffer.
- void DAC_EnableBufferInterrupts (DAC_Type *base, uint32_t mask)
  Enables interrupts for the DAC buffer.
- void DAC_DisableBufferInterrupts (DAC_Type *base, uint32_t mask)
  Disables interrupts for the DAC buffer.
- uint8_t DAC_GetBufferStatusFlags (DAC_Type *base)
  Gets the flags of events for the DAC buffer.
- void DAC_ClearBufferStatusFlags (DAC_Type *base, uint32_t mask)
  Clears the flags of events for the DAC buffer.

Data Structure Documentation

13.3.1 struct dac_config_t

Data Fields

- dac_reference_voltage_source_t referenceVoltageSource
  Select the DAC reference voltage source.
- bool enableLowPowerMode
  Enable the low-power mode.

13.3.1.0.0.15 Field Documentation

13.3.1.0.0.15.1 dac_reference_voltage_source_t dac_config_t::referenceVoltageSource

13.3.1.0.0.15.2 bool dac_config_t::enableLowPowerMode

13.3.2 struct dac_buffer_config_t

Data Fields

- dac_buffer_trigger_mode_t triggerMode
  Select the buffer's trigger mode.
- dac_buffer_watermark_t watermark
  Select the buffer's watermark.
- dac_buffer_work_mode_t workMode
  Select the buffer's work mode.
- uint8_t upperLimit

Set the upper limit for the buffer index.

13.3.2.0.0.16 Field Documentation

13.3.2.0.0.16.1 dac_buffer_trigger_mode_t dac_buffer_config_t::triggerMode

13.3.2.0.0.16.2 dac_buffer_watermark_t dac_buffer_config_t::watermark

13.3.2.0.0.16.3 dac_buffer_work_mode_t dac_buffer_config_t::workMode

13.3.2.0.0.16.4 uint8_t dac_buffer_config_t::upperLimit

Normally, 0-15 is available for a buffer with 16 items.

Macro Definition Documentation

13.4.1 #define FSL_DAC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

Enumeration Type Documentation

13.5.1 enum _dac_buffer_status_flags

Enumerator

kDAC_BufferWatermarkFlag DAC Buffer Watermark Flag.
kDAC_BufferReadPointerTopPositionFlag DAC Buffer Read Pointer Top Position Flag.
kDAC_BufferReadPointerBottomPositionFlag DAC Buffer Read Pointer Bottom Position Flag.

13.5.2 enum _dac_buffer_interrupt_enable

Enumerator

kDAC_BufferWatermarkInterruptEnable DAC Buffer Watermark Interrupt Enable.
kDAC_BufferReadPointerTopInterruptEnable DAC Buffer Read Pointer Top Flag Interrupt Enable.
kDAC_BufferReadPointerBottomInterruptEnable DAC Buffer Read Pointer Bottom Flag Interrupt Enable.

13.5.3 enum dac_reference_voltage_source_t

Enumerator

kDAC_ReferenceVoltageSourceVref1 The DAC selects DACREF_1 as the reference voltage.
kDAC_ReferenceVoltageSourceVref2 The DAC selects DACREF_2 as the reference voltage.

13.5.4 enum dac_buffer_trigger_mode_t

Enumerator

kDAC_BufferTriggerByHardwareMode The DAC hardware trigger is selected.
kDAC_BufferTriggerBySoftwareMode The DAC software trigger is selected.

13.5.5 enum dac_buffer_watermark_t

Enumerator

kDAC_BufferWatermark1Word 1 word away from the upper limit.
kDAC_BufferWatermark2Word 2 words away from the upper limit.
kDAC_BufferWatermark3Word 3 words away from the upper limit.
kDAC_BufferWatermark4Word 4 words away from the upper limit.

13.5.6 enum dac_buffer_work_mode_t

Enumerator

kDAC_BufferWorkAsNormalMode Normal mode.
kDAC_BufferWorkAsSwingMode Swing mode.
kDAC_BufferWorkAsOneTimeScanMode One-Time Scan mode.

Function Documentation

13.6.1 void DAC_Init (DAC_Type * *base*, const dac_config_t * *config*)

This function initializes the DAC module including the following operations.

- Enabling the clock for DAC module.
- Configuring the DAC converter with a user configuration.
- Enabling the DAC module.

Parameters

base	DAC peripheral base address.
config	Pointer to the configuration structure. See "dac_config_t".

13.6.2 void DAC_Deinit (DAC_Type * *base*)

This function de-initializes the DAC module including the following operations.

- Disabling the DAC module.
- Disabling the clock for the DAC module.

Parameters

base	DAC peripheral base address.

13.6.3 void DAC_GetDefaultConfig (dac_config_t * *config*)

This function initializes the user configuration structure to a default value. The default values are as follows.

```
*   config->referenceVoltageSource = kDAC_ReferenceVoltageSourceVref2;
*   config->enableLowPowerMode = false;
*
```

Parameters

config	Pointer to the configuration structure. See "dac_config_t".

13.6.4 static void DAC_Enable (DAC_Type * *base,* bool *enable*) [inline], [static]

Parameters

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

13.6.5 static void DAC_EnableBuffer (DAC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	DAC peripheral base address.

enable	Enables or disables the feature.

13.6.6 void DAC_SetBufferConfig (DAC_Type * *base,* const dac_buffer_config_t * *config*)

Parameters

base	DAC peripheral base address.
config	Pointer to the configuration structure. See "dac_buffer_config_t".

13.6.7 void DAC_GetDefaultBufferConfig (dac_buffer_config_t * *config*)

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

```
*   config->triggerMode = kDAC_BufferTriggerBySoftwareMode;
*   config->watermark   = kDAC_BufferWatermark1Word;
*   config->workMode    = kDAC_BufferWorkAsNormalMode;
*   config->upperLimit  = DAC_DATL_COUNT - 1U;
*
```

Parameters

config	Pointer to the configuration structure. See "dac_buffer_config_t".

13.6.8 static void DAC_EnableBufferDMA (DAC_Type * *base,* bool *enable*) [inline], [static]

Parameters

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

13.6.9 void DAC_SetBufferValue (DAC_Type * *base,* uint8_t *index,* uint16_t *value*)

Parameters

base	DAC peripheral base address.
index	Setting the index for items in the buffer. The available index should not exceed the size of the DAC buffer.
value	Setting the value for items in the buffer. 12-bits are available.

13.6.10 static void DAC_DoSoftwareTriggerBuffer (DAC_Type * *base*) [inline], [static]

This function triggers the function using software. The read pointer of the DAC buffer is updated with one step after this function is called. Changing the read pointer depends on the buffer's work mode.

Parameters

base	DAC peripheral base address.

13.6.11 static uint8_t DAC_GetBufferReadPointer (DAC_Type * *base*) [inline], [static]

This function gets the current read pointer of the DAC buffer. The current output value depends on the item indexed by the read pointer. It is updated either by a software trigger or a hardware trigger.

Parameters

base	DAC peripheral base address.

Returns

The current read pointer of the DAC buffer.

13.6.12 void DAC_SetBufferReadPointer (DAC_Type * *base,* uint8_t *index*)

This function sets the current read pointer of the DAC buffer. The current output value depends on the item indexed by the read pointer. It is updated either by a software trigger or a hardware trigger. After the read pointer changes, the DAC output value also changes.

Parameters

base	DAC peripheral base address.
index	Setting an index value for the pointer.

13.6.13 void DAC_EnableBufferInterrupts (DAC_Type * *base*, uint32_t *mask*)

Parameters

base	DAC peripheral base address.
mask	Mask value for interrupts. See "_dac_buffer_interrupt_enable".

13.6.14 void DAC_DisableBufferInterrupts (DAC_Type * *base*, uint32_t *mask*)

Parameters

base	DAC peripheral base address.
mask	Mask value for interrupts. See "_dac_buffer_interrupt_enable".

13.6.15 uint8_t DAC_GetBufferStatusFlags (DAC_Type * *base*)

Parameters

base	DAC peripheral base address.

Returns

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

13.6.16 void DAC_ClearBufferStatusFlags (DAC_Type * *base*, uint32_t *mask*)

Parameters

base	DAC peripheral base address.
mask	Mask value for flags. See "_dac_buffer_status_flags_t".

Chapter 14
DMAMUX: Direct Memory Access Multiplexer Driver

Overview

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

Typical use case

14.2.1 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.

Macro Definition Documentation

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

Function Documentation

14.4.1 void DMAMUX_Init (DMAMUX_Type * *base*)

This function ungates the DMAMUX clock.

Parameters

base	DMAMUX peripheral base address.

14.4.2 void DMAMUX_Deinit (DMAMUX_Type * *base*)

This function gates the DMAMUX clock.

Parameters

base	DMAMUX peripheral base address.

14.4.3 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.

14.4.4 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.

14.4.5 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.

14.4.6 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.

14.4.7 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 15
DSPI: Serial Peripheral Interface Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the Serial Peripheral Interface (SPI) module of MCUXpresso SDK devices.

Modules

- DSPI Driver

DSPI Driver

15.2.1 Overview

This section describes the programming interface of the DSPI peripheral driver. The DSPI driver configures the DSPI module and provides functional and transactional interfaces to build the DSPI application.

15.2.2 Typical use case

15.2.2.1 Master Operation

Refer to the driver examples codes located at *<SDK_ROOT>/boards/<BOARD>/driver_examples/dspi*.

15.2.2.2 Slave Operation

Refer to the driver examples codes located at *<SDK_ROOT>/boards/<BOARD>/driver_examples/dspi*.

Data Structures

- struct dspi_command_data_config_t
  DSPI master command date configuration used for the SPIx_PUSHR. More...
- struct dspi_master_ctar_config_t
  DSPI master ctar configuration structure. More...
- struct dspi_master_config_t
  DSPI master configuration structure. More...
- struct dspi_slave_ctar_config_t
  DSPI slave ctar configuration structure. More...
- struct dspi_slave_config_t
  DSPI slave configuration structure. More...
- struct dspi_transfer_t
  DSPI master/slave transfer structure. More...
- struct dspi_half_duplex_transfer_t
  DSPI half-duplex(master) transfer structure. More...
- struct dspi_master_handle_t
  DSPI master transfer handle structure used for transactional API. More...
- struct dspi_slave_handle_t
  DSPI slave transfer handle structure used for the transactional API. More...

Macros

- #define DSPI_DUMMY_DATA (0x00U)
  DSPI dummy data if there is no Tx data.
- #define DSPI_MASTER_CTAR_SHIFT (0U)
  DSPI master CTAR shift macro; used internally.
- #define DSPI_MASTER_CTAR_MASK (0x0FU)

DSPI master CTAR mask macro; used internally.
- #define DSPI_MASTER_PCS_SHIFT (4U)
  DSPI master PCS shift macro; used internally.
- #define DSPI_MASTER_PCS_MASK (0xF0U)
  DSPI master PCS mask macro; used internally.
- #define DSPI_SLAVE_CTAR_SHIFT (0U)
  DSPI slave CTAR shift macro; used internally.
- #define DSPI_SLAVE_CTAR_MASK (0x07U)
  DSPI slave CTAR mask macro; used internally.

Typedefs

- typedef void(∗ dspi_master_transfer_callback_t)(SPI_Type ∗base, dspi_master_handle_t ∗handle, status_t status, void ∗userData)
  Completion callback function pointer type.
- typedef void(∗ dspi_slave_transfer_callback_t)(SPI_Type ∗base, dspi_slave_handle_t ∗handle, status_t status, void ∗userData)
  Completion callback function pointer type.

Enumerations

- enum {
  kStatus_DSPI_Busy = MAKE_STATUS(kStatusGroup_DSPI, 0),
  kStatus_DSPI_Error = MAKE_STATUS(kStatusGroup_DSPI, 1),
  kStatus_DSPI_Idle = MAKE_STATUS(kStatusGroup_DSPI, 2),
  kStatus_DSPI_OutOfRange = MAKE_STATUS(kStatusGroup_DSPI, 3) }
  Status for the DSPI driver.
- enum _dspi_flags {
  kDSPI_TxCompleteFlag = (int)SPI_SR_TCF_MASK,
  kDSPI_EndOfQueueFlag = SPI_SR_EOQF_MASK,
  kDSPI_TxFifoUnderflowFlag = SPI_SR_TFUF_MASK,
  kDSPI_TxFifoFillRequestFlag = SPI_SR_TFFF_MASK,
  kDSPI_RxFifoOverflowFlag = SPI_SR_RFOF_MASK,
  kDSPI_RxFifoDrainRequestFlag = SPI_SR_RFDF_MASK,
  kDSPI_TxAndRxStatusFlag = SPI_SR_TXRXS_MASK,
  kDSPI_AllStatusFlag }
  DSPI status flags in SPIx_SR register.
- enum _dspi_interrupt_enable {
  kDSPI_TxCompleteInterruptEnable = (int)SPI_RSER_TCF_RE_MASK,
  kDSPI_EndOfQueueInterruptEnable = SPI_RSER_EOQF_RE_MASK,
  kDSPI_TxFifoUnderflowInterruptEnable = SPI_RSER_TFUF_RE_MASK,
  kDSPI_TxFifoFillRequestInterruptEnable = SPI_RSER_TFFF_RE_MASK,
  kDSPI_RxFifoOverflowInterruptEnable = SPI_RSER_RFOF_RE_MASK,
  kDSPI_RxFifoDrainRequestInterruptEnable = SPI_RSER_RFDF_RE_MASK,
  kDSPI_AllInterruptEnable }

DSPI interrupt source.
- enum _dspi_dma_enable {
  kDSPI_TxDmaEnable = (SPI_RSER_TFFF_RE_MASK | SPI_RSER_TFFF_DIRS_MASK),
  kDSPI_RxDmaEnable = (SPI_RSER_RFDF_RE_MASK | SPI_RSER_RFDF_DIRS_MASK) }
  DSPI DMA source.
- enum dspi_master_slave_mode_t {
  kDSPI_Master = 1U,
  kDSPI_Slave = 0U }
  DSPI master or slave mode configuration.
- enum dspi_master_sample_point_t {
  kDSPI_SckToSin0Clock = 0U,
  kDSPI_SckToSin1Clock = 1U,
  kDSPI_SckToSin2Clock = 2U }
  DSPI Sample Point: Controls when the DSPI master samples SIN in the Modified Transfer Format.
- enum dspi_which_pcs_t {
  kDSPI_Pcs0 = 1U << 0,
  kDSPI_Pcs1 = 1U << 1,
  kDSPI_Pcs2 = 1U << 2,
  kDSPI_Pcs3 = 1U << 3,
  kDSPI_Pcs4 = 1U << 4,
  kDSPI_Pcs5 = 1U << 5 }
  DSPI Peripheral Chip Select (Pcs) configuration (which Pcs to configure).
- enum dspi_pcs_polarity_config_t {
  kDSPI_PcsActiveHigh = 0U,
  kDSPI_PcsActiveLow = 1U }
  DSPI Peripheral Chip Select (Pcs) Polarity configuration.
- enum _dspi_pcs_polarity {
  kDSPI_Pcs0ActiveLow = 1U << 0,
  kDSPI_Pcs1ActiveLow = 1U << 1,
  kDSPI_Pcs2ActiveLow = 1U << 2,
  kDSPI_Pcs3ActiveLow = 1U << 3,
  kDSPI_Pcs4ActiveLow = 1U << 4,
  kDSPI_Pcs5ActiveLow = 1U << 5,
  kDSPI_PcsAllActiveLow = 0xFFU }
  DSPI Peripheral Chip Select (Pcs) Polarity.
- enum dspi_clock_polarity_t {
  kDSPI_ClockPolarityActiveHigh = 0U,
  kDSPI_ClockPolarityActiveLow = 1U }
  DSPI clock polarity configuration for a given CTAR.
- enum dspi_clock_phase_t {
  kDSPI_ClockPhaseFirstEdge = 0U,
  kDSPI_ClockPhaseSecondEdge = 1U }
  DSPI clock phase configuration for a given CTAR.
- enum dspi_shift_direction_t {
  kDSPI_MsbFirst = 0U,
  kDSPI_LsbFirst = 1U }
  DSPI data shifter direction options for a given CTAR.

- enum dspi_delay_type_t {
kDSPI_PcsToSck = 1U,
kDSPI_LastSckToPcs,
kDSPI_BetweenTransfer }
  DSPI delay type selection.
- enum dspi_ctar_selection_t {
kDSPI_Ctar0 = 0U,
kDSPI_Ctar1 = 1U,
kDSPI_Ctar2 = 2U,
kDSPI_Ctar3 = 3U,
kDSPI_Ctar4 = 4U,
kDSPI_Ctar5 = 5U,
kDSPI_Ctar6 = 6U,
kDSPI_Ctar7 = 7U }
  DSPI Clock and Transfer Attributes Register (CTAR) selection.
- enum _dspi_transfer_config_flag_for_master {
kDSPI_MasterCtar0 = 0U << DSPI_MASTER_CTAR_SHIFT,
kDSPI_MasterCtar1 = 1U << DSPI_MASTER_CTAR_SHIFT,
kDSPI_MasterCtar2 = 2U << DSPI_MASTER_CTAR_SHIFT,
kDSPI_MasterCtar3 = 3U << DSPI_MASTER_CTAR_SHIFT,
kDSPI_MasterCtar4 = 4U << DSPI_MASTER_CTAR_SHIFT,
kDSPI_MasterCtar5 = 5U << DSPI_MASTER_CTAR_SHIFT,
kDSPI_MasterCtar6 = 6U << DSPI_MASTER_CTAR_SHIFT,
kDSPI_MasterCtar7 = 7U << DSPI_MASTER_CTAR_SHIFT,
kDSPI_MasterPcs0 = 0U << DSPI_MASTER_PCS_SHIFT,
kDSPI_MasterPcs1 = 1U << DSPI_MASTER_PCS_SHIFT,
kDSPI_MasterPcs2 = 2U << DSPI_MASTER_PCS_SHIFT,
kDSPI_MasterPcs3 = 3U << DSPI_MASTER_PCS_SHIFT,
kDSPI_MasterPcs4 = 4U << DSPI_MASTER_PCS_SHIFT,
kDSPI_MasterPcs5 = 5U << DSPI_MASTER_PCS_SHIFT,
kDSPI_MasterPcsContinuous = 1U << 20,
kDSPI_MasterActiveAfterTransfer = 1U << 21 }
  Use this enumeration for the DSPI master transfer configFlags.
- enum _dspi_transfer_config_flag_for_slave { kDSPI_SlaveCtar0 = 0U << DSPI_SLAVE_CTAR-_SHIFT }
  Use this enumeration for the DSPI slave transfer configFlags.
- enum _dspi_transfer_state {
kDSPI_Idle = 0x0U,
kDSPI_Busy,
kDSPI_Error }
  DSPI transfer state, which is used for DSPI transactional API state machine.

Variables

- volatile uint8_t g_dspiDummyData [ ]

Global variable for dummy data value setting.

Driver version

- #define FSL_DSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))
  DSPI driver version 2.2.4.

Initialization and deinitialization

- void DSPI_MasterInit (SPI_Type ∗base, const dspi_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes the DSPI master.
- void DSPI_MasterGetDefaultConfig (dspi_master_config_t ∗masterConfig)
  Sets the dspi_master_config_t structure to default values.
- void DSPI_SlaveInit (SPI_Type ∗base, const dspi_slave_config_t ∗slaveConfig)
  DSPI slave configuration.
- void DSPI_SlaveGetDefaultConfig (dspi_slave_config_t ∗slaveConfig)
  Sets the dspi_slave_config_t structure to a default value.
- void DSPI_Deinit (SPI_Type ∗base)
  De-initializes the DSPI peripheral.
- static void DSPI_Enable (SPI_Type ∗base, bool enable)
  Enables the DSPI peripheral and sets the MCR MDIS to 0.

Status

- static uint32_t DSPI_GetStatusFlags (SPI_Type ∗base)
  Gets the DSPI status flag state.
- static void DSPI_ClearStatusFlags (SPI_Type ∗base, uint32_t statusFlags)
  Clears the DSPI status flag.

Interrupts

- void DSPI_EnableInterrupts (SPI_Type ∗base, uint32_t mask)
  Enables the DSPI interrupts.
- static void DSPI_DisableInterrupts (SPI_Type ∗base, uint32_t mask)
  Disables the DSPI interrupts.

DMA Control

- static void DSPI_EnableDMA (SPI_Type ∗base, uint32_t mask)
  Enables the DSPI DMA request.
- static void DSPI_DisableDMA (SPI_Type ∗base, uint32_t mask)
  Disables the DSPI DMA request.
- static uint32_t DSPI_MasterGetTxRegisterAddress (SPI_Type ∗base)

Gets the DSPI master PUSHR data register address for the DMA operation.
- static uint32_t DSPI_SlaveGetTxRegisterAddress (SPI_Type *base)
  Gets the DSPI slave PUSHR data register address for the DMA operation.
- static uint32_t DSPI_GetRxRegisterAddress (SPI_Type *base)
  Gets the DSPI POPR data register address for the DMA operation.

Bus Operations

- uint32_t DSPI_GetInstance (SPI_Type *base)
  Get instance number for DSPI module.
- static void DSPI_SetMasterSlaveMode (SPI_Type *base, dspi_master_slave_mode_t mode)
  Configures the DSPI for master or slave.
- static bool DSPI_IsMaster (SPI_Type *base)
  Returns whether the DSPI module is in master mode.
- static void DSPI_StartTransfer (SPI_Type *base)
  Starts the DSPI transfers and clears HALT bit in MCR.
- static void DSPI_StopTransfer (SPI_Type *base)
  Stops DSPI transfers and sets the HALT bit in MCR.
- static void DSPI_SetFifoEnable (SPI_Type *base, bool enableTxFifo, bool enableRxFifo)
  Enables or disables the DSPI FIFOs.
- static void DSPI_FlushFifo (SPI_Type *base, bool flushTxFifo, bool flushRxFifo)
  Flushes the DSPI FIFOs.
- static void DSPI_SetAllPcsPolarity (SPI_Type *base, uint32_t mask)
  Configures the DSPI peripheral chip select polarity simultaneously.
- uint32_t DSPI_MasterSetBaudRate (SPI_Type *base, dspi_ctar_selection_t whichCtar, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the DSPI baud rate in bits per second.
- void DSPI_MasterSetDelayScaler (SPI_Type *base, dspi_ctar_selection_t whichCtar, uint32_t prescaler, uint32_t scaler, dspi_delay_type_t whichDelay)
  Manually configures the delay prescaler and scaler for a particular CTAR.
- uint32_t DSPI_MasterSetDelayTimes (SPI_Type *base, dspi_ctar_selection_t whichCtar, dspi_delay_type_t whichDelay, uint32_t srcClock_Hz, uint32_t delayTimeInNanoSec)
  Calculates the delay prescaler and scaler based on the desired delay input in nanoseconds.
- static void DSPI_MasterWriteData (SPI_Type *base, dspi_command_data_config_t *command, uint16_t data)
  Writes data into the data buffer for master mode.
- void DSPI_GetDefaultDataCommandConfig (dspi_command_data_config_t *command)
  Sets the dspi_command_data_config_t structure to default values.
- void DSPI_MasterWriteDataBlocking (SPI_Type *base, dspi_command_data_config_t *command, uint16_t data)
  Writes data into the data buffer master mode and waits till complete to return.
- static uint32_t DSPI_MasterGetFormattedCommand (dspi_command_data_config_t *command)
  Returns the DSPI command word formatted to the PUSHR data register bit field.
- void DSPI_MasterWriteCommandDataBlocking (SPI_Type *base, uint32_t data)
  Writes a 32-bit data word (16-bit command appended with 16-bit data) into the data buffer master mode and waits till complete to return.
- static void DSPI_SlaveWriteData (SPI_Type *base, uint32_t data)
  Writes data into the data buffer in slave mode.
- void DSPI_SlaveWriteDataBlocking (SPI_Type *base, uint32_t data)

Writes data into the data buffer in slave mode, waits till data was transmitted, and returns.
- static uint32_t DSPI_ReadData (SPI_Type *base)
  Reads data from the data buffer.
- void DSPI_SetDummyData (SPI_Type *base, uint8_t dummyData)
  Set up the dummy data.

Transactional APIs

- void DSPI_MasterTransferCreateHandle (SPI_Type *base, dspi_master_handle_t *handle, dspi_master_transfer_callback_t callback, void *userData)
  Initializes the DSPI master handle.
- status_t DSPI_MasterTransferBlocking (SPI_Type *base, dspi_transfer_t *transfer)
  DSPI master transfer data using polling.
- status_t DSPI_MasterTransferNonBlocking (SPI_Type *base, dspi_master_handle_t *handle, dspi_transfer_t *transfer)
  DSPI master transfer data using interrupts.
- status_t DSPI_MasterHalfDuplexTransferBlocking (SPI_Type *base, dspi_half_duplex_transfer_t *xfer)
  Transfers a block of data using a polling method.
- status_t DSPI_MasterHalfDuplexTransferNonBlocking (SPI_Type *base, dspi_master_handle_t *handle, dspi_half_duplex_transfer_t *xfer)
  Performs a non-blocking DSPI interrupt transfer.
- status_t DSPI_MasterTransferGetCount (SPI_Type *base, dspi_master_handle_t *handle, size_t *count)
  Gets the master transfer count.
- void DSPI_MasterTransferAbort (SPI_Type *base, dspi_master_handle_t *handle)
  DSPI master aborts a transfer using an interrupt.
- void DSPI_MasterTransferHandleIRQ (SPI_Type *base, dspi_master_handle_t *handle)
  DSPI Master IRQ handler function.
- void DSPI_SlaveTransferCreateHandle (SPI_Type *base, dspi_slave_handle_t *handle, dspi_slave_transfer_callback_t callback, void *userData)
  Initializes the DSPI slave handle.
- status_t DSPI_SlaveTransferNonBlocking (SPI_Type *base, dspi_slave_handle_t *handle, dspi_transfer_t *transfer)
  DSPI slave transfers data using an interrupt.
- status_t DSPI_SlaveTransferGetCount (SPI_Type *base, dspi_slave_handle_t *handle, size_t *count)
  Gets the slave transfer count.
- void DSPI_SlaveTransferAbort (SPI_Type *base, dspi_slave_handle_t *handle)
  DSPI slave aborts a transfer using an interrupt.
- void DSPI_SlaveTransferHandleIRQ (SPI_Type *base, dspi_slave_handle_t *handle)
  DSPI Master IRQ handler function.
- uint8_t DSPI_GetDummyDataInstance (SPI_Type *base)
  brief Dummy data for each instance.

15.2.3 Data Structure Documentation

15.2.3.1 struct dspi_command_data_config_t

Data Fields

- bool isPcsContinuous
  Option to enable the continuous assertion of the chip select between transfers.
- uint8_t whichCtar
  The desired Clock and Transfer Attributes Register (CTAR) to use for CTAS.
- uint8_t whichPcs
  The desired PCS signal to use for the data transfer.
- bool isEndOfQueue
  Signals that the current transfer is the last in the queue.
- bool clearTransferCount
  Clears the SPI Transfer Counter (SPI_TCNT) before transmission starts.

15.2.3.1.0.17 Field Documentation

15.2.3.1.0.17.1 bool dspi_command_data_config_t::isPcsContinuous

15.2.3.1.0.17.2 uint8_t dspi_command_data_config_t::whichCtar

15.2.3.1.0.17.3 uint8_t dspi_command_data_config_t::whichPcs

15.2.3.1.0.17.4 bool dspi_command_data_config_t::isEndOfQueue

15.2.3.1.0.17.5 bool dspi_command_data_config_t::clearTransferCount

15.2.3.2 struct dspi_master_ctar_config_t

Data Fields

- uint32_t baudRate
  Baud Rate for DSPI.
- uint32_t bitsPerFrame
  Bits per frame, minimum 4, maximum 16.
- dspi_clock_polarity_t cpol
  Clock polarity.
- dspi_clock_phase_t cpha
  Clock phase.
- dspi_shift_direction_t direction
  MSB or LSB data shift direction.
- uint32_t pcsToSckDelayInNanoSec
  PCS to SCK delay time in nanoseconds; setting to 0 sets the minimum delay.
- uint32_t lastSckToPcsDelayInNanoSec
  The last SCK to PCS delay time in nanoseconds; setting to 0 sets the minimum delay.
- uint32_t betweenTransferDelayInNanoSec
  After the SCK delay time in nanoseconds; setting to 0 sets the minimum delay.

15.2.3.2.0.18 Field Documentation

15.2.3.2.0.18.1 uint32_t dspi_master_ctar_config_t::baudRate

15.2.3.2.0.18.2 uint32_t dspi_master_ctar_config_t::bitsPerFrame

15.2.3.2.0.18.3 dspi_clock_polarity_t dspi_master_ctar_config_t::cpol

15.2.3.2.0.18.4 dspi_clock_phase_t dspi_master_ctar_config_t::cpha

15.2.3.2.0.18.5 dspi_shift_direction_t dspi_master_ctar_config_t::direction

15.2.3.2.0.18.6 uint32_t dspi_master_ctar_config_t::pcsToSckDelayInNanoSec

It also sets the boundary value if out of range.

15.2.3.2.0.18.7 uint32_t dspi_master_ctar_config_t::lastSckToPcsDelayInNanoSec

It also sets the boundary value if out of range.

15.2.3.2.0.18.8 uint32_t dspi_master_ctar_config_t::betweenTransferDelayInNanoSec

It also sets the boundary value if out of range.

15.2.3.3 struct dspi_master_config_t

Data Fields

- dspi_ctar_selection_t whichCtar
  The desired CTAR to use.
- dspi_master_ctar_config_t ctarConfig
  Set the ctarConfig to the desired CTAR.
- dspi_which_pcs_t whichPcs
  The desired Peripheral Chip Select (pcs).
- dspi_pcs_polarity_config_t pcsActiveHighOrLow
  The desired PCS active high or low.
- bool enableContinuousSCK
  CONT_SCKE, continuous SCK enable.
- bool enableRxFifoOverWrite
  ROOE, receive FIFO overflow overwrite enable.
- bool enableModifiedTimingFormat
  Enables a modified transfer format to be used if true.
- dspi_master_sample_point_t samplePoint
  Controls when the module master samples SIN in the Modified Transfer Format.

15.2.3.3.0.19 Field Documentation

15.2.3.3.0.19.1 dspi_ctar_selection_t dspi_master_config_t::whichCtar

15.2.3.3.0.19.2 dspi_master_ctar_config_t dspi_master_config_t::ctarConfig

15.2.3.3.0.19.3 dspi_which_pcs_t dspi_master_config_t::whichPcs

15.2.3.3.0.19.4 dspi_pcs_polarity_config_t dspi_master_config_t::pcsActiveHighOrLow

15.2.3.3.0.19.5 bool dspi_master_config_t::enableContinuousSCK

Note that the continuous SCK is only supported for CPHA = 1.

15.2.3.3.0.19.6 bool dspi_master_config_t::enableRxFifoOverWrite

If ROOE = 0, the incoming data is ignored and the data from the transfer that generated the overflow is also ignored. If ROOE = 1, the incoming data is shifted to the shift register.

15.2.3.3.0.19.7 bool dspi_master_config_t::enableModifiedTimingFormat

15.2.3.3.0.19.8 dspi_master_sample_point_t dspi_master_config_t::samplePoint

It's valid only when CPHA=0.

15.2.3.4 struct dspi_slave_ctar_config_t

Data Fields

- uint32_t bitsPerFrame
  Bits per frame, minimum 4, maximum 16.
- dspi_clock_polarity_t cpol
  Clock polarity.
- dspi_clock_phase_t cpha
  Clock phase.

15.2.3.4.0.20 Field Documentation

15.2.3.4.0.20.1 uint32_t dspi_slave_ctar_config_t::bitsPerFrame

15.2.3.4.0.20.2 dspi_clock_polarity_t dspi_slave_ctar_config_t::cpol

15.2.3.4.0.20.3 dspi_clock_phase_t dspi_slave_ctar_config_t::cpha

Slave only supports MSB and does not support LSB.

15.2.3.5 struct dspi_slave_config_t

Data Fields

- dspi_ctar_selection_t whichCtar
  The desired CTAR to use.
- dspi_slave_ctar_config_t ctarConfig
  Set the ctarConfig to the desired CTAR.
- bool enableContinuousSCK
  CONT_SCKE, continuous SCK enable.
- bool enableRxFifoOverWrite
  ROOE, receive FIFO overflow overwrite enable.
- bool enableModifiedTimingFormat
  Enables a modified transfer format to be used if true.
- dspi_master_sample_point_t samplePoint
  Controls when the module master samples SIN in the Modified Transfer Format.

15.2.3.5.0.21 Field Documentation

15.2.3.5.0.21.1 dspi_ctar_selection_t dspi_slave_config_t::whichCtar

15.2.3.5.0.21.2 dspi_slave_ctar_config_t dspi_slave_config_t::ctarConfig

15.2.3.5.0.21.3 bool dspi_slave_config_t::enableContinuousSCK

Note that the continuous SCK is only supported for CPHA = 1.

15.2.3.5.0.21.4 bool dspi_slave_config_t::enableRxFifoOverWrite

If ROOE = 0, the incoming data is ignored and the data from the transfer that generated the overflow is also ignored. If ROOE = 1, the incoming data is shifted to the shift register.

15.2.3.5.0.21.5 bool dspi_slave_config_t::enableModifiedTimingFormat

15.2.3.5.0.21.6 dspi_master_sample_point_t dspi_slave_config_t::samplePoint

It's valid only when CPHA=0.

15.2.3.6 struct dspi_transfer_t

Data Fields

- uint8_t * txData
  Send buffer.
- uint8_t * rxData
  Receive buffer.
- volatile size_t dataSize
  Transfer bytes.
- uint32_t configFlags
  Transfer transfer configuration flags.

15.2.3.6.0.22 Field Documentation

15.2.3.6.0.22.1 uint8_t* dspi_transfer_t::txData

15.2.3.6.0.22.2 uint8_t* dspi_transfer_t::rxData

15.2.3.6.0.22.3 volatile size_t dspi_transfer_t::dataSize

15.2.3.6.0.22.4 uint32_t dspi_transfer_t::configFlags

Set from _dspi_transfer_config_flag_for_master if the transfer is used for master or _dspi_transfer_config-_flag_for_slave enumeration if the transfer is used for slave.

15.2.3.7 struct dspi_half_duplex_transfer_t

Data Fields

- uint8_t * txData
  Send buffer.
- uint8_t * rxData
  Receive buffer.
- size_t txDataSize
  Transfer bytes for transmit.
- size_t rxDataSize
  Transfer bytes.
- uint32_t configFlags
  Transfer configuration flags; set from _dspi_transfer_config_flag_for_master.
- bool isPcsAssertInTransfer
  If Pcs pin keep assert between transmit and receive.
- bool isTransmitFirst
  True for transmit first and false for receive first.

15.2.3.7.0.23 Field Documentation

15.2.3.7.0.23.1 uint32_t dspi_half_duplex_transfer_t::configFlags

15.2.3.7.0.23.2 bool dspi_half_duplex_transfer_t::isPcsAssertInTransfer

true for assert and false for de-assert.

15.2.3.7.0.23.3 bool dspi_half_duplex_transfer_t::isTransmitFirst

15.2.3.8 struct _dspi_master_handle

Forward declaration of the _dspi_master_handle typedefs.

The master handle.

Data Fields

- uint32_t bitsPerFrame
  The desired number of bits per frame.
- volatile uint32_t command
  The desired data command.
- volatile uint32_t lastCommand
  The desired last data command.
- uint8_t fifoSize
  FIFO dataSize.
- volatile bool isPcsActiveAfterTransfer
  Indicates whether the PCS signal is active after the last frame transfer.
- volatile bool isThereExtraByte
  Indicates whether there are extra bytes.
- uint8_t ∗volatile txData
  Send buffer.
- uint8_t ∗volatile rxData
  Receive buffer.
- volatile size_t remainingSendByteCount
  A number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount
  A number of bytes remaining to receive.
- size_t totalByteCount
  A number of transfer bytes.
- volatile uint8_t state
  DSPI transfer state, see _dspi_transfer_state.
- dspi_master_transfer_callback_t callback
  Completion callback.
- void ∗ userData
  Callback user data.

15.2.3.8.0.24 Field Documentation

15.2.3.8.0.24.1 uint32_t dspi_master_handle_t::bitsPerFrame

15.2.3.8.0.24.2 volatile uint32_t dspi_master_handle_t::command

15.2.3.8.0.24.3 volatile uint32_t dspi_master_handle_t::lastCommand

15.2.3.8.0.24.4 uint8_t dspi_master_handle_t::fifoSize

15.2.3.8.0.24.5 volatile bool dspi_master_handle_t::isPcsActiveAfterTransfer

15.2.3.8.0.24.6 volatile bool dspi_master_handle_t::isThereExtraByte

15.2.3.8.0.24.7 uint8_t∗ volatile dspi_master_handle_t::txData

15.2.3.8.0.24.8 uint8_t∗ volatile dspi_master_handle_t::rxData

15.2.3.8.0.24.9 volatile size_t dspi_master_handle_t::remainingSendByteCount

15.2.3.8.0.24.10 volatile size_t dspi_master_handle_t::remainingReceiveByteCount

15.2.3.8.0.24.11 volatile uint8_t dspi_master_handle_t::state

15.2.3.8.0.24.12 dspi_master_transfer_callback_t dspi_master_handle_t::callback

15.2.3.8.0.24.13 void∗ dspi_master_handle_t::userData

15.2.3.9 struct _dspi_slave_handle

Forward declaration of the _dspi_slave_handle typedefs.

The slave handle.

Data Fields

- uint32_t bitsPerFrame
  The desired number of bits per frame.
- volatile bool isThereExtraByte
  Indicates whether there are extra bytes.
- uint8_t ∗volatile txData
  Send buffer.
- uint8_t ∗volatile rxData
  Receive buffer.
- volatile size_t remainingSendByteCount
  A number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount
  A number of bytes remaining to receive.
- size_t totalByteCount
  A number of transfer bytes.
- volatile uint8_t state

DSPI transfer state.
- volatile uint32_t errorCount
  Error count for slave transfer.
- dspi_slave_transfer_callback_t callback
  Completion callback.
- void * userData
  Callback user data.

15.2.3.9.0.25 Field Documentation

15.2.3.9.0.25.1 uint32_t dspi_slave_handle_t::bitsPerFrame

15.2.3.9.0.25.2 volatile bool dspi_slave_handle_t::isThereExtraByte

15.2.3.9.0.25.3 uint8_t* volatile dspi_slave_handle_t::txData

15.2.3.9.0.25.4 uint8_t* volatile dspi_slave_handle_t::rxData

15.2.3.9.0.25.5 volatile size_t dspi_slave_handle_t::remainingSendByteCount

15.2.3.9.0.25.6 volatile size_t dspi_slave_handle_t::remainingReceiveByteCount

15.2.3.9.0.25.7 volatile uint8_t dspi_slave_handle_t::state

15.2.3.9.0.25.8 volatile uint32_t dspi_slave_handle_t::errorCount

15.2.3.9.0.25.9 dspi_slave_transfer_callback_t dspi_slave_handle_t::callback

15.2.3.9.0.25.10 void* dspi_slave_handle_t::userData

15.2.4 Macro Definition Documentation

15.2.4.1 #define FSL_DSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))

15.2.4.2 #define DSPI_DUMMY_DATA (0x00U)

Dummy data used for Tx if there is no txData.

15.2.4.3 #define DSPI_MASTER_CTAR_SHIFT (0U)

15.2.4.4 #define DSPI_MASTER_CTAR_MASK (0x0FU)

15.2.4.5 #define DSPI_MASTER_PCS_SHIFT (4U)

15.2.4.6 #define DSPI_MASTER_PCS_MASK (0xF0U)

15.2.4.7 #define DSPI_SLAVE_CTAR_SHIFT (0U)

15.2.4.8 #define DSPI_SLAVE_CTAR_MASK (0x07U)

15.2.5 Typedef Documentation

15.2.5.1 typedef void(* dspi_master_transfer_callback_t)(SPI_Type *base, dspi_master_handle_t *handle, status_t status, void *userData)

Parameters

base	DSPI peripheral address.
handle	Pointer to the handle for the DSPI master.
status	Success or error code describing whether the transfer completed.
userData	Arbitrary pointer-dataSized value passed from the application.

15.2.5.2 typedef void(* dspi_slave_transfer_callback_t)(SPI_Type *base, dspi_slave_handle_t *handle, status_t status, void *userData)

Parameters

base	DSPI peripheral address.
handle	Pointer to the handle for the DSPI slave.
status	Success or error code describing whether the transfer completed.
userData	Arbitrary pointer-dataSized value passed from the application.

15.2.6 Enumeration Type Documentation

15.2.6.1 anonymous enum

Enumerator

kStatus_DSPI_Busy DSPI transfer is busy.
kStatus_DSPI_Error DSPI driver error.
kStatus_DSPI_Idle DSPI is idle.
kStatus_DSPI_OutOfRange DSPI transfer out of range.

15.2.6.2 enum _dspi_flags

Enumerator

kDSPI_TxCompleteFlag Transfer Complete Flag.
kDSPI_EndOfQueueFlag End of Queue Flag.
kDSPI_TxFifoUnderflowFlag Transmit FIFO Underflow Flag.
kDSPI_TxFifoFillRequestFlag Transmit FIFO Fill Flag.
kDSPI_RxFifoOverflowFlag Receive FIFO Overflow Flag.
kDSPI_RxFifoDrainRequestFlag Receive FIFO Drain Flag.
kDSPI_TxAndRxStatusFlag The module is in Stopped/Running state.
kDSPI_AllStatusFlag All statuses above.

15.2.6.3 enum _dspi_interrupt_enable

Enumerator

kDSPI_TxCompleteInterruptEnable TCF interrupt enable.
kDSPI_EndOfQueueInterruptEnable EOQF interrupt enable.
kDSPI_TxFifoUnderflowInterruptEnable TFUF interrupt enable.
kDSPI_TxFifoFillRequestInterruptEnable TFFF interrupt enable, DMA disable.
kDSPI_RxFifoOverflowInterruptEnable RFOF interrupt enable.
kDSPI_RxFifoDrainRequestInterruptEnable RFDF interrupt enable, DMA disable.
kDSPI_AllInterruptEnable All above interrupts enable.

15.2.6.4 enum _dspi_dma_enable

Enumerator

kDSPI_TxDmaEnable TFFF flag generates DMA requests. No Tx interrupt request.
kDSPI_RxDmaEnable RFDF flag generates DMA requests. No Rx interrupt request.

15.2.6.5 enum dspi_master_slave_mode_t

Enumerator

kDSPI_Master DSPI peripheral operates in master mode.
kDSPI_Slave DSPI peripheral operates in slave mode.

15.2.6.6 enum dspi_master_sample_point_t

This field is valid only when the CPHA bit in the CTAR register is 0.

Enumerator

kDSPI_SckToSin0Clock 0 system clocks between SCK edge and SIN sample.
kDSPI_SckToSin1Clock 1 system clock between SCK edge and SIN sample.
kDSPI_SckToSin2Clock 2 system clocks between SCK edge and SIN sample.

15.2.6.7 enum dspi_which_pcs_t

Enumerator

kDSPI_Pcs0 Pcs[0].
kDSPI_Pcs1 Pcs[1].
kDSPI_Pcs2 Pcs[2].

kDSPI_Pcs3 Pcs[3].
kDSPI_Pcs4 Pcs[4].
kDSPI_Pcs5 Pcs[5].

15.2.6.8 enum dspi_pcs_polarity_config_t

Enumerator

kDSPI_PcsActiveHigh Pcs Active High (idles low).
kDSPI_PcsActiveLow Pcs Active Low (idles high).

15.2.6.9 enum _dspi_pcs_polarity

Enumerator

kDSPI_Pcs0ActiveLow Pcs0 Active Low (idles high).
kDSPI_Pcs1ActiveLow Pcs1 Active Low (idles high).
kDSPI_Pcs2ActiveLow Pcs2 Active Low (idles high).
kDSPI_Pcs3ActiveLow Pcs3 Active Low (idles high).
kDSPI_Pcs4ActiveLow Pcs4 Active Low (idles high).
kDSPI_Pcs5ActiveLow Pcs5 Active Low (idles high).
kDSPI_PcsAllActiveLow Pcs0 to Pcs5 Active Low (idles high).

15.2.6.10 enum dspi_clock_polarity_t

Enumerator

kDSPI_ClockPolarityActiveHigh CPOL=0. Active-high DSPI clock (idles low).
kDSPI_ClockPolarityActiveLow CPOL=1. Active-low DSPI clock (idles high).

15.2.6.11 enum dspi_clock_phase_t

Enumerator

kDSPI_ClockPhaseFirstEdge CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.
kDSPI_ClockPhaseSecondEdge CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

15.2.6.12 enum dspi_shift_direction_t

Enumerator

kDSPI_MsbFirst Data transfers start with most significant bit.
kDSPI_LsbFirst Data transfers start with least significant bit. Shifting out of LSB is not supported for slave

15.2.6.13 enum dspi_delay_type_t

Enumerator

kDSPI_PcsToSck Pcs-to-SCK delay.
kDSPI_LastSckToPcs The last SCK edge to Pcs delay.
kDSPI_BetweenTransfer Delay between transfers.

15.2.6.14 enum dspi_ctar_selection_t

Enumerator

kDSPI_Ctar0 CTAR0 selection option for master or slave mode; note that CTAR0 and CTAR0_SLAVE are the same register address.
kDSPI_Ctar1 CTAR1 selection option for master mode only.
kDSPI_Ctar2 CTAR2 selection option for master mode only; note that some devices do not support CTAR2.
kDSPI_Ctar3 CTAR3 selection option for master mode only; note that some devices do not support CTAR3.
kDSPI_Ctar4 CTAR4 selection option for master mode only; note that some devices do not support CTAR4.
kDSPI_Ctar5 CTAR5 selection option for master mode only; note that some devices do not support CTAR5.
kDSPI_Ctar6 CTAR6 selection option for master mode only; note that some devices do not support CTAR6.
kDSPI_Ctar7 CTAR7 selection option for master mode only; note that some devices do not support CTAR7.

15.2.6.15 enum _dspi_transfer_config_flag_for_master

Enumerator

kDSPI_MasterCtar0 DSPI master transfer use CTAR0 setting.
kDSPI_MasterCtar1 DSPI master transfer use CTAR1 setting.
kDSPI_MasterCtar2 DSPI master transfer use CTAR2 setting.

kDSPI_MasterCtar3 DSPI master transfer use CTAR3 setting.
kDSPI_MasterCtar4 DSPI master transfer use CTAR4 setting.
kDSPI_MasterCtar5 DSPI master transfer use CTAR5 setting.
kDSPI_MasterCtar6 DSPI master transfer use CTAR6 setting.
kDSPI_MasterCtar7 DSPI master transfer use CTAR7 setting.
kDSPI_MasterPcs0 DSPI master transfer use PCS0 signal.
kDSPI_MasterPcs1 DSPI master transfer use PCS1 signal.
kDSPI_MasterPcs2 DSPI master transfer use PCS2 signal.
kDSPI_MasterPcs3 DSPI master transfer use PCS3 signal.
kDSPI_MasterPcs4 DSPI master transfer use PCS4 signal.
kDSPI_MasterPcs5 DSPI master transfer use PCS5 signal.
kDSPI_MasterPcsContinuous Indicates whether the PCS signal is continuous.
kDSPI_MasterActiveAfterTransfer Indicates whether the PCS signal is active after the last frame transfer.

15.2.6.16 enum _dspi_transfer_config_flag_for_slave

Enumerator

kDSPI_SlaveCtar0 DSPI slave transfer use CTAR0 setting. DSPI slave can only use PCS0.

15.2.6.17 enum _dspi_transfer_state

Enumerator

kDSPI_Idle Nothing in the transmitter/receiver.
kDSPI_Busy Transfer queue is not finished.
kDSPI_Error Transfer error.

15.2.7 Function Documentation

15.2.7.1 void DSPI_MasterInit (SPI_Type ∗ *base*, const dspi_master_config_t ∗ *masterConfig*, uint32_t *srcClock_Hz*)

This function initializes the DSPI master configuration. This is an example use case.

```
*   dspi_master_config_t  masterConfig;
*   masterConfig.whichCtar                                  = kDSPI_Ctar0;
*   masterConfig.ctarConfig.baudRate                        = 500000000U;
*   masterConfig.ctarConfig.bitsPerFrame                    = 8;
*   masterConfig.ctarConfig.cpol                            =
      kDSPI_ClockPolarityActiveHigh;
*   masterConfig.ctarConfig.cpha                            =
      kDSPI_ClockPhaseFirstEdge;
*   masterConfig.ctarConfig.direction                       =
```

```
     kDSPI_MsbFirst;
*   masterConfig.ctarConfig.pcsToSckDelayInNanoSec        = 1000000000U /
     masterConfig.ctarConfig.baudRate ;
*   masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec    = 1000000000U
      / masterConfig.ctarConfig.baudRate ;
*   masterConfig.ctarConfig.betweenTransferDelayInNanoSec =
     1000000000U / masterConfig.ctarConfig.baudRate ;
*   masterConfig.whichPcs                                 = kDSPI_Pcs0;
*   masterConfig.pcsActiveHighOrLow                       =
     kDSPI_PcsActiveLow;
*   masterConfig.enableContinuousSCK                      = false;
*   masterConfig.enableRxFifoOverWrite                    = false;
*   masterConfig.enableModifiedTimingFormat               = false;
*   masterConfig.samplePoint                              =
     kDSPI_SckToSin0Clock;
*   DSPI_MasterInit(base, &masterConfig, srcClock_Hz);
*
```

Parameters

base	DSPI peripheral address.
masterConfig	Pointer to the structure dspi_master_config_t.
srcClock_Hz	Module source input clock in Hertz.

15.2.7.2 void DSPI_MasterGetDefaultConfig (dspi_master_config_t * *masterConfig*)

The purpose of this API is to get the configuration structure initialized for the DSPI_MasterInit(). Users may use the initialized structure unchanged in the DSPI_MasterInit() or modify the structure before calling the DSPI_MasterInit(). Example:

```
*  dspi_master_config_t  masterConfig;
*  DSPI_MasterGetDefaultConfig(&masterConfig);
*
```

Parameters

masterConfig	pointer to dspi_master_config_t structure

15.2.7.3 void DSPI_SlaveInit (SPI_Type * *base,* const dspi_slave_config_t * *slaveConfig*)

This function initializes the DSPI slave configuration. This is an example use case.

```
*   dspi_slave_config_t  slaveConfig;
*  slaveConfig->whichCtar                  = kDSPI_Ctar0;
*  slaveConfig->ctarConfig.bitsPerFrame    = 8;
*  slaveConfig->ctarConfig.cpol            =
     kDSPI_ClockPolarityActiveHigh;
*  slaveConfig->ctarConfig.cpha            =
     kDSPI_ClockPhaseFirstEdge;
*  slaveConfig->enableContinuousSCK        = false;
```

```
*  slaveConfig->enableRxFifoOverWrite      = false;
*  slaveConfig->enableModifiedTimingFormat = false;
*  slaveConfig->samplePoint                = kDSPI_SckToSin0Clock;
*   DSPI_SlaveInit(base, &slaveConfig);
*
```

Parameters

base	DSPI peripheral address.
slaveConfig	Pointer to the structure dspi_master_config_t.

15.2.7.4 void DSPI_SlaveGetDefaultConfig (dspi_slave_config_t * *slaveConfig*)

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

```
*  dspi_slave_config_t  slaveConfig;
*  DSPI_SlaveGetDefaultConfig(&slaveConfig);
*
```

Parameters

slaveConfig	Pointer to the dspi_slave_config_t structure.

15.2.7.5 void DSPI_Deinit (SPI_Type * *base*)

Call this API to disable the DSPI clock.

Parameters

base	DSPI peripheral address.

15.2.7.6 static void DSPI_Enable (SPI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	DSPI peripheral address.
enable	Pass true to enable module, false to disable module.

15.2.7.7 static uint32_t DSPI_GetStatusFlags (SPI_Type * *base*) [inline], [static]

Parameters

base	DSPI peripheral address.

Returns

DSPI status (in SR register).

15.2.7.8 static void DSPI_ClearStatusFlags (SPI_Type * *base,* uint32_t *statusFlags*) [inline], [static]

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status bit to clear. The list of status bits is defined in the **dspi_status_and_interrupt_request_t**. The function uses these bit positions in its algorithm to clear the desired flag state. This is an example.

```
*  DSPI_ClearStatusFlags(base, kDSPI_TxCompleteFlag|
     kDSPI_EndOfQueueFlag);
*
```

Parameters

base	DSPI peripheral address.
statusFlags	The status flag used from the type dspi_flags.

< The status flags are cleared by writing 1 (w1c).

15.2.7.9 void DSPI_EnableInterrupts (SPI_Type * *base,* uint32_t *mask*)

This function configures various interrupt masks of the DSPI. The parameters are a base and an interrupt mask.

Note

For Tx Fill and Rx FIFO drain requests, enable the interrupt request and disable the DMA request. Do not use this API(write to RSER register) while DSPI is in running state.

```
*  DSPI_EnableInterrupts(base,
     kDSPI_TxCompleteInterruptEnable |
     kDSPI_EndOfQueueInterruptEnable );
*
```

Parameters

base	DSPI peripheral address.
mask	The interrupt mask; use the enum _dspi_interrupt_enable.

15.2.7.10 static void DSPI_DisableInterrupts (SPI_Type * *base*, uint32_t *mask*) [inline], [static]

```
* DSPI_DisableInterrupts(base,
     kDSPI_TxCompleteInterruptEnable |
     kDSPI_EndOfQueueInterruptEnable );
*
```

Parameters

base	DSPI peripheral address.
mask	The interrupt mask; use the enum _dspi_interrupt_enable.

15.2.7.11 static void DSPI_EnableDMA (SPI_Type * *base*, uint32_t *mask*) [inline], [static]

This function configures the Rx and Tx DMA mask of the DSPI. The parameters are a base and a DMA mask.

```
* DSPI_EnableDMA(base, kDSPI_TxDmaEnable |
     kDSPI_RxDmaEnable);
*
```

Parameters

base	DSPI peripheral address.
mask	The interrupt mask; use the enum _dspi_dma_enable.

15.2.7.12 static void DSPI_DisableDMA (SPI_Type * *base*, uint32_t *mask*) [inline], [static]

This function configures the Rx and Tx DMA mask of the DSPI. The parameters are a base and a DMA mask.

```
* SPI_DisableDMA(base, kDSPI_TxDmaEnable | kDSPI_RxDmaEnable);
*
```

Parameters

base	DSPI peripheral address.
mask	The interrupt mask; use the enum _dspi_dma_enable.

15.2.7.13 static uint32_t DSPI_MasterGetTxRegisterAddress (SPI_Type * *base*) [inline], [static]

This function gets the DSPI master PUSHR data register address because this value is needed for the DMA operation.

Parameters

base	DSPI peripheral address.

Returns

The DSPI master PUSHR data register address.

15.2.7.14 static uint32_t DSPI_SlaveGetTxRegisterAddress (SPI_Type * *base*) [inline], [static]

This function gets the DSPI slave PUSHR data register address as this value is needed for the DMA operation.

Parameters

base	DSPI peripheral address.

Returns

The DSPI slave PUSHR data register address.

15.2.7.15 static uint32_t DSPI_GetRxRegisterAddress (SPI_Type * *base*) [inline], [static]

This function gets the DSPI POPR data register address as this value is needed for the DMA operation.

Parameters

base	DSPI peripheral address.

Returns

The DSPI POPR data register address.

15.2.7.16 uint32_t DSPI_GetInstance (SPI_Type * *base*)

Parameters

base	DSPI peripheral base address.

15.2.7.17 static void DSPI_SetMasterSlaveMode (SPI_Type * *base*, dspi_master_slave_mode_t *mode*) [inline], [static]

Parameters

base	DSPI peripheral address.
mode	Mode setting (master or slave) of type dspi_master_slave_mode_t.

15.2.7.18 static bool DSPI_IsMaster (SPI_Type * *base*) [inline], [static]

Parameters

base	DSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

15.2.7.19 static void DSPI_StartTransfer (SPI_Type * *base*) [inline], [static]

This function sets the module to start data transfer in either master or slave mode.

Parameters

base	DSPI peripheral address.

15.2.7.20 static void DSPI_StopTransfer (SPI_Type * *base*) [inline], [static]

This function stops data transfers in either master or slave modes.

Parameters

base	DSPI peripheral address.

15.2.7.21 static void DSPI_SetFifoEnable (SPI_Type * *base*, bool *enableTxFifo*, bool *enableRxFifo*) [inline], [static]

This function allows the caller to disable/enable the Tx and Rx FIFOs independently.

Note

> To disable, pass in a logic 0 (false) for the particular FIFO configuration. To enable, pass in a logic 1 (true).

Parameters

base	DSPI peripheral address.
enableTxFifo	Disables (false) the TX FIFO; Otherwise, enables (true) the TX FIFO
enableRxFifo	Disables (false) the RX FIFO; Otherwise, enables (true) the RX FIFO

15.2.7.22 static void DSPI_FlushFifo (SPI_Type * *base*, bool *flushTxFifo*, bool *flushRxFifo*) [inline], [static]

Parameters

base	DSPI peripheral address.
flushTxFifo	Flushes (true) the Tx FIFO; Otherwise, does not flush (false) the Tx FIFO

flushRxFifo	Flushes (true) the Rx FIFO; Otherwise, does not flush (false) the Rx FIFO

15.2.7.23 static void DSPI_SetAllPcsPolarity (SPI_Type ∗ *base*, uint32_t *mask*) [inline], [static]

For example, PCS0 and PCS1 are set to active low and other PCS is set to active high. Note that the number of PCSs is specific to the device.

```
* DSPI_SetAllPcsPolarity(base, kDSPI_Pcs0ActiveLow |
     kDSPI_Pcs1ActiveLow);
```

Parameters

base	DSPI peripheral address.
mask	The PCS polarity mask; use the enum _dspi_pcs_polarity.

15.2.7.24 uint32_t DSPI_MasterSetBaudRate (SPI_Type ∗ *base*, dspi_ctar_selection_t *whichCtar*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

This function takes in the desired baudRate_Bps (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate, and returns the calculated baud rate in bits-per-second. It requires that the caller also provide the frequency of the module source clock (in Hertz).

Parameters

base	DSPI peripheral address.
whichCtar	The desired Clock and Transfer Attributes Register (CTAR) of the type dspi_ctar_selection_t
baudRate_Bps	The desired baud rate in bits per second
srcClock_Hz	Module source input clock in Hertz

Returns

The actual calculated baud rate

15.2.7.25 void DSPI_MasterSetDelayScaler (SPI_Type ∗ *base*, dspi_ctar_selection_t *whichCtar*, uint32_t *prescaler*, uint32_t *scaler*, dspi_delay_type_t *whichDelay*)

This function configures the PCS to SCK delay pre-scalar (PcsSCK) and scalar (CSSCK), after SCK delay pre-scalar (PASC) and scalar (ASC), and the delay after transfer pre-scalar (PDT) and scalar (DT).

These delay names are available in the type dspi_delay_type_t.

The user passes the delay to the configuration along with the prescaler and scaler value. This allows the user to directly set the prescaler/scaler values if pre-calculated or to manually increment either value.

Parameters

base	DSPI peripheral address.
whichCtar	The desired Clock and Transfer Attributes Register (CTAR) of type dspi_ctar_-selection_t.
prescaler	The prescaler delay value (can be an integer 0, 1, 2, or 3).
scaler	The scaler delay value (can be any integer between 0 to 15).
whichDelay	The desired delay to configure; must be of type dspi_delay_type_t

15.2.7.26 uint32_t DSPI_MasterSetDelayTimes (SPI_Type ∗ *base*, dspi_ctar_selection_t *whichCtar*, dspi_delay_type_t *whichDelay*, uint32_t *srcClock_Hz*, uint32_t *delayTimeInNanoSec*)

This function calculates the values for the following. PCS to SCK delay pre-scalar (PCSSCK) and scalar (CSSCK), or After SCK delay pre-scalar (PASC) and scalar (ASC), or Delay after transfer pre-scalar (PDT) and scalar (DT).

These delay names are available in the type dspi_delay_type_t.

The user passes which delay to configure along with the desired delay value in nanoseconds. The function calculates the values needed for the prescaler and scaler. Note that returning the calculated delay as an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. The higher-level peripheral driver alerts the user of an out of range delay input.

Parameters

base	DSPI peripheral address.
whichCtar	The desired Clock and Transfer Attributes Register (CTAR) of type dspi_ctar_-selection_t.
whichDelay	The desired delay to configure, must be of type dspi_delay_type_t
srcClock_Hz	Module source input clock in Hertz

delayTimeIn-NanoSec	The desired delay value in nanoseconds.

Returns

The actual calculated delay value.

15.2.7.27 static void DSPI_MasterWriteData (SPI_Type * *base*, dspi_-command_data_config_t * *command*, uint16_t *data*) [inline], [static]

In master mode, the 16-bit data is appended to the 16-bit command info. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). This is an example.

```
*  dspi_command_data_config_t commandConfig;
*  commandConfig.isPcsContinuous = true;
*  commandConfig.whichCtar = kDSPICtar0;
*  commandConfig.whichPcs = kDSPIPcs0;
*  commandConfig.clearTransferCount = false;
*  commandConfig.isEndOfQueue = false;
*  DSPI_MasterWriteData(base, &commandConfig, dataWord);
```

Parameters

base	DSPI peripheral address.
command	Pointer to the command structure.
data	The data word to be sent.

15.2.7.28 void DSPI_GetDefaultDataCommandConfig (dspi_command_data_config_t * *command*)

The purpose of this API is to get the configuration structure initialized for use in the **DSPI_MasterWrite-_xx()**. Users may use the initialized structure unchanged in the DSPI_MasterWrite_xx() or modify the structure before calling the DSPI_MasterWrite_xx(). This is an example.

```
*  dspi_command_data_config_t  command;
*  DSPI_GetDefaultDataCommandConfig(&command);
*
```

Parameters

command	Pointer to the dspi_command_data_config_t structure.

15.2.7.29 void DSPI_MasterWriteDataBlocking (SPI_Type ∗ *base*, dspi_command_data_config_t ∗ *command*, uint16_t *data*)

In master mode, the 16-bit data is appended to the 16-bit command info. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). This is an example.

```
*  dspi_command_config_t commandConfig;
*  commandConfig.isPcsContinuous = true;
*  commandConfig.whichCtar = kDSPICtar0;
*  commandConfig.whichPcs = kDSPIPcs1;
*  commandConfig.clearTransferCount = false;
*  commandConfig.isEndOfQueue = false;
*  DSPI_MasterWriteDataBlocking(base, &commandConfig, dataWord);
*
```

Note

This function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running to transmit data (MCR[MDIS] & [HALT] = 0). Because the SPI is a synchronous protocol, the received data is available when the transmit completes.

Parameters

base	DSPI peripheral address.
command	Pointer to the command structure.
data	The data word to be sent.

15.2.7.30 static uint32_t DSPI_MasterGetFormattedCommand (dspi_command_data_config_t ∗ *command*) [inline], [static]

This function allows the caller to pass in the data command structure and returns the command word formatted according to the DSPI PUSHR register bit field placement. The user can then "OR" the returned command word with the desired data to send and use the function **DSPI_HAL_WriteCommandDataMastermode** or **DSPI_HAL_WriteCommandDataMastermodeBlocking** to write the entire 32-bit command data word to the PUSHR. This helps improve performance in cases where the command structure is constant. For example, the user calls this function before starting a transfer to generate the command word. When they are ready to transmit the data, they OR this formatted command word with the desired data to transmit. This process increases transmit performance when compared to calling send functions, such as **DSPI_HAL_WriteDataMastermode**, which format the command word each time a data word is to be sent.

Parameters

command	Pointer to the command structure.

Returns

The command word formatted to the PUSHR data register bit field.

15.2.7.31 void DSPI_MasterWriteCommandDataBlocking (SPI_Type ∗ *base*, uint32_t *data*)

In this function, the user must append the 16-bit data to the 16-bit command information and then provide the total 32-bit word as the data to send. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). The user is responsible for appending this command with the data to send. This is an example:

```
*  dataWord = <16-bit command> | <16-bit data>;
*  DSPI_MasterWriteCommandDataBlocking(base, dataWord);
*
```

Note

This function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running to transmit data (MCR[MDIS] & [HALT] = 0). Because the SPI is a synchronous protocol, the received data is available when the transmit completes.

For a blocking polling transfer, see methods below.

Option 1
uint32_t command_to_send = DSPI_MasterGetFormattedCommand(&command);
uint32_t data0 = command_to_send | data_need_to_send_0;
uint32_t data1 = command_to_send | data_need_to_send_1;
uint32_t data2 = command_to_send | data_need_to_send_2;
DSPI_MasterWriteCommandDataBlocking(base,data0);
DSPI_MasterWriteCommandDataBlocking(base,data1);
DSPI_MasterWriteCommandDataBlocking(base,data2);

Option 2
DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_0);
DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_1);
DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_2);

Parameters

base	DSPI peripheral address.
data	The data word (command and data combined) to be sent.

15.2.7.32 static void DSPI_SlaveWriteData (SPI_Type * *base*, uint32_t *data*) [inline], [static]

In slave mode, up to 16-bit words may be written.

Parameters

base	DSPI peripheral address.
data	The data to send.

15.2.7.33 void DSPI_SlaveWriteDataBlocking (SPI_Type * *base*, uint32_t *data*)

In slave mode, up to 16-bit words may be written. The function first clears the transmit complete flag, writes data into data register, and finally waits until the data is transmitted.

Parameters

base	DSPI peripheral address.
data	The data to send.

15.2.7.34 static uint32_t DSPI_ReadData (SPI_Type * *base*) [inline], [static]

Parameters

base	DSPI peripheral address.

Returns

The data from the read data buffer.

15.2.7.35 void DSPI_SetDummyData (SPI_Type * *base,* uint8_t *dummyData*)

Parameters

base	DSPI peripheral address.
dummyData	Data to be transferred when tx buffer is NULL.

15.2.7.36 void DSPI_MasterTransferCreateHandle (SPI_Type * *base,* dspi_master_-handle_t * *handle,* dspi_master_transfer_callback_t *callback,* void * *userData*)

This function initializes the DSPI handle, which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Parameters

base	DSPI peripheral base address.
handle	DSPI handle pointer to _dspi_master_handle.
callback	DSPI callback.
userData	Callback function parameter.

15.2.7.37 status_t DSPI_MasterTransferBlocking (SPI_Type * *base,* dspi_transfer_t * *transfer*)

This function transfers data using polling. This is a blocking function, which does not return until all transfers have been completed.

Parameters

base	DSPI peripheral base address.
transfer	Pointer to the dspi_transfer_t structure.

Returns

status of status_t.

15.2.7.38 status_t DSPI_MasterTransferNonBlocking (SPI_Type * *base*, dspi_master_handle_t * *handle*, dspi_transfer_t * *transfer*)

This function transfers data using interrupts. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the _dspi_master_handle structure which stores the transfer state.
transfer	Pointer to the dspi_transfer_t structure.

Returns

status of status_t.

15.2.7.39 status_t DSPI_MasterHalfDuplexTransferBlocking (SPI_Type * *base*, dspi_half_duplex_transfer_t * *xfer*)

This function will do a half-duplex transfer for DSPI master, This is a blocking function, which does not retuen until all transfer have been completed. And data transfer will be half-duplex, users can set transmit first or receive first.

Parameters

base	DSPI base pointer
xfer	pointer to dspi_half_duplex_transfer_t structure

Returns

status of status_t.

15.2.7.40 status_t DSPI_MasterHalfDuplexTransferNonBlocking (SPI_Type * *base*, dspi_master_handle_t * *handle*, dspi_half_duplex_transfer_t * *xfer*)

This function transfers data using interrupts, the transfer mechanism is half-duplex. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

base	DSPI peripheral base address.
handle	pointer to _dspi_master_handle structure which stores the transfer state
xfer	pointer to dspi_half_duplex_transfer_t structure

Returns

status of status_t.

15.2.7.41 status_t DSPI_MasterTransferGetCount (SPI_Type ∗ *base*, dspi_master_handle_t ∗ *handle*, size_t ∗ *count*)

This function gets the master transfer count.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the _dspi_master_handle structure which stores the transfer state.
count	The number of bytes transferred by using the non-blocking transaction.

Returns

status of status_t.

15.2.7.42 void DSPI_MasterTransferAbort (SPI_Type ∗ *base*, dspi_master_handle_t ∗ *handle*)

This function aborts a transfer using an interrupt.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the _dspi_master_handle structure which stores the transfer state.

15.2.7.43 void DSPI_MasterTransferHandleIRQ (SPI_Type ∗ *base*, dspi_master_handle_t ∗ *handle*)

This function processes the DSPI transmit and receive IRQ.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the _dspi_master_handle structure which stores the transfer state.

15.2.7.44 void DSPI_SlaveTransferCreateHandle (SPI_Type * *base*, dspi_slave_handle_t * *handle*, dspi_slave_transfer_callback_t *callback*, void * *userData*)

This function initializes the DSPI handle, which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Parameters

handle	DSPI handle pointer to the _dspi_slave_handle.
base	DSPI peripheral base address.
callback	DSPI callback.
userData	Callback function parameter.

15.2.7.45 status_t DSPI_SlaveTransferNonBlocking (SPI_Type * *base*, dspi_slave_handle_t * *handle*, dspi_transfer_t * *transfer*)

This function transfers data using an interrupt. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the _dspi_slave_handle structure which stores the transfer state.
transfer	Pointer to the dspi_transfer_t structure.

Returns

status of status_t.

15.2.7.46 status_t DSPI_SlaveTransferGetCount (SPI_Type * *base*, dspi_slave_handle_t * *handle*, size_t * *count*)

This function gets the slave transfer count.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the _dspi_master_handle structure which stores the transfer state.
count	The number of bytes transferred by using the non-blocking transaction.

Returns

status of status_t.

15.2.7.47 void DSPI_SlaveTransferAbort (SPI_Type * *base,* dspi_slave_handle_t * *handle*)

This function aborts a transfer using an interrupt.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the _dspi_slave_handle structure which stores the transfer state.

15.2.7.48 void DSPI_SlaveTransferHandleIRQ (SPI_Type * *base,* dspi_slave_handle_t * *handle*)

This function processes the DSPI transmit and receive IRQ.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the _dspi_slave_handle structure which stores the transfer state.

15.2.7.49 uint8_t DSPI_GetDummyDataInstance (SPI_Type * *base*)

The purpose of this API is to avoid MISRA rule8.5 : Multiple declarations of externally-linked object or function g_dspiDummyData.

param base DSPI peripheral base address.

15.2.8 Variable Documentation

15.2.8.1 volatile uint8_t g_dspiDummyData[]

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

Overview

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

Typical use case

16.2.1 eDMA Operation

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

Data Structures

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

Macros

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

Typedefs

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

Enumerations

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

kEDMA_BandwidthStallNone = 0x0U,
kEDMA_BandwidthStall4Cycle = 0x2U,
kEDMA_BandwidthStall8Cycle = 0x3U }
Bandwidth control.
- enum edma_channel_link_type_t {
kEDMA_LinkNone = 0x0U,
kEDMA_MinorLink,
kEDMA_MajorLink }
Channel link type.
- enum {
kEDMA_DoneFlag = 0x1U,
kEDMA_ErrorFlag = 0x2U,
kEDMA_InterruptFlag = 0x4U }
_edma_channel_status_flags eDMA channel status flags.
- enum {
kEDMA_DestinationBusErrorFlag = DMA_ES_DBE_MASK,
kEDMA_SourceBusErrorFlag = DMA_ES_SBE_MASK,
kEDMA_ScatterGatherErrorFlag = DMA_ES_SGE_MASK,
kEDMA_NbytesErrorFlag = DMA_ES_NCE_MASK,
kEDMA_DestinationOffsetErrorFlag = DMA_ES_DOE_MASK,
kEDMA_DestinationAddressErrorFlag = DMA_ES_DAE_MASK,
kEDMA_SourceOffsetErrorFlag = DMA_ES_SOE_MASK,
kEDMA_SourceAddressErrorFlag = DMA_ES_SAE_MASK,
kEDMA_ErrorChannelFlag = DMA_ES_ERRCHN_MASK,
kEDMA_ChannelPriorityErrorFlag = DMA_ES_CPE_MASK,
kEDMA_TransferCanceledFlag = DMA_ES_ECX_MASK,
kEDMA_GroupPriorityErrorFlag = DMA_ES_GPE_MASK,
kEDMA_ValidFlag = (int)DMA_ES_VLD_MASK }
_edma_error_status_flags eDMA channel error status flags.
- enum edma_interrupt_enable_t {
kEDMA_ErrorInterruptEnable = 0x1U,
kEDMA_MajorInterruptEnable = DMA_CSR_INTMAJOR_MASK,
kEDMA_HalfInterruptEnable = DMA_CSR_INTHALF_MASK }
eDMA interrupt source
- enum edma_transfer_type_t {
kEDMA_MemoryToMemory = 0x0U,
kEDMA_PeripheralToMemory,
kEDMA_MemoryToPeripheral,
kEDMA_PeripheralToPeripheral }
eDMA transfer type
- enum {
kStatus_EDMA_QueueFull = MAKE_STATUS(kStatusGroup_EDMA, 0),
kStatus_EDMA_Busy = MAKE_STATUS(kStatusGroup_EDMA, 1) }
_edma_transfer_status eDMA transfer status

Driver version

- #define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))
  eDMA driver version

eDMA initialization and de-initialization

- void EDMA_Init (DMA_Type ∗base, const edma_config_t ∗config)
  Initializes the eDMA peripheral.
- void EDMA_Deinit (DMA_Type ∗base)
  Deinitializes the eDMA peripheral.
- void EDMA_InstallTCD (DMA_Type ∗base, uint32_t channel, edma_tcd_t ∗tcd)
  Push content of TCD structure into hardware TCD register.
- void EDMA_GetDefaultConfig (edma_config_t ∗config)
  Gets the eDMA default configuration structure.
- static void EDMA_EnableContinuousChannelLinkMode (DMA_Type ∗base, bool enable)
  Enable/Disable continuous channel link mode.
- static void EDMA_EnableMinorLoopMapping (DMA_Type ∗base, bool enable)
  Enable/Disable minor loop mapping.

eDMA Channel Operation

- void EDMA_ResetChannel (DMA_Type ∗base, uint32_t channel)
  Sets all TCD registers to default values.
- void EDMA_SetTransferConfig (DMA_Type ∗base, uint32_t channel, const edma_transfer_config-_t ∗config, edma_tcd_t ∗nextTcd)
  Configures the eDMA transfer attribute.
- void EDMA_SetMinorOffsetConfig (DMA_Type ∗base, uint32_t channel, const edma_minor_-offset_config_t ∗config)
  Configures the eDMA minor offset feature.
- void EDMA_SetChannelPreemptionConfig (DMA_Type ∗base, uint32_t channel, const edma_-channel_Preemption_config_t ∗config)
  Configures the eDMA channel preemption feature.
- void EDMA_SetChannelLink (DMA_Type ∗base, uint32_t channel, edma_channel_link_type_-t type, uint32_t linkedChannel)
  Sets the channel link for the eDMA transfer.
- void EDMA_SetBandWidth (DMA_Type ∗base, uint32_t channel, edma_bandwidth_t bandWidth)
  Sets the bandwidth for the eDMA transfer.
- void EDMA_SetModulo (DMA_Type ∗base, uint32_t channel, edma_modulo_t srcModulo, edma-_modulo_t destModulo)
  Sets the source modulo and the destination modulo for the eDMA transfer.
- static void EDMA_EnableAsyncRequest (DMA_Type ∗base, uint32_t channel, bool enable)
  Enables an async request for the eDMA transfer.
- static void EDMA_EnableAutoStopRequest (DMA_Type ∗base, uint32_t channel, bool enable)
  Enables an auto stop request for the eDMA transfer.
- void EDMA_EnableChannelInterrupts (DMA_Type ∗base, uint32_t channel, uint32_t mask)
  Enables the interrupt source for the eDMA transfer.
- void EDMA_DisableChannelInterrupts (DMA_Type ∗base, uint32_t channel, uint32_t mask)
  Disables the interrupt source for the eDMA transfer.
- void EDMA_SetMajorOffsetConfig (DMA_Type ∗base, uint32_t channel, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA channel TCD major offset feature.

eDMA TCD Operation

- void EDMA_TcdReset (edma_tcd_t ∗tcd)
  Sets all fields to default values for the TCD structure.
- void EDMA_TcdSetTransferConfig (edma_tcd_t ∗tcd, const edma_transfer_config_t ∗config, edma_tcd_t ∗nextTcd)
  Configures the eDMA TCD transfer attribute.
- void EDMA_TcdSetMinorOffsetConfig (edma_tcd_t ∗tcd, const edma_minor_offset_config_t ∗config)
  Configures the eDMA TCD minor offset feature.
- void EDMA_TcdSetChannelLink (edma_tcd_t ∗tcd, edma_channel_link_type_t type, uint32_t linkedChannel)
  Sets the channel link for the eDMA TCD.
- static void EDMA_TcdSetBandWidth (edma_tcd_t ∗tcd, edma_bandwidth_t bandWidth)
  Sets the bandwidth for the eDMA TCD.
- void EDMA_TcdSetModulo (edma_tcd_t ∗tcd, edma_modulo_t srcModulo, edma_modulo_t destModulo)
  Sets the source modulo and the destination modulo for the eDMA TCD.
- static void EDMA_TcdEnableAutoStopRequest (edma_tcd_t ∗tcd, bool enable)
  Sets the auto stop request for the eDMA TCD.
- void EDMA_TcdEnableInterrupts (edma_tcd_t ∗tcd, uint32_t mask)
  Enables the interrupt source for the eDMA TCD.
- void EDMA_TcdDisableInterrupts (edma_tcd_t ∗tcd, uint32_t mask)
  Disables the interrupt source for the eDMA TCD.
- void EDMA_TcdSetMajorOffsetConfig (edma_tcd_t ∗tcd, int32_t sourceOffset, int32_t destOffset)
  Configures the eDMA TCD major offset feature.

eDMA Channel Transfer Operation

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

eDMA Channel Status Operation

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

eDMA Transactional Operation

- void EDMA_CreateHandle (edma_handle_t *handle, DMA_Type *base, uint32_t channel)
  Creates the eDMA handle.
- void EDMA_InstallTCDMemory (edma_handle_t *handle, edma_tcd_t *tcdPool, uint32_t tcdSize)
  Installs the TCDs memory pool into the eDMA handle.
- void EDMA_SetCallback (edma_handle_t *handle, edma_callback callback, void *userData)
  Installs a callback function for the eDMA transfer.
- void EDMA_PrepareTransferConfig (edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)
  Prepares the eDMA transfer structure configurations.
- void EDMA_PrepareTransfer (edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth, void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferBytes, edma_transfer_type_t type)
  Prepares the eDMA transfer structure.
- status_t EDMA_SubmitTransfer (edma_handle_t *handle, const edma_transfer_config_t *config)
  Submits the eDMA transfer request.
- void EDMA_StartTransfer (edma_handle_t *handle)
  eDMA starts transfer.
- void EDMA_StopTransfer (edma_handle_t *handle)
  eDMA stops transfer.
- void EDMA_AbortTransfer (edma_handle_t *handle)
  eDMA aborts transfer.
- static uint32_t EDMA_GetUnusedTCDNumber (edma_handle_t *handle)
  Get unused TCD slot number.
- static uint32_t EDMA_GetNextTCDAddress (edma_handle_t *handle)
  Get the next tcd address.
- void EDMA_HandleIRQ (edma_handle_t *handle)
  eDMA IRQ handler for the current major loop transfer completion.

Data Structure Documentation

16.3.1 struct edma_config_t

Data Fields

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

16.3.1.0.0.26 Field Documentation

16.3.1.0.0.26.1 bool edma_config_t::enableContinuousLinkMode

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

16.3.1.0.0.26.2 bool edma_config_t::enableHaltOnError

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

16.3.1.0.0.26.3 bool edma_config_t::enableDebugMode

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

16.3.2 struct edma_transfer_config_t

This structure configures the source/destination transfer attribute.

Data Fields

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

16.3.2.0.0.27 Field Documentation

16.3.2.0.0.27.1 uint32_t edma_transfer_config_t::srcAddr

16.3.2.0.0.27.2 uint32_t edma_transfer_config_t::destAddr

16.3.2.0.0.27.3 edma_transfer_size_t edma_transfer_config_t::srcTransferSize

16.3.2.0.0.27.4 edma_transfer_size_t edma_transfer_config_t::destTransferSize

16.3.2.0.0.27.5 int16_t edma_transfer_config_t::srcOffset

16.3.2.0.0.27.6 int16_t edma_transfer_config_t::destOffset

16.3.2.0.0.27.7 uint32_t edma_transfer_config_t::majorLoopCounts

16.3.3 struct edma_channel_Preemption_config_t

Data Fields

- bool enableChannelPreemption
  If true: a channel can be suspended by other channel with higher priority.
- bool enablePreemptAbility
  If true: a channel can suspend other channel with low priority.
- uint8_t channelPriority
  Channel priority.

16.3.4 struct edma_minor_offset_config_t

Data Fields

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

16.3.4.0.0.28 Field Documentation

16.3.4.0.0.28.1 bool edma_minor_offset_config_t::enableSrcMinorOffset

16.3.4.0.0.28.2 bool edma_minor_offset_config_t::enableDestMinorOffset

16.3.4.0.0.28.3 uint32_t edma_minor_offset_config_t::minorOffset

16.3.5 struct edma_tcd_t

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

Data Fields

- __IO uint32_t SADDR
  SADDR register, used to save source address.
- __IO uint16_t SOFF
  SOFF register, save offset bytes every transfer.
- __IO uint16_t ATTR
  ATTR register, source/destination transfer size and modulo.
- __IO uint32_t NBYTES
  Nbytes register, minor loop length in bytes.
- __IO uint32_t SLAST
  SLAST register.
- __IO uint32_t DADDR
  DADDR register, used for destination address.
- __IO uint16_t DOFF
  DOFF register, used for destination offset.
- __IO uint16_t CITER
  CITER register, current minor loop numbers, for unfinished minor loop.
- __IO uint32_t DLAST_SGA
  DLASTSGA register, next tcd address used in scatter-gather mode.
- __IO uint16_t CSR
  CSR register, for TCD control status.
- __IO uint16_t BITER
  BITER register, begin minor loop count.

16.3.5.0.0.29 Field Documentation

16.3.5.0.0.29.1 __IO uint16_t edma_tcd_t::CITER

16.3.5.0.0.29.2 __IO uint16_t edma_tcd_t::BITER

16.3.6 struct edma_handle_t

Data Fields

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

16.3.6.0.0.30 Field Documentation

16.3.6.0.0.30.1 edma_callback edma_handle_t::callback

16.3.6.0.0.30.2 void* edma_handle_t::userData

16.3.6.0.0.30.3 DMA_Type* edma_handle_t::base

16.3.6.0.0.30.4 edma_tcd_t* edma_handle_t::tcdPool

16.3.6.0.0.30.5 uint8_t edma_handle_t::channel

16.3.6.0.0.30.6 volatile int8_t edma_handle_t::header

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

16.3.6.0.0.30.7 volatile int8_t edma_handle_t::tail

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

16.3.6.0.0.30.8 volatile int8_t edma_handle_t::tcdUsed

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

16.3.6.0.0.30.9 volatile int8_t edma_handle_t::tcdSize

16.3.6.0.0.30.10 uint8_t edma_handle_t::flags

Macro Definition Documentation

16.4.1 #define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))

Version 2.4.1.

Typedef Documentation

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

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

Parameters

handle	EDMA handle pointer, users shall not touch the values inside.
userData	The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.
transferDone	If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDM-A register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

tcds	How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this.

Enumeration Type Documentation

16.6.1 enum edma_transfer_size_t

Enumerator

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

16.6.2 enum edma_modulo_t

Enumerator

kEDMA_ModuloDisable Disable modulo.
kEDMA_Modulo2bytes Circular buffer size is 2 bytes.
kEDMA_Modulo4bytes Circular buffer size is 4 bytes.
kEDMA_Modulo8bytes Circular buffer size is 8 bytes.
kEDMA_Modulo16bytes Circular buffer size is 16 bytes.
kEDMA_Modulo32bytes Circular buffer size is 32 bytes.
kEDMA_Modulo64bytes Circular buffer size is 64 bytes.
kEDMA_Modulo128bytes Circular buffer size is 128 bytes.
kEDMA_Modulo256bytes Circular buffer size is 256 bytes.
kEDMA_Modulo512bytes Circular buffer size is 512 bytes.
kEDMA_Modulo1Kbytes Circular buffer size is 1 K bytes.
kEDMA_Modulo2Kbytes Circular buffer size is 2 K bytes.
kEDMA_Modulo4Kbytes Circular buffer size is 4 K bytes.
kEDMA_Modulo8Kbytes Circular buffer size is 8 K bytes.
kEDMA_Modulo16Kbytes Circular buffer size is 16 K bytes.
kEDMA_Modulo32Kbytes Circular buffer size is 32 K bytes.
kEDMA_Modulo64Kbytes Circular buffer size is 64 K bytes.
kEDMA_Modulo128Kbytes Circular buffer size is 128 K bytes.
kEDMA_Modulo256Kbytes Circular buffer size is 256 K bytes.
kEDMA_Modulo512Kbytes Circular buffer size is 512 K bytes.
kEDMA_Modulo1Mbytes Circular buffer size is 1 M bytes.
kEDMA_Modulo2Mbytes Circular buffer size is 2 M bytes.
kEDMA_Modulo4Mbytes Circular buffer size is 4 M bytes.

kEDMA_Modulo8Mbytes Circular buffer size is 8 M bytes.
kEDMA_Modulo16Mbytes Circular buffer size is 16 M bytes.
kEDMA_Modulo32Mbytes Circular buffer size is 32 M bytes.
kEDMA_Modulo64Mbytes Circular buffer size is 64 M bytes.
kEDMA_Modulo128Mbytes Circular buffer size is 128 M bytes.
kEDMA_Modulo256Mbytes Circular buffer size is 256 M bytes.
kEDMA_Modulo512Mbytes Circular buffer size is 512 M bytes.
kEDMA_Modulo1Gbytes Circular buffer size is 1 G bytes.
kEDMA_Modulo2Gbytes Circular buffer size is 2 G bytes.

16.6.3 enum edma_bandwidth_t

Enumerator

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

16.6.4 enum edma_channel_link_type_t

Enumerator

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

16.6.5 anonymous enum

Enumerator

kEDMA_DoneFlag DONE flag, set while transfer finished, CITER value exhausted.
kEDMA_ErrorFlag eDMA error flag, an error occurred in a transfer
kEDMA_InterruptFlag eDMA interrupt flag, set while an interrupt occurred of this channel

16.6.6 anonymous enum

Enumerator

kEDMA_DestinationBusErrorFlag Bus error on destination address.
kEDMA_SourceBusErrorFlag Bus error on the source address.
kEDMA_ScatterGatherErrorFlag Error on the Scatter/Gather address, not 32byte aligned.
kEDMA_NbytesErrorFlag NBYTES/CITER configuration error.

kEDMA_DestinationOffsetErrorFlag Destination offset not aligned with destination size.
kEDMA_DestinationAddressErrorFlag Destination address not aligned with destination size.
kEDMA_SourceOffsetErrorFlag Source offset not aligned with source size.
kEDMA_SourceAddressErrorFlag Source address not aligned with source size.
kEDMA_ErrorChannelFlag Error channel number of the cancelled channel number.
kEDMA_ChannelPriorityErrorFlag Channel priority is not unique.
kEDMA_TransferCanceledFlag Transfer cancelled.
kEDMA_GroupPriorityErrorFlag Group priority is not unique.
kEDMA_ValidFlag No error occurred, this bit is 0. Otherwise, it is 1.

16.6.7 enum edma_interrupt_enable_t

Enumerator

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

16.6.8 enum edma_transfer_type_t

Enumerator

kEDMA_MemoryToMemory Transfer from memory to memory.
kEDMA_PeripheralToMemory Transfer from peripheral to memory.
kEDMA_MemoryToPeripheral Transfer from memory to peripheral.
kEDMA_PeripheralToPeripheral Transfer from Peripheral to peripheral.

16.6.9 anonymous enum

Enumerator

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

Function Documentation

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

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

Parameters

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

Note

This function enables the minor loop map feature.

16.7.2 void EDMA_Deinit (DMA_Type ∗ *base*)

This function gates the eDMA clock.

Parameters

base	eDMA peripheral base address.

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

Parameters

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

16.7.4 void EDMA_GetDefaultConfig (edma_config_t ∗ *config*)

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

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

Parameters

config	A pointer to the eDMA configuration structure.

16.7.5 static void EDMA_EnableContinuousChannelLinkMode (DMA_Type * *base*, bool *enable*) [inline], [static]

Note

Do not use continuous link mode with a channel linking to itself if there is only one minor loop iteration per service request, for example, if the channel's NBYTES value is the same as either the source or destination size. The same data transfer profile can be achieved by simply increasing the NBYTES value, which provides more efficient, faster processing.

Parameters

base	EDMA peripheral base address.
enable	true is enable, false is disable.

16.7.6 static void EDMA_EnableMinorLoopMapping (DMA_Type * *base*, bool *enable*) [inline], [static]

The TCDn.word2 is redefined to include individual enable fields, an offset field, and the NBYTES field.

Parameters

base	EDMA peripheral base address.
enable	true is enable, false is disable.

16.7.7 void EDMA_ResetChannel (DMA_Type * *base*, uint32_t *channel*)

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

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

Note

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

16.7.8 void EDMA_SetTransferConfig (DMA_Type ∗ *base*, uint32_t *channel*, const edma_transfer_config_t ∗ *config*, edma_tcd_t ∗ *nextTcd*)

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

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

Parameters

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

Note

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

16.7.9 void EDMA_SetMinorOffsetConfig (DMA_Type ∗ *base*, uint32_t *channel*, const edma_minor_offset_config_t ∗ *config*)

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

Parameters

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

16.7.10 void EDMA_SetChannelPreemptionConfig (DMA_Type ∗ *base*, uint32_t *channel*, const edma_channel_Preemption_config_t ∗ *config*)

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

Parameters

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

16.7.11 void EDMA_SetChannelLink (DMA_Type ∗ *base*, uint32_t *channel*, edma_channel_link_type_t *type*, uint32_t *linkedChannel*)

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

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.
type	A channel link type, which can be one of the following: • kEDMA_LinkNone • kEDMA_MinorLink • kEDMA_MajorLink

linkedChannel	The linked channel number.

Note

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

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

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

Parameters

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

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

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

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

srcModulo	A source modulo value.
destModulo	A destination modulo value.

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

Parameters

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

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

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

Parameters

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

16.7.16 void EDMA_EnableChannelInterrupts (DMA_Type * *base*, uint32_t *channel*, uint32_t *mask*)

Parameters

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

16.7.17 void EDMA_DisableChannelInterrupts (DMA_Type * *base,* uint32_t *channel,* uint32_t *mask*)

Parameters

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

16.7.18 void EDMA_SetMajorOffsetConfig (DMA_Type * *base*, uint32_t *channel*, int32_t *sourceOffset*, int32_t *destOffset*)

Adjustment value added to the source address at the completion of the major iteration count

Parameters

base	eDMA peripheral base address.
channel	edma channel number.
sourceOffset	source address offset will be applied to source address after major loop done.
destOffset	destination address offset will be applied to source address after major loop done.

16.7.19 void EDMA_TcdReset (edma_tcd_t * *tcd*)

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

Parameters

tcd	Pointer to the TCD structure.

Note

This function enables the auto stop request feature.

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

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

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

Parameters

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

Note

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

16.7.21 void EDMA_TcdSetMinorOffsetConfig (edma_tcd_t * *tcd*, const edma_minor_offset_config_t * *config*)

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

Parameters

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

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

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

Note

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

Parameters

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

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

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

Parameters

tcd	A pointer to the TCD structure.
bandWidth	A bandwidth setting, which can be one of the following: • kEDMABandwidthStallNone • kEDMABandwidthStall4Cycle • kEDMABandwidthStall8Cycle

16.7.24 void EDMA_TcdSetModulo (edma_tcd_t * *tcd*, edma_modulo_t *srcModulo*, edma_modulo_t *destModulo*)

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

Parameters

tcd	A pointer to the TCD structure.

srcModulo	A source modulo value.
destModulo	A destination modulo value.

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

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

Parameters

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

16.7.26 void EDMA_TcdEnableInterrupts (edma_tcd_t * *tcd*, uint32_t *mask*)

Parameters

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

16.7.27 void EDMA_TcdDisableInterrupts (edma_tcd_t * *tcd*, uint32_t *mask*)

Parameters

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

16.7.28 void EDMA_TcdSetMajorOffsetConfig (edma_tcd_t * *tcd*, int32_t *sourceOffset*, int32_t *destOffset*)

Adjustment value added to the source address at the completion of the major iteration count

Parameters

tcd	A point to the TCD structure.
sourceOffset	source address offset wiil be applied to source address after major loop done.
destOffset	destination address offset will be applied to source address after major loop done.

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

This function enables the hardware channel request.

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

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

This function disables the hardware channel request.

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

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

This function starts a minor loop transfer.

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

16.7.32 uint32_t EDMA_GetRemainingMajorLoopCount (DMA_Type * *base*, uint32_t *channel*)

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

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

Returns

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

Note

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

16.7.33 static uint32_t EDMA_GetErrorStatusFlags (DMA_Type ∗ *base*) [inline], [static]

Parameters

base	eDMA peripheral base address.

Returns

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

16.7.34 uint32_t EDMA_GetChannelStatusFlags (DMA_Type ∗ *base,* uint32_t *channel*)

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

Returns

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

16.7.35 void EDMA_ClearChannelStatusFlags (DMA_Type * *base*, uint32_t *channel*, uint32_t *mask*)

Parameters

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

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

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

Parameters

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

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

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

Parameters

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

16.7.38 void EDMA_SetCallback (edma_handle_t * *handle*, edma_callback *callback*, void * *userData*)

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

Parameters

handle	eDMA handle pointer.
callback	eDMA callback function pointer.
userData	A parameter for the callback function.

16.7.39 void EDMA_PrepareTransferConfig (edma_transfer_config_t * *config*, void * *srcAddr*, uint32_t *srcWidth*, int16_t *srcOffset*, void * *destAddr*, uint32_t *destWidth*, int16_t *destOffset*, uint32_t *bytesEachRequest*, uint32_t *transferBytes*)

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

Parameters

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

Note

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

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

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

Parameters

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

Note

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

16.7.41 status_t EDMA_SubmitTransfer (edma_handle_t * *handle*, const edma_transfer_config_t * *config*)

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

Parameters

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

Return values

kStatus_EDMA_Success	It means submit transfer request succeed.
kStatus_EDMA_QueueFull	It means TCD queue is full. Submit transfer request is not allowed.
kStatus_EDMA_Busy	It means the given channel is busy, need to submit request later.

16.7.42 void EDMA_StartTransfer (edma_handle_t * *handle*)

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

Parameters

handle	eDMA handle pointer.

16.7.43 void EDMA_StopTransfer (edma_handle_t * *handle*)

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

Parameters

handle	eDMA handle pointer.

16.7.44 void EDMA_AbortTransfer (edma_handle_t * *handle*)

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

Parameters

handle	DMA handle pointer.

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

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

Parameters

handle	DMA handle pointer.

Returns

The unused tcd slot number.

16.7.46 static uint32_t EDMA_GetNextTCDAddress (edma_handle_t * *handle*) [inline], [static]

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

handle	DMA handle pointer.

Returns

The next TCD address.

16.7.47 void EDMA_HandleIRQ (edma_handle_t * *handle*)

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

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga_index are calculated based on the DLAST_SGA bitfield lies in the TC-D_CSR register, the sga_index in this case should be 2 (DLAST_SGA of TCD[1] stores the address of TCD[2]). Thus, the "tcdUsed" updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description section of the Reference Manual for further details.

Parameters

handle	eDMA handle pointer.

Chapter 17
ENET: Ethernet MAC Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the 10/100 Mbps Ethernet MAC (ENET) module of MCUXpresso SDK devices.

ENET: Ethernet MAC Driver {EthernetMACDriver}

Operations of Ethernet MAC Driver

17.2.1 MII interface Operation

The MII interface is the interface connected with MAC and PHY. the Serial management interface - MII management interface should be set before any access to the external PHY chip register. Call ENET_SetSMI() to initialize the MII management interface. Use ENET_StartSMIRead(), ENET_StartSMIWrite(), and ENET_ReadSMIData() to read/write to PHY registers. This function group sets up the MII and serial management SMI interface, gets data from the SMI interface, and starts the SMI read and write command. Use ENET_SetMII() to configure the MII before successfully getting data from the external PHY.

17.2.2 MAC address filter

This group sets/gets the ENET mac address and the multicast group address filter. ENET_AddMulticastGroup() should be called to add the ENET MAC to the multicast group. The IEEE 1588 feature requires receiving the PTP message.

17.2.3 Other Baisc control Operations

This group has the receive active API ENET_ActiveRead() for single and multiple rings. The ENET_AVBConfigure() is provided to configure the AVB features to support the AVB frames transmission. Note that due to the AVB frames transmission scheme being a credit-based TX scheme, it is only supported with the Enhanced buffer descriptors. Because of this, the AVB configuration should only be done with the Enhanced buffer descriptor. When the AVB feature is required, make sure the the "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" is defined before using this feature.

17.2.4 Transactional Operation

For ENET receive, the ENET_GetRxFrameSize() function needs to be called to get the received data size. Then, call the ENET_ReadFrame() function to get the received data. If the received error occurs, call the

ENET_GetRxErrBeforeReadFrame() function after ENET_GetRxFrameSize() and before ENET_ReadFrame() functions to get the detailed error information.

For ENET transmit, call the ENET_SendFrame() function to send the data out. The transmit data error information is only accessible for the IEEE 1588 enhanced buffer descriptor mode. When the ENET_ENHANCEDBUFFERDESCRIPTOR_MODE is defined, the ENET_GetTxErrAfterSendFrame() can be used to get the detail transmit error information. The transmit error information can only be updated by uDMA after the data is transmitted. The ENET_GetTxErrAfterSendFrame() function is recommended to be called on the transmit interrupt handler.

If send/read frame with zero-copy mechanism is needed, there're special APIs like ENET_GetRxBuffer(), ENET_ReleaseRxBuffer(), ENET_SendFrameZeroCopy() and ENET_SetTxBuffer(). The send frame zero-copy APIs can't be used mixed with ENET_SendFrame() for the same ENET peripheral, same as read frame zero-copy APIs.

17.2.5 PTP IEEE 1588 Feature Operation

This function group configures the PTP IEEE 1588 feature, starts/stops/gets/sets/adjusts the PTP IEEE 1588 timer, gets the receive/transmit frame timestamp, and PTP IEEE 1588 timer channel feature setting.

The ENET_Ptp1588Configure() function needs to be called when the ENET_ENHANCEDBUFFERDESCRIPTOR_MODE is defined and the IEEE 1588 feature is required.

Typical use case

17.3.1 ENET Initialization, receive, and transmit operations

For the ENET_ENHANCEDBUFFERDESCRIPTOR_MODE undefined use case, use the legacy type buffer descriptor transmit/receive the frame as follows. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/enet For the ENET_ENHANCEDBUFFERDESCRIPTOR_MODE defined use case, add the PTP IEEE 1588 configuration to enable the PTP IEEE 1588 feature. The initialization occurs as follows. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/enet

Data Structures

- struct enet_rx_bd_struct_t
  Defines the receive buffer descriptor structure for the little endian system. More...
- struct enet_tx_bd_struct_t
  Defines the enhanced transmit buffer descriptor structure for the little endian system. More...
- struct enet_data_error_stats_t
  Defines the ENET data error statistics structure. More...
- struct enet_frame_info_t
  Defines the frame info structure. More...
- struct enet_tx_dirty_ring_t
  Defines the ENET transmit dirty addresses ring/queue structure. More...
- struct enet_buffer_config_t

Defines the receive buffer descriptor configuration structure. More...
- struct enet_config_t
  Defines the basic configuration structure for the ENET device. More...
- struct enet_tx_bd_ring_t
  Defines the ENET transmit buffer descriptor ring/queue structure. More...
- struct enet_rx_bd_ring_t
  Defines the ENET receive buffer descriptor ring/queue structure. More...
- struct enet_handle_t
  Defines the ENET handler structure. More...

Macros

- #define ENET_BUFFDESCRIPTOR_RX_ERR_MASK
  Defines the receive error status flag mask.

Typedefs

- typedef void(∗ enet_callback_t)(ENET_Type ∗base, enet_handle_t ∗handle, enet_event_t event, enet_frame_info_t ∗frameInfo, void ∗userData)
  ENET callback function.
- typedef void(∗ enet_isr_t)(ENET_Type ∗base, enet_handle_t ∗handle)
  Define interrupt IRQ handler.

Enumerations

- enum {
  kStatus_ENET_RxFrameError = MAKE_STATUS(kStatusGroup_ENET, 0U),
  kStatus_ENET_RxFrameFail = MAKE_STATUS(kStatusGroup_ENET, 1U),
  kStatus_ENET_RxFrameEmpty = MAKE_STATUS(kStatusGroup_ENET, 2U),
  kStatus_ENET_TxFrameOverLen = MAKE_STATUS(kStatusGroup_ENET, 3U),
  kStatus_ENET_TxFrameBusy = MAKE_STATUS(kStatusGroup_ENET, 4U),
  kStatus_ENET_TxFrameFail = MAKE_STATUS(kStatusGroup_ENET, 5U) }
  Defines the status return codes for transaction.
- enum enet_mii_mode_t {
  kENET_MiiMode = 0U,
  kENET_RmiiMode = 1U }
  Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.
- enum enet_mii_speed_t {
  kENET_MiiSpeed10M = 0U,
  kENET_MiiSpeed100M = 1U }
  Defines the 10/100/1000 Mbps speed for the MII data interface.
- enum enet_mii_duplex_t {
  kENET_MiiHalfDuplex = 0U,
  kENET_MiiFullDuplex }
  Defines the half or full duplex for the MII data interface.
- enum enet_mii_write_t {
  kENET_MiiWriteNoCompliant = 0U,
  kENET_MiiWriteValidFrame }
  Define the MII opcode for normal MDIO_CLAUSES_22 Frame.

- enum enet_mii_read_t {
  kENET_MiiReadValidFrame = 2U,
  kENET_MiiReadNoCompliant = 3U }
  Defines the read operation for the MII management frame.
- enum enet_mii_extend_opcode {
  kENET_MiiAddrWrite_C45 = 0U,
  kENET_MiiWriteFrame_C45 = 1U,
  kENET_MiiReadFrame_C45 = 3U }
  Define the MII opcode for extended MDIO_CLAUSES_45 Frame.
- enum enet_special_control_flag_t {
  kENET_ControlFlowControlEnable = 0x0001U,
  kENET_ControlRxPayloadCheckEnable = 0x0002U,
  kENET_ControlRxPadRemoveEnable = 0x0004U,
  kENET_ControlRxBroadCastRejectEnable = 0x0008U,
  kENET_ControlMacAddrInsert = 0x0010U,
  kENET_ControlStoreAndFwdDisable = 0x0020U,
  kENET_ControlSMIPreambleDisable = 0x0040U,
  kENET_ControlPromiscuousEnable = 0x0080U,
  kENET_ControlMIILoopEnable = 0x0100U,
  kENET_ControlVLANTagEnable = 0x0200U }
  Defines a special configuration for ENET MAC controller.
- enum enet_interrupt_enable_t {
  kENET_BabrInterrupt = ENET_EIR_BABR_MASK,
  kENET_BabtInterrupt = ENET_EIR_BABT_MASK,
  kENET_GraceStopInterrupt = ENET_EIR_GRA_MASK,
  kENET_TxFrameInterrupt = ENET_EIR_TXF_MASK,
  kENET_TxBufferInterrupt = ENET_EIR_TXB_MASK,
  kENET_RxFrameInterrupt = ENET_EIR_RXF_MASK,
  kENET_RxBufferInterrupt = ENET_EIR_RXB_MASK,
  kENET_MiiInterrupt = ENET_EIR_MII_MASK,
  kENET_EBusERInterrupt = ENET_EIR_EBERR_MASK,
  kENET_LateCollisionInterrupt = ENET_EIR_LC_MASK,
  kENET_RetryLimitInterrupt = ENET_EIR_RL_MASK,
  kENET_UnderrunInterrupt = ENET_EIR_UN_MASK,
  kENET_PayloadRxInterrupt = ENET_EIR_PLR_MASK,
  kENET_WakeupInterrupt = ENET_EIR_WAKEUP_MASK,
  kENET_TsAvailInterrupt = ENET_EIR_TS_AVAIL_MASK,
  kENET_TsTimerInterrupt = ENET_EIR_TS_TIMER_MASK }
  List of interrupts supported by the peripheral.
- enum enet_event_t {
  kENET_RxEvent,
  kENET_TxEvent,
  kENET_ErrEvent,
  kENET_WakeUpEvent,
  kENET_TimeStampEvent,
  kENET_TimeStampAvailEvent }

Defines the common interrupt event for callback use.
- enum enet_tx_accelerator_t {
kENET_TxAccelIsShift16Enabled = ENET_TACC_SHIFT16_MASK,
kENET_TxAccelIpCheckEnabled = ENET_TACC_IPCHK_MASK,
kENET_TxAccelProtoCheckEnabled = ENET_TACC_PROCHK_MASK }
Defines the transmit accelerator configuration.
- enum enet_rx_accelerator_t {
kENET_RxAccelPadRemoveEnabled = ENET_RACC_PADREM_MASK,
kENET_RxAccelIpCheckEnabled = ENET_RACC_IPDIS_MASK,
kENET_RxAccelProtoCheckEnabled = ENET_RACC_PRODIS_MASK,
kENET_RxAccelMacCheckEnabled = ENET_RACC_LINEDIS_MASK,
kENET_RxAccelisShift16Enabled = ENET_RACC_SHIFT16_MASK }
Defines the receive accelerator configuration.

Functions

- uint32_t ENET_GetInstance (ENET_Type ∗base)
Get the ENET instance from peripheral base address.

Variables

- const clock_ip_name_t s_enetClock [ ]
Pointers to enet clocks for each instance.

Driver version

- #define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 3, 4))
Defines the driver version.

Control and status region bit masks of the receive buffer descriptor.

Defines the queue number.

- #define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U
Empty bit mask.
- #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U
Software owner one mask.
- #define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U
Next buffer descriptor is the start address.
- #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U
Software owner two mask.
- #define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U
Last BD of the frame mask.
- #define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U
Received because of the promiscuous mode.
- #define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U
Broadcast packet mask.
- #define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U
Multicast packet mask.
- #define ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK 0x0020U

Length violation mask.
- #define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U
  Non-octet aligned frame mask.
- #define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U
  CRC error mask.
- #define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U
  FIFO overrun mask.
- #define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U
  Frame is truncated mask.

Control and status bit masks of the transmit buffer descriptor.

- #define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U
  Ready bit mask.
- #define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK 0x4000U
  Software owner one mask.
- #define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U
  Wrap buffer descriptor mask.
- #define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK 0x1000U
  Software owner two mask.
- #define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U
  Last BD of the frame mask.
- #define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK 0x0400U
  Transmit CRC mask.

Defines some Ethernet parameters.

- #define ENET_FRAME_MAX_FRAMELEN 1518U
  Default maximum Ethernet frame size.
- #define ENET_FIFO_MIN_RX_FULL 5U
  ENET minimum receive FIFO full.
- #define ENET_RX_MIN_BUFFERSIZE 256U
  ENET minimum buffer size.
- #define ENET_PHY_MAXADDRESS (ENET_MMFR_PA_MASK >> ENET_MMFR_PA_SHIFT)
  Maximum PHY address.
- #define ENET_TX_INTERRUPT ((uint32_t)kENET_TxFrameInterrupt | (uint32_t)kENET_TxBufferInterrupt)
  Enet Tx interrupt flag.
- #define ENET_RX_INTERRUPT ((uint32_t)kENET_RxFrameInterrupt | (uint32_t)kENET_RxBufferInterrupt)
  Enet Rx interrupt flag.
- #define ENET_TS_INTERRUPT ((uint32_t)kENET_TsTimerInterrupt | (uint32_t)kENET_TsAvailInterrupt)
  Enet timestamp interrupt flag.
- #define ENET_ERR_INTERRUPT
  Enet error interrupt flag.

Defines Tx operation flags.

- #define ENET_TX_LAST_BD_FLAG 0x01U

Tx set last buffer descriptor flag.
- #define ENET_TX_TIMESTAMP_FLAG 0x02U
  Tx timestamp flag.

Initialization and De-initialization

- void ENET_GetDefaultConfig (enet_config_t ∗config)
  Gets the ENET default configuration structure.
- void ENET_Up (ENET_Type ∗base, enet_handle_t ∗handle, const enet_config_t ∗config, const enet_buffer_config_t ∗bufferConfig, uint8_t ∗macAddr, uint32_t srcClock_Hz)
  Initializes the ENET module.
- void ENET_Init (ENET_Type ∗base, enet_handle_t ∗handle, const enet_config_t ∗config, const enet_buffer_config_t ∗bufferConfig, uint8_t ∗macAddr, uint32_t srcClock_Hz)
  Initializes the ENET module.
- void ENET_Down (ENET_Type ∗base)
  Stops the ENET module.
- void ENET_Deinit (ENET_Type ∗base)
  Deinitializes the ENET module.
- static void ENET_Reset (ENET_Type ∗base)
  Resets the ENET module.

MII interface operation

- void ENET_SetMII (ENET_Type ∗base, enet_mii_speed_t speed, enet_mii_duplex_t duplex)
  Sets the ENET MII speed and duplex.
- void ENET_SetSMI (ENET_Type ∗base, uint32_t srcClock_Hz, bool isPreambleDisabled)
  Sets the ENET SMI(serial management interface)- MII management interface.
- static bool ENET_GetSMI (ENET_Type ∗base)
  Gets the ENET SMI- MII management interface configuration.
- static uint32_t ENET_ReadSMIData (ENET_Type ∗base)
  Reads data from the PHY register through an SMI interface.
- void ENET_StartSMIRead (ENET_Type ∗base, uint32_t phyAddr, uint32_t phyReg, enet_mii_read_t operation)
  Starts an SMI (Serial Management Interface) read command.
- void ENET_StartSMIWrite (ENET_Type ∗base, uint32_t phyAddr, uint32_t phyReg, enet_mii_write_t operation, uint32_t data)
  Starts an SMI write command.
- void ENET_StartExtC45SMIRead (ENET_Type ∗base, uint32_t phyAddr, uint32_t phyReg)
  Starts the extended IEEE802.3 Clause 45 MDIO format SMI read command.
- void ENET_StartExtC45SMIWrite (ENET_Type ∗base, uint32_t phyAddr, uint32_t phyReg, uint32_t data)
  Starts the extended IEEE802.3 Clause 45 MDIO format SMI write command.
- void ENET_StartExtC45SMIWriteReg (ENET_Type ∗base, uint32_t phyAddr, uint32_t phyReg)
  Starts the extended IEEE802.3 Clause 45 MDIO format SMI write register command.
- void ENET_StartExtC45SMIWriteData (ENET_Type ∗base, uint32_t phyAddr, uint32_t phyReg, uint32_t data)
  Starts the extended IEEE802.3 Clause 45 MDIO format SMI write data command.
- void ENET_StartExtC45SMIReadData (ENET_Type ∗base, uint32_t phyAddr, uint32_t phyReg)
  Starts the extended IEEE802.3 Clause 45 MDIO format SMI read data command.

MAC Address Filter

- void ENET_SetMacAddr (ENET_Type *base, uint8_t *macAddr)
  Sets the ENET module Mac address.
- void ENET_GetMacAddr (ENET_Type *base, uint8_t *macAddr)
  Gets the ENET module Mac address.
- void ENET_AddMulticastGroup (ENET_Type *base, uint8_t *address)
  Adds the ENET device to a multicast group.
- void ENET_LeaveMulticastGroup (ENET_Type *base, uint8_t *address)
  Moves the ENET device from a multicast group.

Other basic operation

- static void ENET_ActiveRead (ENET_Type *base)
  Activates ENET read or receive.
- static void ENET_EnableSleepMode (ENET_Type *base, bool enable)
  Enables/disables the MAC to enter sleep mode.
- static void ENET_GetAccelFunction (ENET_Type *base, uint32_t *txAccelOption, uint32_t *rxAccelOption)
  Gets ENET transmit and receive accelerator functions from MAC controller.

Interrupts.

- static void ENET_EnableInterrupts (ENET_Type *base, uint32_t mask)
  Enables the ENET interrupt.
- static void ENET_DisableInterrupts (ENET_Type *base, uint32_t mask)
  Disables the ENET interrupt.
- static uint32_t ENET_GetInterruptStatus (ENET_Type *base)
  Gets the ENET interrupt status flag.
- static void ENET_ClearInterruptStatus (ENET_Type *base, uint32_t mask)
  Clears the ENET interrupt events status flag.
- void ENET_SetRxISRHandler (ENET_Type *base, enet_isr_t ISRHandler)
  Set the second level Rx IRQ handler.
- void ENET_SetTxISRHandler (ENET_Type *base, enet_isr_t ISRHandler)
  Set the second level Tx IRQ handler.
- void ENET_SetErrISRHandler (ENET_Type *base, enet_isr_t ISRHandler)
  Set the second level Err IRQ handler.

Transactional operation

- void ENET_SetCallback (enet_handle_t *handle, enet_callback_t callback, void *userData)
  Sets the callback function.
- void ENET_GetRxErrBeforeReadFrame (enet_handle_t *handle, enet_data_error_stats_t *eErrorStatic, uint8_t ringId)
  Gets the error statistics of a received frame for ENET specified ring.
- status_t ENET_GetRxFrameSize (enet_handle_t *handle, uint32_t *length, uint8_t ringId)
  Gets the size of the read frame for specified ring.
- status_t ENET_ReadFrame (ENET_Type *base, enet_handle_t *handle, uint8_t *data, uint32_t length, uint8_t ringId, uint32_t *ts)
  Reads a frame from the ENET device.

- status_t ENET_SendFrame (ENET_Type ∗base, enet_handle_t ∗handle, const uint8_t ∗data, uint32-_t length, uint8_t ringId, bool tsFlag, void ∗context)
  Transmits an ENET frame for specified ring.
- status_t ENET_SetTxReclaim (enet_handle_t ∗handle, bool isEnable, uint8_t ringId)
  Enable or disable tx descriptors reclaim mechanism.
- status_t ENET_GetRxBuffer (ENET_Type ∗base, enet_handle_t ∗handle, void ∗∗buffer, uint32_t ∗length, uint8_t ringId, bool ∗isLastBuff, uint32_t ∗ts)
  Get a receive buffer pointer of the ENET device for specified ring.
- void ENET_ReleaseRxBuffer (ENET_Type ∗base, enet_handle_t ∗handle, void ∗buffer, uint8_t ringId)
  Release receive buffer descriptor to DMA.
- status_t ENET_SendFrameZeroCopy (ENET_Type ∗base, enet_handle_t ∗handle, const uint8_-t ∗data, uint32_t length, uint8_t ringId, bool tsFlag, void ∗context)
  Transmits an ENET frame for specified ring with zero-copy.
- status_t ENET_SetTxBuffer (ENET_Type ∗base, enet_handle_t ∗handle, const uint8_t ∗data, uint32_t length, uint8_t ringId, uint8_t txFlag, void ∗context)
  Set up ENET Tx buffer descriptor, preparing for one frame stores in scattered buffer.
- void ENET_TransmitIRQHandler (ENET_Type ∗base, enet_handle_t ∗handle)
  The transmit IRQ handler.
- void ENET_ReceiveIRQHandler (ENET_Type ∗base, enet_handle_t ∗handle)
  The receive IRQ handler.
- void ENET_ErrorIRQHandler (ENET_Type ∗base, enet_handle_t ∗handle)
  Some special IRQ handler including the error, mii, wakeup irq handler.
- void ENET_CommonFrame0IRQHandler (ENET_Type ∗base)
  the common IRQ handler for the tx/rx/error etc irq handler.

Data Structure Documentation

17.4.1 struct enet_rx_bd_struct_t

Data Fields

- uint16_t length
  Buffer descriptor data length.
- uint16_t control
  Buffer descriptor control and status.
- uint8_t ∗ buffer
  Data buffer pointer.

17.4.1.0.0.31 Field Documentation

17.4.1.0.0.31.1 uint16_t enet_rx_bd_struct_t::length

17.4.1.0.0.31.2 uint16_t enet_rx_bd_struct_t::control

17.4.1.0.0.31.3 uint8_t* enet_rx_bd_struct_t::buffer

17.4.2 struct enet_tx_bd_struct_t

Data Fields

- uint16_t length
  Buffer descriptor data length.
- uint16_t control
  Buffer descriptor control and status.
- uint8_t * buffer
  Data buffer pointer.

17.4.2.0.0.32 Field Documentation

17.4.2.0.0.32.1 uint16_t enet_tx_bd_struct_t::length

17.4.2.0.0.32.2 uint16_t enet_tx_bd_struct_t::control

17.4.2.0.0.32.3 uint8_t* enet_tx_bd_struct_t::buffer

17.4.3 struct enet_data_error_stats_t

Data Fields

- uint32_t statsRxLenGreaterErr
  Receive length greater than RCR[MAX_FL].
- uint32_t statsRxAlignErr
  Receive non-octet alignment/.
- uint32_t statsRxFcsErr
  Receive CRC error.
- uint32_t statsRxOverRunErr
  Receive over run.
- uint32_t statsRxTruncateErr
  Receive truncate.

17.4.3.0.0.33 Field Documentation

17.4.3.0.0.33.1 uint32_t enet_data_error_stats_t::statsRxLenGreaterErr

17.4.3.0.0.33.2 uint32_t enet_data_error_stats_t::statsRxFcsErr

17.4.3.0.0.33.3 uint32_t enet_data_error_stats_t::statsRxOverRunErr

17.4.3.0.0.33.4 uint32_t enet_data_error_stats_t::statsRxTruncateErr

17.4.4 struct enet_frame_info_t

Data Fields

- void * context
  User specified data.

17.4.5 struct enet_tx_dirty_ring_t

Data Fields

- enet_frame_info_t * txDirtyBase
  Dirty buffer descriptor base address pointer.
- uint16_t txGenIdx
  tx generate index.
- uint16_t txConsumIdx
  tx consume index.
- uint16_t txRingLen
  tx ring length.
- bool isFull
  tx ring is full flag.

17.4.5.0.0.34 Field Documentation

17.4.5.0.0.34.1 enet_frame_info_t* enet_tx_dirty_ring_t::txDirtyBase

17.4.5.0.0.34.2 uint16_t enet_tx_dirty_ring_t::txGenIdx

17.4.5.0.0.34.3 uint16_t enet_tx_dirty_ring_t::txConsumIdx

17.4.5.0.0.34.4 uint16_t enet_tx_dirty_ring_t::txRingLen

17.4.5.0.0.34.5 bool enet_tx_dirty_ring_t::isFull

17.4.6 struct enet_buffer_config_t

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

1. The aligned receive and transmit buffer size must be evenly divisible by ENET_BUFF_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.
2. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET_BUFF_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
3. The aligned transmit and receive data buffer start address must be evenly divisible by ENET_BUFF_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber * rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber * txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.

Data Fields

- uint16_t rxBdNumber
  Receive buffer descriptor number.
- uint16_t txBdNumber
  Transmit buffer descriptor number.
- uint16_t rxBuffSizeAlign
  Aligned receive data buffer size.
- uint16_t txBuffSizeAlign
  Aligned transmit data buffer size.
- volatile enet_rx_bd_struct_t * rxBdStartAddrAlign
  Aligned receive buffer descriptor start address: should be non-cacheable.
- volatile enet_tx_bd_struct_t * txBdStartAddrAlign
  Aligned transmit buffer descriptor start address: should be non-cacheable.
- uint8_t * rxBufferAlign
  Receive data buffer start address.
- uint8_t * txBufferAlign
  Transmit data buffer start address.
- bool rxMaintainEnable
  Receive buffer cache maintain.
- bool txMaintainEnable
  Transmit buffer cache maintain.
- enet_frame_info_t * txFrameInfo
  Transmit frame information start address.

17.4.6.0.0.35 Field Documentation

17.4.6.0.0.35.1 uint16_t enet_buffer_config_t::rxBdNumber

17.4.6.0.0.35.2 uint16_t enet_buffer_config_t::txBdNumber

17.4.6.0.0.35.3 uint16_t enet_buffer_config_t::rxBuffSizeAlign

17.4.6.0.0.35.4 uint16_t enet_buffer_config_t::txBuffSizeAlign

17.4.6.0.0.35.5 volatile enet_rx_bd_struct_t* enet_buffer_config_t::rxBdStartAddrAlign

17.4.6.0.0.35.6 volatile enet_tx_bd_struct_t* enet_buffer_config_t::txBdStartAddrAlign

17.4.6.0.0.35.7 uint8_t* enet_buffer_config_t::rxBufferAlign

17.4.6.0.0.35.8 uint8_t* enet_buffer_config_t::txBufferAlign

17.4.6.0.0.35.9 bool enet_buffer_config_t::rxMaintainEnable

17.4.6.0.0.35.10 bool enet_buffer_config_t::txMaintainEnable

17.4.6.0.0.35.11 enet_frame_info_t* enet_buffer_config_t::txFrameInfo

17.4.7 struct enet_config_t

Note:

1. macSpecialConfig is used for a special control configuration, A logical OR of "enet_special_control-_flag_t". For a special configuration for MAC, set this parameter to 0.
2. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO .... txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
3. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size if smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.
4. When "kENET_ControlFlowControlEnable" is set in the macSpecialConfig, ensure that the pause-Duration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.
5. When "kENET_ControlStoreAndFwdDisabled" is set in the macSpecialConfig, ensure that the rx-FifoFullThreshold and txFifoWatermark are set for store and forward disable.
6. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The "enet_tx_accelerator_t" and "enet_rx_accelerator_t" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.

7. The intCoalesceCfg can be used in the rx or tx enabled cases to decrese the CPU loading.

Data Fields

- uint32_t macSpecialConfig
  Mac special configuration.
- uint32_t interrupt
  Mac interrupt source.
- uint16_t rxMaxFrameLen
  Receive maximum frame length.
- enet_mii_mode_t miiMode
  MII mode.
- enet_mii_speed_t miiSpeed
  MII Speed.
- enet_mii_duplex_t miiDuplex
  MII duplex.
- uint8_t rxAccelerConfig
  Receive accelerator, A logical OR of "enet_rx_accelerator_t".
- uint8_t txAccelerConfig
  Transmit accelerator, A logical OR of "enet_rx_accelerator_t".
- uint16_t pauseDuration
  For flow control enabled case: Pause duration.
- uint8_t rxFifoEmptyThreshold
  For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.
- uint8_t rxFifoStatEmptyThreshold
  `For flow control enabled case: number of frames in the receive FIFO,` *independent of size, that can be accept.*
- uint8_t rxFifoFullThreshold
  `For store and forward disable case, the data required in RX FIFO to notify` *the MAC receive ready status.*
- uint8_t txFifoWatermark
  `For store and forward disable case, the data required in TX FIFO` *before a frame transmit start.*
- uint8_t ringNum
  Number of used rings.

17.4.7.0.0.36 Field Documentation

17.4.7.0.0.36.1 uint32_t enet_config_t::macSpecialConfig

A logical OR of "enet_special_control_flag_t".

17.4.7.0.0.36.2 uint32_t enet_config_t::interrupt

A logical OR of "enet_interrupt_enable_t".

17.4.7.0.0.36.3 uint16_t enet_config_t::rxMaxFrameLen

17.4.7.0.0.36.4 enet_mii_mode_t enet_config_t::miiMode

17.4.7.0.0.36.5 enet_mii_speed_t enet_config_t::miiSpeed

17.4.7.0.0.36.6 enet_mii_duplex_t enet_config_t::miiDuplex

17.4.7.0.0.36.7 uint8_t enet_config_t::rxAccelerConfig

17.4.7.0.0.36.8 uint8_t enet_config_t::txAccelerConfig

17.4.7.0.0.36.9 uint16_t enet_config_t::pauseDuration

17.4.7.0.0.36.10 uint8_t enet_config_t::rxFifoEmptyThreshold

17.4.7.0.0.36.11 uint8_t enet_config_t::rxFifoStatEmptyThreshold

If the limit is reached, reception continues and a pause frame is triggered.

17.4.7.0.0.36.12 uint8_t enet_config_t::rxFifoFullThreshold

17.4.7.0.0.36.13 uint8_t enet_config_t::txFifoWatermark

17.4.7.0.0.36.14 uint8_t enet_config_t::ringNum

default with 1 – single ring.

17.4.8 struct enet_tx_bd_ring_t

Data Fields

- volatile enet_tx_bd_struct_t ∗ txBdBase
  Buffer descriptor base address pointer.
- uint16_t txGenIdx
  The current available transmit buffer descriptor pointer.
- uint16_t txConsumIdx
  Transmit consume index.
- volatile uint16_t txDescUsed
  Transmit descriptor used number.
- uint16_t txRingLen
  Transmit ring length.

17.4.8.0.0.37 Field Documentation

17.4.8.0.0.37.1 volatile enet_tx_bd_struct_t* enet_tx_bd_ring_t::txBdBase

17.4.8.0.0.37.2 uint16_t enet_tx_bd_ring_t::txGenIdx

17.4.8.0.0.37.3 uint16_t enet_tx_bd_ring_t::txConsumIdx

17.4.8.0.0.37.4 volatile uint16_t enet_tx_bd_ring_t::txDescUsed

17.4.8.0.0.37.5 uint16_t enet_tx_bd_ring_t::txRingLen

17.4.9 struct enet_rx_bd_ring_t

Data Fields

- volatile enet_rx_bd_struct_t * rxBdBase
  Buffer descriptor base address pointer.
- uint16_t rxGenIdx
  The current available receive buffer descriptor pointer.
- uint16_t rxRingLen
  Receive ring length.

17.4.9.0.0.38 Field Documentation

17.4.9.0.0.38.1 volatile enet_rx_bd_struct_t* enet_rx_bd_ring_t::rxBdBase

17.4.9.0.0.38.2 uint16_t enet_rx_bd_ring_t::rxGenIdx

17.4.9.0.0.38.3 uint16_t enet_rx_bd_ring_t::rxRingLen

17.4.10 struct _enet_handle

Data Fields

- enet_rx_bd_ring_t rxBdRing [FSL_FEATURE_ENET_QUEUE]
  Receive buffer descriptor.
- enet_tx_bd_ring_t txBdRing [FSL_FEATURE_ENET_QUEUE]
  Transmit buffer descriptor.
- uint16_t rxBuffSizeAlign [FSL_FEATURE_ENET_QUEUE]
  Receive buffer size alignment.
- uint16_t txBuffSizeAlign [FSL_FEATURE_ENET_QUEUE]
  Transmit buffer size alignment.
- bool rxMaintainEnable [FSL_FEATURE_ENET_QUEUE]
  Receive buffer cache maintain.
- bool txMaintainEnable [FSL_FEATURE_ENET_QUEUE]
  Transmit buffer cache maintain.
- uint8_t ringNum
  Number of used rings.

- enet_callback_t callback
  Callback function.
- void * userData
  Callback function parameter.
- enet_tx_dirty_ring_t txDirtyRing [FSL_FEATURE_ENET_QUEUE]
  Ring to store tx frame information.
- bool TxReclaimEnable [FSL_FEATURE_ENET_QUEUE]
  Tx reclaim enable flag.

17.4.10.0.0.39 Field Documentation

17.4.10.0.0.39.1 enet_rx_bd_ring_t enet_handle_t::rxBdRing[FSL_FEATURE_ENET_QUEUE]

17.4.10.0.0.39.2 enet_tx_bd_ring_t enet_handle_t::txBdRing[FSL_FEATURE_ENET_QUEUE]

17.4.10.0.0.39.3 uint16_t enet_handle_t::rxBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]

17.4.10.0.0.39.4 uint16_t enet_handle_t::txBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]

17.4.10.0.0.39.5 bool enet_handle_t::rxMaintainEnable[FSL_FEATURE_ENET_QUEUE]

17.4.10.0.0.39.6 bool enet_handle_t::txMaintainEnable[FSL_FEATURE_ENET_QUEUE]

17.4.10.0.0.39.7 uint8_t enet_handle_t::ringNum

17.4.10.0.0.39.8 enet_callback_t enet_handle_t::callback

17.4.10.0.0.39.9 void* enet_handle_t::userData

17.4.10.0.0.39.10 enet_tx_dirty_ring_t enet_handle_t::txDirtyRing[FSL_FEATURE_ENET_QUEUE]

17.4.10.0.0.39.11 bool enet_handle_t::TxReclaimEnable[FSL_FEATURE_ENET_QUEUE]

Macro Definition Documentation

17.5.1 #define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 3, 4))

17.5.2 #define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U

17.5.3 #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U

17.5.4 #define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U

17.5.5 #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U

17.5.6 #define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U

17.5.7 #define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U

17.5.8 #define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U

17.5.9 #define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U

17.5.10 #define ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK 0x0020U

17.5.11 #define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U

17.5.12 #define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U

17.5.13 #define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U

17.5.14 #define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U

17.5.15 #define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U

17.5.16 #define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK 0x4000U

17.5.17 #define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U

17.5.18 #define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK 0x1000U

17.5.19 #define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U

17.5.20 #define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK 0x0400U

17.5.21 #define ENET_BUFFDESCRIPTOR_RX_ERR_MASK

```
(ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK |
      ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK | \
     ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK |
      ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK |
      ENET_BUFFDESCRIPTOR_RX_CRC_MASK)
```

17.5.22 #define ENET_FRAME_MAX_FRAMELEN 1518U

17.5.23 #define ENET_FIFO_MIN_RX_FULL 5U

17.5.24 #define ENET_RX_MIN_BUFFERSIZE 256U

17.5.25 #define ENET_PHY_MAXADDRESS (ENET_MMFR_PA_MASK >> ENET_MMFR_PA_SHIFT)

17.5.26 #define ENET_TX_INTERRUPT ((uint32_t)kENET_TxFrameInterrupt | (uint32_t)kENET_TxBufferInterrupt)

17.5.27 #define ENET_RX_INTERRUPT ((uint32_t)kENET_RxFrameInterrupt | (uint32_t)kENET_RxBufferInterrupt)

17.5.28 #define ENET_TS_INTERRUPT ((uint32_t)kENET_TsTimerInterrupt | (uint32_t)kENET_TsAvailInterrupt)

17.5.29 #define ENET_ERR_INTERRUPT

Value:

```
((uint32_t)kENET_BabrInterrupt | (uint32_t)
      kENET_BabtInterrupt | (uint32_t)kENET_EBusERInterrupt | \
     (uint32_t)kENET_LateCollisionInterrupt | (uint32_t)
      kENET_RetryLimitInterrupt |                    \
     (uint32_t)kENET_UnderrunInterrupt | (uint32_t)
      kENET_PayloadRxInterrupt)
```

17.5.30 #define ENET_TX_LAST_BD_FLAG 0x01U

17.5.31 #define ENET_TX_TIMESTAMP_FLAG 0x02U

Typedef Documentation

17.6.1 typedef void(∗ enet_callback_t)(ENET_Type ∗base, enet_handle_t ∗handle,enet_event_t event, enet_frame_info_t ∗frameInfo, void ∗userData)

17.6.2 typedef void(∗ enet_isr_t)(ENET_Type ∗base, enet_handle_t ∗handle)

Enumeration Type Documentation

17.7.1 anonymous enum

Enumerator

kStatus_ENET_RxFrameError A frame received but data error happen.
kStatus_ENET_RxFrameFail Failed to receive a frame.
kStatus_ENET_RxFrameEmpty No frame arrive.
kStatus_ENET_TxFrameOverLen Tx frame over length.
kStatus_ENET_TxFrameBusy Tx buffer descriptors are under process.
kStatus_ENET_TxFrameFail Transmit frame fail.

17.7.2 enum enet_mii_mode_t

Enumerator

kENET_MiiMode MII mode for data interface.
kENET_RmiiMode RMII mode for data interface.

17.7.3 enum enet_mii_speed_t

Notice: "kENET_MiiSpeed1000M" only supported when mii mode is "kENET_RgmiiMode".

Enumerator

kENET_MiiSpeed10M Speed 10 Mbps.
kENET_MiiSpeed100M Speed 100 Mbps.

17.7.4 enum enet_mii_duplex_t

Enumerator

kENET_MiiHalfDuplex Half duplex mode.
kENET_MiiFullDuplex Full duplex mode.

17.7.5 enum enet_mii_write_t

Enumerator

kENET_MiiWriteNoCompliant Write frame operation, but not MII-compliant.
kENET_MiiWriteValidFrame Write frame operation for a valid MII management frame.

17.7.6 enum enet_mii_read_t

Enumerator

kENET_MiiReadValidFrame Read frame operation for a valid MII management frame.
kENET_MiiReadNoCompliant Read frame operation, but not MII-compliant.

17.7.7 enum enet_mii_extend_opcode

Enumerator

kENET_MiiAddrWrite_C45 Address Write operation.
kENET_MiiWriteFrame_C45 Write frame operation for a valid MII management frame.
kENET_MiiReadFrame_C45 Read frame operation for a valid MII management frame.

17.7.8 enum enet_special_control_flag_t

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the enet_config_t. The kENET_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifoFullThreshold and txFifoWatermark in the enet_config_t.

Enumerator

kENET_ControlFlowControlEnable Enable ENET flow control: pause frame.
kENET_ControlRxPayloadCheckEnable Enable ENET receive payload length check.
kENET_ControlRxPadRemoveEnable Padding is removed from received frames.
kENET_ControlRxBroadCastRejectEnable Enable broadcast frame reject.
kENET_ControlMacAddrInsert Enable MAC address insert.
kENET_ControlStoreAndFwdDisable Enable FIFO store and forward.
kENET_ControlSMIPreambleDisable Enable SMI preamble.
kENET_ControlPromiscuousEnable Enable promiscuous mode.
kENET_ControlMIILoopEnable Enable ENET MII loop back.
kENET_ControlVLANTagEnable Enable normal VLAN (single vlan tag).

17.7.9 enum enet_interrupt_enable_t

This enumeration uses one-bot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Enumerator

kENET_BabrInterrupt Babbling receive error interrupt source.
kENET_BabtInterrupt Babbling transmit error interrupt source.
kENET_GraceStopInterrupt Graceful stop complete interrupt source.
kENET_TxFrameInterrupt TX FRAME interrupt source.
kENET_TxBufferInterrupt TX BUFFER interrupt source.
kENET_RxFrameInterrupt RX FRAME interrupt source.
kENET_RxBufferInterrupt RX BUFFER interrupt source.
kENET_MiiInterrupt MII interrupt source.
kENET_EBusERInterrupt Ethernet bus error interrupt source.
kENET_LateCollisionInterrupt Late collision interrupt source.
kENET_RetryLimitInterrupt Collision Retry Limit interrupt source.
kENET_UnderrunInterrupt Transmit FIFO underrun interrupt source.
kENET_PayloadRxInterrupt Payload Receive error interrupt source.
kENET_WakeupInterrupt WAKEUP interrupt source.
kENET_TsAvailInterrupt TS AVAIL interrupt source for PTP.
kENET_TsTimerInterrupt TS WRAP interrupt source for PTP.

17.7.10 enum enet_event_t

Enumerator

kENET_RxEvent Receive event.
kENET_TxEvent Transmit event.
kENET_ErrEvent Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .
kENET_WakeUpEvent Wake up from sleep mode event.
kENET_TimeStampEvent Time stamp event.
kENET_TimeStampAvailEvent Time stamp available event.

17.7.11 enum enet_tx_accelerator_t

Enumerator

kENET_TxAccelIsShift16Enabled Transmit FIFO shift-16.
kENET_TxAccelIpCheckEnabled Insert IP header checksum.
kENET_TxAccelProtoCheckEnabled Insert protocol checksum.

17.7.12 enum enet_rx_accelerator_t

Enumerator

kENET_RxAccelPadRemoveEnabled Padding removal for short IP frames.
kENET_RxAccelIpCheckEnabled Discard with wrong IP header checksum.
kENET_RxAccelProtoCheckEnabled Discard with wrong protocol checksum.
kENET_RxAccelMacCheckEnabled Discard with Mac layer errors.
kENET_RxAccelisShift16Enabled Receive FIFO shift-16.

Function Documentation

17.8.1 uint32_t ENET_GetInstance (ENET_Type * *base*)

Parameters

base	ENET peripheral base address.

Returns

ENET instance.

17.8.2 void ENET_GetDefaultConfig (enet_config_t * *config*)

The purpose of this API is to get the default ENET MAC controller configure structure for ENET_Init(). User may use the initialized structure unchanged in ENET_Init(), or modify some fields of the structure before calling ENET_Init(). Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

Parameters

config	The ENET mac controller configuration structure pointer.

17.8.3 void ENET_Up (ENET_Type * *base,* enet_handle_t * *handle,* const enet_config_t * *config,* const enet_buffer_config_t * *bufferConfig,* uint8_t * *macAddr,* uint32_t *srcClock_Hz*)

This function initializes the module with the ENET configuration.

Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling ENET_Up().

Parameters

base	ENET peripheral base address.
handle	ENET handler pointer.
config	ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
bufferConfig	ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.
macAddr	ENET mac address of Ethernet device. This MAC address should be provided.
srcClock_Hz	The internal module clock source for MII clock.

17.8.4 void ENET_Init (ENET_Type * *base*, enet_handle_t * *handle*, const enet_config_t * *config*, const enet_buffer_config_t * *bufferConfig*, uint8_t * *macAddr*, uint32_t *srcClock_Hz*)

This function ungates the module clock and initializes it with the ENET configuration.

Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling ENET_Init().

Parameters

base	ENET peripheral base address.
handle	ENET handler pointer.
config	ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
bufferConfig	ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.
macAddr	ENET mac address of Ethernet device. This MAC address should be provided.
srcClock_Hz	The internal module clock source for MII clock.

17.8.5 void ENET_Down (ENET_Type * *base*)

This function disables the ENET module.

Parameters

base	ENET peripheral base address.

17.8.6 void ENET_Deinit (ENET_Type * *base*)

This function gates the module clock, clears ENET interrupts, and disables the ENET module.

Parameters

base	ENET peripheral base address.

17.8.7 static void ENET_Reset (ENET_Type * *base*) `[inline]`, `[static]`

This function restores the ENET module to reset state. Note that this function sets all registers to reset state. As a result, the ENET module can't work after calling this function.

Parameters

base	ENET peripheral base address.

17.8.8 void ENET_SetMII (ENET_Type * *base,* enet_mii_speed_t *speed,* enet_mii_duplex_t *duplex*)

This API is provided to dynamically change the speed and dulpex for MAC.

Parameters

base	ENET peripheral base address.
speed	The speed of the RMII mode.
duplex	The duplex of the RMII mode.

17.8.9 void ENET_SetSMI (ENET_Type * *base,* uint32_t *srcClock_Hz,* bool *isPreambleDisabled*)

Parameters

base	ENET peripheral base address.
srcClock_Hz	This is the ENET module clock frequency. Normally it's the system clock. See clock distribution.
isPreamble-Disabled	The preamble disable flag. • true Enables the preamble. • false Disables the preamble.

17.8.10 static bool ENET_GetSMI (ENET_Type * *base*) [inline], [static]

This API is used to get the SMI configuration to check whether the MII management interface has been set.

Parameters

base	ENET peripheral base address.

Returns

The SMI setup status true or false.

17.8.11 static uint32_t ENET_ReadSMIData (ENET_Type * base) [inline], [static]

Parameters

base	ENET peripheral base address.

Returns

The data read from PHY

17.8.12 void ENET_StartSMIRead (ENET_Type * base, uint32_t phyAddr, uint32_t phyReg, enet_mii_read_t operation)

Used for standard IEEE802.3 MDIO Clause 22 format.

Parameters

base	ENET peripheral base address.
phyAddr	The PHY address.
phyReg	The PHY register. Range from 0 $\sim$ 31.
operation	The read operation.

17.8.13 void ENET_StartSMIWrite (ENET_Type * base, uint32_t phyAddr, uint32_t phyReg, enet_mii_write_t operation, uint32_t data)

Used for standard IEEE802.3 MDIO Clause 22 format.

Parameters

base	ENET peripheral base address.
phyAddr	The PHY address.
phyReg	The PHY register. Range from 0 ∼ 31.
operation	The write operation.
data	The data written to PHY.

17.8.14 void ENET_StartExtC45SMIRead (ENET_Type ∗ *base*, uint32_t *phyAddr*, uint32_t *phyReg*)

Deprecated Do not use this function. It has been superceded by ENET_StartExtC45SMIWriteReg and ENET_StartExtC45SMIReadData.

Parameters

base	ENET peripheral base address.
phyAddr	The PHY address.
phyReg	The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): phyReg = (devaddr << 16) | regAddr.

17.8.15 void ENET_StartExtC45SMIWrite (ENET_Type ∗ *base*, uint32_t *phyAddr*, uint32_t *phyReg*, uint32_t *data*)

Deprecated Do not use this function. It has been superceded by ENET_StartExtC45SMIWriteReg and ENET_StartExtC45SMIWriteData.

Parameters

base	ENET peripheral base address.
phyAddr	The PHY address.

phyReg	The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): phyReg = (devaddr << 16) | regAddr.
data	The data written to PHY.

17.8.16 void ENET_StartExtC45SMIWriteReg (ENET_Type ∗ *base,* uint32_t *phyAddr,* uint32_t *phyReg*)

Parameters

base	ENET peripheral base address.
phyAddr	The PHY address.
phyReg	The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): phyReg = (devaddr << 16) | regAddr.

17.8.17 void ENET_StartExtC45SMIWriteData (ENET_Type ∗ *base,* uint32_t *phyAddr,* uint32_t *phyReg,* uint32_t *data*)

After writing MMFR register, we need to check whether the transmission is over. This is an example for whole precedure of clause 45 MDIO write.

```
*       ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*       ENET_StartExtC45SMIWriteReg(base, phyAddr, phyReg);
*       while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
*       {
*       }
*       ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*       ENET_StartExtC45SMIWriteData(base, phyAddr, phyReg, data);
*       while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
*       {
*       }
*       ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*
```

Parameters

base	ENET peripheral base address.

phyAddr	The PHY address.
phyReg	The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): phyReg = (devaddr << 16) | regAddr.
data	The data written to PHY.

17.8.18 void ENET_StartExtC45SMIReadData (ENET_Type ∗ *base,* uint32_t *phyAddr,* uint32_t *phyReg*)

After writing MMFR register, we need to check whether the transmission is over. This is an example for whole precedure of clause 45 MDIO read.

```
*       uint32_t data;
*       ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*       ENET_StartExtC45SMIWriteReg(base, phyAddr, phyReg);
*       while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
*       {
*       }
*       ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*       ENET_StartExtC45SMIReadData(base, phyAddr, phyReg);
*       while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
*       {
*       }
*       ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*       data = ENET_ReadSMIData(base);
*
```

Parameters

base	ENET peripheral base address.
phyAddr	The PHY address.
phyReg	The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): phyReg = (devaddr << 16) | regAddr.

17.8.19 void ENET_SetMacAddr (ENET_Type ∗ *base,* uint8_t ∗ *macAddr*)

Parameters

base	ENET peripheral base address.
macAddr	The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

17.8.20 void ENET_GetMacAddr (ENET_Type * *base*, uint8_t * *macAddr*)

Parameters

base	ENET peripheral base address.
macAddr	The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

17.8.21 void ENET_AddMulticastGroup (ENET_Type * *base*, uint8_t * *address*)

Parameters

base	ENET peripheral base address.
address	The six-byte multicast group address which is provided by application.

17.8.22 void ENET_LeaveMulticastGroup (ENET_Type * *base*, uint8_t * *address*)

Parameters

base	ENET peripheral base address.
address	The six-byte multicast group address which is provided by application.

17.8.23 static void ENET_ActiveRead (ENET_Type * *base*) `[inline]`, `[static]`

This function is to active the enet read process.

Note

> This must be called after the MAC configuration and state are ready. It must be called after the ENET_Init() and ENET_Ptp1588Configure(). This should be called when the ENET receive required.

Parameters

base	ENET peripheral base address.

17.8.24 static void ENET_EnableSleepMode (ENET_Type * *base*, bool *enable*) [inline], [static]

This function is used to set the MAC enter sleep mode. When entering sleep mode, the magic frame wakeup interrupt should be enabled to wake up MAC from the sleep mode and reset it to normal mode.

Parameters

base	ENET peripheral base address.
enable	True enable sleep mode, false disable sleep mode.

17.8.25 static void ENET_GetAccelFunction (ENET_Type * *base*, uint32_t * *txAccelOption*, uint32_t * *rxAccelOption*) [inline], [static]

Parameters

base	ENET peripheral base address.
txAccelOption	The transmit accelerator option. The "enet_tx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option.
rxAccelOption	The receive accelerator option. The "enet_rx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option.

17.8.26 static void ENET_EnableInterrupts (ENET_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enumeration members. See enet_interrupt_enable_t. For example, to enable the TX frame interrupt and RX frame interrupt, do the following.

```
*      ENET_EnableInterrupts(ENET, kENET_TxFrameInterrupt |
       kENET_RxFrameInterrupt);
*
```

Parameters

base	ENET peripheral base address.
mask	ENET interrupts to enable. This is a logical OR of the enumeration enet_interrupt_enable_t.

17.8.27 static void ENET_DisableInterrupts (ENET_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See enet_interrupt_enable_t. For example, to disable the TX frame interrupt and RX frame interrupt, do the following.

```
*     ENET_DisableInterrupts(ENET, kENET_TxFrameInterrupt |
      kENET_RxFrameInterrupt);
*
```

Parameters

base	ENET peripheral base address.
mask	ENET interrupts to disable. This is a logical OR of the enumeration enet_interrupt_enable_t.

17.8.28 static uint32_t ENET_GetInterruptStatus (ENET_Type * *base*) [inline], [static]

Parameters

base	ENET peripheral base address.

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration enet_interrupt_enable_t.

17.8.29 static void ENET_ClearInterruptStatus (ENET_Type * *base*, uint32_t *mask*) [inline], [static]

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the enet_interrupt_enable_t. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
*     ENET_ClearInterruptStatus(ENET,
      kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
*
```

Parameters

base	ENET peripheral base address.
mask	ENET interrupt source to be cleared. This is the logical OR of members of the enumeration enet_interrupt_enable_t.

17.8.30 void ENET_SetRxISRHandler (ENET_Type * *base*, enet_isr_t *ISRHandler*)

Parameters

base	ENET peripheral base address.
ISRHandler	The handler to install.

17.8.31 void ENET_SetTxISRHandler (ENET_Type * *base*, enet_isr_t *ISRHandler*)

Parameters

base	ENET peripheral base address.
ISRHandler	The handler to install.

17.8.32 void ENET_SetErrISRHandler (ENET_Type * *base*, enet_isr_t *ISRHandler*)

Parameters

base	ENET peripheral base address.
ISRHandler	The handler to install.

17.8.33 void ENET_SetCallback (enet_handle_t * *handle*, enet_callback_t *callback*, void * *userData*)

This API is provided for the application callback required case when ENET interrupt is enabled. This API should be called after calling ENET_Init.

Parameters

handle	ENET handler pointer. Should be provided by application.
callback	The ENET callback function.
userData	The callback function parameter.

17.8.34 void ENET_GetRxErrBeforeReadFrame (enet_handle_t ∗ *handle*, enet_data_error_stats_t ∗ *eErrorStatic*, uint8_t *ringId*)

This API must be called after the ENET_GetRxFrameSize and before the ENET_ReadFrame(). If the ENET_GetRxFrameSize returns kStatus_ENET_RxFrameError, the ENET_GetRxErrBeforeReadFrame can be used to get the exact error statistics. This is an example.

```
*       status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*       if (status == kStatus_ENET_RxFrameError)
*       {
*           Comments: Get the error information of the received frame.
*           ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic, 0);
*           Comments: update the receive buffer.
*           ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
*       }
*
```

Parameters

handle	The ENET handler structure pointer. This is the same handler pointer used in the ENET_Init.
eErrorStatic	The error statistics structure pointer.
ringId	The ring index, range from 0 ∼ FSL_FEATURE_ENET_QUEUE - 1.

17.8.35 status_t ENET_GetRxFrameSize (enet_handle_t ∗ *handle*, uint32_t ∗ *length*, uint8_t *ringId*)

This function gets a received frame size from the ENET buffer descriptors.

Note

The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET_GetRxFrameSize, ENET_ReadFrame() should be called to update the receive buffers if the result is not "kStatus_ENET_RxFrameEmpty".

Parameters

handle	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
length	The length of the valid frame received.
ringId	The ring index or ring number.

Return values

kStatus_ENET_RxFrame-Empty	No frame received. Should not call ENET_ReadFrame to read frame.
kStatus_ENET_RxFrame-Error	Data error happens. ENET_ReadFrame should be called with NULL data and NULL length to update the receive buffers.
kStatus_Success	Receive a frame Successfully then the ENET_ReadFrame should be called with the right data buffer and the captured data length input.

17.8.36 status_t ENET_ReadFrame (ENET_Type ∗ *base*, enet_handle_t ∗ *handle*, uint8_t ∗ *data*, uint32_t *length*, uint8_t *ringId*, uint32_t ∗ *ts*)

This function reads a frame (both the data and the length) from the ENET buffer descriptors. User can get timestamp through ts pointer if the ts is not NULL. Note that it doesn't store the timestamp in the receive timestamp queue. The ENET_GetRxFrameSize should be used to get the size of the prepared data buffer. This is an example:

```
*       uint32_t length;
*       enet_handle_t g_handle;
*       Comments: Get the received frame size firstly.
*       status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*       if (length != 0)
*       {
*           Comments: Allocate memory here with the size of "length"
*           uint8_t *data = memory allocate interface;
*           if (!data)
*           {
*               ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
*               Comments: Add the console warning log.
*           }
*           else
*           {
*               status = ENET_ReadFrame(ENET, &g_handle, data, length, 0, NULL);
*               Comments: Call stack input API to deliver the data to stack
*           }
*       }
*       else if (status == kStatus_ENET_RxFrameError)
*       {
*           Comments: Update the received buffer when a error frame is received.
*           ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
*       }
*
```

Parameters

base	ENET peripheral base address.
handle	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
data	The data buffer provided by user to store the frame which memory size should be at least "length".
length	The size of the data buffer which is still the length of the received frame.
ringId	The ring index or ring number.
ts	The timestamp address to store received timestamp.

Returns

The execute status, successful or failure.

17.8.37 status_t ENET_SendFrame (ENET_Type * *base*, enet_handle_t * *handle*, const uint8_t * *data*, uint32_t *length*, uint8_t *ringId*, bool *tsFlag*, void * *context*)

Note

The CRC is automatically appended to the data. Input the data to send without the CRC.

Parameters

base	ENET peripheral base address.
handle	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
data	The data buffer provided by user to send.
length	The length of the data to send.
ringId	The ring index or ring number.
tsFlag	Timestamp enable flag.
context	Used by user to handle some events after transmit over.

Return values

kStatus_Success	Send frame succeed.
kStatus_ENET_TxFrame-Busy	Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus-_ENET_TxFrameBusy.

17.8.38 status_t ENET_SetTxReclaim (enet_handle_t * *handle*, bool *isEnable*, uint8_t *ringId*)

Note

This function must be called when no pending send frame action. Set enable if you want to reclaim context or timestamp in interrupt.

Parameters

handle	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
isEnable	Enable or disable flag.
ringId	The ring index or ring number.

Return values

kStatus_Success	Succeed to enable/disable Tx reclaim.
kStatus_Fail	Fail to enable/disable Tx reclaim.

17.8.39 status_t ENET_GetRxBuffer (ENET_Type * *base*, enet_handle_t * *handle*, void ** *buffer*, uint32_t * *length*, uint8_t *ringId*, bool * *isLastBuff*, uint32_t * *ts*)

This function can get the data address which stores frame. Then can analyze these data directly without doing any memory copy. When the frame locates in multiple BD buffer, need to repeat calling this function until isLastBuff=true (need to store the temp buf pointer everytime call this function). After finishing the analysis of this frame, call ENET_ReleaseRxBuffer to release rxbuff memory to DMA. This is an example:

```
*       uint32_t length;
*       uint8_t *buf = NULL;
*       uint32_t data_len = 0;
*       bool isLastBuff = false;
*       enet_handle_t g_handle;
*       status_t status;
*       status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*       if (length != 0)
*       {
```

```
*           ENET_GetRxBuffer(EXAMPLE_ENET, &g_handle, &buf, &data_len, 0, &isLastBuff, NULL
      );
*           ENET_ReleaseRxBuffer(EXAMPLE_ENET, &g_handle, buf, 0);
*       }
*
```

Parameters

base	ENET peripheral base address.
handle	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
buffer	The data buffer pointer to store the frame.
length	The size of the data buffer. If isLastBuff=false, it represents data length of this buffer. If isLastBuff=true, it represents data length of total frame.
ringId	The ring index, range from 0 ∼ FSL_FEATURE_ENET_QUEUE - 1.
isLastBuff	The flag represents whether this buffer is the last buffer to store frame.
ts	The 1588 timestamp value, vaild in last buffer.

Return values

kStatus_Success	Get receive buffer succeed.
kStatus_ENET_RxFrame-Fail	Get receive buffer fails, it's owned by application, should wait app to release this buffer.

17.8.40 void ENET_ReleaseRxBuffer (ENET_Type ∗ *base*, enet_handle_t ∗ *handle*, void ∗ *buffer*, uint8_t *ringId*)

This function can release specified BD owned by application, meanwhile it may rearrange the BD to let the no-owned BDs always in back of the index of DMA transfer. So for the situation that releasing order is not same as the getting order, the rearrangement makes all ready BDs can be used by DMA.

Note

> This function can't be interrupted by ENET_GetRxBuffer, so in application must make sure ENET-_GetRxBuffer is called before or after this function. And this function itself isn't thread safe due to BD content exchanging.

Parameters

base	ENET peripheral base address.
handle	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
buffer	The buffer address to store frame, using it to find the correspond BD and release it.
ringId	The ring index, range from 0 ∼ FSL_FEATURE_ENET_QUEUE - 1.

17.8.41 status_t ENET_SendFrameZeroCopy (ENET_Type ∗ *base,* enet_handle_t ∗ *handle,* const uint8_t ∗ *data,* uint32_t *length,* uint8_t *ringId,* bool *tsFlag,* void ∗ *context*)

Note

The CRC is automatically appended to the data. Input the data to send without the CRC. The frame must store in continuous memeory and need to check the buffer start address alignment based on your device, otherwise it has issue or can't get highest DMA transmit speed.

Parameters

base	ENET peripheral base address.
handle	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
data	The data buffer provided by user to send.
length	The length of the data to send.
ringId	The ring index or ring number.
tsFlag	Timestamp enable flag.
context	Used by user to handle some events after transmit over.

Return values

kStatus_Success	Send frame succeed.
kStatus_ENET_TxFrame-Busy	Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus-_ENET_TxFrameBusy.

17.8.42 status_t ENET_SetTxBuffer (ENET_Type ∗ *base,* enet_handle_t ∗ *handle,* const uint8_t ∗ *data,* uint32_t *length,* uint8_t *ringId,* uint8_t *txFlag,* void ∗ *context*)

This function only set one Tx BD everytime calls, all ready data will be sent out with last flag sets or gets error. Send frame succeeds with last flag sets, then you can get context from frameInfo in callback.

Note

> The CRC is automatically appended to the data. Input the data to send without the CRC. And if doesn't succeed to call this function, user can't get context in frameInfo of callback.

Parameters

base	ENET peripheral base address.
handle	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
data	The data buffer provided by user to send.
length	The length of the data to send.
ringId	The ring index, range from 0 ∼ FSL_FEATURE_ENET_QUEUE - 1.
txFlag	This function uses timestamp enable flag, last BD flag.
context	Used by user to handle some events after transmit over.

Return values

kStatus_Success	Send frame succeed.
kStatus_ENET_TxFrame-OverLen	Buffer length isn't enough to store data.
kStatus_ENET_TxFrame-Busy	Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity.

17.8.43 void ENET_TransmitIRQHandler (ENET_Type ∗ *base,* enet_handle_t ∗ *handle*)

Parameters

base	ENET peripheral base address.
handle	The ENET handler pointer.

17.8.44 void ENET_ReceiveIRQHandler (ENET_Type ∗ *base,* enet_handle_t ∗ *handle*)

Parameters

base	ENET peripheral base address.
handle	The ENET handler pointer.

17.8.45 void ENET_ErrorIRQHandler (ENET_Type ∗ *base,* enet_handle_t ∗ *handle*)

Parameters

base	ENET peripheral base address.
handle	The ENET handler pointer.

17.8.46 void ENET_CommonFrame0IRQHandler (ENET_Type ∗ *base*)

This is used for the combined tx/rx/error interrupt for single/mutli-ring (frame 0).

Parameters

base	ENET peripheral base address.

Variable Documentation

17.9.1 const clock_ip_name_t s_enetClock[ ]

Chapter 18
EWM: External Watchdog Monitor Driver

Overview

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

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, 2))
  EWM driver version 2.0.2.

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.

Data Structure Documentation

18.3.1 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.

Macro Definition Documentation

18.4.1 #define FSL_EWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

Enumeration Type Documentation

18.5.1 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.

18.5.2 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.

Function Documentation

18.6.1 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

18.6.2 void EWM_Deinit (EWM_Type * *base*)

This function is used to shut down the EWM.

Parameters

base	EWM peripheral base address

18.6.3 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

18.6.4 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

18.6.5 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

18.6.6 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.

18.6.7 void EWM_Refresh (EWM_Type ∗ *base*)

This function resets the EWM counter to zero.

Parameters

base	EWM peripheral base address

Chapter 19
C90TFS Flash Driver

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

Ftftx FLASH Driver

19.2.1 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_xacc_state_t {
  kFLASH_AccessStateUnLimited,
  kFLASH_AccessStateExecuteOnly,
  kFLASH_AccessStateMixed }
  Enumeration for the three possible flash execute access 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.
- status_t FLASH_ProgramSection (flash_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)
  Programs flash with data at locations passed in through parameters via the Program Section command.

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.

FlexRAM

- status_t FLASH_SetFlexramFunction (flash_config_t *config, ftfx_flexram_func_opt_t option)
  Sets the FlexRAM function command.

Swap

- status_t FLASH_Swap (flash_config_t *config, uint32_t address, bool isSetEnable)
  Swaps the lower half flash with the higher half flash.

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_IsExecuteOnly (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, flash_xacc_state_t *access_state)
  Returns the access state of the desired flash area via the pointer passed into the function.
- 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.

19.2.2 Data Structure Documentation

19.2.2.1 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] .

19.2.2.1.0.40 Field Documentation

19.2.2.1.0.40.1 uint32_t pflash_prot_status_t::protl

19.2.2.1.0.40.2 uint32_t pflash_prot_status_t::proth

19.2.2.1.0.40.3 uint8_t pflash_prot_status_t::protsl

19.2.2.1.0.40.4 uint8_t pflash_prot_status_t::protsh

19.2.2.2 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.

19.2.3 Macro Definition Documentation

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

Version 3.1.2.

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

Version 3.0.0.

19.2.4 Enumeration Type Documentation

19.2.4.1 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.

19.2.4.2 enum flash_xacc_state_t

Enumerator

kFLASH_AccessStateUnLimited Flash region is unlimited.
kFLASH_AccessStateExecuteOnly Flash region is execute only.
kFLASH_AccessStateMixed Flash is mixed with unlimited and execute only region.

19.2.4.3 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.

19.2.5 Function Documentation

19.2.5.1 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.

19.2.5.2 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.

19.2.5.3 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_AlignmentError	The parameter is not aligned with the specified baseline.
kStatus_FTFx_AddressError	The address is out of range.
kStatus_FTFx_EraseKeyError	The API erase key is invalid.

19.2.5.4 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_EraseKeyError	API erase key is invalid.
kStatus_FTFx_ExecuteInRamFunctionNotReady	Execute-in-RAM function is not available.

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

19.2.5.5 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FTFx_AddressError	Address is out of range.

kStatus_FTFx_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.

19.2.5.6 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.

19.2.5.7 status_t FLASH_ProgramSection (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 length.

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the desired data have been programed successfully into flash based on start address and length.
kStatus_FTFx_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_Set-FlexramAsRamError	Failed to set flexram as RAM.
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_Recover-FlexramAsEepromError	Failed to recover FlexRAM as EEPROM.

19.2.5.8 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.

19.2.5.9 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.

19.2.5.10 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.

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

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

Parameters

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

Return values

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

19.2.5.12 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_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.

19.2.5.13 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.

19.2.5.14 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.

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the FlexRAM has been successfully configured as RAM or EEPROM.
kStatus_FTFx_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.

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

Parameters

config	A pointer to the storage for the driver runtime state.
address	Address used to configure the flash swap function

isSetEnable	The possible option used to configure the Flash Swap function or check the flash Swap status.

Return values

kStatus_FTFx_Success	API was executed successfully; the lower half flash and higher half flash have been swaped.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FTFx_Swap-IndicatorAddressError	Swap indicator address 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_Swap-SystemNotInUninitialized	Swap system is not in an uninitialized state.

19.2.5.17 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.

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

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

Parameters

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

Return values

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

kStatus_FTFx_-AlignmentError	The parameter is not aligned to the specified baseline.
kStatus_FTFx_Address-Error	The address is out of range.

19.2.5.19 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.

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

Parameters

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

Return values

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

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

Parameters

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

Return values

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

19.2.5.22 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_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.

Ftftx CACHE Driver

19.3.1 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.

19.3.2 Data Structure Documentation

19.3.2.1 struct ftfx_prefetch_speculation_status_t

Data Fields

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

19.3.2.1.0.41 Field Documentation

19.3.2.1.0.41.1 bool ftfx_prefetch_speculation_status_t::instructionOff

19.3.2.1.0.41.2 bool ftfx_prefetch_speculation_status_t::dataOff

19.3.2.2 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.

19.3.2.2.0.42 Field Documentation

19.3.2.2.0.42.1 function_bit_operation_ptr_t ftfx_cache_config_t::bitOperFuncAddr

19.3.3 Enumeration Type Documentation

19.3.3.1 enum _ftfx_cache_ram_func_constants

Enumerator

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

19.3.4 Function Documentation

19.3.4.1 status_t FTFx_CACHE_Init (ftfx_cache_config_t ∗ *config*)

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

Parameters

config	Pointer to the storage for the driver runtime state.

Return values

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

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

Parameters

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

Return values

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

19.3.4.3 status_t FTFx_CACHE_PflashSetPrefetchSpeculation (ftfx_prefetch_-speculation_status_t * *speculationStatus*)

Parameters

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

Return values

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

19.3.4.4 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.

Ftftx FLEXNVM Driver

19.4.1 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_DflashProgramSection (flexnvm_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)
  Programs flash with data at locations passed in through parameters via the Program Section command.
- 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.

FlexRAM

- status_t FLEXNVM_SetFlexramFunction (flexnvm_config_t *config, ftfx_flexram_func_opt_t option)
  Sets the FlexRAM function command.

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.

19.4.2 Data Structure Documentation

19.4.2.1 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.

19.4.3 Enumeration Type Documentation

19.4.3.1 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.

19.4.4 Function Documentation

19.4.4.1 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.

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

19.4.4.3 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_EraseKeyError	API erase key is invalid.
kStatus_FTFx_ExecuteInRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_AccessError	Invalid instruction codes and out-of bounds addresses.

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

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

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

Parameters

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

Return values

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

kStatus_FTFx_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.

19.4.4.5 status_t FLEXNVM_DflashProgramSection (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 length.

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the desired date have been successfully programed into specified date flash area.
kStatus_FTFx_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FTFx_AddressError	Address is out of range.
kStatus_FTFx_SetFlexramAsRamError	Failed to set flexram as RAM.

kStatus_FTFx_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_Recover-FlexramAsEepromError	Failed to recover FlexRAM as EEPROM.

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
kStatus_FTFx_Invalid-Argument	Invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.

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

19.4.4.7 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.

19.4.4.8 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.

19.4.4.9 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.

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

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

Parameters

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

Return values

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

19.4.4.11 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.

19.4.4.12 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.

19.4.4.13 status_t FLEXNVM_SetFlexramFunction (flexnvm_config_t * *config*, ftfx_flexram_func_opt_t *option*)

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully; the FlexRAM has been successfully configured as RAM or EEPROM
kStatus_FTFx_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.

19.4.4.14 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_AddressError	Address is out of range.
kStatus_FTFx_SetFlexramAsEepromError	Failed to set flexram as eeprom.
kStatus_FTFx_ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FTFx_RecoverFlexramAsRamError	Failed to recover the FlexRAM as RAM.

19.4.4.15 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.

19.4.4.16 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.

19.4.4.17 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_CommandNotSupported	Flash API is not supported.
kStatus_FTFx_CommandFailure	Run-time error during command execution.

19.4.4.18 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.

19.4.4.19 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.

ftfx feature

19.5.1 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.

19.5.2 Macro Definition Documentation

19.5.2.1 #define FTFx_DRIVER_IS_FLASH_RESIDENT 1U

Used for the flash resident application.

19.5.2.2 #define FTFx_DRIVER_IS_EXPORTED 0U

Used for the MCUXpresso SDK application.

19.5.2.3 #define FTFx_FLASH1_HAS_PROT_CONTROL (0U)

19.5.2.4 #define FTFx_FLASH1_HAS_XACC_CONTROL (0U)

19.5.3 ftfx adapter

ftfx controller

19.6.1 Overview

Modules

- ftfx utilities

Data Structures

- struct ftfx_swap_state_config_t
  Flash Swap information. More...
- 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 ftfx_swap_control_opt_t {
  kFTFx_SwapControlOptionIntializeSystem = 0x01U,
  kFTFx_SwapControlOptionSetInUpdateState = 0x02U,
  kFTFx_SwapControlOptionSetInCompleteState = 0x04U,
  kFTFx_SwapControlOptionReportStatus = 0x08U,
  kFTFx_SwapControlOptionDisableSystem = 0x10U }
  Enumeration for the possible options of Swap control commands.
- 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_swap_block_status_t {
  kFTFx_SwapBlockStatusLowerHalfProgramBlocksAtZero,
  kFTFx_SwapBlockStatusUpperHalfProgramBlocksAtZero }
  Enumeration for the possible flash Swap block 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.
- status_t FTFx_API_UpdateFlexnvmPartitionStatus (ftfx_config_t ∗config)
Updates FlexNVM memory partition status according to data flash 0 IFR.

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.
- status_t FTFx_CMD_ProgramSection (ftfx_config_t *config, uint32_t start, const uint8_t *src, uint32_t lengthInBytes)
  Programs flash with data at locations passed in through parameters via the Program Section command.
- status_t FTFx_CMD_ProgramPartition (ftfx_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 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.

FlexRAM

- status_t FTFx_CMD_SetFlexramFunction (ftfx_config_t ∗config, ftfx_flexram_func_opt_t option)
  Sets the FlexRAM function command.

Swap

- status_t FTFx_CMD_SwapControl (ftfx_config_t ∗config, uint32_t address, ftfx_swap_control_opt_t option, ftfx_swap_state_config_t ∗returnInfo)
  Configures the Swap function or checks the swap state of the Flash module.

19.6.2 Data Structure Documentation

19.6.2.1 struct ftfx_swap_state_config_t

Data Fields

- ftfx_swap_state_t flashSwapState
  The current Swap system status.
- ftfx_swap_block_status_t currentSwapBlockStatus
  The current Swap block status.
- ftfx_swap_block_status_t nextSwapBlockStatus
  The next Swap block status.

19.6.2.1.0.43 Field Documentation

19.6.2.1.0.43.1 ftfx_swap_state_t ftfx_swap_state_config_t::flashSwapState

19.6.2.1.0.43.2 ftfx_swap_block_status_t ftfx_swap_state_config_t::currentSwapBlockStatus

19.6.2.1.0.43.3 ftfx_swap_block_status_t ftfx_swap_state_config_t::nextSwapBlockStatus

19.6.2.2 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.

19.6.2.2.0.44 Field Documentation

19.6.2.2.0.44.1 uint32_t ftfx_spec_mem_t::base

19.6.2.2.0.44.2 uint32_t ftfx_spec_mem_t::size

19.6.2.2.0.44.3 uint32_t ftfx_spec_mem_t::count

19.6.2.3 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.

19.6.2.3.0.45 Field Documentation

19.6.2.3.0.45.1 uint8_t ftfx_mem_desc_t::type

19.6.2.3.0.45.2 uint8_t ftfx_mem_desc_t::index

19.6.2.3.0.45.3 uint32_t ftfx_mem_desc_t::totalSize

19.6.2.3.0.45.4 uint32_t ftfx_mem_desc_t::sectorSize

19.6.2.3.0.45.5 uint32_t ftfx_mem_desc_t::blockCount

19.6.2.4 struct ftfx_ops_config_t

Data Fields

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

19.6.2.4.0.46 Field Documentation

19.6.2.4.0.46.1 uint32_t ftfx_ops_config_t::convertedAddress

19.6.2.5 struct ftfx_ifr_desc_t

19.6.2.6 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.

19.6.2.6.0.47 Field Documentation

19.6.2.6.0.47.1 function_ptr_t ftfx_config_t::runCmdFuncAddr

19.6.3 Macro Definition Documentation

19.6.3.1 #define kStatusGroupGeneric 0

19.6.4 Enumeration Type Documentation

19.6.4.1 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.

19.6.4.2 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.

19.6.4.3 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.

19.6.4.4 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.

19.6.4.5 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.

19.6.4.6 enum ftfx_security_state_t

Enumerator

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

19.6.4.7 enum ftfx_flexram_func_opt_t

Enumerator

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

19.6.4.8 enum ftfx_swap_control_opt_t

Enumerator

kFTFx_SwapControlOptionIntializeSystem An option used to initialize the Swap system.
kFTFx_SwapControlOptionSetInUpdateState An option used to set the Swap in an update state.
kFTFx_SwapControlOptionSetInCompleteState An option used to set the Swap in a complete state.

kFTFx_SwapControlOptionReportStatus An option used to report the Swap status.
kFTFx_SwapControlOptionDisableSystem An option used to disable the Swap status.

19.6.4.9 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.

19.6.4.10 enum ftfx_swap_block_status_t

Enumerator

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

19.6.5 Function Documentation

19.6.5.1 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.

19.6.5.2 status_t FTFx_API_UpdateFlexnvmPartitionStatus (ftfx_config_t * *config*)

This function updates FlexNVM memory partition status.

Parameters

config	Pointer to the storage for the driver runtime state.

Return values

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

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

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

Parameters

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

Return values

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

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

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

Parameters

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

Return values

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

kStatus_FTFx_-AlignmentError	The parameter is not aligned with the specified baseline.
kStatus_FTFx_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.

19.6.5.5 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.

19.6.5.6 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.

19.6.5.7 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.

19.6.5.8 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_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.

19.6.5.9 status_t FTFx_CMD_ProgramSection (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 length.

Parameters

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

Return values

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

kStatus_FTFx_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FTFx_Set-FlexramAsRamError	Failed to set flexram as RAM.
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_Recover-FlexramAsEepromError	Failed to recover FlexRAM as EEPROM.

19.6.5.10 status_t FTFx_CMD_ProgramPartition (ftfx_config_t * *config*, ftfx_partition_flexram_load_opt_t *option*, uint32_t *eepromDataSizeCode*, uint32_t *flexnvmPartitionCode*)

Parameters

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

Return values

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

19.6.5.11 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_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.

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

19.6.5.12 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_InvalidArgument	An invalid argument is provided.
kStatus_FTFx_AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FTFx_ExecuteInRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_AccessError	Invalid instruction codes and out-of bounds addresses.

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

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FTFx_ExecuteIn-RamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FTFx_Access-Error	Invalid instruction codes and out-of bounds addresses.

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

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

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

Parameters

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

Return values

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

19.6.5.15 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_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.

19.6.5.16 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_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.

19.6.5.17 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.

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

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

Parameters

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

Return values

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

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_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.

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

Parameters

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

Return values

kStatus_FTFx_Success	API was executed successfully.
kStatus_FTFx_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_-AlignmentError	Parameter is not aligned with specified baseline.

kStatus_FTFx_Swap-IndicatorAddressError	Swap indicator address 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.

19.6.6 ftfx utilities

19.6.6.1 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 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.
- #define B1P4(b) (((uint32_t)(b)&0xFFU) << 24U)
  bytes2word utility.

19.6.6.2 Macro Definition Documentation

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

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

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

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

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

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

Chapter 20
FlexBus: External Bus Interface Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the Crossbar External Bus Interface (FlexBus) block of MCUXpresso SDK devices.

A multifunction external bus interface is provided on the device with a basic functionality to interface to slave-only devices. It can be directly connected to the following asynchronous or synchronous devices with little or no additional circuitry.

- External ROMs
- Flash memories
- Programmable logic devices
- Other simple target (slave) devices

For asynchronous devices, a simple chip-select based interface can be used. The FlexBus interface has up to six general purpose chip-selects, FB_CS[5:0]. The number of chip selects available depends on the device and its pin configuration.

FlexBus functional operation

To configure the FlexBus driver, use on of the two ways to configure the flexbus_config_t structure.

1. Using the FLEXBUS_GetDefaultConfig() function.
2. Set parameters in the flexbus_config_t structure.

To initialize and configure the FlexBus driver, call the FLEXBUS_Init() function and pass a pointer to the flexbus_config_t structure.

To de-initialize the FlexBus driver, call the FLEXBUS_Deinit() function.

Typical use case and example

This example shows how to write/read to external memory (MRAM) by using the FlexBus module.

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

Data Structures

- struct flexbus_config_t
  Configuration structure that the user needs to set. More...

Enumerations

- enum flexbus_port_size_t {
  kFLEXBUS_4Bytes = 0x00U,
  kFLEXBUS_1Byte = 0x01U,

kFLEXBUS_2Bytes = 0x02U }
Defines port size for FlexBus peripheral.
- enum flexbus_write_address_hold_t {
kFLEXBUS_Hold1Cycle = 0x00U,
kFLEXBUS_Hold2Cycles = 0x01U,
kFLEXBUS_Hold3Cycles = 0x02U,
kFLEXBUS_Hold4Cycles = 0x03U }
Defines number of cycles to hold address and attributes for FlexBus peripheral.
- enum flexbus_read_address_hold_t {
kFLEXBUS_Hold1Or0Cycles = 0x00U,
kFLEXBUS_Hold2Or1Cycles = 0x01U,
kFLEXBUS_Hold3Or2Cycle = 0x02U,
kFLEXBUS_Hold4Or3Cycle = 0x03U }
Defines number of cycles to hold address and attributes for FlexBus peripheral.
- enum flexbus_address_setup_t {
kFLEXBUS_FirstRisingEdge = 0x00U,
kFLEXBUS_SecondRisingEdge = 0x01U,
kFLEXBUS_ThirdRisingEdge = 0x02U,
kFLEXBUS_FourthRisingEdge = 0x03U }
Address setup for FlexBus peripheral.
- enum flexbus_bytelane_shift_t {
kFLEXBUS_NotShifted = 0x00U,
kFLEXBUS_Shifted = 0x01U }
Defines byte-lane shift for FlexBus peripheral.
- enum flexbus_multiplex_group1_t {
kFLEXBUS_MultiplexGroup1_FB_ALE = 0x00U,
kFLEXBUS_MultiplexGroup1_FB_CS1 = 0x01U,
kFLEXBUS_MultiplexGroup1_FB_TS = 0x02U }
Defines multiplex group1 valid signals.
- enum flexbus_multiplex_group2_t {
kFLEXBUS_MultiplexGroup2_FB_CS4 = 0x00U,
kFLEXBUS_MultiplexGroup2_FB_TSIZ0 = 0x01U,
kFLEXBUS_MultiplexGroup2_FB_BE_31_24 = 0x02U }
Defines multiplex group2 valid signals.
- enum flexbus_multiplex_group3_t {
kFLEXBUS_MultiplexGroup3_FB_CS5 = 0x00U,
kFLEXBUS_MultiplexGroup3_FB_TSIZ1 = 0x01U,
kFLEXBUS_MultiplexGroup3_FB_BE_23_16 = 0x02U }
Defines multiplex group3 valid signals.
- enum flexbus_multiplex_group4_t {
kFLEXBUS_MultiplexGroup4_FB_TBST = 0x00U,
kFLEXBUS_MultiplexGroup4_FB_CS2 = 0x01U,
kFLEXBUS_MultiplexGroup4_FB_BE_15_8 = 0x02U }
Defines multiplex group4 valid signals.
- enum flexbus_multiplex_group5_t {
kFLEXBUS_MultiplexGroup5_FB_TA = 0x00U,
kFLEXBUS_MultiplexGroup5_FB_CS3 = 0x01U,

kFLEXBUS_MultiplexGroup5_FB_BE_7_0 = 0x02U }
Defines multiplex group5 valid signals.

Driver version

- #define FSL_FLEXBUS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  Version 2.1.1.

FlexBus functional operation

- void FLEXBUS_Init (FB_Type *base, const flexbus_config_t *config)
  Initializes and configures the FlexBus module.
- void FLEXBUS_Deinit (FB_Type *base)
  De-initializes a FlexBus instance.
- void FLEXBUS_GetDefaultConfig (flexbus_config_t *config)
  Initializes the FlexBus configuration structure.

Data Structure Documentation

20.4.1 struct flexbus_config_t

Data Fields

- uint8_t chip
  Chip FlexBus for validation.
- uint8_t waitStates
  Value of wait states.
- uint8_t secondaryWaitStates
  Value of secondary wait states.
- uint32_t chipBaseAddress
  Chip base address for using FlexBus.
- uint32_t chipBaseAddressMask
  Chip base address mask.
- bool writeProtect
  Write protected.
- bool burstWrite
  Burst-Write enable.
- bool burstRead
  Burst-Read enable.
- bool byteEnableMode
  Byte-enable mode support.
- bool autoAcknowledge
  Auto acknowledge setting.
- bool extendTransferAddress
  Extend transfer start/extend address latch enable.
- bool secondaryWaitStatesEnable
  Enable secondary wait states.
- flexbus_port_size_t portSize
  Port size of transfer.
- flexbus_bytelane_shift_t byteLaneShift

Byte-lane shift enable.
- flexbus_write_address_hold_t writeAddressHold
  Write address hold or deselect option.
- flexbus_read_address_hold_t readAddressHold
  Read address hold or deselect option.
- flexbus_address_setup_t addressSetup
  Address setup setting.
- flexbus_multiplex_group1_t group1MultiplexControl
  FlexBus Signal Group 1 Multiplex control.
- flexbus_multiplex_group2_t group2MultiplexControl
  FlexBus Signal Group 2 Multiplex control.
- flexbus_multiplex_group3_t group3MultiplexControl
  FlexBus Signal Group 3 Multiplex control.
- flexbus_multiplex_group4_t group4MultiplexControl
  FlexBus Signal Group 4 Multiplex control.
- flexbus_multiplex_group5_t group5MultiplexControl
  FlexBus Signal Group 5 Multiplex control.

Macro Definition Documentation

20.5.1 #define FSL_FLEXBUS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

Enumeration Type Documentation

20.6.1 enum flexbus_port_size_t

Enumerator

kFLEXBUS_4Bytes 32-bit port size
kFLEXBUS_1Byte 8-bit port size
kFLEXBUS_2Bytes 16-bit port size

20.6.2 enum flexbus_write_address_hold_t

Enumerator

kFLEXBUS_Hold1Cycle Hold address and attributes one cycles after FB_CSn negates on writes.
kFLEXBUS_Hold2Cycles Hold address and attributes two cycles after FB_CSn negates on writes.
kFLEXBUS_Hold3Cycles Hold address and attributes three cycles after FB_CSn negates on writes.

kFLEXBUS_Hold4Cycles Hold address and attributes four cycles after FB_CSn negates on writes.

20.6.3 enum flexbus_read_address_hold_t

Enumerator

kFLEXBUS_Hold1Or0Cycles Hold address and attributes 1 or 0 cycles on reads.

kFLEXBUS_Hold2Or1Cycles Hold address and attributes 2 or 1 cycles on reads.
kFLEXBUS_Hold3Or2Cycle Hold address and attributes 3 or 2 cycles on reads.
kFLEXBUS_Hold4Or3Cycle Hold address and attributes 4 or 3 cycles on reads.

20.6.4 enum flexbus_address_setup_t

Enumerator

kFLEXBUS_FirstRisingEdge Assert FB_CSn on first rising clock edge after address is asserted.
kFLEXBUS_SecondRisingEdge Assert FB_CSn on second rising clock edge after address is asserted.
kFLEXBUS_ThirdRisingEdge Assert FB_CSn on third rising clock edge after address is asserted.
kFLEXBUS_FourthRisingEdge Assert FB_CSn on fourth rising clock edge after address is asserted.

20.6.5 enum flexbus_bytelane_shift_t

Enumerator

kFLEXBUS_NotShifted Not shifted. Data is left-justified on FB_AD
kFLEXBUS_Shifted Shifted. Data is right justified on FB_AD

20.6.6 enum flexbus_multiplex_group1_t

Enumerator

kFLEXBUS_MultiplexGroup1_FB_ALE FB_ALE.
kFLEXBUS_MultiplexGroup1_FB_CS1 FB_CS1.
kFLEXBUS_MultiplexGroup1_FB_TS FB_TS.

20.6.7 enum flexbus_multiplex_group2_t

Enumerator

kFLEXBUS_MultiplexGroup2_FB_CS4 FB_CS4.
kFLEXBUS_MultiplexGroup2_FB_TSIZ0 FB_TSIZ0.
kFLEXBUS_MultiplexGroup2_FB_BE_31_24 FB_BE_31_24.

20.6.8 enum flexbus_multiplex_group3_t

Enumerator

kFLEXBUS_MultiplexGroup3_FB_CS5 FB_CS5.
kFLEXBUS_MultiplexGroup3_FB_TSIZ1 FB_TSIZ1.
kFLEXBUS_MultiplexGroup3_FB_BE_23_16 FB_BE_23_16.

20.6.9 enum flexbus_multiplex_group4_t

Enumerator

kFLEXBUS_MultiplexGroup4_FB_TBST FB_TBST.
kFLEXBUS_MultiplexGroup4_FB_CS2 FB_CS2.
kFLEXBUS_MultiplexGroup4_FB_BE_15_8 FB_BE_15_8.

20.6.10 enum flexbus_multiplex_group5_t

Enumerator

kFLEXBUS_MultiplexGroup5_FB_TA FB_TA.
kFLEXBUS_MultiplexGroup5_FB_CS3 FB_CS3.
kFLEXBUS_MultiplexGroup5_FB_BE_7_0 FB_BE_7_0.

Function Documentation

20.7.1 void FLEXBUS_Init (FB_Type * *base*, const flexbus_config_t * *config*)

This function enables the clock gate for FlexBus module. Only chip 0 is validated and set to known values. Other chips are disabled. Note that in this function, certain parameters, depending on external memories, must be set before using the FLEXBUS_Init() function. This example shows how to set up the uart_state_t and the flexbus_config_t parameters and how to call the FLEXBUS_Init function by passing in these parameters.

```
flexbus_config_t flexbusConfig;
FLEXBUS_GetDefaultConfig(&flexbusConfig);
flexbusConfig.waitStates            = 2U;
flexbusConfig.chipBaseAddress       = 0x60000000U;
flexbusConfig.chipBaseAddressMask   = 7U;
FLEXBUS_Init(FB, &flexbusConfig);
```

Parameters

base	FlexBus peripheral address.
config	Pointer to the configuration structure

20.7.2 void FLEXBUS_Deinit (FB_Type * *base*)

This function disables the clock gate of the FlexBus module clock.

Parameters

base	FlexBus peripheral address.

20.7.3 void FLEXBUS_GetDefaultConfig (flexbus_config_t * *config*)

This function initializes the FlexBus configuration structure to default value. The default values are.

```
fbConfig->chip                  = 0;
fbConfig->writeProtect          = 0;
fbConfig->burstWrite            = 0;
fbConfig->burstRead             = 0;
fbConfig->byteEnableMode        = 0;
fbConfig->autoAcknowledge       = true;
fbConfig->extendTransferAddress = 0;
fbConfig->secondaryWaitStates   = 0;
fbConfig->byteLaneShift         = kFLEXBUS_NotShifted;
fbConfig->writeAddressHold      = kFLEXBUS_Hold1Cycle;
fbConfig->readAddressHold       = kFLEXBUS_Hold1Or0Cycles;
fbConfig->addressSetup          = kFLEXBUS_FirstRisingEdge;
fbConfig->portSize              = kFLEXBUS_1Byte;
fbConfig->group1MultiplexControl = kFLEXBUS_MultiplexGroup1_FB_ALE;
fbConfig->group2MultiplexControl = kFLEXBUS_MultiplexGroup2_FB_CS4 ;
fbConfig->group3MultiplexControl = kFLEXBUS_MultiplexGroup3_FB_CS5;
fbConfig->group4MultiplexControl = kFLEXBUS_MultiplexGroup4_FB_TBST;
fbConfig->group5MultiplexControl = kFLEXBUS_MultiplexGroup5_FB_TA;
```

Parameters

config	Pointer to the initialization structure.

See Also

FLEXBUS_Init

Chapter 21
FlexCAN: Flex Controller Area Network Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the Flex Controller Area Network (FlexCAN) module of MCUXpresso SDK devices.

Modules

- FlexCAN Driver

FlexCAN Driver

21.2.1 Overview

This section describes the programming interface of the FlexCAN driver. The FlexCAN driver configures FlexCAN module and provides functional and transactional interfaces to build the FlexCAN application.

21.2.2 Typical use case

21.2.2.1 Message Buffer Send Operation

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

21.2.2.2 Message Buffer Receive Operation

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

21.2.2.3 Receive FIFO Operation

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

Data Structures

- struct flexcan_frame_t
  FlexCAN message frame structure. More...
- struct flexcan_timing_config_t
  FlexCAN protocol timing characteristic configuration structure. More...
- struct flexcan_config_t
  FlexCAN module configuration structure. More...
- struct flexcan_rx_mb_config_t
  FlexCAN Receive Message Buffer configuration structure. More...
- struct flexcan_rx_fifo_config_t
  FlexCAN Rx FIFO configuration structure. More...
- struct flexcan_mb_transfer_t
  FlexCAN Message Buffer transfer. More...
- struct flexcan_fifo_transfer_t
  FlexCAN Rx FIFO transfer. More...
- struct flexcan_handle_t
  FlexCAN handle structure. More...

Macros

- #define FLEXCAN_ID_STD(id) (((uint32_t)(((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)
  FlexCAN Frame ID helper macro.
- #define FLEXCAN_ID_EXT(id)
  Extend Frame ID helper macro.
- #define FLEXCAN_RX_MB_STD_MASK(id, rtr, ide)
  FlexCAN Rx Message Buffer Mask helper macro.
- #define FLEXCAN_RX_MB_EXT_MASK(id, rtr, ide)
  Extend Rx Message Buffer Mask helper macro.
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)
  FlexCAN Rx FIFO Mask helper macro.
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(id, rtr, ide)
  Standard Rx FIFO Mask helper macro Type B upper part helper macro.
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(id, rtr, ide)
  Standard Rx FIFO Mask helper macro Type B lower part helper macro.
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(id) (((uint32_t)(id)&0x7F8) << 21)
  Standard Rx FIFO Mask helper macro Type C upper part helper macro.
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(id) (((uint32_t)(id)&0x7F8) << 13)
  Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(id) (((uint32_t)(id)&0x7F8) << 5)
  Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.
- #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(id) (((uint32_t)(id)&0x7F8) >> 3)
  Standard Rx FIFO Mask helper macro Type C lower part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)
  Extend Rx FIFO Mask helper macro Type A helper macro.
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(id, rtr, ide)
  Extend Rx FIFO Mask helper macro Type B upper part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(id, rtr, ide)
  Extend Rx FIFO Mask helper macro Type B lower part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) << 3)
  Extend Rx FIFO Mask helper macro Type C upper part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(id)
  Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)
  Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 21)
  Extend Rx FIFO Mask helper macro Type C lower part helper macro.
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide) FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)
  FlexCAN Rx FIFO Filter helper macro.
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)
  Standard Rx FIFO Filter helper macro Type B upper part helper macro.
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)

Standard Rx FIFO Filter helper macro Type B lower part helper macro.
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)
  Standard Rx FIFO Filter helper macro Type C upper part helper macro.
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)
  Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)
  Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.
- #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)
  Standard Rx FIFO Filter helper macro Type C lower part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide) FLEXCAN_RX_FIFO_EXT-_MASK_TYPE_A(id, rtr, ide)
  Extend Rx FIFO Filter helper macro Type A helper macro.
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)
  Extend Rx FIFO Filter helper macro Type B upper part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)
  Extend Rx FIFO Filter helper macro Type B lower part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)
  Extend Rx FIFO Filter helper macro Type C upper part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)
  Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)
  Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.
- #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id) FLEXCAN_RX_FIFO_EXT_-MASK_TYPE_C_LOW(id)
  Extend Rx FIFO Filter helper macro Type C lower part helper macro.

Typedefs

- typedef void(∗ flexcan_transfer_callback_t)(CAN_Type ∗base, flexcan_handle_t ∗handle, status_t status, uint32_t result, void ∗userData)
  FlexCAN transfer callback function.

Enumerations

- enum {
kStatus_FLEXCAN_TxBusy = MAKE_STATUS(kStatusGroup_FLEXCAN, 0),
kStatus_FLEXCAN_TxIdle = MAKE_STATUS(kStatusGroup_FLEXCAN, 1),
kStatus_FLEXCAN_TxSwitchToRx,
kStatus_FLEXCAN_RxBusy = MAKE_STATUS(kStatusGroup_FLEXCAN, 3),
kStatus_FLEXCAN_RxIdle = MAKE_STATUS(kStatusGroup_FLEXCAN, 4),
kStatus_FLEXCAN_RxOverflow = MAKE_STATUS(kStatusGroup_FLEXCAN, 5),
kStatus_FLEXCAN_RxFifoBusy = MAKE_STATUS(kStatusGroup_FLEXCAN, 6),
kStatus_FLEXCAN_RxFifoIdle = MAKE_STATUS(kStatusGroup_FLEXCAN, 7),
kStatus_FLEXCAN_RxFifoOverflow = MAKE_STATUS(kStatusGroup_FLEXCAN, 8),
kStatus_FLEXCAN_RxFifoWarning = MAKE_STATUS(kStatusGroup_FLEXCAN, 9),
kStatus_FLEXCAN_ErrorStatus = MAKE_STATUS(kStatusGroup_FLEXCAN, 10),
kStatus_FLEXCAN_WakeUp = MAKE_STATUS(kStatusGroup_FLEXCAN, 11),
kStatus_FLEXCAN_UnHandled = MAKE_STATUS(kStatusGroup_FLEXCAN, 12),
kStatus_FLEXCAN_RxRemote = MAKE_STATUS(kStatusGroup_FLEXCAN, 13) }
  FlexCAN transfer status.
- enum flexcan_frame_format_t {
kFLEXCAN_FrameFormatStandard = 0x0U,
kFLEXCAN_FrameFormatExtend = 0x1U }
  FlexCAN frame format.
- enum flexcan_frame_type_t {
kFLEXCAN_FrameTypeData = 0x0U,
kFLEXCAN_FrameTypeRemote = 0x1U }
  FlexCAN frame type.
- enum flexcan_clock_source_t {
kFLEXCAN_ClkSrcOsc = 0x0U,
kFLEXCAN_ClkSrcPeri = 0x1U,
kFLEXCAN_ClkSrc0 = 0x0U,
kFLEXCAN_ClkSrc1 = 0x1U }
  FlexCAN clock source.
- enum flexcan_wake_up_source_t {
kFLEXCAN_WakeupSrcUnfiltered = 0x0U,
kFLEXCAN_WakeupSrcFiltered = 0x1U }
  FlexCAN wake up source.
- enum flexcan_rx_fifo_filter_type_t {
kFLEXCAN_RxFifoFilterTypeA = 0x0U,
kFLEXCAN_RxFifoFilterTypeB,
kFLEXCAN_RxFifoFilterTypeC,
kFLEXCAN_RxFifoFilterTypeD = 0x3U }
  FlexCAN Rx Fifo Filter type.
- enum flexcan_rx_fifo_priority_t {
kFLEXCAN_RxFifoPrioLow = 0x0U,
kFLEXCAN_RxFifoPrioHigh = 0x1U }
  FlexCAN Rx FIFO priority.
- enum _flexcan_interrupt_enable {

kFLEXCAN_BusOffInterruptEnable = CAN_CTRL1_BOFFMSK_MASK,
kFLEXCAN_ErrorInterruptEnable = CAN_CTRL1_ERRMSK_MASK,
kFLEXCAN_RxWarningInterruptEnable = CAN_CTRL1_RWRNMSK_MASK,
kFLEXCAN_TxWarningInterruptEnable = CAN_CTRL1_TWRNMSK_MASK,
kFLEXCAN_WakeUpInterruptEnable = CAN_MCR_WAKMSK_MASK }
FlexCAN interrupt configuration structure, default settings all disabled.
- enum _flexcan_flags {
kFLEXCAN_SynchFlag = CAN_ESR1_SYNCH_MASK,
kFLEXCAN_TxWarningIntFlag = CAN_ESR1_TWRNINT_MASK,
kFLEXCAN_RxWarningIntFlag = CAN_ESR1_RWRNINT_MASK,
kFLEXCAN_TxErrorWarningFlag = CAN_ESR1_TXWRN_MASK,
kFLEXCAN_RxErrorWarningFlag = CAN_ESR1_RXWRN_MASK,
kFLEXCAN_IdleFlag = CAN_ESR1_IDLE_MASK,
kFLEXCAN_FaultConfinementFlag = CAN_ESR1_FLTCONF_MASK,
kFLEXCAN_TransmittingFlag = CAN_ESR1_TX_MASK,
kFLEXCAN_ReceivingFlag = CAN_ESR1_RX_MASK,
kFLEXCAN_BusOffIntFlag = CAN_ESR1_BOFFINT_MASK,
kFLEXCAN_ErrorIntFlag = CAN_ESR1_ERRINT_MASK,
kFLEXCAN_WakeUpIntFlag = CAN_ESR1_WAKINT_MASK }
FlexCAN status flags.
- enum _flexcan_error_flags {
kFLEXCAN_StuffingError = CAN_ESR1_STFERR_MASK,
kFLEXCAN_FormError = CAN_ESR1_FRMERR_MASK,
kFLEXCAN_CrcError = CAN_ESR1_CRCERR_MASK,
kFLEXCAN_AckError = CAN_ESR1_ACKERR_MASK,
kFLEXCAN_Bit0Error = CAN_ESR1_BIT0ERR_MASK,
kFLEXCAN_Bit1Error = CAN_ESR1_BIT1ERR_MASK }
FlexCAN error status flags.
- enum {
kFLEXCAN_RxFifoOverflowFlag = CAN_IFLAG1_BUF7I_MASK,
kFLEXCAN_RxFifoWarningFlag = CAN_IFLAG1_BUF6I_MASK,
kFLEXCAN_RxFifoFrameAvlFlag = CAN_IFLAG1_BUF5I_MASK }
FlexCAN Rx FIFO status flags.

Driver version

- #define FSL_FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
FlexCAN driver version.

Initialization and deinitialization

- void FLEXCAN_EnterFreezeMode (CAN_Type *base)
Enter FlexCAN Freeze Mode.
- void FLEXCAN_ExitFreezeMode (CAN_Type *base)
Exit FlexCAN Freeze Mode.

- uint32_t FLEXCAN_GetInstance (CAN_Type *base)
  Get the FlexCAN instance from peripheral base address.
- bool FLEXCAN_CalculateImprovedTimingValues (uint32_t baudRate, uint32_t sourceClock_Hz, flexcan_timing_config_t *pTimingConfig)
  Calculates the improved timing values by specific baudrates for classical CAN.
- void FLEXCAN_Init (CAN_Type *base, const flexcan_config_t *pConfig, uint32_t sourceClock_Hz)
  Initializes a FlexCAN instance.
- void FLEXCAN_Deinit (CAN_Type *base)
  De-initializes a FlexCAN instance.
- void FLEXCAN_GetDefaultConfig (flexcan_config_t *pConfig)
  Gets the default configuration structure.

Configuration.

- void FLEXCAN_SetTimingConfig (CAN_Type *base, const flexcan_timing_config_t *pConfig)
  Sets the FlexCAN protocol timing characteristic.
- void FLEXCAN_SetRxMbGlobalMask (CAN_Type *base, uint32_t mask)
  Sets the FlexCAN receive message buffer global mask.
- void FLEXCAN_SetRxFifoGlobalMask (CAN_Type *base, uint32_t mask)
  Sets the FlexCAN receive FIFO global mask.
- void FLEXCAN_SetRxIndividualMask (CAN_Type *base, uint8_t maskIdx, uint32_t mask)
  Sets the FlexCAN receive individual mask.
- void FLEXCAN_SetTxMbConfig (CAN_Type *base, uint8_t mbIdx, bool enable)
  Configures a FlexCAN transmit message buffer.
- void FLEXCAN_SetRxMbConfig (CAN_Type *base, uint8_t mbIdx, const flexcan_rx_mb_config_t *pRxMbConfig, bool enable)
  Configures a FlexCAN Receive Message Buffer.
- void FLEXCAN_SetRxFifoConfig (CAN_Type *base, const flexcan_rx_fifo_config_t *pRxFifoConfig, bool enable)
  Configures the FlexCAN Rx FIFO.

Status

- static uint32_t FLEXCAN_GetStatusFlags (CAN_Type *base)
  Gets the FlexCAN module interrupt flags.
- static void FLEXCAN_ClearStatusFlags (CAN_Type *base, uint32_t mask)
  Clears status flags with the provided mask.
- static void FLEXCAN_GetBusErrCount (CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)
  Gets the FlexCAN Bus Error Counter value.
- static uint32_t FLEXCAN_GetMbStatusFlags (CAN_Type *base, uint32_t mask)
  Gets the FlexCAN Message Buffer interrupt flags.
- static void FLEXCAN_ClearMbStatusFlags (CAN_Type *base, uint32_t mask)
  Clears the FlexCAN Message Buffer interrupt flags.

Interrupts

- static void FLEXCAN_EnableInterrupts (CAN_Type *base, uint32_t mask)
    Enables FlexCAN interrupts according to the provided mask.
- static void FLEXCAN_DisableInterrupts (CAN_Type *base, uint32_t mask)
    Disables FlexCAN interrupts according to the provided mask.
- static void FLEXCAN_EnableMbInterrupts (CAN_Type *base, uint32_t mask)
    Enables FlexCAN Message Buffer interrupts.
- static void FLEXCAN_DisableMbInterrupts (CAN_Type *base, uint32_t mask)
    Disables FlexCAN Message Buffer interrupts.

Bus Operations

- static void FLEXCAN_Enable (CAN_Type *base, bool enable)
    Enables or disables the FlexCAN module operation.
- status_t FLEXCAN_WriteTxMb (CAN_Type *base, uint8_t mbIdx, const flexcan_frame_t *pTxFrame)
    Writes a FlexCAN Message to the Transmit Message Buffer.
- status_t FLEXCAN_ReadRxMb (CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pRxFrame)
    Reads a FlexCAN Message from Receive Message Buffer.
- status_t FLEXCAN_ReadRxFifo (CAN_Type *base, flexcan_frame_t *pRxFrame)
    Reads a FlexCAN Message from Rx FIFO.

Transactional

- status_t FLEXCAN_TransferSendBlocking (CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pTxFrame)
    Performs a polling send transaction on the CAN bus.
- status_t FLEXCAN_TransferReceiveBlocking (CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pRxFrame)
    Performs a polling receive transaction on the CAN bus.
- status_t FLEXCAN_TransferReceiveFifoBlocking (CAN_Type *base, flexcan_frame_t *pRxFrame)
    Performs a polling receive transaction from Rx FIFO on the CAN bus.
- void FLEXCAN_TransferCreateHandle (CAN_Type *base, flexcan_handle_t *handle, flexcan_transfer_callback_t callback, void *userData)
    Initializes the FlexCAN handle.
- status_t FLEXCAN_TransferSendNonBlocking (CAN_Type *base, flexcan_handle_t *handle, flexcan_mb_transfer_t *pMbXfer)
    Sends a message using IRQ.
- status_t FLEXCAN_TransferReceiveNonBlocking (CAN_Type *base, flexcan_handle_t *handle, flexcan_mb_transfer_t *pMbXfer)
    Receives a message using IRQ.
- status_t FLEXCAN_TransferReceiveFifoNonBlocking (CAN_Type *base, flexcan_handle_t *handle, flexcan_fifo_transfer_t *pFifoXfer)
    Receives a message from Rx FIFO using IRQ.
- uint32_t FLEXCAN_GetTimeStamp (flexcan_handle_t *handle, uint8_t mbIdx)

Gets the detail index of Mailbox's Timestamp by handle.
- void FLEXCAN_TransferAbortSend (CAN_Type ∗base, flexcan_handle_t ∗handle, uint8_t mbIdx)
  Aborts the interrupt driven message send process.
- void FLEXCAN_TransferAbortReceive (CAN_Type ∗base, flexcan_handle_t ∗handle, uint8_t mbIdx)
  Aborts the interrupt driven message receive process.
- void FLEXCAN_TransferAbortReceiveFifo (CAN_Type ∗base, flexcan_handle_t ∗handle)
  Aborts the interrupt driven message receive from Rx FIFO process.
- void FLEXCAN_TransferHandleIRQ (CAN_Type ∗base, flexcan_handle_t ∗handle)
  FlexCAN IRQ handle function.

21.2.3 Data Structure Documentation

21.2.3.1 struct flexcan_frame_t

21.2.3.1.0.1 Field Documentation

21.2.3.1.0.1.1 uint32_t flexcan_frame_t::timestamp

21.2.3.1.0.1.2 uint32_t flexcan_frame_t::length

21.2.3.1.0.1.3 uint32_t flexcan_frame_t::type

21.2.3.1.0.1.4 uint32_t flexcan_frame_t::format

21.2.3.1.0.1.5 uint32_t flexcan_frame_t::__pad0__

21.2.3.1.0.1.6 uint32_t flexcan_frame_t::idhit

21.2.3.1.0.1.7 uint32_t flexcan_frame_t::id

21.2.3.1.0.1.8 uint32_t flexcan_frame_t::dataWord0

21.2.3.1.0.1.9 uint32_t flexcan_frame_t::dataWord1

21.2.3.1.0.1.10 uint8_t flexcan_frame_t::dataByte3

21.2.3.1.0.1.11 uint8_t flexcan_frame_t::dataByte2

21.2.3.1.0.1.12 uint8_t flexcan_frame_t::dataByte1

21.2.3.1.0.1.13 uint8_t flexcan_frame_t::dataByte0

21.2.3.1.0.1.14 uint8_t flexcan_frame_t::dataByte7

21.2.3.1.0.1.15 uint8_t flexcan_frame_t::dataByte6

21.2.3.1.0.1.16 uint8_t flexcan_frame_t::dataByte5

21.2.3.1.0.1.17 uint8_t flexcan_frame_t::dataByte4

21.2.3.2 struct flexcan_timing_config_t

Data Fields

- uint16_t preDivider
  Clock Pre-scaler Division Factor.
- uint8_t rJumpwidth
  Re-sync Jump Width.
- uint8_t phaseSeg1
  Phase Segment 1.

- uint8_t phaseSeg2
  Phase Segment 2.
- uint8_t propSeg
  Propagation Segment.

21.2.3.2.0.2 Field Documentation

21.2.3.2.0.2.1 uint16_t flexcan_timing_config_t::preDivider

21.2.3.2.0.2.2 uint8_t flexcan_timing_config_t::rJumpwidth

21.2.3.2.0.2.3 uint8_t flexcan_timing_config_t::phaseSeg1

21.2.3.2.0.2.4 uint8_t flexcan_timing_config_t::phaseSeg2

21.2.3.2.0.2.5 uint8_t flexcan_timing_config_t::propSeg

21.2.3.3 struct flexcan_config_t

Data Fields

- uint32_t baudRate
  FlexCAN baud rate in bps.
- flexcan_clock_source_t clkSrc
  Clock source for FlexCAN Protocol Engine.
- flexcan_wake_up_source_t wakeupSrc
  Wake up source selection.
- uint8_t maxMbNum
  The maximum number of Message Buffers used by user.
- bool enableLoopBack
  Enable or Disable Loop Back Self Test Mode.
- bool enableTimerSync
  Enable or Disable Timer Synchronization.
- bool enableSelfWakeup
  Enable or Disable Self Wakeup Mode.
- bool enableIndividMask
  Enable or Disable Rx Individual Mask.
- bool disableSelfReception
  Enable or Disable Self Reflection.
- bool enableListenOnlyMode
  Enable or Disable Listen Only Mode.

21.2.3.3.0.3 Field Documentation

21.2.3.3.0.3.1 uint32_t flexcan_config_t::baudRate

21.2.3.3.0.3.2 flexcan_clock_source_t flexcan_config_t::clkSrc

21.2.3.3.0.3.3 flexcan_wake_up_source_t flexcan_config_t::wakeupSrc

21.2.3.3.0.3.4 uint8_t flexcan_config_t::maxMbNum

21.2.3.3.0.3.5 bool flexcan_config_t::enableLoopBack

21.2.3.3.0.3.6 bool flexcan_config_t::enableTimerSync

21.2.3.3.0.3.7 bool flexcan_config_t::enableSelfWakeup

21.2.3.3.0.3.8 bool flexcan_config_t::enableIndividMask

21.2.3.3.0.3.9 bool flexcan_config_t::disableSelfReception

21.2.3.3.0.3.10 bool flexcan_config_t::enableListenOnlyMode

21.2.3.4 struct flexcan_rx_mb_config_t

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_-SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

Data Fields

- uint32_t id
  CAN Message Buffer Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.
- flexcan_frame_format_t format
  CAN Frame Identifier format(Standard of Extend).
- flexcan_frame_type_t type
  CAN Frame Type(Data or Remote).

21.2.3.4.0.4 Field Documentation

21.2.3.4.0.4.1 uint32_t flexcan_rx_mb_config_t::id

21.2.3.4.0.4.2 flexcan_frame_format_t flexcan_rx_mb_config_t::format

21.2.3.4.0.4.3 flexcan_frame_type_t flexcan_rx_mb_config_t::type

21.2.3.5 struct flexcan_rx_fifo_config_t

Data Fields

- uint32_t * idFilterTable
  Pointer to the FlexCAN Rx FIFO identifier filter table.
- uint8_t idFilterNum
  The quantity of filter elements.
- flexcan_rx_fifo_filter_type_t idFilterType
  The FlexCAN Rx FIFO Filter type.
- flexcan_rx_fifo_priority_t priority
  The FlexCAN Rx FIFO receive priority.

21.2.3.5.0.5 Field Documentation

21.2.3.5.0.5.1 uint32_t* flexcan_rx_fifo_config_t::idFilterTable

21.2.3.5.0.5.2 uint8_t flexcan_rx_fifo_config_t::idFilterNum

21.2.3.5.0.5.3 flexcan_rx_fifo_filter_type_t flexcan_rx_fifo_config_t::idFilterType

21.2.3.5.0.5.4 flexcan_rx_fifo_priority_t flexcan_rx_fifo_config_t::priority

21.2.3.6 struct flexcan_mb_transfer_t

Data Fields

- flexcan_frame_t * frame
  The buffer of CAN Message to be transfer.
- uint8_t mbIdx
  The index of Message buffer used to transfer Message.

21.2.3.6.0.6 Field Documentation

21.2.3.6.0.6.1 flexcan_frame_t* flexcan_mb_transfer_t::frame

21.2.3.6.0.6.2 uint8_t flexcan_mb_transfer_t::mbIdx

21.2.3.7 struct flexcan_fifo_transfer_t

Data Fields

- flexcan_frame_t * frame

The buffer of CAN Message to be received from Rx FIFO.

21.2.3.7.0.7 Field Documentation

21.2.3.7.0.7.1 flexcan_frame_t* flexcan_fifo_transfer_t::frame

21.2.3.8 struct _flexcan_handle

FlexCAN handle structure definition.

Data Fields

- flexcan_transfer_callback_t callback
  Callback function.
- void * userData
  FlexCAN callback function parameter.
- flexcan_frame_t *volatile mbFrameBuf [CAN_WORD1_COUNT]
  The buffer for received data from Message Buffers.
- flexcan_frame_t *volatile rxFifoFrameBuf
  The buffer for received data from Rx FIFO.
- volatile uint8_t mbState [CAN_WORD1_COUNT]
  Message Buffer transfer state.
- volatile uint8_t rxFifoState
  Rx FIFO transfer state.
- volatile uint32_t timestamp [CAN_WORD1_COUNT]
  Mailbox transfer timestamp.

21.2.3.8.0.8 Field Documentation

21.2.3.8.0.8.1 flexcan_transfer_callback_t flexcan_handle_t::callback

21.2.3.8.0.8.2 void* flexcan_handle_t::userData

21.2.3.8.0.8.3 flexcan_frame_t* volatile flexcan_handle_t::mbFrameBuf[CAN_WORD1_COUNT]

21.2.3.8.0.8.4 flexcan_frame_t* volatile flexcan_handle_t::rxFifoFrameBuf

21.2.3.8.0.8.5 volatile uint8_t flexcan_handle_t::mbState[CAN_WORD1_COUNT]

21.2.3.8.0.8.6 volatile uint8_t flexcan_handle_t::rxFifoState

21.2.3.8.0.8.7 volatile uint32_t flexcan_handle_t::timestamp[CAN_WORD1_COUNT]

21.2.4 Macro Definition Documentation

21.2.4.1 #define FSL_FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

21.2.4.2 #define FLEXCAN_ID_STD(*id*) (((uint32_t)(((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)

Standard Frame ID helper macro.

21.2.4.3 #define FLEXCAN_ID_EXT(*id*)

Value:

```
(((uint32_t)(((uint32_t)(id)) << CAN_ID_EXT_SHIFT)) & \
     (CAN_ID_EXT_MASK | CAN_ID_STD_MASK))
```

21.2.4.4 #define FLEXCAN_RX_MB_STD_MASK(*id, rtr, ide*)

Value:

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
     FLEXCAN_ID_STD(id))
```

Standard Rx Message Buffer Mask helper macro.

21.2.4.5 #define FLEXCAN_RX_MB_EXT_MASK(*id, rtr, ide*)

Value:

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
     FLEXCAN_ID_EXT(id))
```

21.2.4.6 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(*id, rtr, ide*)

Value:

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
      (FLEXCAN_ID_STD(id) << 1))
```

Standard Rx FIFO Mask helper macro Type A helper macro.

21.2.4.7 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(*id, rtr, ide*)

Value:

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
      (((uint32_t)(id)&0x7FF) << 19))
```

21.2.4.8 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(*id, rtr, ide*)

Value:

```
(((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
      (((uint32_t)(id)&0x7FF) << 3))
```

21.2.4.9 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(*id*) (((uint32_t)(id)&0x7F8) << 21)

21.2.4.10 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(*id*) (((uint32_t)(id)&0x7F8) << 13)

21.2.4.11 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(*id*) (((uint32_t)(id)&0x7F8) << 5)

21.2.4.12 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(*id*) (((uint32_t)(id)&0x7F8) >> 3)

21.2.4.13 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(*id, rtr, ide*)

Value:

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
      (FLEXCAN_ID_EXT(id) << 1))
```

21.2.4.14 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(*id, rtr, ide*)

Value:

```
(                                                                             \
        ((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
        ((FLEXCAN_ID_EXT(id) & 0x1FFF8000)                                        \
         << 1))
```

21.2.4.15 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(*id, rtr, ide*)

Value:

```
(((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
     ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) >>                                    \
      15))
```

21.2.4.16 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(*id*) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) << 3)

21.2.4.17 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(*id*)

Value:

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >>            \
     5)
```

21.2.4.18 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(*id*)

Value:

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >>           \
     13)
```

21.2.4.19 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(*id*) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 21)

21.2.4.20 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(*id, rtr, ide*) FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)

Standard Rx FIFO Filter helper macro Type A helper macro.

21.2.4.21 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(*id*, *rtr*, *ide*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(                    \
        id, rtr, ide)
```

21.2.4.22 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(*id*, *rtr*, *ide*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(                     \
        id, rtr, ide)
```

21.2.4.23 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(*id*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(          \
        id)
```

21.2.4.24 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(*id*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(          \
        id)
```

21.2.4.25 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(*id*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(          \
        id)
```

21.2.4.26 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(*id*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(          \
        id)
```

21.2.4.27 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(*id, rtr, ide*) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)

21.2.4.28 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(*id, rtr, ide*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(                    \
        id, rtr, ide)
```

21.2.4.29 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(*id, rtr, ide*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(                    \
        id, rtr, ide)
```

21.2.4.30 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(*id*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(          \
        id)
```

21.2.4.31 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(*id*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(          \
        id)
```

21.2.4.32 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(*id*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(          \
        id)
```

21.2.4.33 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(*id*) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)

21.2.5 Typedef Documentation

21.2.5.1 typedef void(* flexcan_transfer_callback_t)(CAN_Type *base, flexcan_handle_t *handle, status_t status, uint32_t result, void *userData)

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus-_FLEXCAN_ErrorStatus, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

21.2.6 Enumeration Type Documentation

21.2.6.1 anonymous enum

Enumerator

kStatus_FLEXCAN_TxBusy Tx Message Buffer is Busy.
kStatus_FLEXCAN_TxIdle Tx Message Buffer is Idle.
kStatus_FLEXCAN_TxSwitchToRx Remote Message is send out and Message buffer changed to Receive one.
kStatus_FLEXCAN_RxBusy Rx Message Buffer is Busy.
kStatus_FLEXCAN_RxIdle Rx Message Buffer is Idle.
kStatus_FLEXCAN_RxOverflow Rx Message Buffer is Overflowed.
kStatus_FLEXCAN_RxFifoBusy Rx Message FIFO is Busy.
kStatus_FLEXCAN_RxFifoIdle Rx Message FIFO is Idle.
kStatus_FLEXCAN_RxFifoOverflow Rx Message FIFO is overflowed.
kStatus_FLEXCAN_RxFifoWarning Rx Message FIFO is almost overflowed.
kStatus_FLEXCAN_ErrorStatus FlexCAN Module Error and Status.
kStatus_FLEXCAN_WakeUp FlexCAN is waken up from STOP mode.
kStatus_FLEXCAN_UnHandled UnHadled Interrupt asserted.
kStatus_FLEXCAN_RxRemote Rx Remote Message Received in Mail box.

21.2.6.2 enum flexcan_frame_format_t

Enumerator

kFLEXCAN_FrameFormatStandard Standard frame format attribute.
kFLEXCAN_FrameFormatExtend Extend frame format attribute.

21.2.6.3 enum flexcan_frame_type_t

Enumerator

kFLEXCAN_FrameTypeData Data frame type attribute.
kFLEXCAN_FrameTypeRemote Remote frame type attribute.

21.2.6.4 enum flexcan_clock_source_t

Deprecated Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

Enumerator

kFLEXCAN_ClkSrcOsc FlexCAN Protocol Engine clock from Oscillator.
kFLEXCAN_ClkSrcPeri FlexCAN Protocol Engine clock from Peripheral Clock.
kFLEXCAN_ClkSrc0 FlexCAN Protocol Engine clock selected by user as SRC == 0.
kFLEXCAN_ClkSrc1 FlexCAN Protocol Engine clock selected by user as SRC == 1.

21.2.6.5 enum flexcan_wake_up_source_t

Enumerator

kFLEXCAN_WakeupSrcUnfiltered FlexCAN uses unfiltered Rx input to detect edge.
kFLEXCAN_WakeupSrcFiltered FlexCAN uses filtered Rx input to detect edge.

21.2.6.6 enum flexcan_rx_fifo_filter_type_t

Enumerator

kFLEXCAN_RxFifoFilterTypeA One full ID (standard and extended) per ID Filter element.
kFLEXCAN_RxFifoFilterTypeB Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.
kFLEXCAN_RxFifoFilterTypeC Four partial 8-bit Standard or extended ID slices per ID Filter Table element.
kFLEXCAN_RxFifoFilterTypeD All frames rejected.

21.2.6.7 enum flexcan_rx_fifo_priority_t

The matching process starts from the Rx MB(or Rx FIFO) with higher priority. If no MB(or Rx FIFO filter) is satisfied, the matching process goes on with the Rx FIFO(or Rx MB) with lower priority.

Enumerator

kFLEXCAN_RxFifoPrioLow Matching process start from Rx Message Buffer first.
kFLEXCAN_RxFifoPrioHigh Matching process start from Rx FIFO first.

21.2.6.8 enum _flexcan_interrupt_enable

This structure contains the settings for all of the FlexCAN Module interrupt configurations. Note: FlexCAN Message Buffers and Rx FIFO have their own interrupts.

Enumerator

kFLEXCAN_BusOffInterruptEnable Bus Off interrupt.
kFLEXCAN_ErrorInterruptEnable Error interrupt.
kFLEXCAN_RxWarningInterruptEnable Rx Warning interrupt.
kFLEXCAN_TxWarningInterruptEnable Tx Warning interrupt.
kFLEXCAN_WakeUpInterruptEnable Wake Up interrupt.

21.2.6.9 enum _flexcan_flags

This provides constants for the FlexCAN status flags for use in the FlexCAN functions. Note: The CPU read action clears FlEXCAN_ErrorFlag, therefore user need to read FlEXCAN_ErrorFlag and distinguish which error is occur using _flexcan_error_flags enumerations.

Enumerator

kFLEXCAN_SynchFlag CAN Synchronization Status.
kFLEXCAN_TxWarningIntFlag Tx Warning Interrupt Flag.
kFLEXCAN_RxWarningIntFlag Rx Warning Interrupt Flag.
kFLEXCAN_TxErrorWarningFlag Tx Error Warning Status.
kFLEXCAN_RxErrorWarningFlag Rx Error Warning Status.
kFLEXCAN_IdleFlag CAN IDLE Status Flag.
kFLEXCAN_FaultConfinementFlag Fault Confinement State Flag.
kFLEXCAN_TransmittingFlag FlexCAN In Transmission Status.
kFLEXCAN_ReceivingFlag FlexCAN In Reception Status.
kFLEXCAN_BusOffIntFlag Bus Off Interrupt Flag.
kFLEXCAN_ErrorIntFlag Error Interrupt Flag.
kFLEXCAN_WakeUpIntFlag Wake-Up Interrupt Flag.

21.2.6.10 enum _flexcan_error_flags

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN_ErrorFlag in _flexcan_flags enumerations to ditermine which error is generated.

Enumerator

kFLEXCAN_StuffingError Stuffing Error.
kFLEXCAN_FormError Form Error.
kFLEXCAN_CrcError Cyclic Redundancy Check Error.
kFLEXCAN_AckError Received no ACK on transmission.
kFLEXCAN_Bit0Error Unable to send dominant bit.
kFLEXCAN_Bit1Error Unable to send recessive bit.

21.2.6.11 anonymous enum

The FlexCAN Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 $\sim$ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Enumerator

kFLEXCAN_RxFifoOverflowFlag Rx FIFO overflow flag.
kFLEXCAN_RxFifoWarningFlag Rx FIFO almost full flag.
kFLEXCAN_RxFifoFrameAvlFlag Frames available in Rx FIFO flag.

21.2.7 Function Documentation

21.2.7.1 void FLEXCAN_EnterFreezeMode (CAN_Type ∗ *base*)

This function makes the FlexCAN work under Freeze Mode.

Parameters

base	FlexCAN peripheral base address.

21.2.7.2 void FLEXCAN_ExitFreezeMode (CAN_Type ∗ *base*)

This function makes the FlexCAN leave Freeze Mode.

Parameters

base	FlexCAN peripheral base address.

21.2.7.3 uint32_t FLEXCAN_GetInstance (CAN_Type ∗ *base*)

Parameters

base	FlexCAN peripheral base address.

Returns

FlexCAN instance.

21.2.7.4 bool FLEXCAN_CalculateImprovedTimingValues (uint32_t *baudRate*, uint32_t *sourceClock_Hz*, flexcan_timing_config_t ∗ *pTimingConfig*)

Parameters

baudRate	The classical CAN speed in bps defined by user
sourceClock_-Hz	The Source clock data speed in bps. Zero to disable baudrate switching
pTimingConfig	Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

21.2.7.5 void FLEXCAN_Init (CAN_Type ∗ *base*, const flexcan_config_t ∗ *pConfig*, uint32_t *sourceClock_Hz*)

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_Init function by passing in these parameters.

```
*   flexcan_config_t flexcanConfig;
*   flexcanConfig.clkSrc               = kFLEXCAN_ClkSrc0;
*   flexcanConfig.baudRate             = 1000000U;
*   flexcanConfig.maxMbNum             = 16;
*   flexcanConfig.enableLoopBack       = false;
*   flexcanConfig.enableSelfWakeup     = false;
*   flexcanConfig.enableIndividMask    = false;
*   flexcanConfig.enableDoze           = false;
*   flexcanConfig.disableSelfReception = false;
*   flexcanConfig.enableListenOnlyMode = false;
*   flexcanConfig.timingConfig         = timingConfig;
*   FLEXCAN_Init(CAN0, &flexcanConfig, 8000000UL);
*
```

Parameters

base	FlexCAN peripheral base address.
pConfig	Pointer to the user-defined configuration structure.
sourceClock_-Hz	FlexCAN Protocol Engine clock source frequency in Hz.

21.2.7.6 void FLEXCAN_Deinit (CAN_Type ∗ *base*)

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

base	FlexCAN peripheral base address.

21.2.7.7 void FLEXCAN_GetDefaultConfig (flexcan_config_t ∗ *pConfig*)

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0; flexcanConfig->baudRate = 1000000U; flexcanConfig->baudRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig->enableMemoryErrorControl = true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true; flexcanConfig.timingConfig = timingConfig;

Parameters

pConfig	Pointer to the FlexCAN configuration structure.

21.2.7.8 void FLEXCAN_SetTimingConfig (CAN_Type ∗ *base*, const flexcan_timing_config_t ∗ *pConfig*)

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_Init() and fill the baud rate field with a desired value. This provides the default timing characteristics to the module.

Note that calling FLEXCAN_SetTimingConfig() overrides the baud rate set in FLEXCAN_Init().

Parameters

base	FlexCAN peripheral base address.
pConfig	Pointer to the timing configuration structure.

21.2.7.9 void FLEXCAN_SetRxMbGlobalMask (CAN_Type * *base*, uint32_t *mask*)

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the FLEXCAN_Init().

Parameters

base	FlexCAN peripheral base address.
mask	Rx Message Buffer Global Mask value.

21.2.7.10 void FLEXCAN_SetRxFifoGlobalMask (CAN_Type * *base*, uint32_t *mask*)

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

base	FlexCAN peripheral base address.
mask	Rx Fifo Global Mask value.

21.2.7.11 void FLEXCAN_SetRxIndividualMask (CAN_Type * *base*, uint8_t *maskIdx*, uint32_t *mask*)

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the FLEXCAN_Init(). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

base	FlexCAN peripheral base address.

maskIdx	The Index of individual Mask.
mask	Rx Individual Mask value.

21.2.7.12 void FLEXCAN_SetTxMbConfig (CAN_Type ∗ *base,* uint8_t *mbIdx,* bool *enable*)

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

base	FlexCAN peripheral base address.
mbIdx	The Message Buffer index.
enable	Enable/disable Tx Message Buffer. • true: Enable Tx Message Buffer. • false: Disable Tx Message Buffer.

21.2.7.13 void FLEXCAN_SetRxMbConfig (CAN_Type ∗ *base,* uint8_t *mbIdx,* const flexcan_rx_mb_config_t ∗ *pRxMbConfig,* bool *enable*)

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

base	FlexCAN peripheral base address.
mbIdx	The Message Buffer index.
pRxMbConfig	Pointer to the FlexCAN Message Buffer configuration structure.
enable	Enable/disable Rx Message Buffer. • true: Enable Rx Message Buffer. • false: Disable Rx Message Buffer.

21.2.7.14 void FLEXCAN_SetRxFifoConfig (CAN_Type ∗ *base,* const flexcan_rx_fifo_config_t ∗ *pRxFifoConfig,* bool *enable*)

This function configures the Rx FIFO with given Rx FIFO configuration.

Parameters

base	FlexCAN peripheral base address.
pRxFifoConfig	Pointer to the FlexCAN Rx FIFO configuration structure.
enable	Enable/disable Rx FIFO. • true: Enable Rx FIFO. • false: Disable Rx FIFO.

21.2.7.15 static uint32_t FLEXCAN_GetStatusFlags (CAN_Type * *base*) [inline], [static]

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators _flexcan_flags. To check the specific status, compare the return value with enumerators in _flexcan_flags.

Parameters

base	FlexCAN peripheral base address.

Returns

FlexCAN status flags which are ORed by the enumerators in the _flexcan_flags.

21.2.7.16 static void FLEXCAN_ClearStatusFlags (CAN_Type * *base*, uint32_t *mask*) [inline], [static]

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

base	FlexCAN peripheral base address.
mask	The status flags to be cleared, it is logical OR value of _flexcan_flags.

21.2.7.17 static void FLEXCAN_GetBusErrCount (CAN_Type * *base*, uint8_t * *txErrBuf*, uint8_t * *rxErrBuf*) [inline], [static]

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

base	FlexCAN peripheral base address.
txErrBuf	Buffer to store Tx Error Counter value.
rxErrBuf	Buffer to store Rx Error Counter value.

21.2.7.18 static uint32_t FLEXCAN_GetMbStatusFlags (CAN_Type * *base,* uint32_t *mask*) [inline], [static]

This function gets the interrupt flags of a given Message Buffers.

Parameters

base	FlexCAN peripheral base address.
mask	The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

21.2.7.19 static void FLEXCAN_ClearMbStatusFlags (CAN_Type * *base,* uint32_t *mask*) [inline], [static]

This function clears the interrupt flags of a given Message Buffers.

Parameters

base	FlexCAN peripheral base address.
mask	The ORed FlexCAN Message Buffer mask.

21.2.7.20 static void FLEXCAN_EnableInterrupts (CAN_Type * *base,* uint32_t *mask*) [inline], [static]

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see _flexcan_interrupt_enable.

Parameters

base	FlexCAN peripheral base address.
mask	The interrupts to enable. Logical OR of _flexcan_interrupt_enable.

21.2.7.21 static void FLEXCAN_DisableInterrupts (CAN_Type ∗ *base*, uint32_t *mask*) [inline], [static]

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see _flexcan_interrupt_enable.

Parameters

base	FlexCAN peripheral base address.
mask	The interrupts to disable. Logical OR of _flexcan_interrupt_enable.

21.2.7.22 static void FLEXCAN_EnableMbInterrupts (CAN_Type ∗ *base*, uint32_t *mask*) [inline], [static]

This function enables the interrupts of given Message Buffers.

Parameters

base	FlexCAN peripheral base address.
mask	The ORed FlexCAN Message Buffer mask.

21.2.7.23 static void FLEXCAN_DisableMbInterrupts (CAN_Type ∗ *base*, uint32_t *mask*) [inline], [static]

This function disables the interrupts of given Message Buffers.

Parameters

base	FlexCAN peripheral base address.
mask	The ORed FlexCAN Message Buffer mask.

21.2.7.24 static void FLEXCAN_Enable (CAN_Type ∗ *base*, bool *enable*) [inline], [static]

This function enables or disables the FlexCAN module.

Parameters

base	FlexCAN base pointer.
enable	true to enable, false to disable.

21.2.7.25 status_t FLEXCAN_WriteTxMb (CAN_Type * *base*, uint8_t *mbIdx*, const flexcan_frame_t * *pTxFrame*)

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

base	FlexCAN peripheral base address.
mbIdx	The FlexCAN Message Buffer index.
pTxFrame	Pointer to CAN message frame to be sent.

Return values

kStatus_Success	- Write Tx Message Buffer Successfully.
kStatus_Fail	- Tx Message Buffer is currently in use.

21.2.7.26 status_t FLEXCAN_ReadRxMb (CAN_Type * *base*, uint8_t *mbIdx*, flexcan_frame_t * *pRxFrame*)

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

base	FlexCAN peripheral base address.
mbIdx	The FlexCAN Message Buffer index.
pRxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Rx Message Buffer is full and has been read successfully.
kStatus_FLEXCAN_Rx-Overflow	- Rx Message Buffer is already overflowed and has been read successfully.
kStatus_Fail	- Rx Message Buffer is empty.

21.2.7.27 status_t FLEXCAN_ReadRxFifo (CAN_Type * *base*, flexcan_frame_t * *pRxFrame*)

This function reads a CAN message from the FlexCAN build-in Rx FIFO.

Parameters

base	FlexCAN peripheral base address.
pRxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Read Message from Rx FIFO successfully.
kStatus_Fail	- Rx FIFO is not enabled.

21.2.7.28 status_t FLEXCAN_TransferSendBlocking (CAN_Type * *base*, uint8_t *mbIdx*, flexcan_frame_t * *pTxFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	FlexCAN peripheral base pointer.
mbIdx	The FlexCAN Message Buffer index.
pTxFrame	Pointer to CAN message frame to be sent.

Return values

kStatus_Success	- Write Tx Message Buffer Successfully.
kStatus_Fail	- Tx Message Buffer is currently in use.

21.2.7.29 status_t FLEXCAN_TransferReceiveBlocking (CAN_Type ∗ *base*, uint8_t *mbIdx*, flexcan_frame_t ∗ *pRxFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	FlexCAN peripheral base pointer.
mbIdx	The FlexCAN Message Buffer index.
pRxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Rx Message Buffer is full and has been read successfully.
kStatus_FLEXCAN_Rx-Overflow	- Rx Message Buffer is already overflowed and has been read successfully.
kStatus_Fail	- Rx Message Buffer is empty.

21.2.7.30 status_t FLEXCAN_TransferReceiveFifoBlocking (CAN_Type * *base*, flexcan_frame_t * *pRxFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	FlexCAN peripheral base pointer.
pRxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Read Message from Rx FIFO successfully.
kStatus_Fail	- Rx FIFO is not enabled.

21.2.7.31 void FLEXCAN_TransferCreateHandle (CAN_Type * *base*, flexcan_handle_t * *handle*, flexcan_transfer_callback_t *callback*, void * *userData*)

This function initializes the FlexCAN handle, which can be used for other FlexCAN transactional APIs. Usually, for a specified FlexCAN instance, call this API once to get the initialized handle.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
callback	The callback function.
userData	The parameter of the callback function.

21.2.7.32 status_t FLEXCAN_TransferSendNonBlocking (CAN_Type * *base*, flexcan_handle_t * *handle*, flexcan_mb_transfer_t * *pMbXfer*)

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
pMbXfer	FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

kStatus_Success	Start Tx Message Buffer sending process successfully.
kStatus_Fail	Write Tx Message Buffer failed.
kStatus_FLEXCAN_Tx-Busy	Tx Message Buffer is in use.

21.2.7.33 status_t FLEXCAN_TransferReceiveNonBlocking (CAN_Type * *base*, flexcan_handle_t * *handle*, flexcan_mb_transfer_t * *pMbXfer*)

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
pMbXfer	FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

kStatus_Success	- Start Rx Message Buffer receiving process successfully.
kStatus_FLEXCAN_Rx-Busy	- Rx Message Buffer is in use.

21.2.7.34 status_t FLEXCAN_TransferReceiveFifoNonBlocking (CAN_Type ∗ *base*, flexcan_handle_t ∗ *handle*, flexcan_fifo_transfer_t ∗ *pFifoXfer*)

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
pFifoXfer	FlexCAN Rx FIFO transfer structure. See the flexcan_fifo_transfer_t.

Return values

kStatus_Success	- Start Rx FIFO receiving process successfully.
kStatus_FLEXCAN_Rx-FifoBusy	- Rx FIFO is currently in use.

21.2.7.35 uint32_t FLEXCAN_GetTimeStamp (flexcan_handle_t ∗ *handle*, uint8_t *mbIdx*)

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN_TransferSendNonBlocking -FLEXCAN_TransferFDSendNonBlocking -FLEXCAN_TransferReceiveNonBlocking -FLEXCAN_TransferFDReceiveNonBlocking -FLEXCAN_TransferReceiveFifoNonBlocking

Parameters

handle	FlexCAN handle pointer.
mbIdx	The FlexCAN FD Message Buffer index.

Return values

the	index of mailbox 's timestamp stored in the handle.

21.2.7.36 void FLEXCAN_TransferAbortSend (CAN_Type * *base*, flexcan_handle_t * *handle*, uint8_t *mbIdx*)

This function aborts the interrupt driven message send process.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
mbIdx	The FlexCAN Message Buffer index.

21.2.7.37 void FLEXCAN_TransferAbortReceive (CAN_Type * *base*, flexcan_handle_t * *handle*, uint8_t *mbIdx*)

This function aborts the interrupt driven message receive process.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
mbIdx	The FlexCAN Message Buffer index.

21.2.7.38 void FLEXCAN_TransferAbortReceiveFifo (CAN_Type * *base*, flexcan_handle_t * *handle*)

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.

21.2.7.39 void FLEXCAN_TransferHandleIRQ (CAN_Type * *base*, flexcan_handle_t * *handle*)

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.

Chapter 22
FTM: FlexTimer Driver

Overview

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

Function groups

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

22.2.1 Initialization and deinitialization

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

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

22.2.2 PWM Operations

The function FTM_SetupPwm() sets up FTM channels for the PWM output. The function sets up the PWM signal properties for multiple channels. Each channel has its own duty cycle and level-mode specified. However, the same PWM period and PWM mode is applied to all channels requesting the PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100 0=inactive signal (0% duty cycle) and 100=always active signal (100% duty cycle).

The function FTM_UpdatePwmDutycycle() updates the PWM signal duty cycle of a particular FTM channel.

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

22.2.3 Input capture operations

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

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

22.2.4 Output compare operations

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

22.2.5 Quad decode

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

22.2.6 Fault operation

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

Register Update

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

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

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

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

Typical use case

22.4.1 PWM output

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

Data Structures

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

Enumerations

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

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

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

kFTM_Chnl0Match = (1U << 0),
kFTM_Chnl1Match = (1U << 1),
kFTM_Chnl2Match = (1U << 2),
kFTM_Chnl3Match = (1U << 3),
kFTM_Chnl4Match = (1U << 4),
kFTM_Chnl5Match = (1U << 5),
kFTM_Chnl6Match = (1U << 6),
kFTM_Chnl7Match = (1U << 7),
kFTM_CntMax = (1U << 8),
kFTM_CntMin = (1U << 9),
kFTM_HalfCycMatch = (1U << 10) }

FTM options available as loading point for register reload.

- enum ftm_interrupt_enable_t {
kFTM_Chnl0InterruptEnable = (1U << 0),
kFTM_Chnl1InterruptEnable = (1U << 1),
kFTM_Chnl2InterruptEnable = (1U << 2),
kFTM_Chnl3InterruptEnable = (1U << 3),
kFTM_Chnl4InterruptEnable = (1U << 4),
kFTM_Chnl5InterruptEnable = (1U << 5),
kFTM_Chnl6InterruptEnable = (1U << 6),
kFTM_Chnl7InterruptEnable = (1U << 7),
kFTM_FaultInterruptEnable = (1U << 8),
kFTM_TimeOverflowInterruptEnable = (1U << 9),
kFTM_ReloadInterruptEnable = (1U << 10) }

  List of FTM interrupts.
- enum ftm_status_flags_t {
kFTM_Chnl0Flag = (1U << 0),
kFTM_Chnl1Flag = (1U << 1),
kFTM_Chnl2Flag = (1U << 2),
kFTM_Chnl3Flag = (1U << 3),
kFTM_Chnl4Flag = (1U << 4),
kFTM_Chnl5Flag = (1U << 5),
kFTM_Chnl6Flag = (1U << 6),
kFTM_Chnl7Flag = (1U << 7),
kFTM_FaultFlag = (1U << 8),
kFTM_TimeOverflowFlag = (1U << 9),
kFTM_ChnlTriggerFlag = (1U << 10),
kFTM_ReloadFlag = (1U << 11) }

  List of FTM flags.
- enum {
kFTM_QuadDecoderCountingIncreaseFlag = FTM_QDCTRL_QUADIR_MASK,
kFTM_QuadDecoderCountingOverflowOnTopFlag = FTM_QDCTRL_TOFDIR_MASK }

  List of FTM Quad Decoder flags.

Functions

- void FTM_SetupFaultInput (FTM_Type ∗base, ftm_fault_input_t faultNumber, const ftm_fault_param_t ∗faultParams)
  Sets up the working of the FTM fault inputs protection.
- static void FTM_SetGlobalTimeBaseOutputEnable (FTM_Type ∗base, bool enable)
  Enables or disables the FTM global time base signal generation to other FTMs.
- static void FTM_SetOutputMask (FTM_Type ∗base, ftm_chnl_t chnlNumber, bool mask)
  Sets the FTM peripheral timer channel output mask.
- static void FTM_SetSoftwareTrigger (FTM_Type ∗base, bool enable)
  Enables or disables the FTM software trigger for PWM synchronization.
- static void FTM_SetWriteProtection (FTM_Type ∗base, bool enable)
  Enables or disables the FTM write protection.
- static void FTM_EnableDmaTransfer (FTM_Type ∗base, ftm_chnl_t chnlNumber, bool enable)
  Enable DMA transfer or not.

Driver version

- #define FSL_FTM_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  FTM driver version 2.3.0.

Initialization and deinitialization

- status_t FTM_Init (FTM_Type ∗base, const ftm_config_t ∗config)
  Ungates the FTM clock and configures the peripheral for basic operation.
- void FTM_Deinit (FTM_Type ∗base)
  Gates the FTM clock.
- void FTM_GetDefaultConfig (ftm_config_t ∗config)
  Fills in the FTM configuration structure with the default settings.

Channel mode operations

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

Configures the dual edge capture mode of the FTM.

Interrupt Interface

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

Status Interface

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

Read and write the timer period

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

Timer Start and Stop

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

Software output control

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

Channel pair operations

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

Quad Decoder

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

Data Structure Documentation

22.5.1 struct ftm_chnl_pwm_signal_param_t

Data Fields

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

22.5.1.0.0.9 Field Documentation

22.5.1.0.0.9.1 ftm_chnl_t ftm_chnl_pwm_signal_param_t::chnlNumber

In combined mode, this represents the channel pair number.

22.5.1.0.0.9.2 ftm_pwm_level_select_t ftm_chnl_pwm_signal_param_t::level

22.5.1.0.0.9.3 uint8_t ftm_chnl_pwm_signal_param_t::dutyCyclePercent

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

22.5.1.0.0.9.4 uint8_t ftm_chnl_pwm_signal_param_t::firstEdgeDelayPercent

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

22.5.1.0.0.9.5 bool ftm_chnl_pwm_signal_param_t::enableComplementary

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

22.5.1.0.0.9.6 bool ftm_chnl_pwm_signal_param_t::enableDeadtime

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

22.5.2 struct ftm_chnl_pwm_config_param_t

Data Fields

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

22.5.2.0.0.10 Field Documentation

22.5.2.0.0.10.1 ftm_chnl_t ftm_chnl_pwm_config_param_t::chnlNumber

In combined mode, this represents the channel pair number.

22.5.2.0.0.10.2 ftm_pwm_level_select_t ftm_chnl_pwm_config_param_t::level

22.5.2.0.0.10.3 uint16_t ftm_chnl_pwm_config_param_t::dutyValue

22.5.2.0.0.10.4 uint16_t ftm_chnl_pwm_config_param_t::firstEdgeValue

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

22.5.2.0.0.10.5 bool ftm_chnl_pwm_config_param_t::enableComplementary

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

22.5.2.0.0.10.6 bool ftm_chnl_pwm_config_param_t::enableDeadtime

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

22.5.3 struct ftm_dual_edge_capture_param_t

Data Fields

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

22.5.4 struct ftm_phase_params_t

Data Fields

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

22.5.5 struct ftm_fault_param_t

Data Fields

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

22.5.6 struct ftm_config_t

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

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

Data Fields

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

22.5.6.0.0.11 Field Documentation

22.5.6.0.0.11.1 uint32_t ftm_config_t::pwmSyncMode

22.5.6.0.0.11.2 uint32_t ftm_config_t::reloadPoints

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

22.5.6.0.0.11.3 uint32_t ftm_config_t::deadTimeValue

22.5.6.0.0.11.4 uint32_t ftm_config_t::extTriggers

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

Macro Definition Documentation

22.6.1 #define FSL_FTM_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

Enumeration Type Documentation

22.7.1 enum ftm_chnl_t

Note

Actual number of available channels is SoC dependent

Enumerator

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

22.7.2 enum ftm_fault_input_t

Enumerator

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

22.7.3 enum ftm_pwm_mode_t

Enumerator

kFTM_EdgeAlignedPwm Edge-aligned PWM.
kFTM_CenterAlignedPwm Center-aligned PWM.
kFTM_EdgeAlignedCombinedPwm Edge-aligned combined PWM.
kFTM_CenterAlignedCombinedPwm Center-aligned combined PWM.
kFTM_AsymmetricalCombinedPwm Asymmetrical combined PWM.

22.7.4 enum ftm_pwm_level_select_t

Enumerator

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

22.7.5 enum ftm_output_compare_mode_t

Enumerator

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

22.7.6 enum ftm_input_capture_edge_t

Enumerator

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

22.7.7 enum ftm_dual_edge_capture_mode_t

Enumerator

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

22.7.8 enum ftm_quad_decode_mode_t

Enumerator

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

22.7.9 enum ftm_phase_polarity_t

Enumerator

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

22.7.10 enum ftm_deadtime_prescale_t

Enumerator

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

22.7.11 enum ftm_clock_source_t

Enumerator

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

22.7.12 enum ftm_clock_prescale_t

Enumerator

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

22.7.13 enum ftm_bdm_mode_t

Enumerator

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

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

22.7.14 enum ftm_fault_mode_t

Enumerator

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

22.7.15 enum ftm_external_trigger_t

Note

Actual available external trigger sources are SoC-specific

Enumerator

kFTM_Chnl0Trigger Generate trigger when counter equals chnl 0 CnV reg.
kFTM_Chnl1Trigger Generate trigger when counter equals chnl 1 CnV reg.
kFTM_Chnl2Trigger Generate trigger when counter equals chnl 2 CnV reg.
kFTM_Chnl3Trigger Generate trigger when counter equals chnl 3 CnV reg.
kFTM_Chnl4Trigger Generate trigger when counter equals chnl 4 CnV reg.
kFTM_Chnl5Trigger Generate trigger when counter equals chnl 5 CnV reg.
kFTM_InitTrigger Generate Trigger when counter is updated with CNTIN.

22.7.16 enum ftm_pwm_sync_method_t

Enumerator

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

22.7.17 enum ftm_reload_point_t

Note

Actual available reload points are SoC-specific

Enumerator

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

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

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

22.7.18 enum ftm_interrupt_enable_t

Note

Actual available interrupts are SoC-specific

Enumerator

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

22.7.19 enum ftm_status_flags_t

Note

Actual available flags are SoC-specific

Enumerator

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

22.7.20 anonymous enum

Enumerator

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

Function Documentation

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

Note

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

Parameters

base	FTM peripheral base address

config	Pointer to the user configuration structure.

Returns

kStatus_Success indicates success; Else indicates failure.

22.8.2 void FTM_Deinit (FTM_Type ∗ *base*)

Parameters

base	FTM peripheral base address

22.8.3 void FTM_GetDefaultConfig (ftm_config_t ∗ *config*)

The default values are:

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

Parameters

config	Pointer to the user configuration structure.

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

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

Parameters

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

Returns

kStatus_Success if the PWM setup was successful kStatus_Error on failure

22.8.5 void FTM_UpdatePwmDutycycle (FTM_Type ∗ *base*, ftm_chnl_t *chnlNumber*, ftm_pwm_mode_t *currentPwmMode*, uint8_t *dutyCyclePercent*)

Parameters

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

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

Parameters

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

22.8.7 status_t FTM_SetupPwmMode (FTM_Type * *base*, const ftm_chnl_pwm_config_param_t * *chnlParams*, uint8_t *numOfChnls*, ftm_pwm_mode_t *mode*)

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

Parameters

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

Returns

kStatus_Success if the PWM setup was successful kStatus_Error on failure

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

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

Parameters

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

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

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

Parameters

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

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

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

Parameters

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

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

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

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

Parameters

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

22.8.12 void FTM_EnableInterrupts (FTM_Type * *base*, uint32_t *mask*)

Parameters

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

22.8.13 void FTM_DisableInterrupts (FTM_Type * *base*, uint32_t *mask*)

Parameters

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

22.8.14 uint32_t FTM_GetEnabledInterrupts (FTM_Type * *base*)

Parameters

base	FTM peripheral base address

Returns

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

22.8.15 uint32_t FTM_GetStatusFlags (FTM_Type * *base*)

Parameters

base	FTM peripheral base address

Returns

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

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

Parameters

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

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

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

Note

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

Parameters

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

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

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

Note

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

Parameters

base	FTM peripheral base address

Returns

The current counter value in ticks

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

Parameters

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

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

Parameters

base	FTM peripheral base address

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

base	FTM peripheral base address

chnlPair-Number	The FTM channel pair number; options are 0, 1, 2, 3
value	true: enable complementary mode; false: disable complementary mode

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

Parameters

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

22.8.29 void FTM_SetupQuadDecode (FTM_Type * *base*, const ftm_phase_params_t * *phaseAParams*, const ftm_phase_params_t * *phaseBParams*, ftm_quad_decode_mode_t *quadMode*)

Parameters

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

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

Parameters

base	FTM peripheral base address.

Returns

Flag mask of FTM Quad Decoder, see _ftm_quad_decoder_flags.

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

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

Parameters

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

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

Parameters

base	FTM peripheral base address.

Returns

Current quad Decoder counter value.

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

The counter is set as the initial value.

Parameters

base	FTM peripheral base address.

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

Parameters

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

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

Parameters

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

22.8.36 static void FTM_EnableDmaTransfer (FTM_Type * *base*, ftm_chnl_t *chnlNumber*, bool *enable*) [inline], [static]

Note: CHnIE bit needs to be set when calling this API. The channel DMA transfer request is generated and the channel interrupt is not generated if (CHnF = 1) when DMA and CHnIE bits are set.

Parameters

base	FTM peripheral base address.
chnlNumber	Channel to be configured
enable	true to enable, false to disable

Chapter 23
GPIO: General-Purpose Input/Output Driver

Overview

Modules

- FGPIO Driver
- GPIO Driver

Data Structures

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

Enumerations

- enum gpio_pin_direction_t {
  kGPIO_DigitalInput = 0U,
  kGPIO_DigitalOutput = 1U }
  GPIO direction definition.

Driver version

- #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))
  GPIO driver version 2.5.1.

Data Structure Documentation

23.2.1 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.

Macro Definition Documentation

23.3.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))

Enumeration Type Documentation

23.4.1 enum gpio_pin_direction_t

Enumerator

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

GPIO Driver

23.5.1 Overview

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

23.5.2 Typical use case

23.5.2.1 Output Operation

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

23.5.2.2 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.

23.5.3 Function Documentation

23.5.3.1 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

23.5.3.2 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.

23.5.3.3 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

23.5.3.4 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

23.5.3.5 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

23.5.3.6 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.

23.5.3.7 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.

23.5.3.8 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

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.

23.6.1 Typical use case

23.6.1.1 Output Operation

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

23.6.1.2 Input Operation

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

Chapter 24
I2C: Inter-Integrated Circuit Driver

Overview

Modules

- I2C Driver

I2C Driver

24.2.1 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.

24.2.2 Typical use case

24.2.2.1 Master Operation in functional method

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

24.2.2.2 Master Operation in interrupt transactional method

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

24.2.2.3 Master Operation in DMA transactional method

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

24.2.2.4 Slave Operation in functional method

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

24.2.2.5 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.

24.2.3 Data Structure Documentation

24.2.3.1 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.

24.2.3.1.0.12 Field Documentation

24.2.3.1.0.12.1 bool i2c_master_config_t::enableMaster

24.2.3.1.0.12.2 bool i2c_master_config_t::enableStopHold

24.2.3.1.0.12.3 uint32_t i2c_master_config_t::baudRate_Bps

24.2.3.1.0.12.4 uint8_t i2c_master_config_t::glitchFilterWidth

24.2.3.2 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.

24.2.3.2.0.13 Field Documentation

24.2.3.2.0.13.1 bool i2c_slave_config_t::enableSlave

24.2.3.2.0.13.2 bool i2c_slave_config_t::enableGeneralCall

24.2.3.2.0.13.3 bool i2c_slave_config_t::enableWakeUp

24.2.3.2.0.13.4 bool i2c_slave_config_t::enableBaudRateCtl

24.2.3.2.0.13.5 uint16_t i2c_slave_config_t::slaveAddress

24.2.3.2.0.13.6 uint16_t i2c_slave_config_t::upperAddress

24.2.3.2.0.13.7 i2c_slave_address_mode_t i2c_slave_config_t::addressingMode

24.2.3.2.0.13.8 uint32_t i2c_slave_config_t::sclStopHoldTime_ns

24.2.3.3 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.

24.2.3.3.0.14 Field Documentation

24.2.3.3.0.14.1 uint32_t i2c_master_transfer_t::flags

24.2.3.3.0.14.2 uint8_t i2c_master_transfer_t::slaveAddress

24.2.3.3.0.14.3 i2c_direction_t i2c_master_transfer_t::direction

24.2.3.3.0.14.4 uint32_t i2c_master_transfer_t::subaddress

Transferred MSB first.

24.2.3.3.0.14.5 uint8_t i2c_master_transfer_t::subaddressSize

24.2.3.3.0.14.6 uint8_t* volatile i2c_master_transfer_t::data

24.2.3.3.0.14.7 volatile size_t i2c_master_transfer_t::dataSize

24.2.3.4 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.

24.2.3.4.0.15 Field Documentation

24.2.3.4.0.15.1 i2c_master_transfer_t i2c_master_handle_t::transfer

24.2.3.4.0.15.2 size_t i2c_master_handle_t::transferSize

24.2.3.4.0.15.3 uint8_t i2c_master_handle_t::state

24.2.3.4.0.15.4 i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback

24.2.3.4.0.15.5 void∗ i2c_master_handle_t::userData

24.2.3.5 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.

24.2.3.5.0.16 Field Documentation

24.2.3.5.0.16.1 i2c_slave_transfer_event_t i2c_slave_transfer_t::event

24.2.3.5.0.16.2 uint8_t∗ volatile i2c_slave_transfer_t::data

24.2.3.5.0.16.3 volatile size_t i2c_slave_transfer_t::dataSize

24.2.3.5.0.16.4 status_t i2c_slave_transfer_t::completionStatus

Only applies for kI2C_SlaveCompletionEvent.

24.2.3.5.0.16.5 size_t i2c_slave_transfer_t::transferredCount

24.2.3.6 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.

24.2.3.6.0.17 Field Documentation

24.2.3.6.0.17.1 volatile bool i2c_slave_handle_t::isBusy

24.2.3.6.0.17.2 i2c_slave_transfer_t i2c_slave_handle_t::transfer

24.2.3.6.0.17.3 uint32_t i2c_slave_handle_t::eventMask

24.2.3.6.0.17.4 i2c_slave_transfer_callback_t i2c_slave_handle_t::callback

24.2.3.6.0.17.5 void* i2c_slave_handle_t::userData

24.2.4 Macro Definition Documentation

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

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

24.2.5 Typedef Documentation

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

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

24.2.6 Enumeration Type Documentation

24.2.6.1 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.

24.2.6.2 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.

24.2.6.3 enum _i2c_interrupt_enable

Enumerator

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

24.2.6.4 enum i2c_direction_t

Enumerator

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

24.2.6.5 enum i2c_slave_address_mode_t

Enumerator

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

24.2.6.6 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.

24.2.6.7 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.

24.2.6.8 anonymous enum

Enumerator

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

24.2.7 Function Documentation

24.2.7.1 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

24.2.7.2 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

24.2.7.3 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

24.2.7.4 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

24.2.7.5 uint32_t I2C_GetInstance (I2C_Type * *base*)

Parameters

base	I2C peripheral base address.

24.2.7.6 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.

24.2.7.7 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.

24.2.7.8 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.

24.2.7.9 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.

24.2.7.10 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.

24.2.7.11 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

24.2.7.12 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

24.2.7.13 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

24.2.7.14 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

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

Parameters

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

24.2.7.16 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

24.2.7.17 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

24.2.7.18 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.

24.2.7.19 status_t I2C_MasterStop (I2C_Type ∗ *base*)

Return values

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

24.2.7.20 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.

24.2.7.21 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.

24.2.7.22 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.

24.2.7.23 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.

24.2.7.24 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.

24.2.7.25 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.

24.2.7.26 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.

24.2.7.27 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.

24.2.7.28 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.

24.2.7.29 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.

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

Parameters

base	I2C base pointer.
i2cHandle	pointer to i2c_master_handle_t structure.

24.2.7.31 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.

24.2.7.32 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 k-I2C_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.

24.2.7.33 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.

24.2.7.34 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.

24.2.7.35 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

Chapter 25
LLWU: Low-Leakage Wakeup Unit Driver

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.

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.

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.

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** MF5

Data Structure Documentation

25.5.1 struct llwu_external_pin_filter_mode_t

Data Fields

- uint32_t pinIndex
  A pin number.
- llwu_pin_filter_mode_t filterMode
  Filter mode.

Macro Definition Documentation

25.6.1 #define FSL_LLWU_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

Enumeration Type Documentation

25.7.1 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.

25.7.2 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.

Function Documentation

25.8.1 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.

25.8.2 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.

25.8.3 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.

25.8.4 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

25.8.5 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.

25.8.6 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

25.8.7 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.

25.8.8 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 26
LMEM: Local Memory Controller Cache Control Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the Local Memory Controller Cache Controller module of MCUXpresso SDK devices.

Descriptions

The LMEM Cache peripheral driver allows the user to enable/disable the cache and to perform cache maintenance operations such as invalidate, push, and clear. These maintenance operations may be performed on the Processor Code (PC) bus or Both Processor Code (PC) and Processor System (PS) bus.

The devices contain a Processor Code (PC) bus and a Processor System (PS) bus as follows. The Processor Code (PC) bus - a 32-bit address space bus with low-order addresses (0x0000_0000 through 0x1FFF_FFFF) used normally for code access. The Processor System (PS) bus - a 32-bit address space bus with high-order addresses (0x2000_0000 through 0xFFFF_FFFF) used normally for data accesses.

Some MCU devices have caches available for the PC bus and PS bus, others may only have a PC bus cache, while some do not have PC or PS caches at all. See the appropriate reference manual for cache availability.

Cache maintenance operations:

command			description	
		Invalidate	U	
cline1-2		**Push**	P	ush a cache entry if it is valid and modified, then clear the m
cline1-2		**Clear**	P	ush a cache entry if it is valic

The above cache maintenance operations may be performed on the entire cache or on a line-basis. The peripheral driver API names distinguish between the two using the terms "All" or Line".

Function groups

26.3.1 Local Memory Processor Code Bus Cache Control

The invalidate command can be performed on the entire cache, one line, or multiple lines by calling LMEM_CodeCacheInvalidateAll(), LMEM_CodeCacheInvalidateLine(), and LMEM_CodeCacheInvalidateMultiLines().

The push command can be performed on the entire cache, one line, or multiple lines by calling LMEM_-CodeCachePushAll(), LMEM_CodeCachePushLine(), and LMEM_CodeCachePushMultiLines().

The clear command can be performed on the entire cache, one line, or multiple lines by calling LMEM_-CodeCacheClearAll(), LMEM_CodeCacheClearLine(), and LMEM_CodeCacheClearMultiLines().

Note that the parameter "address" must be supplied, which indicates the physical address of the line to perform the one line cache maintenance operation. In addition, the length of the number of bytes should be supplied for multiple line operation. The function determines if the length meets or exceeds 1/2 the cache size because the cache contains 2 WAYs, half of the cache is in WAY0 and the other half in W-AY1 and if so, performs a cache maintenance "all" operation which is faster than performing the cache maintenance on a line-basis.

Cache Demotion: Cache region demotion - Demoting the cache mode reduces the cache function applied to a memory region from write-back to write-through to non-cacheable. The cache region demote function checks to see if the requested cache mode is higher than or equal to the current cache mode, and if so, returns an error. After a region is demoted, its cache mode can only be raised by a reset, which returns it to its default state. To demote a cache region, call the LMEM_CodeCacheDemoteRegion().

Note that the address region assignment of the 16 subregions is device-specific and is detailed in the Chip Configuration section of the SoC reference manual. The LMEM provides typedef enums for each of the 16 regions, starting with "kLMEM_CacheRegion0" and ending with "kLMEM_CacheRegion15". The parameter cacheMode is of type lmem_cache_mode_t. This provides typedef enums for each of the cache modes, such as "kLMEM_CacheNonCacheable", "kLMEM_CacheWriteThrough", and "kLMEM-_CacheWriteBack".

Cache Enable and Disable: The cache enable function enables the PC bus cache and the write buffer. However, before enabling these, the function first performs an invalidate all. Call LMEM_EnableCode-Cache() to enable a particular bus cache.

26.3.2 Local Memory Processor System Bus Cache Control

The invalidate command can be performed on the entire cache, one line, or multiple lines by calling LMEM_SystemCacheInvalidateAll(), LMEM_SystemCacheInvalidateLine(), and LMEM_SystemCache-InvalidateMultiLines().

The push command can be performed on the entire cache, one line, or multiple lines by calling LMEM_-SystemCachePushAll(), LMEM_SystemCachePushLine(), and LMEM_SystemCachePushMultiLines().

The clear command can be performed on the entire cache, one line, or multiple lines by calling LM-EM_SystemCacheClearAll(), LMEM_SystemCacheClearLine(), and LMEM_SystemCacheClearMulti-Lines().

Note that the parameter "address" must be supplied, which indicates the physical address of the line to perform the one line cache maintenance operation. In addition, the length of the number of bytes should be supplied for multiple lines operation. The function determines if the length meets or exceeds 1/2 the cache size because the cache contains 2 WAYs, half of the cache is in WAY0 and the other half in W-AY1 and if so, performs a cache maintenance "all" operation which is faster than performing the cache maintenance on a line-basis.

Cache Demotion: Cache region demotion - Demoting the cache mode reduces the cache function applied to a memory region from write-back to write-through to non-cacheable. The cache region demote function checks to see if the requested cache mode is higher than or equal to the current cache mode, and if so, returns an error. After a region is demoted, its cache mode can only be raised by a reset, which returns it to its default state. To demote a cache region, call the LMEM_SystemCacheDemoteRegion().

Note that the address region assignment of the 16 subregions is device-specific and is described in the Chip Configuration section of the SoC reference manual. The LMEM provides typedef enumerations for each of the 16 regions, starting with "kLMEM_CacheRegion0" and ending with "kLMEM_CacheRegion15". The parameter cacheMode is of type lmem_cache_mode_t. This provides typedef enumerations for each of the cache modes, such as "kLMEM_CacheNonCacheable", "kLMEM_CacheWriteThrough", and "kLMEM_CacheWriteBack".

Cache Enable and Disable: The cache enable function enables the PS bus cache and the write buffer. However, before enabling these, the function first performs an invalidate all. Call LMEM_EnableSystemCache() to enable a particular bus cache.

Macros

- #define LMEM_CACHE_LINE_SIZE (0x10U)
  Cache line is 16-bytes.
- #define LMEM_CACHE_SIZE_ONEWAY (4096U)
  Cache size is 4K-bytes one way.

Enumerations

- enum lmem_cache_mode_t {
  kLMEM_NonCacheable = 0x0U,
  kLMEM_CacheWriteThrough = 0x2U,
  kLMEM_CacheWriteBack = 0x3U }
  LMEM cache mode options.
- enum lmem_cache_region_t {

kLMEM_CacheRegion15 = 0U,
kLMEM_CacheRegion14,
kLMEM_CacheRegion13,
kLMEM_CacheRegion12,
kLMEM_CacheRegion11,
kLMEM_CacheRegion10,
kLMEM_CacheRegion9,
kLMEM_CacheRegion8,
kLMEM_CacheRegion7,
kLMEM_CacheRegion6,
kLMEM_CacheRegion5,
kLMEM_CacheRegion4,
kLMEM_CacheRegion3,
kLMEM_CacheRegion2,
kLMEM_CacheRegion1,
kLMEM_CacheRegion0 }
LMEM cache regions.

- enum lmem_cache_line_command_t {
kLMEM_CacheLineSearchReadOrWrite = 0U,
kLMEM_CacheLineInvalidate,
kLMEM_CacheLinePush,
kLMEM_CacheLineClear }
LMEM cache line command.

Driver version

- #define FSL_LMEM_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))
LMEM controller driver version 2.1.2.

Local Memory Processor Code Bus Cache Control

- void LMEM_EnableCodeCache (LMEM_Type *base, bool enable)
Enables/disables the processor code bus cache.
- static void LMEM_EnableCodeWriteBuffer (LMEM_Type *base, bool enable)
Enables/disables the processor code bus write buffer.
- void LMEM_CodeCacheInvalidateAll (LMEM_Type *base)
Invalidates the processor code bus cache.
- void LMEM_CodeCachePushAll (LMEM_Type *base)
Pushes all modified lines in the processor code bus cache.
- void LMEM_CodeCacheClearAll (LMEM_Type *base)
Clears the processor code bus cache.
- void LMEM_CodeCacheInvalidateLine (LMEM_Type *base, uint32_t address)
Invalidates a specific line in the processor code bus cache.
- void LMEM_CodeCacheInvalidateMultiLines (LMEM_Type *base, uint32_t address, uint32_t length)
Invalidates multiple lines in the processor code bus cache.
- void LMEM_CodeCachePushLine (LMEM_Type *base, uint32_t address)
Pushes a specific modified line in the processor code bus cache.

- void LMEM_CodeCachePushMultiLines (LMEM_Type ∗base, uint32_t address, uint32_t length)
  Pushes multiple modified lines in the processor code bus cache.
- void LMEM_CodeCacheClearLine (LMEM_Type ∗base, uint32_t address)
  Clears a specific line in the processor code bus cache.
- void LMEM_CodeCacheClearMultiLines (LMEM_Type ∗base, uint32_t address, uint32_t length)
  Clears multiple lines in the processor code bus cache.
- status_t LMEM_CodeCacheDemoteRegion (LMEM_Type ∗base, lmem_cache_region_t region, lmem_cache_mode_t cacheMode)
  Demotes the cache mode of a region in processor code bus cache.

Macro Definition Documentation

26.4.1 #define FSL_LMEM_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

26.4.2 #define LMEM_CACHE_LINE_SIZE (0x10U)

26.4.3 #define LMEM_CACHE_SIZE_ONEWAY (4096U)

Enumeration Type Documentation

26.5.1 enum lmem_cache_mode_t

Enumerator

kLMEM_NonCacheable Cache mode: non-cacheable.
kLMEM_CacheWriteThrough Cache mode: write-through.
kLMEM_CacheWriteBack Cache mode: write-back.

26.5.2 enum lmem_cache_region_t

Enumerator

kLMEM_CacheRegion15 Cache Region 15.
kLMEM_CacheRegion14 Cache Region 14.
kLMEM_CacheRegion13 Cache Region 13.
kLMEM_CacheRegion12 Cache Region 12.
kLMEM_CacheRegion11 Cache Region 11.
kLMEM_CacheRegion10 Cache Region 10.
kLMEM_CacheRegion9 Cache Region 9.
kLMEM_CacheRegion8 Cache Region 8.
kLMEM_CacheRegion7 Cache Region 7.
kLMEM_CacheRegion6 Cache Region 6.
kLMEM_CacheRegion5 Cache Region 5.
kLMEM_CacheRegion4 Cache Region 4.
kLMEM_CacheRegion3 Cache Region 3.

kLMEM_CacheRegion2 Cache Region 2.
kLMEM_CacheRegion1 Cache Region 1.
kLMEM_CacheRegion0 Cache Region 0.

26.5.3 enum lmem_cache_line_command_t

Enumerator

kLMEM_CacheLineSearchReadOrWrite Cache line search and read or write.
kLMEM_CacheLineInvalidate Cache line invalidate.
kLMEM_CacheLinePush Cache line push.
kLMEM_CacheLineClear Cache line clear.

Function Documentation

26.6.1 void LMEM_EnableCodeCache (LMEM_Type * *base*, bool *enable*)

This function enables/disables the cache. The function first invalidates the entire cache and then enables/disables both the cache and write buffers.

Parameters

base	LMEM peripheral base address.
enable	The enable or disable flag. true - enable the code cache. false - disable the code cache.

26.6.2 static void LMEM_EnableCodeWriteBuffer (LMEM_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	LMEM peripheral base address.
enable	The enable or disable flag. true - enable the code bus write buffer. false - disable the code bus write buffer.

26.6.3 void LMEM_CodeCacheInvalidateAll (LMEM_Type * *base*)

This function invalidates the cache both ways, which means that it unconditionally clears valid bits and modifies bits of a cache entry.

Parameters

base	LMEM peripheral base address.

26.6.4 void LMEM_CodeCachePushAll (LMEM_Type * *base*)

This function pushes all modified lines in both ways in the entire cache. It pushes a cache entry if it is valid and modified and clears the modified bit. If the entry is not valid or not modified, leave as is. This action does not clear the valid bit. A cache push is synonymous with a cache flush.

Parameters

base	LMEM peripheral base address.

26.6.5 void LMEM_CodeCacheClearAll (LMEM_Type * *base*)

This function clears the entire cache and pushes (flushes) and invalidates the operation. Clear - Pushes a cache entry if it is valid and modified, then clears the valid and modified bits. If the entry is not valid or not modified, clear the valid bit.

Parameters

base	LMEM peripheral base address.

26.6.6 void LMEM_CodeCacheInvalidateLine (LMEM_Type * *base,* uint32_t *address*)

This function invalidates a specific line in the cache based on the physical address passed in by the user. Invalidate - Unconditionally clears valid and modified bits of a cache entry.

Parameters

base	LMEM peripheral base address.
address	The physical address of the cache line. Should be 16-byte aligned address. If not, it is changed to the 16-byte aligned memory address.

26.6.7 void LMEM_CodeCacheInvalidateMultiLines (LMEM_Type * *base*, uint32_t *address*, uint32_t *length*)

This function invalidates multiple lines in the cache based on the physical address and length in bytes passed in by the user. If the function detects that the length meets or exceeds half the cache, the function performs an entire cache invalidate function, which is more efficient than invalidating the cache line-by-line. Because the cache consists of two ways and line commands based on the physical address searches both ways, check half the total amount of cache. Invalidate - Unconditionally clear valid and modified bits of a cache entry.

Parameters

base	LMEM peripheral base address.
address	The physical address of the cache line. Should be 16-byte aligned address. If not, it is changed to the 16-byte aligned memory address.
length	The length in bytes of the total amount of cache lines.

26.6.8 void LMEM_CodeCachePushLine (LMEM_Type * *base*, uint32_t *address*)

This function pushes a specific modified line based on the physical address passed in by the user. Push - Push a cache entry if it is valid and modified, then clear the modified bit. If the entry is not valid or not modified, leave as is. This action does not clear the valid bit. A cache push is synonymous with a cache flush.

Parameters

base	LMEM peripheral base address.
address	The physical address of the cache line. Should be 16-byte aligned address. If not, it is changed to the 16-byte aligned memory address.

26.6.9 void LMEM_CodeCachePushMultiLines (LMEM_Type * *base*, uint32_t *address*, uint32_t *length*)

This function pushes multiple modified lines in the cache based on the physical address and length in bytes passed in by the user. If the function detects that the length meets or exceeds half of the cache, the function performs an cache push function, which is more efficient than pushing the modified lines in the cache line-by-line. Because the cache consists of two ways and line commands based on the physical address searches both ways, check half the total amount of cache. Push - Push a cache entry if it is valid and modified, then clear the modified bit. If the entry is not valid or not modified, leave as is. This action does not clear the valid bit. A cache push is synonymous with a cache flush.

Parameters

base	LMEM peripheral base address.
address	The physical address of the cache line. Should be 16-byte aligned address. If not, it is changed to the 16-byte aligned memory address.
length	The length in bytes of the total amount of cache lines.

26.6.10 void LMEM_CodeCacheClearLine (LMEM_Type ∗ *base,* uint32_t *address*)

This function clears a specific line based on the physical address passed in by the user. Clear - Push a cache entry if it is valid and modified, then clear the valid and modify bits. If entry not valid or not modified, clear the valid bit.

Parameters

base	LMEM peripheral base address.
address	The physical address of the cache line. Should be 16-byte aligned address. If not, it is changed to the 16-byte aligned memory address.

26.6.11 void LMEM_CodeCacheClearMultiLines (LMEM_Type ∗ *base,* uint32_t *address,* uint32_t *length*)

This function clears multiple lines in the cache based on the physical address and length in bytes passed in by the user. If the function detects that the length meets or exceeds half the total amount of cache, the function performs a cache clear function which is more efficient than clearing the lines in the cache line-by-line. Because the cache consists of two ways and line commands based on the physical address searches both ways, check half the total amount of cache. Clear - Push a cache entry if it is valid and modified, then clear the valid and modify bits. If entry not valid or not modified, clear the valid bit.

Parameters

base	LMEM peripheral base address.
address	The physical address of the cache line. Should be 16-byte aligned address. If not, it is changed to the 16-byte aligned memory address.

length	The length in bytes of the total amount of cache lines.

26.6.12 status_t LMEM_CodeCacheDemoteRegion (LMEM_Type ∗ *base*, lmem_cache_region_t *region*, lmem_cache_mode_t *cacheMode*)

This function allows the user to demote the cache mode of a region within the device's memory map. Demoting the cache mode reduces the cache function applied to a memory region from write-back to write-through to non-cacheable. The function checks to see if the requested cache mode is higher than or equal to the current cache mode, and if so, returns an error. After a region is demoted, its cache mode can only be raised by a reset, which returns it to its default state which is the highest cache configure for each region. To maintain cache coherency, changes to the cache mode should be completed while the address space being changed is not being accessed or the cache is disabled. Before a cache mode change, this function completes a cache clear all command to push and invalidate any cache entries that may have changed.

Parameters

base	LMEM peripheral base address.
region	The desired region to demote of type lmem_cache_region_t.
cacheMode	The new, demoted cache mode of type lmem_cache_mode_t.

Returns

The execution result. kStatus_Success The cache demote operation is successful. kStatus_Fail The cache demote operation is failure.

Chapter 27
LPTMR: Low-Power Timer

Overview

The MCUXpresso SDK provides a driver for the Low-Power Timer (LPTMR) of MCUXpresso SDK devices.

Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

27.2.1 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.

27.2.2 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.

27.2.3 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.

27.2.4 Status

Provides functions to get and clear the LPTMR status.

27.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get the currently enabled interrupts.

Typical use case

27.3.1 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.

Data Structure Documentation

27.4.1 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.

Enumeration Type Documentation

27.5.1 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.

27.5.2 enum lptmr_pin_polarity_t

Enumerator

kLPTMR_PinPolarityActiveHigh Pulse Counter input source is active-high.
kLPTMR_PinPolarityActiveLow Pulse Counter input source is active-low.

27.5.3 enum lptmr_timer_mode_t

Enumerator

kLPTMR_TimerModeTimeCounter Time Counter mode.
kLPTMR_TimerModePulseCounter Pulse Counter mode.

27.5.4 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.

27.5.5 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.

27.5.6 enum lptmr_interrupt_enable_t

Enumerator

kLPTMR_TimerInterruptEnable Timer interrupt enable.

27.5.7 enum lptmr_status_flags_t

Enumerator

kLPTMR_TimerCompareFlag Timer compare flag.

Function Documentation

27.6.1 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.

27.6.2 void LPTMR_Deinit (LPTMR_Type * *base*)

Parameters

base	LPTMR peripheral base address

27.6.3 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.

27.6.4 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

27.6.5 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.

27.6.6 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

27.6.7 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

27.6.8 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.

27.6.9 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.

27.6.10 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

27.6.11 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

27.6.12 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 28
LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

Overview

Modules

- LPUART Driver

LPUART Driver

28.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

28.2.2 Typical use case

28.2.2.1 LPUART Operation

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

Data Structures

- struct lpuart_config_t
  LPUART configuration structure. More...
- struct lpuart_transfer_t
  LPUART transfer structure. More...
- struct lpuart_handle_t
  LPUART handle structure. More...

Macros

- #define UART_RETRY_TIMES 0U /∗ Defining to zero means to keep waiting for the flag until it is assert/deassert. ∗/
  Retry times for waiting flag.

Typedefs

- typedef void(∗ lpuart_transfer_callback_t)(LPUART_Type ∗base, lpuart_handle_t ∗handle, status_t status, void ∗userData)
  LPUART transfer callback function.

Enumerations

- enum {
kStatus_LPUART_TxBusy = MAKE_STATUS(kStatusGroup_LPUART, 0),
kStatus_LPUART_RxBusy = MAKE_STATUS(kStatusGroup_LPUART, 1),
kStatus_LPUART_TxIdle = MAKE_STATUS(kStatusGroup_LPUART, 2),
kStatus_LPUART_RxIdle = MAKE_STATUS(kStatusGroup_LPUART, 3),
kStatus_LPUART_TxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_LPUART, 4),
kStatus_LPUART_RxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_LPUART, 5),
kStatus_LPUART_FlagCannotClearManually = MAKE_STATUS(kStatusGroup_LPUART, 6),
kStatus_LPUART_Error = MAKE_STATUS(kStatusGroup_LPUART, 7),
kStatus_LPUART_RxRingBufferOverrun,
kStatus_LPUART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_LPUART, 9),
kStatus_LPUART_NoiseError = MAKE_STATUS(kStatusGroup_LPUART, 10),
kStatus_LPUART_FramingError = MAKE_STATUS(kStatusGroup_LPUART, 11),
kStatus_LPUART_ParityError = MAKE_STATUS(kStatusGroup_LPUART, 12),
kStatus_LPUART_BaudrateNotSupport,
kStatus_LPUART_IdleLineDetected = MAKE_STATUS(kStatusGroup_LPUART, 14),
kStatus_LPUART_Timeout = MAKE_STATUS(kStatusGroup_LPUART, 15) }
Error codes for the LPUART driver.
- enum lpuart_parity_mode_t {
kLPUART_ParityDisabled = 0x0U,
kLPUART_ParityEven = 0x2U,
kLPUART_ParityOdd = 0x3U }
LPUART parity mode.
- enum lpuart_data_bits_t { kLPUART_EightDataBits = 0x0U }
LPUART data bits count.
- enum lpuart_stop_bit_count_t {
kLPUART_OneStopBit = 0U,
kLPUART_TwoStopBit = 1U }
LPUART stop bit count.
- enum lpuart_transmit_cts_source_t {
kLPUART_CtsSourcePin = 0U,
kLPUART_CtsSourceMatchResult = 1U }
LPUART transmit CTS source.
- enum lpuart_transmit_cts_config_t {
kLPUART_CtsSampleAtStart = 0U,
kLPUART_CtsSampleAtIdle = 1U }
LPUART transmit CTS configure.
- enum lpuart_idle_type_select_t {
kLPUART_IdleTypeStartBit = 0U,
kLPUART_IdleTypeStopBit = 1U }
LPUART idle flag type defines when the receiver starts counting.
- enum lpuart_idle_config_t {

kLPUART_IdleCharacter1 = 0U,
kLPUART_IdleCharacter2 = 1U,
kLPUART_IdleCharacter4 = 2U,
kLPUART_IdleCharacter8 = 3U,
kLPUART_IdleCharacter16 = 4U,
kLPUART_IdleCharacter32 = 5U,
kLPUART_IdleCharacter64 = 6U,
kLPUART_IdleCharacter128 = 7U }
LPUART idle detected configuration.
- enum _lpuart_interrupt_enable {
kLPUART_LinBreakInterruptEnable = (LPUART_BAUD_LBKDIE_MASK >> 8),
kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8),
kLPUART_TxDataRegEmptyInterruptEnable = (LPUART_CTRL_TIE_MASK),
kLPUART_TransmissionCompleteInterruptEnable = (LPUART_CTRL_TCIE_MASK),
kLPUART_RxDataRegFullInterruptEnable = (LPUART_CTRL_RIE_MASK),
kLPUART_IdleLineInterruptEnable = (LPUART_CTRL_ILIE_MASK),
kLPUART_RxOverrunInterruptEnable = (LPUART_CTRL_ORIE_MASK),
kLPUART_NoiseErrorInterruptEnable = (LPUART_CTRL_NEIE_MASK),
kLPUART_FramingErrorInterruptEnable = (LPUART_CTRL_FEIE_MASK),
kLPUART_ParityErrorInterruptEnable = (LPUART_CTRL_PEIE_MASK) }
LPUART interrupt configuration structure, default settings all disabled.
- enum _lpuart_flags {
kLPUART_TxDataRegEmptyFlag,
kLPUART_TransmissionCompleteFlag,
kLPUART_RxDataRegFullFlag,
kLPUART_IdleLineFlag = (LPUART_STAT_IDLE_MASK),
kLPUART_RxOverrunFlag = (LPUART_STAT_OR_MASK),
kLPUART_NoiseErrorFlag = (LPUART_STAT_NF_MASK),
kLPUART_FramingErrorFlag,
kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),
kLPUART_LinBreakFlag = (int)(LPUART_STAT_LBKDIF_MASK),
kLPUART_RxActiveEdgeFlag,
kLPUART_RxActiveFlag,
kLPUART_DataMatch1Flag = LPUART_STAT_MA1F_MASK,
kLPUART_DataMatch2Flag = LPUART_STAT_MA2F_MASK,
kLPUART_NoiseErrorInRxDataRegFlag,
kLPUART_ParityErrorInRxDataRegFlag }
LPUART status flags.

Driver version

- #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))
LPUART driver version.

Initialization and deinitialization

- status_t LPUART_Init (LPUART_Type ∗base, const lpuart_config_t ∗config, uint32_t srcClock_-Hz)
  Initializes an LPUART instance with the user configuration structure and the peripheral clock.
- void LPUART_Deinit (LPUART_Type ∗base)
  Deinitializes a LPUART instance.
- void LPUART_GetDefaultConfig (lpuart_config_t ∗config)
  Gets the default configuration structure.
- status_t LPUART_SetBaudRate (LPUART_Type ∗base, uint32_t baudRate_Bps, uint32_t src-Clock_Hz)
  Sets the LPUART instance baudrate.
- void LPUART_Enable9bitMode (LPUART_Type ∗base, bool enable)
  Enable 9-bit data mode for LPUART.
- static void LPUART_SetMatchAddress (LPUART_Type ∗base, uint16_t address1, uint16_t ad-dress2)
  Set the LPUART address.
- static void LPUART_EnableMatchAddress (LPUART_Type ∗base, bool match1, bool match2)
  Enable the LPUART match address feature.

Status

- uint32_t LPUART_GetStatusFlags (LPUART_Type ∗base)
  Gets LPUART status flags.
- status_t LPUART_ClearStatusFlags (LPUART_Type ∗base, uint32_t mask)
  Clears status flags with a provided mask.

Interrupts

- void LPUART_EnableInterrupts (LPUART_Type ∗base, uint32_t mask)
  Enables LPUART interrupts according to a provided mask.
- void LPUART_DisableInterrupts (LPUART_Type ∗base, uint32_t mask)
  Disables LPUART interrupts according to a provided mask.
- uint32_t LPUART_GetEnabledInterrupts (LPUART_Type ∗base)
  Gets enabled LPUART interrupts.
- static uint32_t LPUART_GetDataRegisterAddress (LPUART_Type ∗base)
  Gets the LPUART data register address.
- static void LPUART_EnableTxDMA (LPUART_Type ∗base, bool enable)
  Enables or disables the LPUART transmitter DMA request.
- static void LPUART_EnableRxDMA (LPUART_Type ∗base, bool enable)
  Enables or disables the LPUART receiver DMA.

Bus Operations

- uint32_t LPUART_GetInstance (LPUART_Type ∗base)
  Get the LPUART instance from peripheral base address.

- static void LPUART_EnableTx (LPUART_Type *base, bool enable)
  Enables or disables the LPUART transmitter.
- static void LPUART_EnableRx (LPUART_Type *base, bool enable)
  Enables or disables the LPUART receiver.
- static void LPUART_WriteByte (LPUART_Type *base, uint8_t data)
  Writes to the transmitter register.
- static uint8_t LPUART_ReadByte (LPUART_Type *base)
  Reads the receiver register.
- void LPUART_SendAddress (LPUART_Type *base, uint8_t address)
  Transmit an address frame in 9-bit data mode.
- status_t LPUART_WriteBlocking (LPUART_Type *base, const uint8_t *data, size_t length)
  Writes to the transmitter register using a blocking method.
- status_t LPUART_ReadBlocking (LPUART_Type *base, uint8_t *data, size_t length)
  Reads the receiver data register using a blocking method.

Transactional

- void LPUART_TransferCreateHandle (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_callback_t callback, void *userData)
  Initializes the LPUART handle.
- status_t LPUART_TransferSendNonBlocking (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer)
  Transmits a buffer of data using the interrupt method.
- void LPUART_TransferStartRingBuffer (LPUART_Type *base, lpuart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)
  Sets up the RX ring buffer.
- void LPUART_TransferStopRingBuffer (LPUART_Type *base, lpuart_handle_t *handle)
  Aborts the background transfer and uninstalls the ring buffer.
- size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type *base, lpuart_handle_t *handle)
  Get the length of received data in RX ring buffer.
- void LPUART_TransferAbortSend (LPUART_Type *base, lpuart_handle_t *handle)
  Aborts the interrupt-driven data transmit.
- status_t LPUART_TransferGetSendCount (LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)
  Gets the number of bytes that have been sent out to bus.
- status_t LPUART_TransferReceiveNonBlocking (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer, size_t *receivedBytes)
  Receives a buffer of data using the interrupt method.
- void LPUART_TransferAbortReceive (LPUART_Type *base, lpuart_handle_t *handle)
  Aborts the interrupt-driven data receiving.
- status_t LPUART_TransferGetReceiveCount (LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)
  Gets the number of bytes that have been received.
- void LPUART_TransferHandleIRQ (LPUART_Type *base, lpuart_handle_t *handle)
  LPUART IRQ handle function.
- void LPUART_TransferHandleErrorIRQ (LPUART_Type *base, lpuart_handle_t *handle)
  LPUART Error IRQ handle function.

28.2.3 Data Structure Documentation

28.2.3.1 struct lpuart_config_t

Data Fields

- uint32_t baudRate_Bps
  LPUART baud rate.
- lpuart_parity_mode_t parityMode
  Parity mode, disabled (default), even, odd.
- lpuart_data_bits_t dataBitsCount
  Data bits count, eight (default), seven.
- bool isMsb
  Data bits order, LSB (default), MSB.
- lpuart_stop_bit_count_t stopBitCount
  Number of stop bits, 1 stop bit (default) or 2 stop bits.
- bool enableRxRTS
  RX RTS enable.
- bool enableTxCTS
  TX CTS enable.
- lpuart_transmit_cts_source_t txCtsSource
  TX CTS source.
- lpuart_transmit_cts_config_t txCtsConfig
  TX CTS configure.
- lpuart_idle_type_select_t rxIdleType
  RX IDLE type.
- lpuart_idle_config_t rxIdleConfig
  RX IDLE configuration.
- bool enableTx
  Enable TX.
- bool enableRx
  Enable RX.

28.2.3.1.0.18 Field Documentation

28.2.3.1.0.18.1 lpuart_idle_type_select_t lpuart_config_t::rxIdleType

28.2.3.1.0.18.2 lpuart_idle_config_t lpuart_config_t::rxIdleConfig

28.2.3.2 struct lpuart_transfer_t

Data Fields

- uint8_t ∗ data
  The buffer of data to be transfer.
- size_t dataSize
  The byte count to be transfer.

28.2.3.2.0.19 Field Documentation

28.2.3.2.0.19.1 uint8_t* lpuart_transfer_t::data

28.2.3.2.0.19.2 size_t lpuart_transfer_t::dataSize

28.2.3.3 struct _lpuart_handle

Data Fields

- uint8_t *volatile txData
  Address of remaining data to send.
- volatile size_t txDataSize
  Size of the remaining data to send.
- size_t txDataSizeAll
  Size of the data to send out.
- uint8_t *volatile rxData
  Address of remaining data to receive.
- volatile size_t rxDataSize
  Size of the remaining data to receive.
- size_t rxDataSizeAll
  Size of the data to receive.
- uint8_t * rxRingBuffer
  Start address of the receiver ring buffer.
- size_t rxRingBufferSize
  Size of the ring buffer.
- volatile uint16_t rxRingBufferHead
  Index for the driver to store received data into ring buffer.
- volatile uint16_t rxRingBufferTail
  Index for the user to get data from the ring buffer.
- lpuart_transfer_callback_t callback
  Callback function.
- void * userData
  LPUART callback function parameter.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.

28.2.3.3.0.20 Field Documentation

28.2.3.3.0.20.1 uint8_t* volatile lpuart_handle_t::txData

28.2.3.3.0.20.2 volatile size_t lpuart_handle_t::txDataSize

28.2.3.3.0.20.3 size_t lpuart_handle_t::txDataSizeAll

28.2.3.3.0.20.4 uint8_t* volatile lpuart_handle_t::rxData

28.2.3.3.0.20.5 volatile size_t lpuart_handle_t::rxDataSize

28.2.3.3.0.20.6 size_t lpuart_handle_t::rxDataSizeAll

28.2.3.3.0.20.7 uint8_t* lpuart_handle_t::rxRingBuffer

28.2.3.3.0.20.8 size_t lpuart_handle_t::rxRingBufferSize

28.2.3.3.0.20.9 volatile uint16_t lpuart_handle_t::rxRingBufferHead

28.2.3.3.0.20.10 volatile uint16_t lpuart_handle_t::rxRingBufferTail

28.2.3.3.0.20.11 lpuart_transfer_callback_t lpuart_handle_t::callback

28.2.3.3.0.20.12 void* lpuart_handle_t::userData

28.2.3.3.0.20.13 volatile uint8_t lpuart_handle_t::txState

28.2.3.3.0.20.14 volatile uint8_t lpuart_handle_t::rxState

28.2.4 Macro Definition Documentation

28.2.4.1 #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))

28.2.4.2 #define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */

28.2.5 Typedef Documentation

28.2.5.1 typedef void(* lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle, status_t status, void *userData)

28.2.6 Enumeration Type Documentation

28.2.6.1 anonymous enum

Enumerator

kStatus_LPUART_TxBusy TX busy.

kStatus_LPUART_RxBusy RX busy.
kStatus_LPUART_TxIdle LPUART transmitter is idle.
kStatus_LPUART_RxIdle LPUART receiver is idle.
kStatus_LPUART_TxWatermarkTooLarge TX FIFO watermark too large.
kStatus_LPUART_RxWatermarkTooLarge RX FIFO watermark too large.
kStatus_LPUART_FlagCannotClearManually Some flag can't manually clear.
kStatus_LPUART_Error Error happens on LPUART.
kStatus_LPUART_RxRingBufferOverrun LPUART RX software ring buffer overrun.
kStatus_LPUART_RxHardwareOverrun LPUART RX receiver overrun.
kStatus_LPUART_NoiseError LPUART noise error.
kStatus_LPUART_FramingError LPUART framing error.
kStatus_LPUART_ParityError LPUART parity error.
kStatus_LPUART_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_LPUART_IdleLineDetected IDLE flag.
kStatus_LPUART_Timeout LPUART times out.

28.2.6.2 enum lpuart_parity_mode_t

Enumerator

kLPUART_ParityDisabled Parity disabled.
kLPUART_ParityEven Parity enabled, type even, bit setting: PE|PT = 10.
kLPUART_ParityOdd Parity enabled, type odd, bit setting: PE|PT = 11.

28.2.6.3 enum lpuart_data_bits_t

Enumerator

kLPUART_EightDataBits Eight data bit.

28.2.6.4 enum lpuart_stop_bit_count_t

Enumerator

kLPUART_OneStopBit One stop bit.
kLPUART_TwoStopBit Two stop bits.

28.2.6.5 enum lpuart_transmit_cts_source_t

Enumerator

kLPUART_CtsSourcePin CTS resource is the LPUART_CTS pin.
kLPUART_CtsSourceMatchResult CTS resource is the match result.

28.2.6.6 enum lpuart_transmit_cts_config_t

Enumerator

kLPUART_CtsSampleAtStart CTS input is sampled at the start of each character.
kLPUART_CtsSampleAtIdle CTS input is sampled when the transmitter is idle.

28.2.6.7 enum lpuart_idle_type_select_t

Enumerator

kLPUART_IdleTypeStartBit Start counting after a valid start bit.
kLPUART_IdleTypeStopBit Start counting after a stop bit.

28.2.6.8 enum lpuart_idle_config_t

This structure defines the number of idle characters that must be received before the IDLE flag is set.

Enumerator

kLPUART_IdleCharacter1 the number of idle characters.
kLPUART_IdleCharacter2 the number of idle characters.
kLPUART_IdleCharacter4 the number of idle characters.
kLPUART_IdleCharacter8 the number of idle characters.
kLPUART_IdleCharacter16 the number of idle characters.
kLPUART_IdleCharacter32 the number of idle characters.
kLPUART_IdleCharacter64 the number of idle characters.
kLPUART_IdleCharacter128 the number of idle characters.

28.2.6.9 enum _lpuart_interrupt_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

kLPUART_LinBreakInterruptEnable LIN break detect.
kLPUART_RxActiveEdgeInterruptEnable Receive Active Edge.
kLPUART_TxDataRegEmptyInterruptEnable Transmit data register empty.
kLPUART_TransmissionCompleteInterruptEnable Transmission complete.
kLPUART_RxDataRegFullInterruptEnable Receiver data register full.
kLPUART_IdleLineInterruptEnable Idle line.
kLPUART_RxOverrunInterruptEnable Receiver Overrun.
kLPUART_NoiseErrorInterruptEnable Noise error flag.
kLPUART_FramingErrorInterruptEnable Framing error flag.
kLPUART_ParityErrorInterruptEnable Parity error flag.

28.2.6.10 enum _lpuart_flags

This provides constants for the LPUART status flags for use in the LPUART functions.

Enumerator

kLPUART_TxDataRegEmptyFlag Transmit data register empty flag, sets when transmit buffer is empty.
kLPUART_TransmissionCompleteFlag Transmission complete flag, sets when transmission activity complete.
kLPUART_RxDataRegFullFlag Receive data register full flag, sets when the receive data buffer is full.
kLPUART_IdleLineFlag Idle line detect flag, sets when idle line detected.
kLPUART_RxOverrunFlag Receive Overrun, sets when new data is received before data is read from receive register.
kLPUART_NoiseErrorFlag Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets
kLPUART_FramingErrorFlag Frame error flag, sets if logic 0 was detected where stop bit expected.
kLPUART_ParityErrorFlag If parity enabled, sets upon parity error detection.
kLPUART_LinBreakFlag LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.
kLPUART_RxActiveEdgeFlag Receive pin active edge interrupt flag, sets when active edge detected.
kLPUART_RxActiveFlag Receiver Active Flag (RAF), sets at beginning of valid start bit.
kLPUART_DataMatch1Flag The next character to be read from LPUART_DATA matches MA1.
kLPUART_DataMatch2Flag The next character to be read from LPUART_DATA matches MA2.
kLPUART_NoiseErrorInRxDataRegFlag NOISY bit, sets if noise detected in current data word.
kLPUART_ParityErrorInRxDataRegFlag PARITY bit, sets if noise detected in current data word.

28.2.7 Function Documentation

28.2.7.1 status_t LPUART_Init (LPUART_Type * *base,* const lpuart_config_t * *config,* uint32_t *srcClock_Hz*)

This function configures the LPUART module with user-defined settings. Call the LPUART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
*  lpuart_config_t lpuartConfig;
*  lpuartConfig.baudRate_Bps = 115200U;
*  lpuartConfig.parityMode = kLPUART_ParityDisabled;
*  lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
*  lpuartConfig.isMsb = false;
*  lpuartConfig.stopBitCount = kLPUART_OneStopBit;
*  lpuartConfig.txFifoWatermark = 0;
*  lpuartConfig.rxFifoWatermark = 1;
```

```
*  LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

Parameters

base	LPUART peripheral base address.
config	Pointer to a user-defined configuration structure.
srcClock_Hz	LPUART clock source frequency in HZ.

Return values

kStatus_LPUART_-BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_Success	LPUART initialize succeed

28.2.7.2 void LPUART_Deinit (LPUART_Type ∗ *base*)

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

base	LPUART peripheral base address.

28.2.7.3 void LPUART_GetDefaultConfig (lpuart_config_t ∗ *config*)

This function initializes the LPUART configuration structure to a default value. The default values are-: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

config	Pointer to a configuration structure.

28.2.7.4 status_t LPUART_SetBaudRate (LPUART_Type ∗ *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
* LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
*
```

Parameters

base	LPUART peripheral base address.
baudRate_Bps	LPUART baudrate to be set.
srcClock_Hz	LPUART clock source frequency in HZ.

Return values

kStatus_LPUART_-BaudrateNotSupport	Baudrate is not supported in the current clock source.
kStatus_Success	Set baudrate succeeded.

28.2.7.5 void LPUART_Enable9bitMode (LPUART_Type * *base,* bool *enable*)

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

base	LPUART peripheral base address.
enable	true to enable, flase to disable.

28.2.7.6 static void LPUART_SetMatchAddress (LPUART_Type * *base,* uint16_t *address1,* uint16_t *address2*) [inline], [static]

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note

> Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

base	LPUART peripheral base address.
address1	LPUART slave address1.
address2	LPUART slave address2.

28.2.7.7 static void LPUART_EnableMatchAddress (LPUART_Type ∗ *base*, bool *match1*, bool *match2*) [inline], [static]

Parameters

base	LPUART peripheral base address.
match1	true to enable match address1, false to disable.
match2	true to enable match address2, false to disable.

28.2.7.8 uint32_t LPUART_GetStatusFlags (LPUART_Type ∗ *base*)

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators _lpuart_flags. To check for a specific status, compare the return value with enumerators in the _lpuart_flags. For example, to check whether the TX is empty:

```
*     if (kLPUART_TxDataRegEmptyFlag &
      LPUART_GetStatusFlags(LPUART1))
*     {
*         ...
*     }
*
```

Parameters

base	LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

28.2.7.9 status_t LPUART_ClearStatusFlags (LPUART_Type ∗ *base*, uint32_t *mask*)

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_-

RxActiveFlag, kLPUART_NoiseErrorInRxDataRegFlag, kLPUART_ParityErrorInRxDataRegFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

base	LPUART peripheral base address.
mask	the status flags to be cleared. The user can use the enumerators in the _lpuart_status-_flag_t to do the OR operation and get the mask.

Returns

0 succeed, others failed.

Return values

kStatus_LPUART_Flag-CannotClearManually	The flag can't be cleared by this function but it is cleared automatically by hardware.
kStatus_Success	Status in the mask are cleared.

28.2.7.10 void LPUART_EnableInterrupts (LPUART_Type ∗ *base*, uint32_t *mask*)

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the _lpuart_interrupt_enable. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
*     LPUART_EnableInterrupts(LPUART1,
      kLPUART_TxDataRegEmptyInterruptEnable |
      kLPUART_RxDataRegFullInterruptEnable);
*
```

Parameters

base	LPUART peripheral base address.
mask	The interrupts to enable. Logical OR of the enumeration _uart_interrupt_enable.

28.2.7.11 void LPUART_DisableInterrupts (LPUART_Type ∗ *base*, uint32_t *mask*)

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _lpuart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
*     LPUART_DisableInterrupts(LPUART1,
      kLPUART_TxDataRegEmptyInterruptEnable |
      kLPUART_RxDataRegFullInterruptEnable);
*
```

Parameters

base	LPUART peripheral base address.
mask	The interrupts to disable. Logical OR of _lpuart_interrupt_enable.

28.2.7.12 uint32_t LPUART_GetEnabledInterrupts (LPUART_Type * *base*)

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _lpuart_interrupt_enable. To check a specific interrupt enable status, compare the return value with enumerators in _lpuart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
*     uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);
*
*     if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
*     {
*         ...
*     }
*
```

Parameters

base	LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in _lpuart_interrupt_enable.

28.2.7.13 static uint32_t LPUART_GetDataRegisterAddress (LPUART_Type * *base*) [inline], [static]

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

base	LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

28.2.7.14 static void LPUART_EnableTxDMA (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

base	LPUART peripheral base address.
enable	True to enable, false to disable.

28.2.7.15 static void LPUART_EnableRxDMA (LPUART_Type ∗ *base*, bool *enable*) [inline], [static]

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

base	LPUART peripheral base address.
enable	True to enable, false to disable.

28.2.7.16 uint32_t LPUART_GetInstance (LPUART_Type ∗ *base*)

Parameters

base	LPUART peripheral base address.

Returns

LPUART instance.

28.2.7.17 static void LPUART_EnableTx (LPUART_Type ∗ *base*, bool *enable*) [inline], [static]

This function enables or disables the LPUART transmitter.

Parameters

base	LPUART peripheral base address.
enable	True to enable, false to disable.

28.2.7.18 static void LPUART_EnableRx (LPUART_Type ∗ *base*, bool *enable*) [inline], [static]

This function enables or disables the LPUART receiver.

Parameters

base	LPUART peripheral base address.
enable	True to enable, false to disable.

28.2.7.19 static void LPUART_WriteByte (LPUART_Type * *base*, uint8_t *data*) [inline], [static]

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

base	LPUART peripheral base address.
data	Data write to the TX register.

28.2.7.20 static uint8_t LPUART_ReadByte (LPUART_Type * *base*) [inline], [static]

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

base	LPUART peripheral base address.

Returns

Data read from data register.

28.2.7.21 void LPUART_SendAddress (LPUART_Type * *base*, uint8_t *address*)

Parameters

base	LPUART peripheral base address.

address	LPUART slave address.

28.2.7.22 status_t LPUART_WriteBlocking (LPUART_Type ∗ *base*, const uint8_t ∗ *data*, size_t *length*)

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

base	LPUART peripheral base address.
data	Start address of the data to write.
length	Size of the data to write.

Return values

kStatus_LPUART_-Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully wrote all data.

28.2.7.23 status_t LPUART_ReadBlocking (LPUART_Type ∗ *base*, uint8_t ∗ *data*, size_t *length*)

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

base	LPUART peripheral base address.
data	Start address of the buffer to store the received data.
length	Size of the buffer.

Return values

kStatus_LPUART_Rx-HardwareOverrun	Receiver overrun happened while receiving data.

kStatus_LPUART_Noise-Error	Noise error happened while receiving data.
kStatus_LPUART_-FramingError	Framing error happened while receiving data.
kStatus_LPUART_Parity-Error	Parity error happened while receiving data.
kStatus_LPUART_-Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully received all data.

28.2.7.24 void LPUART_TransferCreateHandle (LPUART_Type ∗ *base,* lpuart_handle_t ∗ *handle,* lpuart_transfer_callback_t *callback,* void ∗ *userData*)

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the LPUART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
callback	Callback function.
userData	User data.

28.2.7.25 status_t LPUART_TransferSendNonBlocking (LPUART_Type ∗ *base,* lpuart_handle_t ∗ *handle,* lpuart_transfer_t ∗ *xfer*)

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus_LPUART_TxIdle as status parameter.

Note

The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the T-X, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART transfer structure, see lpuart_transfer_t.

Return values

kStatus_Success	Successfully start the data transmission.
kStatus_LPUART_TxBusy	Previous transmission still not finished, data not all written to the TX register.
kStatus_InvalidArgument	Invalid argument.

28.2.7.26 void LPUART_TransferStartRingBuffer (LPUART_Type * *base,* lpuart_handle_t * *handle,* uint8_t * *ringBuffer,* size_t *ringBufferSize*)

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note

When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
ringBuffer	Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
ringBufferSize	size of the ring buffer.

28.2.7.27 void LPUART_TransferStopRingBuffer (LPUART_Type * *base,* lpuart_handle_t * *handle*)

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

28.2.7.28 size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type ∗ *base*, lpuart_handle_t ∗ *handle*)

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

28.2.7.29 void LPUART_TransferAbortSend (LPUART_Type ∗ *base*, lpuart_handle_t ∗ *handle*)

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

28.2.7.30 status_t LPUART_TransferGetSendCount (LPUART_Type ∗ *base*, lpuart_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Send bytes count.

Return values

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

28.2.7.31 status_t LPUART_TransferReceiveNonBlocking (LPUART_Type ∗ *base*, lpuart_handle_t ∗ *handle*, lpuart_transfer_t ∗ *xfer*, size_t ∗ *receivedBytes*)

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter `receivedBytes` set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART transfer structure, see uart_transfer_t.
receivedBytes	Bytes received from the ring buffer directly.

Return values

kStatus_Success	Successfully queue the transfer into the transmit queue.
kStatus_LPUART_Rx-Busy	Previous receive request is not finished.
kStatus_InvalidArgument	Invalid argument.

28.2.7.32 void LPUART_TransferAbortReceive (LPUART_Type ∗ *base,* lpuart_handle_t ∗ *handle*)

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

28.2.7.33 status_t LPUART_TransferGetReceiveCount (LPUART_Type ∗ *base,* lpuart_handle_t ∗ *handle,* uint32_t ∗ *count*)

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

Parameters

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

Return values

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

28.2.7.34 void LPUART_TransferHandleIRQ (LPUART_Type ∗ *base,* lpuart_handle_t ∗ *handle*)

This function handles the LPUART transmit and receive IRQ request.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

28.2.7.35 void LPUART_TransferHandleErrorIRQ (LPUART_Type ∗ *base*, lpuart_handle_t ∗ *handle*)

This function handles the LPUART error IRQ request.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

Chapter 29
PDB: Programmable Delay Block

Overview

The MCUXpresso SDK provides a peripheral driver for the Programmable Delay Block (PDB) module of MCUXpresso SDK devices.

The PDB driver includes a basic PDB counter, trigger generators for ADC, DAC, and pulse-out.

The basic PDB counter can be used as a general programmable timer with an interrupt. The counter increases automatically with the divided clock signal after it is triggered to start by an external trigger input or the software trigger. There are "milestones" for the output trigger event. When the counter is equal to any of these "milestones", the corresponding trigger is generated and sent out to other modules. These "milestones" are for the following events.

- Counter delay interrupt, which is the interrupt for the PDB module
- ADC pre-trigger to trigger the ADC conversion
- DAC interval trigger to trigger the DAC buffer and move the buffer read pointer
- Pulse-out triggers to generate a single of rising and falling edges, which can be assembled to a window.

The "milestone" values have a flexible load mode. To call the APIs to set these value is equivalent to writing data to their buffer. The loading event occurs as the load mode describes. This design ensures that all "milestones" can be updated at the same time.

Typical use case

29.2.1 Working as basic PDB counter with a PDB interrupt.

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

29.2.2 Working with an additional trigger. The ADC trigger is used as an example.

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

Data Structures

- struct pdb_config_t
  PDB module configuration. More...
- struct pdb_adc_pretrigger_config_t
  PDB ADC Pre-trigger configuration. More...
- struct pdb_dac_trigger_config_t
  PDB DAC trigger configuration. More...

Enumerations

- enum _pdb_status_flags {
kPDB_LoadOKFlag = PDB_SC_LDOK_MASK,
kPDB_DelayEventFlag = PDB_SC_PDBIF_MASK }
PDB flags.
- enum _pdb_adc_pretrigger_flags {
kPDB_ADCPreTriggerChannel0Flag = PDB_S_CF(1U << 0),
kPDB_ADCPreTriggerChannel1Flag = PDB_S_CF(1U << 1),
kPDB_ADCPreTriggerChannel0ErrorFlag = PDB_S_ERR(1U << 0),
kPDB_ADCPreTriggerChannel1ErrorFlag = PDB_S_ERR(1U << 1) }
PDB ADC PreTrigger channel flags.
- enum _pdb_interrupt_enable {
kPDB_SequenceErrorInterruptEnable = PDB_SC_PDBEIE_MASK,
kPDB_DelayInterruptEnable = PDB_SC_PDBIE_MASK }
PDB buffer interrupts.
- enum pdb_load_value_mode_t {
kPDB_LoadValueImmediately = 0U,
kPDB_LoadValueOnCounterOverflow = 1U,
kPDB_LoadValueOnTriggerInput = 2U,
kPDB_LoadValueOnCounterOverflowOrTriggerInput = 3U }
PDB load value mode.
- enum pdb_prescaler_divider_t {
kPDB_PrescalerDivider1 = 0U,
kPDB_PrescalerDivider2 = 1U,
kPDB_PrescalerDivider4 = 2U,
kPDB_PrescalerDivider8 = 3U,
kPDB_PrescalerDivider16 = 4U,
kPDB_PrescalerDivider32 = 5U,
kPDB_PrescalerDivider64 = 6U,
kPDB_PrescalerDivider128 = 7U }
Prescaler divider.
- enum pdb_divider_multiplication_factor_t {
kPDB_DividerMultiplicationFactor1 = 0U,
kPDB_DividerMultiplicationFactor10 = 1U,
kPDB_DividerMultiplicationFactor20 = 2U,
kPDB_DividerMultiplicationFactor40 = 3U }
Multiplication factor select for prescaler.
- enum pdb_trigger_input_source_t {

kPDB_TriggerInput0 = 0U,
kPDB_TriggerInput1 = 1U,
kPDB_TriggerInput2 = 2U,
kPDB_TriggerInput3 = 3U,
kPDB_TriggerInput4 = 4U,
kPDB_TriggerInput5 = 5U,
kPDB_TriggerInput6 = 6U,
kPDB_TriggerInput7 = 7U,
kPDB_TriggerInput8 = 8U,
kPDB_TriggerInput9 = 9U,
kPDB_TriggerInput10 = 10U,
kPDB_TriggerInput11 = 11U,
kPDB_TriggerInput12 = 12U,
kPDB_TriggerInput13 = 13U,
kPDB_TriggerInput14 = 14U,
kPDB_TriggerSoftware = 15U }
Trigger input source.

- enum pdb_adc_trigger_channel_t {
kPDB_ADCTriggerChannel0 = 0U,
kPDB_ADCTriggerChannel1 = 1U,
kPDB_ADCTriggerChannel2 = 2U,
kPDB_ADCTriggerChannel3 = 3U }
List of PDB ADC trigger channels.
- enum pdb_adc_pretrigger_t {
kPDB_ADCPreTrigger0 = 0U,
kPDB_ADCPreTrigger1 = 1U,
kPDB_ADCPreTrigger2 = 2U,
kPDB_ADCPreTrigger3 = 3U,
kPDB_ADCPreTrigger4 = 4U,
kPDB_ADCPreTrigger5 = 5U,
kPDB_ADCPreTrigger6 = 6U,
kPDB_ADCPreTrigger7 = 7U }
List of PDB ADC pretrigger.
- enum pdb_dac_trigger_channel_t {
kPDB_DACTriggerChannel0 = 0U,
kPDB_DACTriggerChannel1 = 1U }
List of PDB DAC trigger channels.
- enum pdb_pulse_out_trigger_channel_t {
kPDB_PulseOutTriggerChannel0 = 0U,
kPDB_PulseOutTriggerChannel1 = 1U,
kPDB_PulseOutTriggerChannel2 = 2U,
kPDB_PulseOutTriggerChannel3 = 3U }
List of PDB pulse out trigger channels.
- enum pdb_pulse_out_channel_mask_t {

kPDB_PulseOutChannel0Mask = (1U << 0U),
kPDB_PulseOutChannel1Mask = (1U << 1U),
kPDB_PulseOutChannel2Mask = (1U << 2U),
kPDB_PulseOutChannel3Mask = (1U << 3U) }
List of PDB pulse out trigger channels mask.

Driver version

- #define FSL_PDB_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  PDB driver version 2.0.4.

Initialization

- void PDB_Init (PDB_Type ∗base, const pdb_config_t ∗config)
  Initializes the PDB module.
- void PDB_Deinit (PDB_Type ∗base)
  De-initializes the PDB module.
- void PDB_GetDefaultConfig (pdb_config_t ∗config)
  Initializes the PDB user configuration structure.
- static void PDB_Enable (PDB_Type ∗base, bool enable)
  Enables the PDB module.

Basic Counter

- static void PDB_DoSoftwareTrigger (PDB_Type ∗base)
  Triggers the PDB counter by software.
- static void PDB_DoLoadValues (PDB_Type ∗base)
  Loads the counter values.
- static void PDB_EnableDMA (PDB_Type ∗base, bool enable)
  Enables the DMA for the PDB module.
- static void PDB_EnableInterrupts (PDB_Type ∗base, uint32_t mask)
  Enables the interrupts for the PDB module.
- static void PDB_DisableInterrupts (PDB_Type ∗base, uint32_t mask)
  Disables the interrupts for the PDB module.
- static uint32_t PDB_GetStatusFlags (PDB_Type ∗base)
  Gets the status flags of the PDB module.
- static void PDB_ClearStatusFlags (PDB_Type ∗base, uint32_t mask)
  Clears the status flags of the PDB module.
- static void PDB_SetModulusValue (PDB_Type ∗base, uint32_t value)
  Specifies the counter period.
- static uint32_t PDB_GetCounterValue (PDB_Type ∗base)
  Gets the PDB counter's current value.
- static void PDB_SetCounterDelayValue (PDB_Type ∗base, uint32_t value)
  Sets the value for the PDB counter delay event.

ADC Pre-trigger

- static void PDB_SetADCPreTriggerConfig (PDB_Type ∗base, pdb_adc_trigger_channel_t channel, pdb_adc_pretrigger_config_t ∗config)
  Configures the ADC pre-trigger in the PDB module.

- static void PDB_SetADCPreTriggerDelayValue (PDB_Type *base, pdb_adc_trigger_channel_t channel, pdb_adc_pretrigger_t pretriggerNumber, uint32_t value)
    Sets the value for the ADC pre-trigger delay event.
- static uint32_t PDB_GetADCPreTriggerStatusFlags (PDB_Type *base, pdb_adc_trigger_channel_t channel)
    Gets the ADC pre-trigger's status flags.
- static void PDB_ClearADCPreTriggerStatusFlags (PDB_Type *base, pdb_adc_trigger_channel_t channel, uint32_t mask)
    Clears the ADC pre-trigger status flags.

DAC Interval Trigger

- void PDB_SetDACTriggerConfig (PDB_Type *base, pdb_dac_trigger_channel_t channel, pdb_dac_trigger_config_t *config)
    Configures the DAC trigger in the PDB module.
- static void PDB_SetDACTriggerIntervalValue (PDB_Type *base, pdb_dac_trigger_channel_t channel, uint32_t value)
    Sets the value for the DAC interval event.

Pulse-Out Trigger

- static void PDB_EnablePulseOutTrigger (PDB_Type *base, pdb_pulse_out_channel_mask_t channelMask, bool enable)
    Enables the pulse out trigger channels.
- static void PDB_SetPulseOutTriggerDelayValue (PDB_Type *base, pdb_pulse_out_trigger_channel_t channel, uint32_t value1, uint32_t value2)
    Sets event values for the pulse out trigger.

Data Structure Documentation

29.3.1 struct pdb_config_t

Data Fields

- pdb_load_value_mode_t loadValueMode
    Select the load value mode.
- pdb_prescaler_divider_t prescalerDivider
    Select the prescaler divider.
- pdb_divider_multiplication_factor_t dividerMultiplicationFactor
    Multiplication factor select for prescaler.
- pdb_trigger_input_source_t triggerInputSource
    Select the trigger input source.
- bool enableContinuousMode
    Enable the PDB operation in Continuous mode.

29.3.1.0.0.21 Field Documentation

29.3.1.0.0.21.1 pdb_load_value_mode_t pdb_config_t::loadValueMode

29.3.1.0.0.21.2 pdb_prescaler_divider_t pdb_config_t::prescalerDivider

29.3.1.0.0.21.3 pdb_divider_multiplication_factor_t pdb_config_t::dividerMultiplicationFactor

29.3.1.0.0.21.4 pdb_trigger_input_source_t pdb_config_t::triggerInputSource

29.3.1.0.0.21.5 bool pdb_config_t::enableContinuousMode

29.3.2 struct pdb_adc_pretrigger_config_t

Data Fields

- uint32_t enablePreTriggerMask
  PDB Channel Pre-trigger Enable.
- uint32_t enableOutputMask
  PDB Channel Pre-trigger Output Select.
- uint32_t enableBackToBackOperationMask
  PDB Channel pre-trigger Back-to-Back Operation Enable.

29.3.2.0.0.22 Field Documentation

29.3.2.0.0.22.1 uint32_t pdb_adc_pretrigger_config_t::enablePreTriggerMask

29.3.2.0.0.22.2 uint32_t pdb_adc_pretrigger_config_t::enableOutputMask

PDB channel's corresponding pre-trigger asserts when the counter reaches the channel delay register.

29.3.2.0.0.22.3 uint32_t pdb_adc_pretrigger_config_t::enableBackToBackOperationMask

Back-to-back operation enables the ADC conversions complete to trigger the next PDB channel pre-trigger and trigger output, so that the ADC conversions can be triggered on next set of configuration and results registers.

29.3.3 struct pdb_dac_trigger_config_t

Data Fields

- bool enableExternalTriggerInput
  Enables the external trigger for DAC interval counter.
- bool enableIntervalTrigger
  Enables the DAC interval trigger.

29.3.3.0.0.23 Field Documentation

29.3.3.0.0.23.1 bool pdb_dac_trigger_config_t::enableExternalTriggerInput

29.3.3.0.0.23.2 bool pdb_dac_trigger_config_t::enableIntervalTrigger

Macro Definition Documentation

29.4.1 #define FSL_PDB_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

Enumeration Type Documentation

29.5.1 enum _pdb_status_flags

Enumerator

kPDB_LoadOKFlag This flag is automatically cleared when the values in buffers are loaded into the internal registers after the LDOK bit is set or the PDBEN is cleared.
kPDB_DelayEventFlag PDB timer delay event flag.

29.5.2 enum _pdb_adc_pretrigger_flags

Enumerator

kPDB_ADCPreTriggerChannel0Flag Pre-trigger 0 flag.
kPDB_ADCPreTriggerChannel1Flag Pre-trigger 1 flag.
kPDB_ADCPreTriggerChannel0ErrorFlag Pre-trigger 0 Error.
kPDB_ADCPreTriggerChannel1ErrorFlag Pre-trigger 1 Error.

29.5.3 enum _pdb_interrupt_enable

Enumerator

kPDB_SequenceErrorInterruptEnable PDB sequence error interrupt enable.
kPDB_DelayInterruptEnable PDB delay interrupt enable.

29.5.4 enum pdb_load_value_mode_t

Selects the mode to load the internal values after doing the load operation (write 1 to PDBx_SC[LDOK]). These values are for the following operations.

- PDB counter (PDBx_MOD, PDBx_IDLY)
- ADC trigger (PDBx_CHnDLYm)

- DAC trigger (PDBx_DACINTx)
- CMP trigger (PDBx_POyDLY)

Enumerator

kPDB_LoadValueImmediately Load immediately after 1 is written to LDOK.
kPDB_LoadValueOnCounterOverflow Load when the PDB counter overflows (reaches the MOD register value).
kPDB_LoadValueOnTriggerInput Load a trigger input event is detected.
kPDB_LoadValueOnCounterOverflowOrTriggerInput Load either when the PDB counter overflows or a trigger input is detected.

29.5.5 enum pdb_prescaler_divider_t

Counting uses the peripheral clock divided by multiplication factor selected by times of MULT.

Enumerator

kPDB_PrescalerDivider1 Divider x1.
kPDB_PrescalerDivider2 Divider x2.
kPDB_PrescalerDivider4 Divider x4.
kPDB_PrescalerDivider8 Divider x8.
kPDB_PrescalerDivider16 Divider x16.
kPDB_PrescalerDivider32 Divider x32.
kPDB_PrescalerDivider64 Divider x64.
kPDB_PrescalerDivider128 Divider x128.

29.5.6 enum pdb_divider_multiplication_factor_t

Selects the multiplication factor of the prescaler divider for the counter clock.

Enumerator

kPDB_DividerMultiplicationFactor1 Multiplication factor is 1.
kPDB_DividerMultiplicationFactor10 Multiplication factor is 10.
kPDB_DividerMultiplicationFactor20 Multiplication factor is 20.
kPDB_DividerMultiplicationFactor40 Multiplication factor is 40.

29.5.7 enum pdb_trigger_input_source_t

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger. See chip configuration details for the actual PDB input trigger connections.

Enumerator

kPDB_TriggerInput0 Trigger-In 0.
kPDB_TriggerInput1 Trigger-In 1.
kPDB_TriggerInput2 Trigger-In 2.
kPDB_TriggerInput3 Trigger-In 3.
kPDB_TriggerInput4 Trigger-In 4.
kPDB_TriggerInput5 Trigger-In 5.
kPDB_TriggerInput6 Trigger-In 6.
kPDB_TriggerInput7 Trigger-In 7.
kPDB_TriggerInput8 Trigger-In 8.
kPDB_TriggerInput9 Trigger-In 9.
kPDB_TriggerInput10 Trigger-In 10.
kPDB_TriggerInput11 Trigger-In 11.
kPDB_TriggerInput12 Trigger-In 12.
kPDB_TriggerInput13 Trigger-In 13.
kPDB_TriggerInput14 Trigger-In 14.
kPDB_TriggerSoftware Trigger-In 15, software trigger.

29.5.8 enum pdb_adc_trigger_channel_t

Note

Actual number of available channels is SoC dependent

Enumerator

kPDB_ADCTriggerChannel0 PDB ADC trigger channel number 0.
kPDB_ADCTriggerChannel1 PDB ADC trigger channel number 1.
kPDB_ADCTriggerChannel2 PDB ADC trigger channel number 2.
kPDB_ADCTriggerChannel3 PDB ADC trigger channel number 3.

29.5.9 enum pdb_adc_pretrigger_t

Note

Actual number of available pretrigger channels is SoC dependent

Enumerator

kPDB_ADCPreTrigger0 PDB ADC pretrigger number 0.
kPDB_ADCPreTrigger1 PDB ADC pretrigger number 1.
kPDB_ADCPreTrigger2 PDB ADC pretrigger number 2.
kPDB_ADCPreTrigger3 PDB ADC pretrigger number 3.

kPDB_ADCPreTrigger4 PDB ADC pretrigger number 4.
kPDB_ADCPreTrigger5 PDB ADC pretrigger number 5.
kPDB_ADCPreTrigger6 PDB ADC pretrigger number 6.
kPDB_ADCPreTrigger7 PDB ADC pretrigger number 7.

29.5.10 enum pdb_dac_trigger_channel_t

Note

Actual number of available channels is SoC dependent

Enumerator

kPDB_DACTriggerChannel0 PDB DAC trigger channel number 0.
kPDB_DACTriggerChannel1 PDB DAC trigger channel number 1.

29.5.11 enum pdb_pulse_out_trigger_channel_t

Note

Actual number of available channels is SoC dependent

Enumerator

kPDB_PulseOutTriggerChannel0 PDB pulse out trigger channel number 0.
kPDB_PulseOutTriggerChannel1 PDB pulse out trigger channel number 1.
kPDB_PulseOutTriggerChannel2 PDB pulse out trigger channel number 2.
kPDB_PulseOutTriggerChannel3 PDB pulse out trigger channel number 3.

29.5.12 enum pdb_pulse_out_channel_mask_t

Note

Actual number of available channels mask is SoC dependent

Enumerator

kPDB_PulseOutChannel0Mask PDB pulse out trigger channel number 0 mask.
kPDB_PulseOutChannel1Mask PDB pulse out trigger channel number 1 mask.
kPDB_PulseOutChannel2Mask PDB pulse out trigger channel number 2 mask.
kPDB_PulseOutChannel3Mask PDB pulse out trigger channel number 3 mask.

Function Documentation

29.6.1 void PDB_Init (PDB_Type * *base*, const pdb_config_t * *config*)

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

- Enable the clock for PDB instance.
- Configure the PDB module.
- Enable the PDB module.

Parameters

base	PDB peripheral base address.
config	Pointer to the configuration structure. See "pdb_config_t".

29.6.2 void PDB_Deinit (PDB_Type * *base*)

Parameters

base	PDB peripheral base address.

29.6.3 void PDB_GetDefaultConfig (pdb_config_t * *config*)

This function initializes the user configuration structure to a default value. The default values are as follows.

```
*   config->loadValueMode = kPDB_LoadValueImmediately;
*   config->prescalerDivider = kPDB_PrescalerDivider1;
*   config->dividerMultiplicationFactor = kPDB_DividerMultiplicationFactor1
    ;
*   config->triggerInputSource = kPDB_TriggerSoftware;
*   config->enableContinuousMode = false;
*
```

Parameters

config	Pointer to configuration structure. See "pdb_config_t".

29.6.4 static void PDB_Enable (PDB_Type * *base*, bool *enable*) [inline], [static]

Parameters

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

29.6.5 static void PDB_DoSoftwareTrigger (PDB_Type ∗ *base*) [inline], [static]

Parameters

base	PDB peripheral base address.

29.6.6 static void PDB_DoLoadValues (PDB_Type ∗ *base*) [inline], [static]

This function loads the counter values from the internal buffer. See "pdb_load_value_mode_t" about PD-B's load mode.

Parameters

base	PDB peripheral base address.

29.6.7 static void PDB_EnableDMA (PDB_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

base	PDB peripheral base address.
enable	Enable the feature or not.

29.6.8 static void PDB_EnableInterrupts (PDB_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PDB peripheral base address.
mask	Mask value for interrupts. See "_pdb_interrupt_enable".

29.6.9 static void PDB_DisableInterrupts (PDB_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PDB peripheral base address.
mask	Mask value for interrupts. See "_pdb_interrupt_enable".

29.6.10 static uint32_t PDB_GetStatusFlags (PDB_Type * *base*) [inline], [static]

Parameters

base	PDB peripheral base address.

Returns

Mask value for asserted flags. See "_pdb_status_flags".

29.6.11 static void PDB_ClearStatusFlags (PDB_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PDB peripheral base address.
mask	Mask value of flags. See "_pdb_status_flags".

29.6.12 static void PDB_SetModulusValue (PDB_Type * *base*, uint32_t *value*) [inline], [static]

Parameters

base	PDB peripheral base address.
value	Setting value for the modulus. 16-bit is available.

29.6.13 static uint32_t PDB_GetCounterValue (PDB_Type * *base*) [inline], [static]

Parameters

base	PDB peripheral base address.

Returns

PDB counter's current value.

29.6.14 static void PDB_SetCounterDelayValue (PDB_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	PDB peripheral base address.
value	Setting value for PDB counter delay event. 16-bit is available.

29.6.15 static void PDB_SetADCPreTriggerConfig (PDB_Type * *base,* pdb_adc_trigger_channel_t *channel,* pdb_adc_pretrigger_config_t * *config*) [inline], [static]

Parameters

base	PDB peripheral base address.
channel	Channel index for ADC instance.

config	Pointer to the configuration structure. See "pdb_adc_pretrigger_config_t".

29.6.16 static void PDB_SetADCPreTriggerDelayValue (PDB_Type * *base*, pdb_adc_trigger_channel_t *channel*, pdb_adc_pretrigger_t *pretriggerNumber*, uint32_t *value*) [inline], [static]

This function sets the value for ADC pre-trigger delay event. It specifies the delay value for the channel's corresponding pre-trigger. The pre-trigger asserts when the PDB counter is equal to the set value.

Parameters

base	PDB peripheral base address.
channel	Channel index for ADC instance.
pretrigger-Number	Channel group index for ADC instance.
value	Setting value for ADC pre-trigger delay event. 16-bit is available.

29.6.17 static uint32_t PDB_GetADCPreTriggerStatusFlags (PDB_Type * *base*, pdb_adc_trigger_channel_t *channel*) [inline], [static]

Parameters

base	PDB peripheral base address.
channel	Channel index for ADC instance.

Returns

Mask value for asserted flags. See "_pdb_adc_pretrigger_flags".

29.6.18 static void PDB_ClearADCPreTriggerStatusFlags (PDB_Type * *base*, pdb_adc_trigger_channel_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	PDB peripheral base address.
channel	Channel index for ADC instance.
mask	Mask value for flags. See "_pdb_adc_pretrigger_flags".

29.6.19 void PDB_SetDACTriggerConfig (PDB_Type ∗ *base*, pdb_dac_trigger_channel_t *channel*, pdb_dac_trigger_config_t ∗ *config*)

Parameters

base	PDB peripheral base address.
channel	Channel index for DAC instance.
config	Pointer to the configuration structure. See "pdb_dac_trigger_config_t".

29.6.20 static void PDB_SetDACTriggerIntervalValue (PDB_Type ∗ *base*, pdb_dac_trigger_channel_t *channel*, uint32_t *value*) [inline], [static]

This function sets the value for DAC interval event. DAC interval trigger triggers the DAC module to update the buffer when the DAC interval counter is equal to the set value.

Parameters

base	PDB peripheral base address.
channel	Channel index for DAC instance.
value	Setting value for the DAC interval event.

29.6.21 static void PDB_EnablePulseOutTrigger (PDB_Type ∗ *base*, pdb_pulse_out_channel_mask_t *channelMask*, bool *enable*) [inline], [static]

Parameters

base	PDB peripheral base address.
channelMask	Channel mask value for multiple pulse out trigger channel.
enable	Whether the feature is enabled or not.

29.6.22 static void PDB_SetPulseOutTriggerDelayValue (PDB_Type * *base*, pdb_pulse_out_trigger_channel_t *channel*, uint32_t *value1*, uint32_t *value2*) [inline], [static]

This function is used to set event values for the pulse output trigger. These pulse output trigger delay values specify the delay for the PDB Pulse-out. Pulse-out goes high when the PDB counter is equal to the pulse output high value (value1). Pulse-out goes low when the PDB counter is equal to the pulse output low value (value2).

Parameters

base	PDB peripheral base address.
channel	Channel index for pulse out trigger channel.
value1	Setting value for pulse out high.
value2	Setting value for pulse out low.

Chapter 30
PIT: Periodic Interrupt Timer

Overview

The MCUXpresso SDK provides a driver for the Periodic Interrupt Timer (PIT) of MCUXpresso SDK devices.

Function groups

The PIT driver supports operating the module as a time counter.

30.2.1 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.

30.2.2 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.

30.2.3 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.

30.2.4 Status

Provides functions to get and clear the PIT status.

30.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

Typical use case

30.3.1 PIT tick example

Updates the PIT period and toggles an LED periodically. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/pit

Data Structures

- struct pit_config_t
  PIT configuration structure. More...

Enumerations

- enum pit_chnl_t {
  kPIT_Chnl_0 = 0U,
  kPIT_Chnl_1,
  kPIT_Chnl_2,
  kPIT_Chnl_3 }
  List of PIT channels.
- enum pit_interrupt_enable_t { kPIT_TimerInterruptEnable = PIT_TCTRL_TIE_MASK }
  List of PIT interrupts.
- enum pit_status_flags_t { kPIT_TimerFlag = PIT_TFLG_TIF_MASK }
  List of PIT status flags.

Functions

- uint64_t PIT_GetLifetimeTimerCount (PIT_Type ∗base)
  Reads the current lifetime counter value.

Driver version

- #define FSL_PIT_DRIVER_VERSION (MAKE_VERSION(2, 0, 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.

Data Structure Documentation

30.4.1 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

Enumeration Type Documentation

30.5.1 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.

30.5.2 enum pit_interrupt_enable_t

Enumerator

kPIT_TimerInterruptEnable Timer interrupt enable.

30.5.3 enum pit_status_flags_t

Enumerator

kPIT_TimerFlag Timer flag.

Function Documentation

30.6.1 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

30.6.2 void PIT_Deinit (PIT_Type ∗ *base*)

Parameters

base	PIT peripheral base address

30.6.3 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.

30.6.4 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.

30.6.5 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

30.6.6 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

30.6.7 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

30.6.8 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

30.6.9 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

30.6.10 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

30.6.11 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

30.6.12 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.

30.6.13 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.

30.6.14 uint64_t PIT_GetLifetimeTimerCount (PIT_Type ∗ *base*)

The lifetime timer is a 64-bit timer which chains timer 0 and timer 1 together. Timer 0 and 1 are chained by calling the PIT_SetTimerChainMode before using this timer. The period of lifetime timer is equal to the "period of timer 0 ∗ period of timer 1". For the 64-bit value, the higher 32-bit has the value of timer 1, and the lower 32-bit has the value of timer 0.

Parameters

base	PIT peripheral base address

Returns

Current lifetime timer value

Chapter 31
PMC: Power Management Controller

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.

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.

Data Structure Documentation

31.2.1 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.

31.2.2 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.

31.2.3 struct pmc_bandgap_buffer_config_t

Data Fields

- bool enable
  Enable bandgap buffer.
- bool enableInLowPowerMode

Enable bandgap buffer in low-power mode.

31.2.3.0.0.24 Field Documentation

31.2.3.0.0.24.1 bool pmc_bandgap_buffer_config_t::enable

31.2.3.0.0.24.2 bool pmc_bandgap_buffer_config_t::enableInLowPowerMode

Macro Definition Documentation

31.3.1 #define FSL_PMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

Version 2.0.3.

Enumeration Type Documentation

31.4.1 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)

31.4.2 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)

Function Documentation

31.5.1 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.

31.5.2 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

31.5.3 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.

31.5.4 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.

31.5.5 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.

31.5.6 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.

31.5.7 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

31.5.8 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.

31.5.9 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.

31.5.10 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 32
PORT: Port Control and Interrupts

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_open_drain_enable {
  kPORT_OpenDrainDisable = 0U,
  kPORT_OpenDrainEnable = 1U }
  Open Drain feature enable/disable.
- enum _port_passive_filter_enable {
  kPORT_PassiveFilterDisable = 0U,
  kPORT_PassiveFilterEnable = 1U }
  Passive filter feature enable/disable.
- enum _port_drive_strength {
  kPORT_LowDriveStrength = 0U,
  kPORT_HighDriveStrength = 1U }
  Configures the drive strength.
- enum _port_lock_register {
  kPORT_UnlockRegister = 0U,
  kPORT_LockRegister = 1U }
  Unlock/lock the pin control register field[15:0].
- enum port_mux_t {

kPORT_PinDisabledOrAnalog = 0U,
kPORT_MuxAsGpio = 1U,
kPORT_MuxAlt2 = 2U,
kPORT_MuxAlt3 = 3U,
kPORT_MuxAlt4 = 4U,
kPORT_MuxAlt5 = 5U,
kPORT_MuxAlt6 = 6U,
kPORT_MuxAlt7 = 7U,
kPORT_MuxAlt8 = 8U,
kPORT_MuxAlt9 = 9U,
kPORT_MuxAlt10 = 10U,
kPORT_MuxAlt11 = 11U,
kPORT_MuxAlt12 = 12U,
kPORT_MuxAlt13 = 13U,
kPORT_MuxAlt14 = 14U,
kPORT_MuxAlt15 = 15U }
Pin mux selection.
- enum port_interrupt_t {
kPORT_InterruptOrDMADisabled = 0x0U,
kPORT_DMARisingEdge = 0x1U,
kPORT_DMAFallingEdge = 0x2U,
kPORT_DMAEitherEdge = 0x3U,
kPORT_FlagRisingEdge = 0x05U,
kPORT_FlagFallingEdge = 0x06U,
kPORT_FlagEitherEdge = 0x07U,
kPORT_InterruptLogicZero = 0x8U,
kPORT_InterruptRisingEdge = 0x9U,
kPORT_InterruptFallingEdge = 0xAU,
kPORT_InterruptEitherEdge = 0xBU,
kPORT_InterruptLogicOne = 0xCU,
kPORT_ActiveHighTriggerOutputEnable = 0xDU,
kPORT_ActiveLowTriggerOutputEnable = 0xEU }
Configures the interrupt generation condition.
- enum port_digital_filter_clock_source_t {
kPORT_BusClock = 0U,
kPORT_LpoClock = 1U }
Digital filter clock source selection.

Driver version

- #define FSL_PORT_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
Version 2.1.1.

Configuration

- static void PORT_SetPinConfig (PORT_Type ∗base, uint32_t pin, const port_pin_config_t ∗config)

Sets the port PCR register.
- static void PORT_SetMultiplePinsConfig (PORT_Type ∗base, uint32_t mask, const port_pin_config_t ∗config)
  Sets the port PCR register for multiple pins.
- static void PORT_SetPinMux (PORT_Type ∗base, uint32_t pin, port_mux_t mux)
  Configures the pin muxing.
- static void PORT_EnablePinsDigitalFilter (PORT_Type ∗base, uint32_t mask, bool enable)
  Enables the digital filter in one port, each bit of the 32-bit register represents one pin.
- static void PORT_SetDigitalFilterConfig (PORT_Type ∗base, const port_digital_filter_config_t ∗config)
  Sets the digital filter in one port, each bit of the 32-bit register represents one pin.

Interrupt

- static void PORT_SetPinInterruptConfig (PORT_Type ∗base, uint32_t pin, port_interrupt_t config)
  Configures the port pin interrupt/DMA request.
- static void PORT_SetPinDriveStrength (PORT_Type ∗base, uint32_t pin, uint8_t strength)
  Configures the port pin drive strength.
- static uint32_t PORT_GetPinsInterruptFlags (PORT_Type ∗base)
  Reads the whole port status flag.
- static void PORT_ClearPinsInterruptFlags (PORT_Type ∗base, uint32_t mask)
  Clears the multiple pin interrupt status flag.

Data Structure Documentation

32.2.1 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.

32.2.2 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 passiveFilterEnable: 1
  Passive filter enable/disable.
- uint16_t openDrainEnable: 1
  Open drain enable/disable.
- uint16_t driveStrength: 1
  Fast/slow drive strength configure.

- uint16_t mux: 3
  Pin mux Configure.
- uint16_t lockRegister: 1
  Lock/unlock the PCR field[15:0].

Macro Definition Documentation

32.3.1 #define FSL_PORT_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

Enumeration Type Documentation

32.4.1 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.

32.4.2 enum _port_slew_rate

Enumerator

kPORT_FastSlewRate Fast slew rate is configured.
kPORT_SlowSlewRate Slow slew rate is configured.

32.4.3 enum _port_open_drain_enable

Enumerator

kPORT_OpenDrainDisable Open drain output is disabled.
kPORT_OpenDrainEnable Open drain output is enabled.

32.4.4 enum _port_passive_filter_enable

Enumerator

kPORT_PassiveFilterDisable Passive input filter is disabled.
kPORT_PassiveFilterEnable Passive input filter is enabled.

32.4.5 enum _port_drive_strength

Enumerator

kPORT_LowDriveStrength Low-drive strength is configured.
kPORT_HighDriveStrength High-drive strength is configured.

32.4.6 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.

32.4.7 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.

32.4.8 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.

32.4.9 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.

Function Documentation

32.5.1 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.

32.5.2 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.

32.5.3 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

32.5.4 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.

32.5.5 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.

32.5.6 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).

32.5.7 static void PORT_SetPinDriveStrength (PORT_Type * *base*, uint32_t *pin*, uint8_t *strength*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.

strength	PORT pin drive strength • kPORT_LowDriveStrength = 0U - Low-drive strength is configured. • kPORT_HighDriveStrength = 1U - High-drive strength is configured.

32.5.8 static uint32_t PORT_GetPinsInterruptFlags (PORT_Type ∗ *base*) [inline], [static]

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

Parameters

base	PORT peripheral base pointer.

Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

32.5.9 static void PORT_ClearPinsInterruptFlags (PORT_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
mask	PORT pin number macro.

Chapter 33
RCM: Reset Control Module Driver

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_SourceJtag = RCM_SRS1_JTAG_MASK << 8U,
  kRCM_SourceLockup = RCM_SRS1_LOCKUP_MASK << 8U,
  kRCM_SourceSw = RCM_SRS1_SW_MASK << 8U,
  kRCM_SourceMdmap = RCM_SRS1_MDM_AP_MASK << 8U,
  kRCM_SourceEzpt = RCM_SRS1_EZPT_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.
- static uint32_t RCM_GetStickyResetSources (RCM_Type ∗base)
  Gets the sticky reset source status.
- static void RCM_ClearStickyResetSources (RCM_Type ∗base, uint32_t sourceMasks)
  Clears the sticky reset source status.
- void RCM_ConfigureResetPinFilter (RCM_Type ∗base, const rcm_reset_pin_filter_config_t ∗config)
  Configures the reset pin filter.
- static bool RCM_GetEasyPortModePinStatus (RCM_Type ∗base)
  Gets the EZP_MS_B pin assert status.

Data Structure Documentation

33.2.1 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.

33.2.1.0.0.25 Field Documentation

33.2.1.0.0.25.1 bool rcm_reset_pin_filter_config_t::enableFilterInStop

33.2.1.0.0.25.2 rcm_run_wait_filter_mode_t rcm_reset_pin_filter_config_t::filterInRunWait

33.2.1.0.0.25.3 uint8_t rcm_reset_pin_filter_config_t::busClockFilterCount

Macro Definition Documentation

33.3.1 #define FSL_RCM_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

Enumeration Type Documentation

33.4.1 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_SourceJtag JTAG generated reset.
kRCM_SourceLockup Core lock up reset.
kRCM_SourceSw Software reset.
kRCM_SourceMdmap MDM-AP system reset.
kRCM_SourceEzpt EzPort reset.
kRCM_SourceSackerr Parameter could get all reset flags.

33.4.2 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.

Function Documentation

33.5.1 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.

33.5.2 static uint32_t RCM_GetStickyResetSources (RCM_Type ∗ *base*) [inline], [static]

This function gets the current reset source status that has not been cleared by software for a specific source.

This is an example.

```
* uint32_t resetStatus;
*
* To get all reset source statuses.
* resetStatus = RCM_GetStickyResetSources(RCM) & kRCM_SourceAll;
*
* To test whether the MCU is reset using Watchdog.
* resetStatus = RCM_GetStickyResetSources(RCM) &
      kRCM_SourceWdog;
*
* To test multiple reset sources.
* resetStatus = RCM_GetStickyResetSources(RCM) & (
      kRCM_SourceWdog | kRCM_SourcePin);
*
```

Parameters

base	RCM peripheral base address.

Returns

All reset source status bit map.

33.5.3 static void RCM_ClearStickyResetSources (RCM_Type ∗ *base*, uint32_t *sourceMasks*) [inline], [static]

This function clears the sticky system reset flags indicated by source masks.

This is an example.

```
* Clears multiple reset sources.
* RCM_ClearStickyResetSources(kRCM_SourceWdog |
      kRCM_SourcePin);
*
```

Parameters

base	RCM peripheral base address.
sourceMasks	reset source status bit map

33.5.4 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.

33.5.5 static bool RCM_GetEasyPortModePinStatus (RCM_Type * *base*) [inline], [static]

This function gets the easy port mode status (EZP_MS_B) pin assert status.

Parameters

base	RCM peripheral base address.

Returns

status true - asserted, false - reasserted

Chapter 34
RNGA: Random Number Generator Accelerator Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the Random Number Generator Accelerator (R-NGA) block of MCUXpresso SDK devices.

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.

Get random data from RNGA

1. RNGA_GetRandomData() function gets random data from the RNGA module.

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.

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.

Macro Definition Documentation

34.6.1 #define FSL_RNGA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

Enumeration Type Documentation

34.7.1 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.

Function Documentation

34.8.1 void RNGA_Init (RNG_Type ∗ *base*)

This function initializes the RNGA. When called, the RNGA entropy generation starts immediately.

Parameters

base	RNGA base address

34.8.2 void RNGA_Deinit (RNG_Type ∗ *base*)

This function shuts down the RNGA.

Parameters

base	RNGA base address

34.8.3 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

34.8.4 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

34.8.5 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

34.8.6 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 35
RTC: Real Time Clock

Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC) of MCUXpresso SDK devices.

Function groups

The RTC driver supports operating the module as a time counter.

35.2.1 Initialization and deinitialization

The function RTC_Init() initializes the RTC with specified configurations. The function RTC_GetDefaultConfig() gets the default configurations.

The function RTC_Deinit() disables the RTC timer and disables the module clock.

35.2.2 Set & Get Datetime

The function RTC_SetDatetime() sets the timer period in seconds. Users pass in the details in date & time format by using the below data structure.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/rtc The function RTC_GetDatetime() reads the current timer value in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

35.2.3 Set & Get Alarm

The function RTC_SetAlarm() sets the alarm time period in seconds. Users pass in the details in date & time format by using the datetime data structure.

The function RTC_GetAlarm() reads the alarm time in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

35.2.4 Start & Stop timer

The function RTC_StartTimer() starts the RTC time counter.

The function RTC_StopTimer() stops the RTC time counter.

35.2.5 Status

Provides functions to get and clear the RTC status.

35.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

35.2.7 RTC Oscillator

Some SoC's allow control of the RTC oscillator through the RTC module.

The function RTC_SetOscCapLoad() allows the user to modify the capacitor load configuration of the RTC oscillator.

35.2.8 Monotonic Counter

Some SoC's have a 64-bit Monotonic counter available in the RTC module.

The function RTC_SetMonotonicCounter() writes a 64-bit to the counter.

The function RTC_GetMonotonicCounter() reads the monotonic counter and returns the 64-bit counter value to the user.

The function RTC_IncrementMonotonicCounter() increments the Monotonic Counter by one.

Typical use case

35.3.1 RTC tick example

Example to set the RTC current time and trigger an alarm. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/rtc

Data Structures

- struct rtc_datetime_t
  Structure is used to hold the date and time. More...
- struct rtc_config_t
  RTC config structure. More...

Enumerations

- enum rtc_interrupt_enable_t {
  kRTC_TimeInvalidInterruptEnable = (1U << 0U),
  kRTC_TimeOverflowInterruptEnable = (1U << 1U),
  kRTC_AlarmInterruptEnable = (1U << 2U),
  kRTC_SecondsInterruptEnable = (1U << 3U),
  kRTC_MonotonicOverflowInterruptEnable = (1U << 4U) }
  List of RTC interrupts.
- enum rtc_status_flags_t {
  kRTC_TimeInvalidFlag = (1U << 0U),
  kRTC_TimeOverflowFlag = (1U << 1U),
  kRTC_AlarmFlag = (1U << 2U),
  kRTC_MonotonicOverflowFlag = (1U << 3U) }
  List of RTC flags.
- enum rtc_osc_cap_load_t {
  kRTC_Capacitor_2p = RTC_CR_SC2P_MASK,
  kRTC_Capacitor_4p = RTC_CR_SC4P_MASK,
  kRTC_Capacitor_8p = RTC_CR_SC8P_MASK,
  kRTC_Capacitor_16p = RTC_CR_SC16P_MASK }
  List of RTC Oscillator capacitor load settings.

Functions

- static void RTC_SetClockSource (RTC_Type ∗base)
  Set RTC clock source.
- static uint32_t RTC_GetTamperTimeSeconds (RTC_Type ∗base)
  Get the RTC tamper time seconds.
- static void RTC_SetOscCapLoad (RTC_Type ∗base, uint32_t capLoad)
  This function sets the specified capacitor configuration for the RTC oscillator.
- static void RTC_Reset (RTC_Type ∗base)
  Performs a software reset on the RTC module.
- static void RTC_EnableWakeUpPin (RTC_Type ∗base, bool enable)
  Enables or disables the RTC Wakeup Pin Operation.

Driver version

- #define FSL_RTC_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
  Version 2.2.1.

Initialization and deinitialization

- void RTC_Init (RTC_Type ∗base, const rtc_config_t ∗config)
  Ungates the RTC clock and configures the peripheral for basic operation.
- static void RTC_Deinit (RTC_Type ∗base)
  Stops the timer and gate the RTC clock.
- void RTC_GetDefaultConfig (rtc_config_t ∗config)
  Fills in the RTC config struct with the default settings.

Current Time & Alarm

- status_t RTC_SetDatetime (RTC_Type ∗base, const rtc_datetime_t ∗datetime)
  Sets the RTC date and time according to the given time structure.
- void RTC_GetDatetime (RTC_Type ∗base, rtc_datetime_t ∗datetime)
  Gets the RTC time and stores it in the given time structure.
- status_t RTC_SetAlarm (RTC_Type ∗base, const rtc_datetime_t ∗alarmTime)
  Sets the RTC alarm time.
- void RTC_GetAlarm (RTC_Type ∗base, rtc_datetime_t ∗datetime)
  Returns the RTC alarm time.

Interrupt Interface

- void RTC_EnableInterrupts (RTC_Type ∗base, uint32_t mask)
  Enables the selected RTC interrupts.
- void RTC_DisableInterrupts (RTC_Type ∗base, uint32_t mask)
  Disables the selected RTC interrupts.
- uint32_t RTC_GetEnabledInterrupts (RTC_Type ∗base)
  Gets the enabled RTC interrupts.

Status Interface

- uint32_t RTC_GetStatusFlags (RTC_Type ∗base)
  Gets the RTC status flags.
- void RTC_ClearStatusFlags (RTC_Type ∗base, uint32_t mask)
  Clears the RTC status flags.

Timer Start and Stop

- static void RTC_StartTimer (RTC_Type ∗base)
  Starts the RTC time counter.
- static void RTC_StopTimer (RTC_Type ∗base)
  Stops the RTC time counter.

Monotonic counter functions

- void RTC_GetMonotonicCounter (RTC_Type ∗base, uint64_t ∗counter)
  Reads the values of the Monotonic Counter High and Monotonic Counter Low and returns them as a single value.
- void RTC_SetMonotonicCounter (RTC_Type ∗base, uint64_t counter)
  Writes values Monotonic Counter High and Monotonic Counter Low by decomposing the given single value.
- status_t RTC_IncrementMonotonicCounter (RTC_Type ∗base)
  Increments the Monotonic Counter by one.

Data Structure Documentation

35.4.1 struct rtc_datetime_t

Data Fields

- uint16_t year
  Range from 1970 to 2099.
- uint8_t month
  Range from 1 to 12.
- uint8_t day
  Range from 1 to 31 (depending on month).
- uint8_t hour
  Range from 0 to 23.
- uint8_t minute
  Range from 0 to 59.
- uint8_t second
  Range from 0 to 59.

35.4.1.0.0.26 Field Documentation

35.4.1.0.0.26.1 uint16_t rtc_datetime_t::year

35.4.1.0.0.26.2 uint8_t rtc_datetime_t::month

35.4.1.0.0.26.3 uint8_t rtc_datetime_t::day

35.4.1.0.0.26.4 uint8_t rtc_datetime_t::hour

35.4.1.0.0.26.5 uint8_t rtc_datetime_t::minute

35.4.1.0.0.26.6 uint8_t rtc_datetime_t::second

35.4.2 struct rtc_config_t

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

The config struct can be made const so it resides in flash

Data Fields

- bool wakeupSelect
  true: Wakeup pin outputs the 32 KHz clock; false:Wakeup pin used to wakeup the chip
- bool updateMode
  true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked
- bool supervisorAccess

true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported
- uint32_t compensationInterval
  Compensation interval that is written to the CIR field in RTC TCR Register.
- uint32_t compensationTime
  Compensation time that is written to the TCR field in RTC TCR Register.

Enumeration Type Documentation

35.5.1 enum rtc_interrupt_enable_t

Enumerator

kRTC_TimeInvalidInterruptEnable Time invalid interrupt.
kRTC_TimeOverflowInterruptEnable Time overflow interrupt.
kRTC_AlarmInterruptEnable Alarm interrupt.
kRTC_SecondsInterruptEnable Seconds interrupt.
kRTC_MonotonicOverflowInterruptEnable Monotonic Overflow Interrupt Enable.

35.5.2 enum rtc_status_flags_t

Enumerator

kRTC_TimeInvalidFlag Time invalid flag.
kRTC_TimeOverflowFlag Time overflow flag.
kRTC_AlarmFlag Alarm flag.
kRTC_MonotonicOverflowFlag Monotonic Overflow Flag.

35.5.3 enum rtc_osc_cap_load_t

Enumerator

kRTC_Capacitor_2p 2 pF capacitor load
kRTC_Capacitor_4p 4 pF capacitor load
kRTC_Capacitor_8p 8 pF capacitor load
kRTC_Capacitor_16p 16 pF capacitor load

Function Documentation

35.6.1 void RTC_Init (RTC_Type * *base*, const rtc_config_t * *config*)

This function issues a software reset if the timer invalid flag is set.

Note

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

Parameters

base	RTC peripheral base address
config	Pointer to the user's RTC configuration structure.

35.6.2 static void RTC_Deinit (RTC_Type ∗ *base*) [inline], [static]

Parameters

base	RTC peripheral base address

35.6.3 void RTC_GetDefaultConfig (rtc_config_t ∗ *config*)

The default values are as follows.

```
*    config->wakeupSelect = false;
*    config->updateMode = false;
*    config->supervisorAccess = false;
*    config->compensationInterval = 0;
*    config->compensationTime = 0;
*
```

Parameters

config	Pointer to the user's RTC configuration structure.

35.6.4 status_t RTC_SetDatetime (RTC_Type ∗ *base,* const rtc_datetime_t ∗ *datetime*)

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

Parameters

base	RTC peripheral base address

datetime	Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

35.6.5 void RTC_GetDatetime (RTC_Type * *base,* rtc_datetime_t * *datetime*)

Parameters

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

35.6.6 status_t RTC_SetAlarm (RTC_Type * *base,* const rtc_datetime_t * *alarmTime*)

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

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

Returns

kStatus_Success: success in setting the RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

35.6.7 void RTC_GetAlarm (RTC_Type * *base,* rtc_datetime_t * *datetime*)

Parameters

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

35.6.8 void RTC_EnableInterrupts (RTC_Type ∗ *base,* uint32_t *mask*)

Parameters

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

35.6.9 void RTC_DisableInterrupts (RTC_Type ∗ *base,* uint32_t *mask*)

Parameters

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

35.6.10 uint32_t RTC_GetEnabledInterrupts (RTC_Type ∗ *base*)

Parameters

base	RTC peripheral base address

Returns

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

35.6.11 uint32_t RTC_GetStatusFlags (RTC_Type ∗ *base*)

Parameters

base	RTC peripheral base address

Returns

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

35.6.12 void RTC_ClearStatusFlags (RTC_Type * *base*, uint32_t *mask*)

Parameters

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

35.6.13 static void RTC_SetClockSource (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Note

After setting this bit, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

35.6.14 static uint32_t RTC_GetTamperTimeSeconds (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

35.6.15 static void RTC_StartTimer (RTC_Type * *base*) [inline], [static]

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

Parameters

base	RTC peripheral base address

35.6.16 static void RTC_StopTimer (RTC_Type * *base*) [inline], [static]

RTC's seconds register can be written to only when the timer is stopped.

Parameters

base	RTC peripheral base address

35.6.17 static void RTC_SetOscCapLoad (RTC_Type * *base,* uint32_t *capLoad*) [inline], [static]

Parameters

base	RTC peripheral base address
capLoad	Oscillator loads to enable. This is a logical OR of members of the enumeration rtc_osc_cap_load_t

35.6.18 static void RTC_Reset (RTC_Type * *base*) [inline], [static]

This resets all RTC registers except for the SWR bit and the RTC_WAR and RTC_RAR registers. The SWR bit is cleared by software explicitly clearing it.

Parameters

base	RTC peripheral base address

35.6.19 void RTC_GetMonotonicCounter (RTC_Type * *base,* uint64_t * *counter*)

Parameters

base	RTC peripheral base address
counter	Pointer to variable where the value is stored.

35.6.20 void RTC_SetMonotonicCounter (RTC_Type * *base,* uint64_t *counter*)

The Monotonic Overflow Flag in RTC_SR is cleared due to the API.

Parameters

base	RTC peripheral base address
counter	Counter value

35.6.21 status_t RTC_IncrementMonotonicCounter (RTC_Type * *base*)

Increments the Monotonic Counter (registers RTC_MCLR and RTC_MCHR accordingly) by setting the monotonic counter enable (MER[MCE]) and then writing to the RTC_MCLR register. A write to the monotonic counter low that causes it to overflow also increments the monotonic counter high.

Parameters

base	RTC peripheral base address

Returns

kStatus_Success: success kStatus_Fail: error occurred, either time invalid or monotonic overflow flag was found

35.6.22 static void RTC_EnableWakeUpPin (RTC_Type * *base*, bool *enable*) [inline], [static]

This function enable or disable RTC Wakeup Pin. The wakeup pin is optional and not available on all devices.

Parameters

base	RTC_Type base pointer.
enable	true to enable, false to disable.

Chapter 36
SAI: Serial Audio Interface

Overview

The MCUXpresso SDK provides a peripheral driver for the Serial Audio Interface (SAI) module of MCUXpresso SDK devices.

SAI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for SAI initialization, configuration and operation, and for optimization and customization purposes. Using the functional API requires the knowledge of the SAI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SAI functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the sai_handle_t as the first parameter. Initialize the handle by calling the SAI_TransferTxCreateHandle() or SAI_TransferRxCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions SAI_TransferSendNonBlocking() and SAI_TransferReceiveNonBlocking() set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_SAI_TxIdle and kStatus_SAI_RxIdle status.

Typical configurations

Bit width configuration

SAI driver support 8/16/24/32bits stereo/mono raw audio data transfer. SAI EDMA driver support 8/16/32bits stereo/mono raw audio data transfer, since the EDMA doesn't support 24bit data width, so application should pre-convert the 24bit data to 32bit. SAI DMA driver support 8/16/32bits stereo/mono raw audio data transfer, since the EDMA doesn't support 24bit data width, so application should pre-convert the 24bit data to 32bit. SAI SDMA driver support 8/16/24/32bits stereo/mono raw audio data transfer.

Frame configuration

SAI driver support I2S, DSP, Left justified, Right justified, TDM mode. Application can call the api directly: SAI_GetClassicI2SConfig SAI_GetLeftJustifiedConfig SAI_GetRightJustifiedConfig SAI_GetTDMConfig SAI_GetDSPConfig

Typical use case

36.3.1 SAI Send/receive using an interrupt method

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

36.3.2 SAI Send/receive using a DMA method

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

Modules

- SAI Driver

SAI Driver

36.4.1 Overview

Data Structures

- struct sai_config_t
  SAI user configuration structure. More...
- struct sai_transfer_format_t
  sai transfer format More...
- struct sai_master_clock_t
  master clock configurations More...
- struct sai_fifo_t
  sai fifo configurations More...
- struct sai_bit_clock_t
  sai bit clock configurations More...
- struct sai_frame_sync_t
  sai frame sync configurations More...
- struct sai_serial_data_t
  sai serial data configurations More...
- struct sai_transceiver_t
  sai transceiver configurations More...
- struct sai_transfer_t
  SAI transfer structure. More...
- struct sai_handle_t
  SAI handle structure. More...

Macros

- #define SAI_XFER_QUEUE_SIZE (4U)
  SAI transfer queue size, user can refine it according to use case.
- #define FSL_SAI_HAS_FIFO_EXTEND_FEATURE 1
  sai fifo feature

Typedefs

- typedef void(∗ sai_transfer_callback_t)(I2S_Type ∗base, sai_handle_t ∗handle, status_t status, void ∗userData)
  SAI transfer callback prototype.

Enumerations

- enum {
  kStatus_SAI_TxBusy = MAKE_STATUS(kStatusGroup_SAI, 0),
  kStatus_SAI_RxBusy = MAKE_STATUS(kStatusGroup_SAI, 1),
  kStatus_SAI_TxError = MAKE_STATUS(kStatusGroup_SAI, 2),
  kStatus_SAI_RxError = MAKE_STATUS(kStatusGroup_SAI, 3),
  kStatus_SAI_QueueFull = MAKE_STATUS(kStatusGroup_SAI, 4),
  kStatus_SAI_TxIdle = MAKE_STATUS(kStatusGroup_SAI, 5),
  kStatus_SAI_RxIdle = MAKE_STATUS(kStatusGroup_SAI, 6) }
  _sai_status_t, SAI return status.
- enum {
  kSAI_Channel0Mask = 1 << 0U,
  kSAI_Channel1Mask = 1 << 1U,
  kSAI_Channel2Mask = 1 << 2U,
  kSAI_Channel3Mask = 1 << 3U,
  kSAI_Channel4Mask = 1 << 4U,
  kSAI_Channel5Mask = 1 << 5U,
  kSAI_Channel6Mask = 1 << 6U,
  kSAI_Channel7Mask = 1 << 7U }
  _sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific
- enum sai_protocol_t {
  kSAI_BusLeftJustified = 0x0U,
  kSAI_BusRightJustified,
  kSAI_BusI2S,
  kSAI_BusPCMA,
  kSAI_BusPCMB }
  Define the SAI bus type.
- enum sai_master_slave_t {
  kSAI_Master = 0x0U,
  kSAI_Slave = 0x1U,
  kSAI_Bclk_Master_FrameSync_Slave = 0x2U,
  kSAI_Bclk_Slave_FrameSync_Master = 0x3U }
  Master or slave mode.
- enum sai_mono_stereo_t {
  kSAI_Stereo = 0x0U,
  kSAI_MonoRight,
  kSAI_MonoLeft }
  Mono or stereo audio format.
- enum sai_data_order_t {
  kSAI_DataLSB = 0x0U,
  kSAI_DataMSB }
  SAI data order, MSB or LSB.
- enum sai_clock_polarity_t {

kSAI_PolarityActiveHigh = 0x0U,
kSAI_PolarityActiveLow = 0x1U,
kSAI_SampleOnFallingEdge = 0x0U,
kSAI_SampleOnRisingEdge = 0x1U }
SAI clock polarity, active high or low.
- enum sai_sync_mode_t {
kSAI_ModeAsync = 0x0U,
kSAI_ModeSync }
Synchronous or asynchronous mode.
- enum sai_mclk_source_t {
kSAI_MclkSourceSysclk = 0x0U,
kSAI_MclkSourceSelect1,
kSAI_MclkSourceSelect2,
kSAI_MclkSourceSelect3 }
Mater clock source.
- enum sai_bclk_source_t {
kSAI_BclkSourceBusclk = 0x0U,
kSAI_BclkSourceMclkOption1 = 0x1U,
kSAI_BclkSourceMclkOption2 = 0x2U,
kSAI_BclkSourceMclkOption3 = 0x3U,
kSAI_BclkSourceMclkDiv = 0x1U,
kSAI_BclkSourceOtherSai0 = 0x2U,
kSAI_BclkSourceOtherSai1 = 0x3U }
Bit clock source.
- enum {
kSAI_WordStartInterruptEnable,
kSAI_SyncErrorInterruptEnable = I2S_TCSR_SEIE_MASK,
kSAI_FIFOWarningInterruptEnable = I2S_TCSR_FWIE_MASK,
kSAI_FIFOErrorInterruptEnable = I2S_TCSR_FEIE_MASK,
kSAI_FIFORequestInterruptEnable = I2S_TCSR_FRIE_MASK }
_sai_interrupt_enable_t, The SAI interrupt enable flag
- enum {
kSAI_FIFOWarningDMAEnable = I2S_TCSR_FWDE_MASK,
kSAI_FIFORequestDMAEnable = I2S_TCSR_FRDE_MASK }
_sai_dma_enable_t, The DMA request sources
- enum {
kSAI_WordStartFlag = I2S_TCSR_WSF_MASK,
kSAI_SyncErrorFlag = I2S_TCSR_SEF_MASK,
kSAI_FIFOErrorFlag = I2S_TCSR_FEF_MASK,
kSAI_FIFORequestFlag = I2S_TCSR_FRF_MASK,
kSAI_FIFOWarningFlag = I2S_TCSR_FWF_MASK }
_sai_flags, The SAI status flag
- enum sai_reset_type_t {
kSAI_ResetTypeSoftware = I2S_TCSR_SR_MASK,
kSAI_ResetTypeFIFO = I2S_TCSR_FR_MASK,
kSAI_ResetAll = I2S_TCSR_SR_MASK | I2S_TCSR_FR_MASK }

The reset type.

- enum sai_fifo_packing_t {
kSAI_FifoPackingDisabled = 0x0U,
kSAI_FifoPacking8bit = 0x2U,
kSAI_FifoPacking16bit = 0x3U }
The SAI packing mode The mode includes 8 bit and 16 bit packing.
- enum sai_sample_rate_t {
kSAI_SampleRate8KHz = 8000U,
kSAI_SampleRate11025Hz = 11025U,
kSAI_SampleRate12KHz = 12000U,
kSAI_SampleRate16KHz = 16000U,
kSAI_SampleRate22050Hz = 22050U,
kSAI_SampleRate24KHz = 24000U,
kSAI_SampleRate32KHz = 32000U,
kSAI_SampleRate44100Hz = 44100U,
kSAI_SampleRate48KHz = 48000U,
kSAI_SampleRate96KHz = 96000U,
kSAI_SampleRate192KHz = 192000U,
kSAI_SampleRate384KHz = 384000U }
Audio sample rate.
- enum sai_word_width_t {
kSAI_WordWidth8bits = 8U,
kSAI_WordWidth16bits = 16U,
kSAI_WordWidth24bits = 24U,
kSAI_WordWidth32bits = 32U }
Audio word width.
- enum sai_transceiver_type_t {
kSAI_Transmitter = 0U,
kSAI_Receiver = 1U }
sai transceiver type
- enum sai_frame_sync_len_t {
kSAI_FrameSyncLenOneBitClk = 0U,
kSAI_FrameSyncLenPerWordWidth = 1U }
sai frame sync len

Driver version

- #define FSL_SAI_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))
Version 2.3.2.

Initialization and deinitialization

- void SAI_TxInit (I2S_Type ∗base, const sai_config_t ∗config)
Initializes the SAI Tx peripheral.
- void SAI_RxInit (I2S_Type ∗base, const sai_config_t ∗config)

Initializes the SAI Rx peripheral.
- void SAI_TxGetDefaultConfig (sai_config_t *config)
  Sets the SAI Tx configuration structure to default values.
- void SAI_RxGetDefaultConfig (sai_config_t *config)
  Sets the SAI Rx configuration structure to default values.
- void SAI_Init (I2S_Type *base)
  Initializes the SAI peripheral.
- void SAI_Deinit (I2S_Type *base)
  De-initializes the SAI peripheral.
- void SAI_TxReset (I2S_Type *base)
  Resets the SAI Tx.
- void SAI_RxReset (I2S_Type *base)
  Resets the SAI Rx.
- void SAI_TxEnable (I2S_Type *base, bool enable)
  Enables/disables the SAI Tx.
- void SAI_RxEnable (I2S_Type *base, bool enable)
  Enables/disables the SAI Rx.
- static void SAI_TxSetBitClockDirection (I2S_Type *base, sai_master_slave_t masterSlave)
  Set Rx bit clock direction.
- static void SAI_RxSetBitClockDirection (I2S_Type *base, sai_master_slave_t masterSlave)
  Set Rx bit clock direction.
- static void SAI_RxSetFrameSyncDirection (I2S_Type *base, sai_master_slave_t masterSlave)
  Set Rx frame sync direction.
- static void SAI_TxSetFrameSyncDirection (I2S_Type *base, sai_master_slave_t masterSlave)
  Set Tx frame sync direction.
- void SAI_TxSetBitClockRate (I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)
  Transmitter bit clock rate configurations.
- void SAI_RxSetBitClockRate (I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)
  Receiver bit clock rate configurations.
- void SAI_TxSetBitclockConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t *config)
  Transmitter Bit clock configurations.
- void SAI_RxSetBitclockConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t *config)
  Receiver Bit clock configurations.
- void SAI_SetMasterClockConfig (I2S_Type *base, sai_master_clock_t *config)
  Master clock configurations.
- void SAI_TxSetFifoConfig (I2S_Type *base, sai_fifo_t *config)
  SAI transmitter fifo configurations.
- void SAI_RxSetFifoConfig (I2S_Type *base, sai_fifo_t *config)
  SAI receiver fifo configurations.
- void SAI_TxSetFrameSyncConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)
  SAI transmitter Frame sync configurations.
- void SAI_RxSetFrameSyncConfig (I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)
  SAI receiver Frame sync configurations.
- void SAI_TxSetSerialDataConfig (I2S_Type *base, sai_serial_data_t *config)

SAI transmitter Serial data configurations.
- void SAI_RxSetSerialDataConfig (I2S_Type ∗base, sai_serial_data_t ∗config)
  SAI receiver Serial data configurations.
- void SAI_TxSetConfig (I2S_Type ∗base, sai_transceiver_t ∗config)
  SAI transmitter configurations.
- void SAI_RxSetConfig (I2S_Type ∗base, sai_transceiver_t ∗config)
  SAI receiver configurations.
- void SAI_GetClassicI2SConfig (sai_transceiver_t ∗config, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask)
  Get classic I2S mode configurations.
- void SAI_GetLeftJustifiedConfig (sai_transceiver_t ∗config, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask)
  Get left justified mode configurations.
- void SAI_GetRightJustifiedConfig (sai_transceiver_t ∗config, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask)
  Get right justified mode configurations.
- void SAI_GetTDMConfig (sai_transceiver_t ∗config, sai_frame_sync_len_t frameSyncWidth, sai_word_width_t bitWidth, uint32_t dataWordNum, uint32_t saiChannelMask)
  Get TDM mode configurations.
- void SAI_GetDSPConfig (sai_transceiver_t ∗config, sai_frame_sync_len_t frameSyncWidth, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask)
  Get DSP mode configurations.

Status

- static uint32_t SAI_TxGetStatusFlag (I2S_Type ∗base)
  Gets the SAI Tx status flag state.
- static void SAI_TxClearStatusFlags (I2S_Type ∗base, uint32_t mask)
  Clears the SAI Tx status flag state.
- static uint32_t SAI_RxGetStatusFlag (I2S_Type ∗base)
  Gets the SAI Tx status flag state.
- static void SAI_RxClearStatusFlags (I2S_Type ∗base, uint32_t mask)
  Clears the SAI Rx status flag state.
- void SAI_TxSoftwareReset (I2S_Type ∗base, sai_reset_type_t type)
  Do software reset or FIFO reset .
- void SAI_RxSoftwareReset (I2S_Type ∗base, sai_reset_type_t type)
  Do software reset or FIFO reset .
- void SAI_TxSetChannelFIFOMask (I2S_Type ∗base, uint8_t mask)
  Set the Tx channel FIFO enable mask.
- void SAI_RxSetChannelFIFOMask (I2S_Type ∗base, uint8_t mask)
  Set the Rx channel FIFO enable mask.
- void SAI_TxSetDataOrder (I2S_Type ∗base, sai_data_order_t order)
  Set the Tx data order.
- void SAI_RxSetDataOrder (I2S_Type ∗base, sai_data_order_t order)
  Set the Rx data order.
- void SAI_TxSetBitClockPolarity (I2S_Type ∗base, sai_clock_polarity_t polarity)
  Set the Tx data order.
- void SAI_RxSetBitClockPolarity (I2S_Type ∗base, sai_clock_polarity_t polarity)
  Set the Rx data order.

- void SAI_TxSetFrameSyncPolarity (I2S_Type *base, sai_clock_polarity_t polarity)
  Set the Tx data order.
- void SAI_RxSetFrameSyncPolarity (I2S_Type *base, sai_clock_polarity_t polarity)
  Set the Rx data order.
- void SAI_TxSetFIFOPacking (I2S_Type *base, sai_fifo_packing_t pack)
  Set Tx FIFO packing feature.
- void SAI_RxSetFIFOPacking (I2S_Type *base, sai_fifo_packing_t pack)
  Set Rx FIFO packing feature.
- static void SAI_TxSetFIFOErrorContinue (I2S_Type *base, bool isEnabled)
  Set Tx FIFO error continue.
- static void SAI_RxSetFIFOErrorContinue (I2S_Type *base, bool isEnabled)
  Set Rx FIFO error continue.

Interrupts

- static void SAI_TxEnableInterrupts (I2S_Type *base, uint32_t mask)
  Enables the SAI Tx interrupt requests.
- static void SAI_RxEnableInterrupts (I2S_Type *base, uint32_t mask)
  Enables the SAI Rx interrupt requests.
- static void SAI_TxDisableInterrupts (I2S_Type *base, uint32_t mask)
  Disables the SAI Tx interrupt requests.
- static void SAI_RxDisableInterrupts (I2S_Type *base, uint32_t mask)
  Disables the SAI Rx interrupt requests.

DMA Control

- static void SAI_TxEnableDMA (I2S_Type *base, uint32_t mask, bool enable)
  Enables/disables the SAI Tx DMA requests.
- static void SAI_RxEnableDMA (I2S_Type *base, uint32_t mask, bool enable)
  Enables/disables the SAI Rx DMA requests.
- static uint32_t SAI_TxGetDataRegisterAddress (I2S_Type *base, uint32_t channel)
  Gets the SAI Tx data register address.
- static uint32_t SAI_RxGetDataRegisterAddress (I2S_Type *base, uint32_t channel)
  Gets the SAI Rx data register address.

Bus Operations

- void SAI_TxSetFormat (I2S_Type *base, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  Configures the SAI Tx audio format.
- void SAI_RxSetFormat (I2S_Type *base, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  Configures the SAI Rx audio format.
- void SAI_WriteBlocking (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
  Sends data using a blocking method.

- void SAI_WriteMultiChannelBlocking (I2S_Type ∗base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t ∗buffer, uint32_t size)
  Sends data to multi channel using a blocking method.
- static void SAI_WriteData (I2S_Type ∗base, uint32_t channel, uint32_t data)
  Writes data into SAI FIFO.
- void SAI_ReadBlocking (I2S_Type ∗base, uint32_t channel, uint32_t bitWidth, uint8_t ∗buffer, uint32_t size)
  Receives data using a blocking method.
- void SAI_ReadMultiChannelBlocking (I2S_Type ∗base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t ∗buffer, uint32_t size)
  Receives multi channel data using a blocking method.
- static uint32_t SAI_ReadData (I2S_Type ∗base, uint32_t channel)
  Reads data from the SAI FIFO.

Transactional

- void SAI_TransferTxCreateHandle (I2S_Type ∗base, sai_handle_t ∗handle, sai_transfer_callback_t callback, void ∗userData)
  Initializes the SAI Tx handle.
- void SAI_TransferRxCreateHandle (I2S_Type ∗base, sai_handle_t ∗handle, sai_transfer_callback_t callback, void ∗userData)
  Initializes the SAI Rx handle.
- void SAI_TransferTxSetConfig (I2S_Type ∗base, sai_handle_t ∗handle, sai_transceiver_t ∗config)
  SAI transmitter transfer configurations.
- void SAI_TransferRxSetConfig (I2S_Type ∗base, sai_handle_t ∗handle, sai_transceiver_t ∗config)
  SAI receiver transfer configurations.
- status_t SAI_TransferTxSetFormat (I2S_Type ∗base, sai_handle_t ∗handle, sai_transfer_format_t ∗format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  Configures the SAI Tx audio format.
- status_t SAI_TransferRxSetFormat (I2S_Type ∗base, sai_handle_t ∗handle, sai_transfer_format_t ∗format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  Configures the SAI Rx audio format.
- status_t SAI_TransferSendNonBlocking (I2S_Type ∗base, sai_handle_t ∗handle, sai_transfer_t ∗xfer)
  Performs an interrupt non-blocking send transfer on SAI.
- status_t SAI_TransferReceiveNonBlocking (I2S_Type ∗base, sai_handle_t ∗handle, sai_transfer_t ∗xfer)
  Performs an interrupt non-blocking receive transfer on SAI.
- status_t SAI_TransferGetSendCount (I2S_Type ∗base, sai_handle_t ∗handle, size_t ∗count)
  Gets a set byte count.
- status_t SAI_TransferGetReceiveCount (I2S_Type ∗base, sai_handle_t ∗handle, size_t ∗count)
  Gets a received byte count.
- void SAI_TransferAbortSend (I2S_Type ∗base, sai_handle_t ∗handle)
  Aborts the current send.
- void SAI_TransferAbortReceive (I2S_Type ∗base, sai_handle_t ∗handle)
  Aborts the current IRQ receive.
- void SAI_TransferTerminateSend (I2S_Type ∗base, sai_handle_t ∗handle)
  Terminate all SAI send.
- void SAI_TransferTerminateReceive (I2S_Type ∗base, sai_handle_t ∗handle)

Terminate all SAI receive.
- void SAI_TransferTxHandleIRQ (I2S_Type ∗base, sai_handle_t ∗handle)
  Tx interrupt handler.
- void SAI_TransferRxHandleIRQ (I2S_Type ∗base, sai_handle_t ∗handle)
  Tx interrupt handler.

36.4.2 Data Structure Documentation

36.4.2.1 struct sai_config_t

Data Fields

- sai_protocol_t protocol
  Audio bus protocol in SAI.
- sai_sync_mode_t syncMode
  SAI sync mode, control Tx/Rx clock sync.
- bool mclkOutputEnable
  Master clock output enable, true means master clock divider enabled.
- sai_mclk_source_t mclkSource
  Master Clock source.
- sai_bclk_source_t bclkSource
  Bit Clock source.
- sai_master_slave_t masterSlave
  Master or slave.

36.4.2.2 struct sai_transfer_format_t

Data Fields

- uint32_t sampleRate_Hz
  Sample rate of audio data.
- uint32_t bitWidth
  Data length of audio data, usually 8/16/24/32 bits.
- sai_mono_stereo_t stereo
  Mono or stereo.
- uint32_t masterClockHz
  Master clock frequency in Hz.
- uint8_t watermark
  Watermark value.
- uint8_t channel
  Transfer start channel.
- uint8_t channelMask
  enabled channel mask value, reference _sai_channel_mask
- uint8_t endChannel
  end channel number
- uint8_t channelNums
  Total enabled channel numbers.
- sai_protocol_t protocol

Which audio protocol used.
- bool isFrameSyncCompact
  True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

36.4.2.2.0.27 Field Documentation

36.4.2.2.0.27.1 bool sai_transfer_format_t::isFrameSyncCompact

36.4.2.3 struct sai_master_clock_t

Data Fields

- bool mclkOutputEnable
  master clock output enable
- sai_mclk_source_t mclkSource
  Master Clock source.
- uint32_t mclkHz
  target mclk frequency
- uint32_t mclkSourceClkHz
  mclk source frequency

36.4.2.4 struct sai_fifo_t

Data Fields

- bool fifoContinueOneError
  fifo continues when error occur
- sai_fifo_packing_t fifoPacking
  fifo packing mode
- uint8_t fifoWatermark
  fifo watermark

36.4.2.5 struct sai_bit_clock_t

Data Fields

- bool bclkSrcSwap
  bit clock source swap
- bool bclkInputDelay
  bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .
- sai_clock_polarity_t bclkPolarity
  bit clock polarity
- sai_bclk_source_t bclkSource
  bit Clock source

36.4.2.5.0.28 Field Documentation

36.4.2.5.0.28.1 bool sai_bit_clock_t::bclkInputDelay

36.4.2.6 struct sai_frame_sync_t

Data Fields

- uint8_t frameSyncWidth
  frame sync width in number of bit clocks
- bool frameSyncEarly
  TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame.
- sai_clock_polarity_t frameSyncPolarity
  frame sync polarity

36.4.2.7 struct sai_serial_data_t

Data Fields

- sai_data_order_t dataOrder
  configure whether the LSB or MSB is transmitted first
- uint8_t dataWord0Length
  configure the number of bits in the first word in each frame
- uint8_t dataWordNLength
  configure the number of bits in the each word in each frame, except the first word
- uint8_t dataWordLength
  used to record the data length for dma transfer
- uint8_t dataFirstBitShifted
  Configure the bit index for the first bit transmitted for each word in the frame.
- uint8_t dataWordNum
  configure the number of words in each frame
- uint32_t dataMaskedWord
  configure whether the transmit word is masked

36.4.2.8 struct sai_transceiver_t

Data Fields

- sai_serial_data_t serialData
  serial data configurations
- sai_frame_sync_t frameSync
  ws configurations
- sai_bit_clock_t bitClock
  bit clock configurations
- sai_fifo_t fifo
  fifo configurations
- sai_master_slave_t masterSlave
  transceiver is master or slave

- sai_sync_mode_t syncMode
  transceiver sync mode
- uint8_t startChannel
  Transfer start channel.
- uint8_t channelMask
  enabled channel mask value, reference _sai_channel_mask
- uint8_t endChannel
  end channel number
- uint8_t channelNums
  Total enabled channel numbers.

36.4.2.9 struct sai_transfer_t

Data Fields

- uint8_t ∗ data
  Data start address to transfer.
- size_t dataSize
  Transfer size.

36.4.2.9.0.29 Field Documentation

36.4.2.9.0.29.1 uint8_t∗ sai_transfer_t::data

36.4.2.9.0.29.2 size_t sai_transfer_t::dataSize

36.4.2.10 struct _sai_handle

Data Fields

- I2S_Type ∗ base
  base address
- uint32_t state
  Transfer status.
- sai_transfer_callback_t callback
  Callback function called at transfer event.
- void ∗ userData
  Callback parameter passed to callback function.
- uint8_t bitWidth
  Bit width for transfer, 8/16/24/32 bits.
- uint8_t channel
  Transfer start channel.
- uint8_t channelMask
  enabled channel mask value, refernece _sai_channel_mask
- uint8_t endChannel
  end channel number
- uint8_t channelNums
  Total enabled channel numbers.
- sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]
  Transfer queue storing queued transfer.

- size_t transferSize [SAI_XFER_QUEUE_SIZE]
  Data bytes need to transfer.
- volatile uint8_t queueUser
  Index for user to queue transfer.
- volatile uint8_t queueDriver
  Index for driver to get the transfer data and size.
- uint8_t watermark
  Watermark value.

36.4.3 Macro Definition Documentation

36.4.3.1 #define SAI_XFER_QUEUE_SIZE (4U)

36.4.4 Enumeration Type Documentation

36.4.4.1 anonymous enum

Enumerator

kStatus_SAI_TxBusy SAI Tx is busy.
kStatus_SAI_RxBusy SAI Rx is busy.
kStatus_SAI_TxError SAI Tx FIFO error.
kStatus_SAI_RxError SAI Rx FIFO error.
kStatus_SAI_QueueFull SAI transfer queue is full.
kStatus_SAI_TxIdle SAI Tx is idle.
kStatus_SAI_RxIdle SAI Rx is idle.

36.4.4.2 anonymous enum

Enumerator

kSAI_Channel0Mask channel 0 mask value
kSAI_Channel1Mask channel 1 mask value
kSAI_Channel2Mask channel 2 mask value
kSAI_Channel3Mask channel 3 mask value
kSAI_Channel4Mask channel 4 mask value
kSAI_Channel5Mask channel 5 mask value
kSAI_Channel6Mask channel 6 mask value
kSAI_Channel7Mask channel 7 mask value

36.4.4.3 enum sai_protocol_t

Enumerator

kSAI_BusLeftJustified Uses left justified format.

kSAI_BusRightJustified Uses right justified format.
kSAI_BusI2S Uses I2S format.
kSAI_BusPCMA Uses I2S PCM A format.
kSAI_BusPCMB Uses I2S PCM B format.

36.4.4.4 enum sai_master_slave_t

Enumerator

kSAI_Master Master mode include bclk and frame sync.
kSAI_Slave Slave mode include bclk and frame sync.
kSAI_Bclk_Master_FrameSync_Slave bclk in master mode, frame sync in slave mode
kSAI_Bclk_Slave_FrameSync_Master bclk in slave mode, frame sync in master mode

36.4.4.5 enum sai_mono_stereo_t

Enumerator

kSAI_Stereo Stereo sound.
kSAI_MonoRight Only Right channel have sound.
kSAI_MonoLeft Only left channel have sound.

36.4.4.6 enum sai_data_order_t

Enumerator

kSAI_DataLSB LSB bit transferred first.
kSAI_DataMSB MSB bit transferred first.

36.4.4.7 enum sai_clock_polarity_t

Enumerator

kSAI_PolarityActiveHigh Drive outputs on rising edge.
kSAI_PolarityActiveLow Drive outputs on falling edge.
kSAI_SampleOnFallingEdge Sample inputs on falling edge.
kSAI_SampleOnRisingEdge Sample inputs on rising edge.

36.4.4.8 enum sai_sync_mode_t

Enumerator

kSAI_ModeAsync Asynchronous mode.
kSAI_ModeSync Synchronous mode (with receiver or transmit)

36.4.4.9 enum sai_mclk_source_t

Enumerator

kSAI_MclkSourceSysclk Master clock from the system clock.
kSAI_MclkSourceSelect1 Master clock from source 1.
kSAI_MclkSourceSelect2 Master clock from source 2.
kSAI_MclkSourceSelect3 Master clock from source 3.

36.4.4.10 enum sai_bclk_source_t

Enumerator

kSAI_BclkSourceBusclk Bit clock using bus clock.
kSAI_BclkSourceMclkOption1 Bit clock MCLK option 1.
kSAI_BclkSourceMclkOption2 Bit clock MCLK option2.
kSAI_BclkSourceMclkOption3 Bit clock MCLK option3.
kSAI_BclkSourceMclkDiv Bit clock using master clock divider.
kSAI_BclkSourceOtherSai0 Bit clock from other SAI device.
kSAI_BclkSourceOtherSai1 Bit clock from other SAI device.

36.4.4.11 anonymous enum

Enumerator

kSAI_WordStartInterruptEnable Word start flag, means the first word in a frame detected.
kSAI_SyncErrorInterruptEnable Sync error flag, means the sync error is detected.
kSAI_FIFOWarningInterruptEnable FIFO warning flag, means the FIFO is empty.
kSAI_FIFOErrorInterruptEnable FIFO error flag.
kSAI_FIFORequestInterruptEnable FIFO request, means reached watermark.

36.4.4.12 anonymous enum

Enumerator

kSAI_FIFOWarningDMAEnable FIFO warning caused by the DMA request.
kSAI_FIFORequestDMAEnable FIFO request caused by the DMA request.

36.4.4.13 anonymous enum

Enumerator

kSAI_WordStartFlag Word start flag, means the first word in a frame detected.
kSAI_SyncErrorFlag Sync error flag, means the sync error is detected.
kSAI_FIFOErrorFlag FIFO error flag.
kSAI_FIFORequestFlag FIFO request flag.
kSAI_FIFOWarningFlag FIFO warning flag.

36.4.4.14 enum sai_reset_type_t

Enumerator

kSAI_ResetTypeSoftware Software reset, reset the logic state.
kSAI_ResetTypeFIFO FIFO reset, reset the FIFO read and write pointer.
kSAI_ResetAll All reset.

36.4.4.15 enum sai_fifo_packing_t

Enumerator

kSAI_FifoPackingDisabled Packing disabled.
kSAI_FifoPacking8bit 8 bit packing enabled
kSAI_FifoPacking16bit 16bit packing enabled

36.4.4.16 enum sai_sample_rate_t

Enumerator

kSAI_SampleRate8KHz Sample rate 8000 Hz.
kSAI_SampleRate11025Hz Sample rate 11025 Hz.
kSAI_SampleRate12KHz Sample rate 12000 Hz.
kSAI_SampleRate16KHz Sample rate 16000 Hz.
kSAI_SampleRate22050Hz Sample rate 22050 Hz.
kSAI_SampleRate24KHz Sample rate 24000 Hz.
kSAI_SampleRate32KHz Sample rate 32000 Hz.
kSAI_SampleRate44100Hz Sample rate 44100 Hz.
kSAI_SampleRate48KHz Sample rate 48000 Hz.
kSAI_SampleRate96KHz Sample rate 96000 Hz.
kSAI_SampleRate192KHz Sample rate 192000 Hz.
kSAI_SampleRate384KHz Sample rate 384000 Hz.

36.4.4.17 enum sai_word_width_t

Enumerator

kSAI_WordWidth8bits Audio data width 8 bits.
kSAI_WordWidth16bits Audio data width 16 bits.
kSAI_WordWidth24bits Audio data width 24 bits.
kSAI_WordWidth32bits Audio data width 32 bits.

36.4.4.18 enum sai_transceiver_type_t

Enumerator

kSAI_Transmitter sai transmitter
kSAI_Receiver sai receiver

36.4.4.19 enum sai_frame_sync_len_t

Enumerator

kSAI_FrameSyncLenOneBitClk 1 bit clock frame sync len for DSP mode
kSAI_FrameSyncLenPerWordWidth Frame sync length decided by word width.

36.4.5 Function Documentation

36.4.5.1 void SAI_TxInit (I2S_Type ∗ *base*, const sai_config_t ∗ *config*)

Deprecated Do not use this function. It has been superceded by SAI_Init

Ungates the SAI clock, resets the module, and configures SAI Tx with a configuration structure. The configuration structure can be custom filled or set with default values by SAI_TxGetDefaultConfig().

Note

> This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAIM module can cause a hard fault because the clock is not enabled.

Parameters

base	SAI base pointer
config	SAI configuration structure.

36.4.5.2 void SAI_RxInit (I2S_Type * *base*, const sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by SAI_Init

Ungates the SAI clock, resets the module, and configures the SAI Rx with a configuration structure. The configuration structure can be custom filled or set with default values by SAI_RxGetDefaultConfig().

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAI module can cause a hard fault because the clock is not enabled.

Parameters

base	SAI base pointer
config	SAI configuration structure.

36.4.5.3 void SAI_TxGetDefaultConfig (sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by SAI_GetClassicI2SConfig, SAI_GetLeftJustifiedConfig , SAI_GetRightJustifiedConfig, SAI_GetDSPConfig, SAI_GetTDMConfig

This API initializes the configuration structure for use in SAI_TxConfig(). The initialized structure can remain unchanged in SAI_TxConfig(), or it can be modified before calling SAI_TxConfig(). This is an example.

```
sai_config_t config;
SAI_TxGetDefaultConfig(&config);
```

Parameters

config	pointer to master configuration structure

36.4.5.4 void SAI_RxGetDefaultConfig (sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by SAI_GetClassicI2SConfig, SAI_GetLeftJustifiedConfig , SAI_GetRightJustifiedConfig, SAI_GetDSPConfig, SAI_GetTDMConfig

This API initializes the configuration structure for use in SAI_RxConfig(). The initialized structure can remain unchanged in SAI_RxConfig() or it can be modified before calling SAI_RxConfig(). This is an example.

```
sai_config_t config;
SAI_RxGetDefaultConfig(&config);
```

Parameters

config	pointer to master configuration structure

36.4.5.5 void SAI_Init (I2S_Type * *base*)

This API gates the SAI clock. The SAI module can't operate unless SAI_Init is called to enable the clock.

Parameters

base	SAI base pointer.

36.4.5.6 void SAI_Deinit (I2S_Type * *base*)

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

base	SAI base pointer.

36.4.5.7 void SAI_TxReset (I2S_Type * *base*)

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

base	SAI base pointer

36.4.5.8 void SAI_RxReset (I2S_Type * *base*)

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

base	SAI base pointer

36.4.5.9 void SAI_TxEnable (I2S_Type * *base*, bool *enable*)

Parameters

base	SAI base pointer.
enable	True means enable SAI Tx, false means disable.

36.4.5.10 void SAI_RxEnable (I2S_Type * *base*, bool *enable*)

Parameters

base	SAI base pointer.
enable	True means enable SAI Rx, false means disable.

36.4.5.11 static void SAI_TxSetBitClockDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select bit clock direction, master or slave.

Parameters

base	SAI base pointer.
masterSlave	reference sai_master_slave_t.

36.4.5.12 static void SAI_RxSetBitClockDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select bit clock direction, master or slave.

Parameters

base	SAI base pointer.
masterSlave	reference sai_master_slave_t.

36.4.5.13 static void SAI_RxSetFrameSyncDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select frame sync direction, master or slave.

Parameters

base	SAI base pointer.
masterSlave	reference sai_master_slave_t.

36.4.5.14 static void SAI_TxSetFrameSyncDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select frame sync direction, master or slave.

Parameters

base	SAI base pointer.
masterSlave	reference sai_master_slave_t.

36.4.5.15 void SAI_TxSetBitClockRate (I2S_Type * *base*, uint32_t *sourceClockHz*, uint32_t *sampleRate*, uint32_t *bitWidth*, uint32_t *channelNumbers*)

Parameters

base	SAI base pointer.
sourceClockHz	Bit clock source frequency.
sampleRate	Audio data sample rate.
bitWidth	Audio data bitWidth.
channel-Numbers	Audio channel numbers.

36.4.5.16 void SAI_RxSetBitClockRate (I2S_Type * *base*, uint32_t *sourceClockHz*, uint32_t *sampleRate*, uint32_t *bitWidth*, uint32_t *channelNumbers*)

Parameters

base	SAI base pointer.
sourceClockHz	Bit clock source frequency.
sampleRate	Audio data sample rate.
bitWidth	Audio data bitWidth.
channel-Numbers	Audio channel numbers.

36.4.5.17 void SAI_TxSetBitclockConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_bit_clock_t * *config*)

Parameters

base	SAI base pointer.
masterSlave	master or slave.
config	bit clock other configurations, can be NULL in slave mode.

36.4.5.18 void SAI_RxSetBitclockConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_bit_clock_t * *config*)

Parameters

base	SAI base pointer.
masterSlave	master or slave.
config	bit clock other configurations, can be NULL in slave mode.

36.4.5.19 void SAI_SetMasterClockConfig (I2S_Type * *base*, sai_master_clock_t * *config*)

Parameters

base	SAI base pointer.
config	master clock configurations.

36.4.5.20 void SAI_TxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

base	SAI base pointer.
config	fifo configurations.

36.4.5.21 void SAI_RxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

base	SAI base pointer.
config	fifo configurations.

36.4.5.22 void SAI_TxSetFrameSyncConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_frame_sync_t * *config*)

Parameters

base	SAI base pointer.
masterSlave	master or slave.
config	frame sync configurations, can be NULL in slave mode.

36.4.5.23 void SAI_RxSetFrameSyncConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_frame_sync_t * *config*)

Parameters

base	SAI base pointer.
masterSlave	master or slave.
config	frame sync configurations, can be NULL in slave mode.

36.4.5.24 void SAI_TxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

base	SAI base pointer.
config	serial data configurations.

36.4.5.25 void SAI_RxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

base	SAI base pointer.
config	serial data configurations.

36.4.5.26 void SAI_TxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

base	SAI base pointer.
config	transmitter configurations.

36.4.5.27 void SAI_RxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

base	SAI base pointer.
config	receiver configurations.

36.4.5.28 void SAI_GetClassicI2SConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

config	transceiver configurations.
bitWidth	audio data bitWidth.
mode	audio data channel.
saiChannelMask	mask value of the channel to be enable.

36.4.5.29 void SAI_GetLeftJustifiedConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

config	transceiver configurations.
bitWidth	audio data bitWidth.
mode	audio data channel.
saiChannel-Mask	mask value of the channel to be enable.

36.4.5.30 void SAI_GetRightJustifiedConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

config	transceiver configurations.
bitWidth	audio data bitWidth.
mode	audio data channel.
saiChannel-Mask	mask value of the channel to be enable.

36.4.5.31 void SAI_GetTDMConfig (sai_transceiver_t * *config*, sai_frame_sync_len_t *frameSyncWidth*, sai_word_width_t *bitWidth*, uint32_t *dataWordNum*, uint32_t *saiChannelMask*)

Parameters

config	transceiver configurations.
frameSync-Width	length of frame sync.
bitWidth	audio data word width.
dataWordNum	word number in one frame.
saiChannel-Mask	mask value of the channel to be enable.

36.4.5.32 void SAI_GetDSPConfig (sai_transceiver_t * *config*, sai_frame_sync_len_t *frameSyncWidth*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Note

DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
      kSAI_Stereo, channelMask)
* config->frameSync.frameSyncEarly    = true;
* SAI_TxSetConfig(base, config)
*
```

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
      kSAI_Stereo, channelMask)
* SAI_TxSetConfig(base, config)
*
```

Parameters

config	transceiver configurations.
frameSync-Width	length of frame sync.
bitWidth	audio data bitWidth.
mode	audio data channel.
saiChannel-Mask	mask value of the channel to enable.

36.4.5.33 static uint32_t SAI_TxGetStatusFlag (I2S_Type * *base*) [inline], [static]

Parameters

base	SAI base pointer

Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

36.4.5.34 static void SAI_TxClearStatusFlags (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	State mask. It can be a combination of the following source if defined: • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

36.4.5.35 static uint32_t SAI_RxGetStatusFlag (I2S_Type * *base*) [inline], [static]

Parameters

base	SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

36.4.5.36 static void SAI_RxClearStatusFlags (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	State mask. It can be a combination of the following sources if defined. • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

36.4.5.37 void SAI_TxSoftwareReset (I2S_Type * *base*, sai_reset_type_t *type*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

base	SAI base pointer
type	Reset type, FIFO reset or software reset

36.4.5.38 void SAI_RxSoftwareReset (I2S_Type * *base*, sai_reset_type_t *type*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

base	SAI base pointer
type	Reset type, FIFO reset or software reset

36.4.5.39 void SAI_TxSetChannelFIFOMask (I2S_Type * *base*, uint8_t *mask*)

Parameters

base	SAI base pointer
mask	Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

36.4.5.40 void SAI_RxSetChannelFIFOMask (I2S_Type * *base*, uint8_t *mask*)

Parameters

base	SAI base pointer
mask	Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

36.4.5.41 void SAI_TxSetDataOrder (I2S_Type * *base*, sai_data_order_t *order*)

Parameters

base	SAI base pointer
order	Data order MSB or LSB

36.4.5.42 void SAI_RxSetDataOrder (I2S_Type * *base*, sai_data_order_t *order*)

Parameters

base	SAI base pointer
order	Data order MSB or LSB

36.4.5.43 void SAI_TxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
polarity	

36.4.5.44 void SAI_RxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
polarity	

36.4.5.45 void SAI_TxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
polarity	

36.4.5.46 void SAI_RxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
polarity	

36.4.5.47 void SAI_TxSetFIFOPacking (I2S_Type * *base*, sai_fifo_packing_t *pack*)

Parameters

base	SAI base pointer.
pack	FIFO pack type. It is element of sai_fifo_packing_t.

36.4.5.48 void SAI_RxSetFIFOPacking (I2S_Type * *base*, sai_fifo_packing_t *pack*)

Parameters

base	SAI base pointer.
pack	FIFO pack type. It is element of sai_fifo_packing_t.

36.4.5.49 static void SAI_TxSetFIFOErrorContinue (I2S_Type * *base*, bool *isEnabled*) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

base	SAI base pointer.
isEnabled	Is FIFO error continue enabled, true means enable, false means disable.

36.4.5.50 static void SAI_RxSetFIFOErrorContinue (I2S_Type * *base,* bool *isEnabled*) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

base	SAI base pointer.
isEnabled	Is FIFO error continue enabled, true means enable, false means disable.

36.4.5.51 static void SAI_TxEnableInterrupts (I2S_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

36.4.5.52 static void SAI_RxEnableInterrupts (I2S_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

36.4.5.53 static void SAI_TxDisableInterrupts (I2S_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

36.4.5.54 static void SAI_RxDisableInterrupts (I2S_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

36.4.5.55 static void SAI_TxEnableDMA (I2S_Type * *base,* uint32_t *mask,* bool *enable*) `[inline]`, `[static]`

Parameters

base	SAI base pointer
mask	DMA source The parameter can be combination of the following sources if defined. • kSAI_FIFOWarningDMAEnable • kSAI_FIFORequestDMAEnable
enable	True means enable DMA, false means disable DMA.

36.4.5.56 static void SAI_RxEnableDMA (I2S_Type * *base*, uint32_t *mask*, bool *enable*) [inline], [static]

Parameters

base	SAI base pointer
mask	DMA source The parameter can be a combination of the following sources if defined. • kSAI_FIFOWarningDMAEnable • kSAI_FIFORequestDMAEnable
enable	True means enable DMA, false means disable DMA.

36.4.5.57 static uint32_t SAI_TxGetDataRegisterAddress (I2S_Type * *base*, uint32_t *channel*) [inline], [static]

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

Parameters

base	SAI base pointer.
channel	Which data channel used.

Returns

data register address.

36.4.5.58 static uint32_t SAI_RxGetDataRegisterAddress (I2S_Type * *base*, uint32_t *channel*) [inline], [static]

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

Parameters

base	SAI base pointer.
channel	Which data channel used.

Returns

data register address.

36.4.5.59 void SAI_TxSetFormat (I2S_Type * *base*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by SAI_TxSetConfig

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

base	SAI base pointer.
format	Pointer to the SAI audio data format structure.
mclkSource-ClockHz	SAI master clock source frequency in Hz.
bclkSource-ClockHz	SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz.

36.4.5.60 void SAI_RxSetFormat (I2S_Type * *base*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by SAI_RxSetConfig

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

base	SAI base pointer.
format	Pointer to the SAI audio data format structure.
mclkSource-ClockHz	SAI master clock source frequency in Hz.
bclkSource-ClockHz	SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz.

36.4.5.61 void SAI_WriteBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

base	SAI base pointer.
channel	Data channel used.
bitWidth	How many bits in an audio word; usually 8/16/24/32 bits.
buffer	Pointer to the data to be written.
size	Bytes to be written.

36.4.5.62 void SAI_WriteMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

base	SAI base pointer.
channel	Data channel used.
channelMask	channel mask.
bitWidth	How many bits in an audio word; usually 8/16/24/32 bits.
buffer	Pointer to the data to be written.
size	Bytes to be written.

36.4.5.63 static void SAI_WriteData (I2S_Type * *base*, uint32_t *channel*, uint32_t *data*) [inline], [static]

Parameters

base	SAI base pointer.
channel	Data channel used.
data	Data needs to be written.

36.4.5.64 void SAI_ReadBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

base	SAI base pointer.
channel	Data channel used.
bitWidth	How many bits in an audio word; usually 8/16/24/32 bits.
buffer	Pointer to the data to be read.
size	Bytes to be read.

36.4.5.65 void SAI_ReadMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

base	SAI base pointer.
channel	Data channel used.
channelMask	channel mask.
bitWidth	How many bits in an audio word; usually 8/16/24/32 bits.
buffer	Pointer to the data to be read.
size	Bytes to be read.

36.4.5.66 static uint32_t SAI_ReadData (I2S_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	SAI base pointer.
channel	Data channel used.

Returns

Data in SAI FIFO.

36.4.5.67 void SAI_TransferTxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

base	SAI base pointer
handle	SAI handle pointer.
callback	Pointer to the user callback function.
userData	User parameter passed to the callback function

36.4.5.68 void SAI_TransferRxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

base	SAI base pointer.
handle	SAI handle pointer.
callback	Pointer to the user callback function.
userData	User parameter passed to the callback function.

36.4.5.69 void SAI_TransferTxSetConfig (I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config*)

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

base	SAI base pointer.
handle	SAI handle pointer.
config	tranmitter configurations.

36.4.5.70 void SAI_TransferRxSetConfig (I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config*)

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

base	SAI base pointer.
handle	SAI handle pointer.
config	receiver configurations.

36.4.5.71 status_t SAI_TransferTxSetFormat (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by SAI_TransferTxSetConfig

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

base	SAI base pointer.
handle	SAI handle pointer.
format	Pointer to the SAI audio data format structure.
mclkSource-ClockHz	SAI master clock source frequency in Hz.
bclkSource-ClockHz	SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format.

Returns

Status of this function. Return value is the status_t.

36.4.5.72 status_t SAI_TransferRxSetFormat (I2S_Type ∗ *base,* sai_handle_t ∗ *handle,* sai_transfer_format_t ∗ *format,* uint32_t *mclkSourceClockHz,* uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superceded by SAI_TransferRxSetConfig

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

base	SAI base pointer.
handle	SAI handle pointer.
format	Pointer to the SAI audio data format structure.
mclkSource-ClockHz	SAI master clock source frequency in Hz.
bclkSource-ClockHz	SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format.

Returns

Status of this function. Return value is one of status_t.

36.4.5.73 status_t SAI_TransferSendNonBlocking (I2S_Type ∗ *base,* sai_handle_t ∗ *handle,* sai_transfer_t ∗ *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_-SAI_Busy, the transfer is finished.

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure which stores the transfer state.
xfer	Pointer to the sai_transfer_t structure.

Return values

kStatus_Success	Successfully started the data receive.
kStatus_SAI_TxBusy	Previous receive still not finished.
kStatus_InvalidArgument	The input parameter is invalid.

36.4.5.74 status_t SAI_TransferReceiveNonBlocking (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*, sai_transfer_t ∗ *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_-SAI_Busy, the transfer is finished.

Parameters

base	SAI base pointer
handle	Pointer to the sai_handle_t structure which stores the transfer state.
xfer	Pointer to the sai_transfer_t structure.

Return values

kStatus_Success	Successfully started the data receive.
kStatus_SAI_RxBusy	Previous receive still not finished.
kStatus_InvalidArgument	The input parameter is invalid.

36.4.5.75 status_t SAI_TransferGetSendCount (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure which stores the transfer state.
count	Bytes count sent.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferIn-Progress	There is not a non-blocking transaction currently in progress.

36.4.5.76 status_t SAI_TransferGetReceiveCount (I2S_Type * *base,* sai_handle_t * *handle,* size_t * *count*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure which stores the transfer state.
count	Bytes count received.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferIn-Progress	There is not a non-blocking transaction currently in progress.

36.4.5.77 void SAI_TransferAbortSend (I2S_Type * *base,* sai_handle_t * *handle*)

Note

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

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure which stores the transfer state.

36.4.5.78 void SAI_TransferAbortReceive (I2S_Type * *base,* sai_handle_t * *handle*)

Note

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

Parameters

base	SAI base pointer
handle	Pointer to the sai_handle_t structure which stores the transfer state.

36.4.5.79 void SAI_TransferTerminateSend (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.

36.4.5.80 void SAI_TransferTerminateReceive (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.

36.4.5.81 void SAI_TransferTxHandleIRQ (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure.

36.4.5.82 void SAI_TransferRxHandleIRQ (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure.

Chapter 37
SDHC: Secure Digital Host Controller Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the Secure Digital Host Controller (SDHC) module of MCUXpresso SDK devices.

Typical use case

37.2.1 SDHC Operation

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

Data Structures

- struct sdhc_adma2_descriptor_t
  Defines the ADMA2 descriptor structure. More...
- struct sdhc_capability_t
  SDHC capability information. More...
- struct sdhc_transfer_config_t
  Card transfer configuration. More...
- struct sdhc_boot_config_t
  Data structure to configure the MMC boot feature. More...
- struct sdhc_config_t
  Data structure to initialize the SDHC. More...
- struct sdhc_data_t
  Card data descriptor. More...
- struct sdhc_command_t
  Card command descriptor. More...
- struct sdhc_transfer_t
  Transfer state. More...
- struct sdhc_transfer_callback_t
  SDHC callback functions. More...
- struct sdhc_handle_t
  SDHC handle. More...
- struct sdhc_host_t
  SDHC host descriptor. More...

Macros

- #define SDHC_MAX_BLOCK_COUNT (SDHC_BLKATTR_BLKCNT_MASK >> SDHC_BLKATTR_BLKCNT_SHIFT)
  Maximum block count can be set one time.
- #define SDHC_ADMA1_ADDRESS_ALIGN (4096U)
  The alignment size for ADDRESS filed in ADMA1's descriptor.

- #define SDHC_ADMA1_LENGTH_ALIGN (4096U)
  The alignment size for LENGTH field in ADMA1's descriptor.
- #define SDHC_ADMA2_ADDRESS_ALIGN (4U)
  The alignment size for ADDRESS field in ADMA2's descriptor.
- #define SDHC_ADMA2_LENGTH_ALIGN (4U)
  The alignment size for LENGTH filed in ADMA2's descriptor.
- #define SDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT (12U)
  The bit shift for ADDRESS filed in ADMA1's descriptor.
- #define SDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK (0xFFFFFU)
  The bit mask for ADDRESS field in ADMA1's descriptor.
- #define SDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT (12U)
  The bit shift for LENGTH filed in ADMA1's descriptor.
- #define SDHC_ADMA1_DESCRIPTOR_LENGTH_MASK (0xFFFFU)
  The mask for LENGTH field in ADMA1's descriptor.
- #define SDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (SDHC_ADMA1_DESCRIPTOR_LENGTH_MASK + 1U - 4096U)
  The maximum value of LENGTH filed in ADMA1's descriptor.
- #define SDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT (16U)
  The bit shift for LENGTH field in ADMA2's descriptor.
- #define SDHC_ADMA2_DESCRIPTOR_LENGTH_MASK (0xFFFFUL)
  The bit mask for LENGTH field in ADMA2's descriptor.
- #define SDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (SDHC_ADMA2_DESCRIPTOR_LENGTH_MASK)
  The maximum value of LENGTH field in ADMA2's descriptor.

Typedefs

- typedef uint32_t sdhc_adma1_descriptor_t
  Defines the adma1 descriptor structure.
- typedef status_t(∗ sdhc_transfer_function_t)(SDHC_Type ∗base, sdhc_transfer_t ∗content)
  SDHC transfer function.

Enumerations

- enum {
  kStatus_SDHC_BusyTransferring = MAKE_STATUS(kStatusGroup_SDHC, 0U),
  kStatus_SDHC_PrepareAdmaDescriptorFailed = MAKE_STATUS(kStatusGroup_SDHC, 1U),
  kStatus_SDHC_SendCommandFailed = MAKE_STATUS(kStatusGroup_SDHC, 2U),
  kStatus_SDHC_TransferDataFailed = MAKE_STATUS(kStatusGroup_SDHC, 3U),
  kStatus_SDHC_DMADataBufferAddrNotAlign,
  kStatus_SDHC_TransferCommandComplete = MAKE_STATUS(kStatusGroup_SDHC, 5U),
  kStatus_SDHC_TransferDataComplete = MAKE_STATUS(kStatusGroup_SDHC, 6U) }
  _sdhc_status SDHC status
- enum {

kSDHC_SupportAdmaFlag = SDHC_HTCAPBLT_ADMAS_MASK,
kSDHC_SupportHighSpeedFlag = SDHC_HTCAPBLT_HSS_MASK,
kSDHC_SupportDmaFlag = SDHC_HTCAPBLT_DMAS_MASK,
kSDHC_SupportSuspendResumeFlag = SDHC_HTCAPBLT_SRS_MASK,
kSDHC_SupportV330Flag = SDHC_HTCAPBLT_VS33_MASK,
kSDHC_Support4BitFlag = (SDHC_HTCAPBLT_MBL_SHIFT << 0U),
kSDHC_Support8BitFlag = (SDHC_HTCAPBLT_MBL_SHIFT << 1U) }
  _sdhc_capability_flag Host controller capabilities flag mask
- enum {
kSDHC_WakeupEventOnCardInt = SDHC_PROCTL_WECINT_MASK,
kSDHC_WakeupEventOnCardInsert = SDHC_PROCTL_WECINS_MASK,
kSDHC_WakeupEventOnCardRemove = SDHC_PROCTL_WECRM_MASK,
kSDHC_WakeupEventsAll }
  _sdhc_wakeup_event Wakeup event mask
- enum {
kSDHC_ResetAll = SDHC_SYSCTL_RSTA_MASK,
kSDHC_ResetCommand = SDHC_SYSCTL_RSTC_MASK,
kSDHC_ResetData = SDHC_SYSCTL_RSTD_MASK,
kSDHC_ResetsAll = (kSDHC_ResetAll | kSDHC_ResetCommand | kSDHC_ResetData) }
  _sdhc_reset Reset type mask
- enum {
kSDHC_EnableDmaFlag = SDHC_XFERTYP_DMAEN_MASK,
kSDHC_CommandTypeSuspendFlag = (SDHC_XFERTYP_CMDTYP(1U)),
kSDHC_CommandTypeResumeFlag = (SDHC_XFERTYP_CMDTYP(2U)),
kSDHC_CommandTypeAbortFlag = (SDHC_XFERTYP_CMDTYP(3U)),
kSDHC_EnableBlockCountFlag = SDHC_XFERTYP_BCEN_MASK,
kSDHC_EnableAutoCommand12Flag = SDHC_XFERTYP_AC12EN_MASK,
kSDHC_DataReadFlag = SDHC_XFERTYP_DTDSEL_MASK,
kSDHC_MultipleBlockFlag = SDHC_XFERTYP_MSBSEL_MASK,
kSDHC_ResponseLength136Flag = SDHC_XFERTYP_RSPTYP(1U),
kSDHC_ResponseLength48Flag = SDHC_XFERTYP_RSPTYP(2U),
kSDHC_ResponseLength48BusyFlag = SDHC_XFERTYP_RSPTYP(3U),
kSDHC_EnableCrcCheckFlag = SDHC_XFERTYP_CCCEN_MASK,
kSDHC_EnableIndexCheckFlag = SDHC_XFERTYP_CICEN_MASK,
kSDHC_DataPresentFlag = SDHC_XFERTYP_DPSEL_MASK }
  _sdhc_transfer_flag Transfer flag mask
- enum {

kSDHC_CommandInhibitFlag = SDHC_PRSSTAT_CIHB_MASK,
kSDHC_DataInhibitFlag = SDHC_PRSSTAT_CDIHB_MASK,
kSDHC_DataLineActiveFlag = SDHC_PRSSTAT_DLA_MASK,
kSDHC_SdClockStableFlag = SDHC_PRSSTAT_SDSTB_MASK,
kSDHC_WriteTransferActiveFlag = SDHC_PRSSTAT_WTA_MASK,
kSDHC_ReadTransferActiveFlag = SDHC_PRSSTAT_RTA_MASK,
kSDHC_BufferWriteEnableFlag = SDHC_PRSSTAT_BWEN_MASK,
kSDHC_BufferReadEnableFlag = SDHC_PRSSTAT_BREN_MASK,
kSDHC_CardInsertedFlag = SDHC_PRSSTAT_CINS_MASK,
kSDHC_CommandLineLevelFlag = SDHC_PRSSTAT_CLSL_MASK,
kSDHC_Data0LineLevelFlag = (1U << 24U),
kSDHC_Data1LineLevelFlag = (1U << 25U),
kSDHC_Data2LineLevelFlag = (1U << 26U),
kSDHC_Data3LineLevelFlag = (1U << 27U),
kSDHC_Data4LineLevelFlag = (1U << 28U),
kSDHC_Data5LineLevelFlag = (1U << 29U),
kSDHC_Data6LineLevelFlag = (1U << 30U),
kSDHC_Data7LineLevelFlag = (int)(1U << 31U) }
_sdhc_present_status_flag Present status flag mask

- enum {
kSDHC_CommandCompleteFlag = SDHC_IRQSTAT_CC_MASK,
kSDHC_DataCompleteFlag = SDHC_IRQSTAT_TC_MASK,
kSDHC_BlockGapEventFlag = SDHC_IRQSTAT_BGE_MASK,
kSDHC_DmaCompleteFlag = SDHC_IRQSTAT_DINT_MASK,
kSDHC_BufferWriteReadyFlag = SDHC_IRQSTAT_BWR_MASK,
kSDHC_BufferReadReadyFlag = SDHC_IRQSTAT_BRR_MASK,
kSDHC_CardInsertionFlag = SDHC_IRQSTAT_CINS_MASK,
kSDHC_CardRemovalFlag = SDHC_IRQSTAT_CRM_MASK,
kSDHC_CardInterruptFlag = SDHC_IRQSTAT_CINT_MASK,
kSDHC_CommandTimeoutFlag = SDHC_IRQSTAT_CTOE_MASK,
kSDHC_CommandCrcErrorFlag = SDHC_IRQSTAT_CCE_MASK,
kSDHC_CommandEndBitErrorFlag = SDHC_IRQSTAT_CEBE_MASK,
kSDHC_CommandIndexErrorFlag = SDHC_IRQSTAT_CIE_MASK,
kSDHC_DataTimeoutFlag = SDHC_IRQSTAT_DTOE_MASK,
kSDHC_DataCrcErrorFlag = SDHC_IRQSTAT_DCE_MASK,
kSDHC_DataEndBitErrorFlag = SDHC_IRQSTAT_DEBE_MASK,
kSDHC_AutoCommand12ErrorFlag = SDHC_IRQSTAT_AC12E_MASK,
kSDHC_DmaErrorFlag = SDHC_IRQSTAT_DMAE_MASK,
kSDHC_CommandErrorFlag,
kSDHC_DataErrorFlag,
kSDHC_ErrorFlag = (kSDHC_CommandErrorFlag | kSDHC_DataErrorFlag | kSDHC_DmaErrorFlag),
kSDHC_DataFlag,
kSDHC_DataDMAFlag = (kSDHC_DataCompleteFlag | kSDHC_DataErrorFlag | kSDHC_Dma-

ErrorFlag),
kSDHC_CommandFlag = (kSDHC_CommandErrorFlag | kSDHC_CommandCompleteFlag),
kSDHC_CardDetectFlag = (kSDHC_CardInsertionFlag | kSDHC_CardRemovalFlag),
kSDHC_AllInterruptFlags }
_sdhc_interrupt_status_flag Interrupt status flag mask
- enum {
kSDHC_AutoCommand12NotExecutedFlag = SDHC_AC12ERR_AC12NE_MASK,
kSDHC_AutoCommand12TimeoutFlag = SDHC_AC12ERR_AC12TOE_MASK,
kSDHC_AutoCommand12EndBitErrorFlag = SDHC_AC12ERR_AC12EBE_MASK,
kSDHC_AutoCommand12CrcErrorFlag = SDHC_AC12ERR_AC12CE_MASK,
kSDHC_AutoCommand12IndexErrorFlag = SDHC_AC12ERR_AC12IE_MASK,
kSDHC_AutoCommand12NotIssuedFlag = SDHC_AC12ERR_CNIBAC12E_MASK }
_sdhc_auto_command12_error_status_flag Auto CMD12 error status flag mask
- enum {
kSDHC_AdmaLenghMismatchFlag = SDHC_ADMAES_ADMALME_MASK,
kSDHC_AdmaDescriptorErrorFlag = SDHC_ADMAES_ADMADCE_MASK }
_sdhc_adma_error_status_flag ADMA error status flag mask
- enum sdhc_adma_error_state_t {
kSDHC_AdmaErrorStateStopDma = 0x00U,
kSDHC_AdmaErrorStateFetchDescriptor = 0x01U,
kSDHC_AdmaErrorStateChangeAddress = 0x02U,
kSDHC_AdmaErrorStateTransferData = 0x03U }
ADMA error state.
- enum {
kSDHC_ForceEventAutoCommand12NotExecuted = SDHC_FEVT_AC12NE_MASK,
kSDHC_ForceEventAutoCommand12Timeout = SDHC_FEVT_AC12TOE_MASK,
kSDHC_ForceEventAutoCommand12CrcError = SDHC_FEVT_AC12CE_MASK,
kSDHC_ForceEventEndBitError = SDHC_FEVT_AC12EBE_MASK,
kSDHC_ForceEventAutoCommand12IndexError = SDHC_FEVT_AC12IE_MASK,
kSDHC_ForceEventAutoCommand12NotIssued = SDHC_FEVT_CNIBAC12E_MASK,
kSDHC_ForceEventCommandTimeout = SDHC_FEVT_CTOE_MASK,
kSDHC_ForceEventCommandCrcError = SDHC_FEVT_CCE_MASK,
kSDHC_ForceEventCommandEndBitError = SDHC_FEVT_CEBE_MASK,
kSDHC_ForceEventCommandIndexError = SDHC_FEVT_CIE_MASK,
kSDHC_ForceEventDataTimeout = SDHC_FEVT_DTOE_MASK,
kSDHC_ForceEventDataCrcError = SDHC_FEVT_DCE_MASK,
kSDHC_ForceEventDataEndBitError = SDHC_FEVT_DEBE_MASK,
kSDHC_ForceEventAutoCommand12Error = SDHC_FEVT_AC12E_MASK,
kSDHC_ForceEventCardInt = (int)SDHC_FEVT_CINT_MASK,
kSDHC_ForceEventDmaError = SDHC_FEVT_DMAE_MASK,
kSDHC_ForceEventsAll }
_sdhc_force_event Force event bit position
- enum sdhc_data_bus_width_t {
kSDHC_DataBusWidth1Bit = 0U,
kSDHC_DataBusWidth4Bit = 1U,
kSDHC_DataBusWidth8Bit = 2U }

Data transfer width.
- enum sdhc_endian_mode_t {
kSDHC_EndianModeBig = 0U,
kSDHC_EndianModeHalfWordBig = 1U,
kSDHC_EndianModeLittle = 2U }
Endian mode.
- enum sdhc_dma_mode_t {
kSDHC_DmaModeNo = 0U,
kSDHC_DmaModeAdma1 = 1U,
kSDHC_DmaModeAdma2 = 2U }
DMA mode.
- enum {
kSDHC_StopAtBlockGapFlag = 0x01,
kSDHC_ReadWaitControlFlag = 0x02,
kSDHC_InterruptAtBlockGapFlag = 0x04,
kSDHC_ExactBlockNumberReadFlag = 0x08 }
_sdhc_sdio_control_flag SDIO control flag mask
- enum sdhc_boot_mode_t {
kSDHC_BootModeNormal = 0U,
kSDHC_BootModeAlternative = 1U }
MMC card boot mode.
- enum sdhc_card_command_type_t {
kCARD_CommandTypeNormal = 0U,
kCARD_CommandTypeSuspend = 1U,
kCARD_CommandTypeResume = 2U,
kCARD_CommandTypeAbort = 3U }
The command type.
- enum sdhc_card_response_type_t {
kCARD_ResponseTypeNone = 0U,
kCARD_ResponseTypeR1 = 1U,
kCARD_ResponseTypeR1b = 2U,
kCARD_ResponseTypeR2 = 3U,
kCARD_ResponseTypeR3 = 4U,
kCARD_ResponseTypeR4 = 5U,
kCARD_ResponseTypeR5 = 6U,
kCARD_ResponseTypeR5b = 7U,
kCARD_ResponseTypeR6 = 8U,
kCARD_ResponseTypeR7 = 9U }
The command response type.
- enum {

kSDHC_Adma1DescriptorValidFlag = (1U << 0U),
kSDHC_Adma1DescriptorEndFlag = (1U << 1U),
kSDHC_Adma1DescriptorInterrupFlag = (1U << 2U),
kSDHC_Adma1DescriptorActivity1Flag = (1U << 4U),
kSDHC_Adma1DescriptorActivity2Flag = (1U << 5U),
kSDHC_Adma1DescriptorTypeNop = (kSDHC_Adma1DescriptorValidFlag),
kSDHC_Adma1DescriptorTypeTransfer,
kSDHC_Adma1DescriptorTypeLink,
kSDHC_Adma1DescriptorTypeSetLength }
_sdhc_adma1_descriptor_flag The mask for the control/status field in ADMA1 descriptor

- enum {
kSDHC_Adma2DescriptorValidFlag = (1U << 0U),
kSDHC_Adma2DescriptorEndFlag = (1U << 1U),
kSDHC_Adma2DescriptorInterruptFlag = (1U << 2U),
kSDHC_Adma2DescriptorActivity1Flag = (1U << 4U),
kSDHC_Adma2DescriptorActivity2Flag = (1U << 5U),
kSDHC_Adma2DescriptorTypeNop = (kSDHC_Adma2DescriptorValidFlag),
kSDHC_Adma2DescriptorTypeReserved,
kSDHC_Adma2DescriptorTypeTransfer,
kSDHC_Adma2DescriptorTypeLink }
_sdhc_adma2_descriptor_flag ADMA1 descriptor control and status mask

Driver version

- #define FSL_SDHC_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 12U))
Driver version 2.1.12.

Initialization and deinitialization

- void SDHC_Init (SDHC_Type ∗base, const sdhc_config_t ∗config)
SDHC module initialization function.
- void SDHC_Deinit (SDHC_Type ∗base)
Deinitializes the SDHC.
- bool SDHC_Reset (SDHC_Type ∗base, uint32_t mask, uint32_t timeout)
Resets the SDHC.

DMA Control

- status_t SDHC_SetAdmaTableConfig (SDHC_Type ∗base, sdhc_dma_mode_t dmaMode, uint32_t ∗table, uint32_t tableWords, const uint32_t ∗data, uint32_t dataBytes)
Sets the ADMA descriptor table configuration.

Interrupts

- static void SDHC_EnableInterruptStatus (SDHC_Type ∗base, uint32_t mask)
Enables the interrupt status.
- static void SDHC_DisableInterruptStatus (SDHC_Type ∗base, uint32_t mask)
Disables the interrupt status.

- static void SDHC_EnableInterruptSignal (SDHC_Type ∗base, uint32_t mask)
  Enables the interrupt signal corresponding to the interrupt status flag.
- static void SDHC_DisableInterruptSignal (SDHC_Type ∗base, uint32_t mask)
  Disables the interrupt signal corresponding to the interrupt status flag.

Status

- static uint32_t SDHC_GetEnabledInterruptStatusFlags (SDHC_Type ∗base)
  Gets the enabled interrupt status.
- static uint32_t SDHC_GetInterruptStatusFlags (SDHC_Type ∗base)
  Gets the current interrupt status.
- static void SDHC_ClearInterruptStatusFlags (SDHC_Type ∗base, uint32_t mask)
  Clears a specified interrupt status.
- static uint32_t SDHC_GetAutoCommand12ErrorStatusFlags (SDHC_Type ∗base)
  Gets the status of auto command 12 error.
- static uint32_t SDHC_GetAdmaErrorStatusFlags (SDHC_Type ∗base)
  Gets the status of the ADMA error.
- static uint32_t SDHC_GetPresentStatusFlags (SDHC_Type ∗base)
  Gets a present status.

Bus Operations

- void SDHC_GetCapability (SDHC_Type ∗base, sdhc_capability_t ∗capability)
  Gets the capability information.
- static void SDHC_EnableSdClock (SDHC_Type ∗base, bool enable)
  Enables or disables the SD bus clock.
- uint32_t SDHC_SetSdClock (SDHC_Type ∗base, uint32_t srcClock_Hz, uint32_t busClock_Hz)
  Sets the SD bus clock frequency.
- bool SDHC_SetCardActive (SDHC_Type ∗base, uint32_t timeout)
  Sends 80 clocks to the card to set it to the active state.
- static void SDHC_SetDataBusWidth (SDHC_Type ∗base, sdhc_data_bus_width_t width)
  Sets the data transfer width.
- static void SDHC_CardDetectByData3 (SDHC_Type ∗base, bool enable)
  detect card insert status.
- void SDHC_SetTransferConfig (SDHC_Type ∗base, const sdhc_transfer_config_t ∗config)
  Sets the card transfer-related configuration.
- static uint32_t SDHC_GetCommandResponse (SDHC_Type ∗base, uint32_t index)
  Gets the command response.
- static void SDHC_WriteData (SDHC_Type ∗base, uint32_t data)
  Fills the data port.
- static uint32_t SDHC_ReadData (SDHC_Type ∗base)
  Retrieves the data from the data port.
- static void SDHC_EnableWakeupEvent (SDHC_Type ∗base, uint32_t mask, bool enable)
  Enables or disables a wakeup event in low-power mode.
- static void SDHC_EnableCardDetectTest (SDHC_Type ∗base, bool enable)
  Enables or disables the card detection level for testing.
- static void SDHC_SetCardDetectTestLevel (SDHC_Type ∗base, bool high)
  Sets the card detection test level.
- void SDHC_EnableSdioControl (SDHC_Type ∗base, uint32_t mask, bool enable)
  Enables or disables the SDIO card control.
- static void SDHC_SetContinueRequest (SDHC_Type ∗base)

Restarts a transaction which has stopped at the block GAP for the SDIO card.
- void SDHC_SetMmcBootConfig (SDHC_Type ∗base, const sdhc_boot_config_t ∗config)
  Configures the MMC boot feature.
- static void SDHC_SetForceEvent (SDHC_Type ∗base, uint32_t mask)
  Forces generating events according to the given mask.

Transactional

- status_t SDHC_TransferBlocking (SDHC_Type ∗base, uint32_t ∗admaTable, uint32_t admaTable-Words, sdhc_transfer_t ∗transfer)
  Transfers the command/data using a blocking method.
- void SDHC_TransferCreateHandle (SDHC_Type ∗base, sdhc_handle_t ∗handle, const sdhc_-transfer_callback_t ∗callback, void ∗userData)
  Creates the SDHC handle.
- status_t SDHC_TransferNonBlocking (SDHC_Type ∗base, sdhc_handle_t ∗handle, uint32_t ∗admaTable, uint32_t admaTableWords, sdhc_transfer_t ∗transfer)
  Transfers the command/data using an interrupt and an asynchronous method.
- void SDHC_TransferHandleIRQ (SDHC_Type ∗base, sdhc_handle_t ∗handle)
  IRQ handler for the SDHC.

Data Structure Documentation

37.3.1 struct sdhc_adma2_descriptor_t

Data Fields

- uint32_t attribute
  The control and status field.
- const uint32_t ∗ address
  The address field.

37.3.2 struct sdhc_capability_t

Defines a structure to save the capability information of SDHC.

Data Fields

- uint32_t specVersion
  Specification version.
- uint32_t vendorVersion
  Vendor version.
- uint32_t maxBlockLength
  Maximum block length united as byte.
- uint32_t maxBlockCount
  Maximum block count can be set one time.
- uint32_t flags
  Capability flags to indicate the support information(_sdhc_capability_flag)

37.3.3 struct sdhc_transfer_config_t

Define structure to configure the transfer-related command index/argument/flags and data block size/data block numbers. This structure needs to be filled each time a command is sent to the card.

Data Fields

- size_t dataBlockSize
  Data block size.
- uint32_t dataBlockCount
  Data block count.
- uint32_t commandArgument
  Command argument.
- uint32_t commandIndex
  Command index.
- uint32_t flags
  Transfer flags(_sdhc_transfer_flag)

37.3.4 struct sdhc_boot_config_t

Data Fields

- uint32_t ackTimeoutCount
  Timeout value for the boot ACK.
- sdhc_boot_mode_t bootMode
  Boot mode selection.
- uint32_t blockCount
  Stop at block gap value of automatic mode.
- bool enableBootAck
  Enable or disable boot ACK.
- bool enableBoot
  Enable or disable fast boot.
- bool enableAutoStopAtBlockGap
  Enable or disable auto stop at block gap function in boot period.

37.3.4.0.0.30 Field Documentation

37.3.4.0.0.30.1 uint32_t sdhc_boot_config_t::ackTimeoutCount

The available range is 0 ∼ 15.

37.3.4.0.0.30.2 sdhc_boot_mode_t sdhc_boot_config_t::bootMode

37.3.4.0.0.30.3 uint32_t sdhc_boot_config_t::blockCount

Available range is 0 ∼ 65535.

37.3.5 struct sdhc_config_t

Data Fields

- bool cardDetectDat3
  Enable DAT3 as card detection pin.
- sdhc_endian_mode_t endianMode
  Endian mode.
- sdhc_dma_mode_t dmaMode
  DMA mode.
- uint32_t readWatermarkLevel
  Watermark level for DMA read operation.
- uint32_t writeWatermarkLevel
  Watermark level for DMA write operation.

37.3.5.0.0.31 Field Documentation

37.3.5.0.0.31.1 uint32_t sdhc_config_t::readWatermarkLevel

Available range is 1 ∼ 128.

37.3.5.0.0.31.2 uint32_t sdhc_config_t::writeWatermarkLevel

Available range is 1 ∼ 128.

37.3.6 struct sdhc_data_t

Defines a structure to contain data-related attribute. 'enableIgnoreError' is used for the case that upper card driver want to ignore the error event to read/write all the data not to stop read/write immediately when error event happen for example bus testing procedure for MMC card.

Data Fields

- bool enableAutoCommand12
  Enable auto CMD12.
- bool enableIgnoreError
  Enable to ignore error event to read/write all the data.
- size_t blockSize
  Block size.
- uint32_t blockCount
  Block count.
- uint32_t ∗ rxData
  Buffer to save data read.
- const uint32_t ∗ txData
  Data buffer to write.

37.3.7 struct sdhc_command_t

Define card command-related attribute.

Data Fields

- uint32_t index
  Command index.
- uint32_t argument
  Command argument.
- sdhc_card_command_type_t type
  Command type.
- sdhc_card_response_type_t responseType
  Command response type.
- uint32_t response [4U]
  Response for this command.
- uint32_t responseErrorFlags
  response error flag, the flag which need to check the command reponse

37.3.8 struct sdhc_transfer_t

Data Fields

- sdhc_data_t ∗ data
  Data to transfer.
- sdhc_command_t ∗ command
  Command to send.

37.3.9 struct sdhc_transfer_callback_t

Data Fields

- void(∗ CardInserted)(SDHC_Type ∗base, void ∗userData)
  Card inserted occurs when DAT3/CD pin is for card detect.
- void(∗ CardRemoved)(SDHC_Type ∗base, void ∗userData)
  Card removed occurs.
- void(∗ SdioInterrupt)(SDHC_Type ∗base, void ∗userData)
  SDIO card interrupt occurs.
- void(∗ SdioBlockGap)(SDHC_Type ∗base, void ∗userData)
  SDIO card stopped at block gap occurs.
- void(∗ TransferComplete)(SDHC_Type ∗base, sdhc_handle_t ∗handle, status_t status, void ∗userData)
  Transfer complete callback.

37.3.10 struct _sdhc_handle

SDHC handle typedef.

Defines the structure to save the SDHC state information and callback function. The detailed interrupt status when sending a command or transfering data can be obtained from the interruptFlags field by using the mask defined in sdhc_interrupt_flag_t.

Note

All the fields except interruptFlags and transferredWords must be allocated by the user.

Data Fields

- sdhc_data_t ∗volatile data
  Data to transfer.
- sdhc_command_t ∗volatile command
  Command to send.
- volatile uint32_t transferredWords
  Words transferred by DATAPORT way.
- sdhc_transfer_callback_t callback
  Callback function.
- void ∗ userData
  Parameter for transfer complete callback.

37.3.11 struct sdhc_host_t

Data Fields

- SDHC_Type ∗ base
  SDHC peripheral base address.
- uint32_t sourceClock_Hz
  SDHC source clock frequency united in Hz.
- sdhc_config_t config
  SDHC configuration.
- sdhc_capability_t capability
  SDHC capability information.
- sdhc_transfer_function_t transfer
  SDHC transfer function.

Macro Definition Documentation

37.4.1 #define FSL_SDHC_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 12U))

Typedef Documentation

37.5.1 typedef uint32_t sdhc_adma1_descriptor_t

37.5.2 typedef status_t(∗ sdhc_transfer_function_t)(SDHC_Type ∗base, sdhc_transfer_t ∗content)

Enumeration Type Documentation

37.6.1 anonymous enum

Enumerator

kStatus_SDHC_BusyTransferring Transfer is on-going.
kStatus_SDHC_PrepareAdmaDescriptorFailed Set DMA descriptor failed.
kStatus_SDHC_SendCommandFailed Send command failed.
kStatus_SDHC_TransferDataFailed Transfer data failed.
kStatus_SDHC_DMADataBufferAddrNotAlign data buffer addr not align in DMA mode
kStatus_SDHC_TransferCommandComplete command transfer complete
kStatus_SDHC_TransferDataComplete data transfer complete

37.6.2 anonymous enum

Enumerator

kSDHC_SupportAdmaFlag Support ADMA.
kSDHC_SupportHighSpeedFlag Support high-speed.
kSDHC_SupportDmaFlag Support DMA.
kSDHC_SupportSuspendResumeFlag Support suspend/resume.
kSDHC_SupportV330Flag Support voltage 3.3V.
kSDHC_Support4BitFlag Support 4 bit mode.
kSDHC_Support8BitFlag Support 8 bit mode.

37.6.3 anonymous enum

Enumerator

kSDHC_WakeupEventOnCardInt Wakeup on card interrupt.
kSDHC_WakeupEventOnCardInsert Wakeup on card insertion.
kSDHC_WakeupEventOnCardRemove Wakeup on card removal.
kSDHC_WakeupEventsAll All wakeup events.

37.6.4 anonymous enum

Enumerator

kSDHC_ResetAll Reset all except card detection.
kSDHC_ResetCommand Reset command line.
kSDHC_ResetData Reset data line.
kSDHC_ResetsAll All reset types.

37.6.5 anonymous enum

Enumerator

kSDHC_EnableDmaFlag Enable DMA.
kSDHC_CommandTypeSuspendFlag Suspend command.
kSDHC_CommandTypeResumeFlag Resume command.
kSDHC_CommandTypeAbortFlag Abort command.
kSDHC_EnableBlockCountFlag Enable block count.
kSDHC_EnableAutoCommand12Flag Enable auto CMD12.
kSDHC_DataReadFlag Enable data read.
kSDHC_MultipleBlockFlag Multiple block data read/write.
kSDHC_ResponseLength136Flag 136 bit response length
kSDHC_ResponseLength48Flag 48 bit response length
kSDHC_ResponseLength48BusyFlag 48 bit response length with busy status
kSDHC_EnableCrcCheckFlag Enable CRC check.
kSDHC_EnableIndexCheckFlag Enable index check.
kSDHC_DataPresentFlag Data present flag.

37.6.6 anonymous enum

Enumerator

kSDHC_CommandInhibitFlag Command inhibit.
kSDHC_DataInhibitFlag Data inhibit.
kSDHC_DataLineActiveFlag Data line active.
kSDHC_SdClockStableFlag SD bus clock stable.
kSDHC_WriteTransferActiveFlag Write transfer active.
kSDHC_ReadTransferActiveFlag Read transfer active.
kSDHC_BufferWriteEnableFlag Buffer write enable.
kSDHC_BufferReadEnableFlag Buffer read enable.
kSDHC_CardInsertedFlag Card inserted.
kSDHC_CommandLineLevelFlag Command line signal level.
kSDHC_Data0LineLevelFlag Data0 line signal level.
kSDHC_Data1LineLevelFlag Data1 line signal level.

kSDHC_Data2LineLevelFlag Data2 line signal level.
kSDHC_Data3LineLevelFlag Data3 line signal level.
kSDHC_Data4LineLevelFlag Data4 line signal level.
kSDHC_Data5LineLevelFlag Data5 line signal level.
kSDHC_Data6LineLevelFlag Data6 line signal level.
kSDHC_Data7LineLevelFlag Data7 line signal level.

37.6.7 anonymous enum

Enumerator

kSDHC_CommandCompleteFlag Command complete.
kSDHC_DataCompleteFlag Data complete.
kSDHC_BlockGapEventFlag Block gap event.
kSDHC_DmaCompleteFlag DMA interrupt.
kSDHC_BufferWriteReadyFlag Buffer write ready.
kSDHC_BufferReadReadyFlag Buffer read ready.
kSDHC_CardInsertionFlag Card inserted.
kSDHC_CardRemovalFlag Card removed.
kSDHC_CardInterruptFlag Card interrupt.
kSDHC_CommandTimeoutFlag Command timeout error.
kSDHC_CommandCrcErrorFlag Command CRC error.
kSDHC_CommandEndBitErrorFlag Command end bit error.
kSDHC_CommandIndexErrorFlag Command index error.
kSDHC_DataTimeoutFlag Data timeout error.
kSDHC_DataCrcErrorFlag Data CRC error.
kSDHC_DataEndBitErrorFlag Data end bit error.
kSDHC_AutoCommand12ErrorFlag Auto CMD12 error.
kSDHC_DmaErrorFlag DMA error.
kSDHC_CommandErrorFlag Command error.
kSDHC_DataErrorFlag Data error.
kSDHC_ErrorFlag All error.
kSDHC_DataFlag Data interrupts.
kSDHC_DataDMAFlag Data interrupts.
kSDHC_CommandFlag Command interrupts.
kSDHC_CardDetectFlag Card detection interrupts.
kSDHC_AllInterruptFlags All flags mask.

37.6.8 anonymous enum

Enumerator

kSDHC_AutoCommand12NotExecutedFlag Not executed error.
kSDHC_AutoCommand12TimeoutFlag Timeout error.

kSDHC_AutoCommand12EndBitErrorFlag End bit error.
kSDHC_AutoCommand12CrcErrorFlag CRC error.
kSDHC_AutoCommand12IndexErrorFlag Index error.
kSDHC_AutoCommand12NotIssuedFlag Not issued error.

37.6.9 anonymous enum

Enumerator

kSDHC_AdmaLenghMismatchFlag Length mismatch error.
kSDHC_AdmaDescriptorErrorFlag Descriptor error.

37.6.10 enum sdhc_adma_error_state_t

This state is the detail state when ADMA error has occurred.

Enumerator

kSDHC_AdmaErrorStateStopDma Stop DMA.
kSDHC_AdmaErrorStateFetchDescriptor Fetch descriptor.
kSDHC_AdmaErrorStateChangeAddress Change address.
kSDHC_AdmaErrorStateTransferData Transfer data.

37.6.11 anonymous enum

Enumerator

kSDHC_ForceEventAutoCommand12NotExecuted Auto CMD12 not executed error.
kSDHC_ForceEventAutoCommand12Timeout Auto CMD12 timeout error.
kSDHC_ForceEventAutoCommand12CrcError Auto CMD12 CRC error.
kSDHC_ForceEventEndBitError Auto CMD12 end bit error.
kSDHC_ForceEventAutoCommand12IndexError Auto CMD12 index error.
kSDHC_ForceEventAutoCommand12NotIssued Auto CMD12 not issued error.
kSDHC_ForceEventCommandTimeout Command timeout error.
kSDHC_ForceEventCommandCrcError Command CRC error.
kSDHC_ForceEventCommandEndBitError Command end bit error.
kSDHC_ForceEventCommandIndexError Command index error.
kSDHC_ForceEventDataTimeout Data timeout error.
kSDHC_ForceEventDataCrcError Data CRC error.
kSDHC_ForceEventDataEndBitError Data end bit error.
kSDHC_ForceEventAutoCommand12Error Auto CMD12 error.
kSDHC_ForceEventCardInt Card interrupt.
kSDHC_ForceEventDmaError Dma error.

kSDHC_ForceEventsAll All force event flags mask.

37.6.12 enum sdhc_data_bus_width_t

Enumerator

kSDHC_DataBusWidth1Bit 1-bit mode
kSDHC_DataBusWidth4Bit 4-bit mode
kSDHC_DataBusWidth8Bit 8-bit mode

37.6.13 enum sdhc_endian_mode_t

Enumerator

kSDHC_EndianModeBig Big endian mode.
kSDHC_EndianModeHalfWordBig Half word big endian mode.
kSDHC_EndianModeLittle Little endian mode.

37.6.14 enum sdhc_dma_mode_t

Enumerator

kSDHC_DmaModeNo No DMA.
kSDHC_DmaModeAdma1 ADMA1 is selected.
kSDHC_DmaModeAdma2 ADMA2 is selected.

37.6.15 anonymous enum

Enumerator

kSDHC_StopAtBlockGapFlag Stop at block gap.
kSDHC_ReadWaitControlFlag Read wait control.
kSDHC_InterruptAtBlockGapFlag Interrupt at block gap.
kSDHC_ExactBlockNumberReadFlag Exact block number read.

37.6.16 enum sdhc_boot_mode_t

Enumerator

kSDHC_BootModeNormal Normal boot.
kSDHC_BootModeAlternative Alternative boot.

37.6.17 enum sdhc_card_command_type_t

Enumerator

kCARD_CommandTypeNormal Normal command.
kCARD_CommandTypeSuspend Suspend command.
kCARD_CommandTypeResume Resume command.
kCARD_CommandTypeAbort Abort command.

37.6.18 enum sdhc_card_response_type_t

Define the command response type from card to host controller.

Enumerator

kCARD_ResponseTypeNone Response type: none.
kCARD_ResponseTypeR1 Response type: R1.
kCARD_ResponseTypeR1b Response type: R1b.
kCARD_ResponseTypeR2 Response type: R2.
kCARD_ResponseTypeR3 Response type: R3.
kCARD_ResponseTypeR4 Response type: R4.
kCARD_ResponseTypeR5 Response type: R5.
kCARD_ResponseTypeR5b Response type: R5b.
kCARD_ResponseTypeR6 Response type: R6.
kCARD_ResponseTypeR7 Response type: R7.

37.6.19 anonymous enum

Enumerator

kSDHC_Adma1DescriptorValidFlag Valid flag.
kSDHC_Adma1DescriptorEndFlag End flag.
kSDHC_Adma1DescriptorInterrupFlag Interrupt flag.
kSDHC_Adma1DescriptorActivity1Flag Activity 1 flag.
kSDHC_Adma1DescriptorActivity2Flag Activity 2 flag.
kSDHC_Adma1DescriptorTypeNop No operation.
kSDHC_Adma1DescriptorTypeTransfer Transfer data.
kSDHC_Adma1DescriptorTypeLink Link descriptor.
kSDHC_Adma1DescriptorTypeSetLength Set data length.

37.6.20 anonymous enum

Enumerator

kSDHC_Adma2DescriptorValidFlag Valid flag.
kSDHC_Adma2DescriptorEndFlag End flag.
kSDHC_Adma2DescriptorInterruptFlag Interrupt flag.
kSDHC_Adma2DescriptorActivity1Flag Activity 1 mask.
kSDHC_Adma2DescriptorActivity2Flag Activity 2 mask.
kSDHC_Adma2DescriptorTypeNop No operation.
kSDHC_Adma2DescriptorTypeReserved Reserved.
kSDHC_Adma2DescriptorTypeTransfer Transfer type.
kSDHC_Adma2DescriptorTypeLink Link type.

Function Documentation

37.7.1 void SDHC_Init (SDHC_Type * *base*, const sdhc_config_t * *config*)

Configures the SDHC according to the user configuration.

Example:

```
sdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kSDHC_EndianModeLittle;
config.dmaMode = kSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
SDHC_Init(SDHC, &config);
```

Parameters

base	SDHC peripheral base address.
config	SDHC configuration information.

Return values

kStatus_Success	Operate successfully.

37.7.2 void SDHC_Deinit (SDHC_Type * *base*)

Parameters

base	SDHC peripheral base address.

37.7.3 bool SDHC_Reset (SDHC_Type ∗ *base,* uint32_t *mask,* uint32_t *timeout*)

Parameters

base	SDHC peripheral base address.
mask	The reset type mask(_sdhc_reset).
timeout	Timeout for reset.

Return values

true	Reset successfully.
false	Reset failed.

37.7.4 status_t SDHC_SetAdmaTableConfig (SDHC_Type ∗ *base,* sdhc_dma_mode_t *dmaMode,* uint32_t ∗ *table,* uint32_t *tableWords,* const uint32_t ∗ *data,* uint32_t *dataBytes*)

Parameters

base	SDHC peripheral base address.
dmaMode	DMA mode.
table	ADMA table address.
tableWords	ADMA table buffer length united as Words.
data	Data buffer address.
dataBytes	Data length united as bytes.

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.

kStatus_Success	Operate successfully.

37.7.5 static void SDHC_EnableInterruptStatus (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SDHC peripheral base address.
mask	Interrupt status flags mask(_sdhc_interrupt_status_flag).

37.7.6 static void SDHC_DisableInterruptStatus (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SDHC peripheral base address.
mask	The interrupt status flags mask(_sdhc_interrupt_status_flag).

37.7.7 static void SDHC_EnableInterruptSignal (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SDHC peripheral base address.
mask	The interrupt status flags mask(_sdhc_interrupt_status_flag).

37.7.8 static void SDHC_DisableInterruptSignal (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SDHC peripheral base address.
mask	The interrupt status flags mask(_sdhc_interrupt_status_flag).

37.7.9 static uint32_t SDHC_GetEnabledInterruptStatusFlags (SDHC_Type * *base*) [inline], [static]

Parameters

base	SDHC peripheral base address.

Returns

Current interrupt status flags mask(_sdhc_interrupt_status_flag).

37.7.10 static uint32_t SDHC_GetInterruptStatusFlags (SDHC_Type * *base*) [inline], [static]

Parameters

base	SDHC peripheral base address.

Returns

Current interrupt status flags mask(_sdhc_interrupt_status_flag).

37.7.11 static void SDHC_ClearInterruptStatusFlags (SDHC_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	SDHC peripheral base address.
mask	The interrupt status flags mask(_sdhc_interrupt_status_flag).

37.7.12 static uint32_t SDHC_GetAutoCommand12ErrorStatusFlags (SDHC_Type * *base*) [inline], [static]

Parameters

base	SDHC peripheral base address.

Returns

Auto command 12 error status flags mask(_sdhc_auto_command12_error_status_flag).

37.7.13 static uint32_t SDHC_GetAdmaErrorStatusFlags (SDHC_Type ∗ *base*) [inline], [static]

Parameters

base	SDHC peripheral base address.

Returns

ADMA error status flags mask(_sdhc_adma_error_status_flag).

37.7.14 static uint32_t SDHC_GetPresentStatusFlags (SDHC_Type ∗ *base*) [inline], [static]

This function gets the present SDHC's status except for an interrupt status and an error status.

Parameters

base	SDHC peripheral base address.

Returns

Present SDHC's status flags mask(_sdhc_present_status_flag).

37.7.15 void SDHC_GetCapability (SDHC_Type ∗ *base*, sdhc_capability_t ∗ *capability*)

Parameters

base	SDHC peripheral base address.
capability	Structure to save capability information.

37.7.16 static void SDHC_EnableSdClock (SDHC_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	SDHC peripheral base address.
enable	True to enable, false to disable.

37.7.17 uint32_t SDHC_SetSdClock (SDHC_Type ∗ *base,* uint32_t *srcClock_Hz,* uint32_t *busClock_Hz*)

Parameters

base	SDHC peripheral base address.
srcClock_Hz	SDHC source clock frequency united in Hz.
busClock_Hz	SD bus clock frequency united in Hz.

Returns

The nearest frequency of busClock_Hz configured to SD bus.

37.7.18 bool SDHC_SetCardActive (SDHC_Type ∗ *base,* uint32_t *timeout*)

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

Parameters

base	SDHC peripheral base address.
timeout	Timeout to initialize card.

Return values

true	Set card active successfully.
false	Set card active failed.

37.7.19 static void SDHC_SetDataBusWidth (SDHC_Type * *base*, sdhc_data_bus_width_t *width*) [inline], [static]

Parameters

base	SDHC peripheral base address.
width	Data transfer width.

37.7.20 static void SDHC_CardDetectByData3 (SDHC_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	SDHC peripheral base address.
enable	Enable/disable flag.

37.7.21 void SDHC_SetTransferConfig (SDHC_Type * *base*, const sdhc_transfer_config_t * *config*)

This function fills the card transfer-related command argument/transfer flag/data size. The command and data are sent by SDHC after calling this function.

Example:

```
sdhc_transfer_config_t transferConfig;
transferConfig.dataBlockSize = 512U;
transferConfig.dataBlockCount = 2U;
transferConfig.commandArgument = 0x01AAU;
transferConfig.commandIndex = 8U;
transferConfig.flags |= (kSDHC_EnableDmaFlag |
      kSDHC_EnableAutoCommand12Flag |
      kSDHC_MultipleBlockFlag);
SDHC_SetTransferConfig(SDHC, &transferConfig);
```

Parameters

base	SDHC peripheral base address.
config	Command configuration structure.

37.7.22 static uint32_t SDHC_GetCommandResponse (SDHC_Type * *base*, uint32_t *index*) [inline], [static]

Parameters

base	SDHC peripheral base address.
index	The index of response register, range from 0 to 3.

Returns

Response register transfer.

37.7.23 static void SDHC_WriteData (SDHC_Type * *base*, uint32_t *data*) [inline], [static]

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

base	SDHC peripheral base address.
data	The data about to be sent.

37.7.24 static uint32_t SDHC_ReadData (SDHC_Type * *base*) [inline], [static]

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

base	SDHC peripheral base address.

Returns

The data has been read.

37.7.25 static void SDHC_EnableWakeupEvent (SDHC_Type ∗ *base*, uint32_t *mask*, bool *enable*) [inline], [static]

Parameters

base	SDHC peripheral base address.
mask	Wakeup events mask(_sdhc_wakeup_event).
enable	True to enable, false to disable.

37.7.26 static void SDHC_EnableCardDetectTest (SDHC_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

base	SDHC peripheral base address.
enable	True to enable, false to disable.

37.7.27 static void SDHC_SetCardDetectTestLevel (SDHC_Type ∗ *base*, bool *high*) [inline], [static]

This function sets the card detection test level to indicate whether the card is inserted into the SDHC when DAT[3]/ CD pin is selected as a card detection pin. This function can also assert the pin logic when DAT[3]/CD pin is selected as the card detection pin.

Parameters

base	SDHC peripheral base address.

high	True to set the card detect level to high.

37.7.28 void SDHC_EnableSdioControl (SDHC_Type ∗ *base*, uint32_t *mask*, bool *enable*)

Parameters

base	SDHC peripheral base address.
mask	SDIO card control flags mask(_sdhc_sdio_control_flag).
enable	True to enable, false to disable.

37.7.29 static void SDHC_SetContinueRequest (SDHC_Type ∗ *base*) [inline], [static]

Parameters

base	SDHC peripheral base address.

37.7.30 void SDHC_SetMmcBootConfig (SDHC_Type ∗ *base*, const sdhc_boot_config_t ∗ *config*)

Example:

```
sdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
SDHC_SetMmcBootConfig(SDHC, &config);
```

Parameters

base	SDHC peripheral base address.
config	The MMC boot configuration information.

37.7.31 static void SDHC_SetForceEvent (SDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SDHC peripheral base address.
mask	The force events mask(_sdhc_force_event).

37.7.32 status_t SDHC_TransferBlocking (SDHC_Type * *base*, uint32_t * *admaTable*, uint32_t *admaTableWords*, sdhc_transfer_t * *transfer*)

This function waits until the command response/data is received or the SDHC encounters an error by polling the status flag. This function support non word align data addr transfer support, if data buffer addr is not align in DMA mode, the API will continue finish the transfer by polling IO directly The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note

There is no need to call the API 'SDHC_TransferCreateHandle' when calling this API.

Parameters

base	SDHC peripheral base address.
admaTable	ADMA table address, can't be null if transfer way is ADMA1/ADMA2.
admaTable-Words	ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.
transfer	Transfer content.

Return values

kStatus_InvalidArgument	Argument is invalid.
kStatus_SDHC_Prepare-AdmaDescriptorFailed	Prepare ADMA descriptor failed.
kStatus_SDHC_Send-CommandFailed	Send command failed.
kStatus_SDHC_Transfer-DataFailed	Transfer data failed.
kStatus_Success	Operate successfully.

37.7.33 void SDHC_TransferCreateHandle (SDHC_Type ∗ *base,* sdhc_handle_t ∗ *handle,* const sdhc_transfer_callback_t ∗ *callback,* void ∗ *userData*)

Parameters

base	SDHC peripheral base address.
handle	SDHC handle pointer.
callback	Structure pointer to contain all callback functions.
userData	Callback function parameter.

37.7.34 status_t SDHC_TransferNonBlocking (SDHC_Type ∗ *base,* sdhc_handle_t ∗ *handle,* uint32_t ∗ *admaTable,* uint32_t *admaTableWords,* sdhc_transfer_t ∗ *transfer*)

This function sends a command and data and returns immediately. It doesn't wait the transfer complete or encounter an error. This function support non word align data addr transfer support, if data buffer addr is not align in DMA mode, the API will continue finish the transfer by polling IO directly The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note

Call the API 'SDHC_TransferCreateHandle' when calling this API.

Parameters

base	SDHC peripheral base address.
handle	SDHC handle.
admaTable	ADMA table address, can't be null if transfer way is ADMA1/ADMA2.
admaTable-Words	ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.
transfer	Transfer content.

Return values

kStatus_InvalidArgument	Argument is invalid.
kStatus_SDHC_Busy-Transferring	Busy transferring.
kStatus_SDHC_Prepare-AdmaDescriptorFailed	Prepare ADMA descriptor failed.
kStatus_Success	Operate successfully.

37.7.35 void SDHC_TransferHandleIRQ (SDHC_Type ∗ *base*, sdhc_handle_t ∗ *handle*)

This function deals with the IRQs on the given host controller.

Parameters

base	SDHC peripheral base address.
handle	SDHC handle.

Chapter 38
SDRAMC: Synchronous DRAM Controller Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the Synchronous DRAM Controller block of MCUXpresso SDK devices.

SDRAMC: Synchronous DRAM Controller Driver

38.2.1 SDRAM Controller Basic Operation

The SDRAM controller commands include the initialization MRS command, precharge command, enter/exit self-refresh command, and enable/disable auto-refresh command. Use the SDRAMC_SendCommand() to send these commands to SDRAM to initialize it. The SDRAMC_EnableWriteProtect() is provided to enable/disable the write protection. The SDRAMC_EnableOperateValid() is provided to enable/disable the operation valid.

Typical use case

This example shows how to use the SDRAM Controller driver to initialize the external 16 bit port-size 8-column SDRAM chip. Initialize the SDRAM controller and run the initialization sequence. The external SDRAM is initialized and the SDRAM read and write is available.

First, initialize the SDRAM Controller. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/sdramc Then, run the initialization sequence.

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

Data Structures

- struct sdramc_blockctl_config_t
  SDRAM controller block control configuration structure. More...
- struct sdramc_refresh_config_t
  SDRAM controller refresh timing configuration structure. More...
- struct sdramc_config_t
  SDRAM controller configuration structure. More...

Enumerations

- enum sdramc_refresh_time_t {
  kSDRAMC_RefreshThreeClocks = 0x0U,
  kSDRAMC_RefreshSixClocks,
  kSDRAMC_RefreshNineClocks }
  SDRAM controller auto-refresh timing.

- enum sdramc_latency_t {
kSDRAMC_LatencyZero = 0x0U,
kSDRAMC_LatencyOne,
kSDRAMC_LatencyTwo,
kSDRAMC_LatencyThree }
Setting latency for SDRAM controller timing specifications.
- enum sdramc_command_bit_location_t {
kSDRAMC_Commandbit17 = 0x0U,
kSDRAMC_Commandbit18,
kSDRAMC_Commandbit19,
kSDRAMC_Commandbit20,
kSDRAMC_Commandbit21,
kSDRAMC_Commandbit22,
kSDRAMC_Commandbit23,
kSDRAMC_Commandbit24 }
SDRAM controller command bit location.
- enum sdramc_command_t {
kSDRAMC_ImrsCommand = 0x0U,
kSDRAMC_PrechargeCommand,
kSDRAMC_SelfrefreshEnterCommand,
kSDRAMC_SelfrefreshExitCommand,
kSDRAMC_AutoRefreshEnableCommand,
kSDRAMC_AutoRefreshDisableCommand }
SDRAM controller command.
- enum sdramc_port_size_t {
kSDRAMC_PortSize32Bit = 0x0U,
kSDRAMC_PortSize8Bit,
kSDRAMC_PortSize16Bit }
SDRAM port size.
- enum sdramc_block_selection_t {
kSDRAMC_Block0 = 0x0U,
kSDRAMC_Block1 }
SDRAM controller block selection.

Driver version

- #define FSL_SDRAMC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
SDRAMC driver version 2.1.1.

SDRAM Controller Initialization and De-initialization

- void SDRAMC_Init (SDRAM_Type ∗base, sdramc_config_t ∗configure)
Initializes the SDRAM controller.
- void SDRAMC_Deinit (SDRAM_Type ∗base)
Deinitializes the SDRAM controller module and gates the clock.

SDRAM Controller Basic Operation

- void SDRAMC_SendCommand (SDRAM_Type *base, sdramc_block_selection_t block, sdramc_command_t command)
  Sends the SDRAM command.
- static void SDRAMC_EnableWriteProtect (SDRAM_Type *base, sdramc_block_selection_t block, bool enable)
  Enables/disables the write protection.
- static void SDRAMC_EnableOperateValid (SDRAM_Type *base, sdramc_block_selection_t block, bool enable)
  Enables/disables the valid operation.

Data Structure Documentation

38.4.1 struct sdramc_blockctl_config_t

Data Fields

- sdramc_block_selection_t block
  The block number.
- sdramc_port_size_t portSize
  The port size of the associated SDRAM block.
- sdramc_command_bit_location_t location
  The command bit location.
- sdramc_latency_t latency
  The latency for some timing specifications.
- uint32_t address
  The base address of the SDRAM block.
- uint32_t addressMask
  The base address mask of the SDRAM block.

38.4.1.0.0.32 Field Documentation

38.4.1.0.0.32.1 sdramc_block_selection_t sdramc_blockctl_config_t::block

38.4.1.0.0.32.2 sdramc_port_size_t sdramc_blockctl_config_t::portSize

38.4.1.0.0.32.3 sdramc_command_bit_location_t sdramc_blockctl_config_t::location

38.4.1.0.0.32.4 sdramc_latency_t sdramc_blockctl_config_t::latency

38.4.1.0.0.32.5 uint32_t sdramc_blockctl_config_t::address

38.4.1.0.0.32.6 uint32_t sdramc_blockctl_config_t::addressMask

38.4.2 struct sdramc_refresh_config_t

Data Fields

- sdramc_refresh_time_t refreshTime
  Trc:The number of bus clocks inserted between a REF and next ACTIVE command.
- uint32_t sdramRefreshRow
  The SDRAM refresh time each row: ns/row.
- uint32_t busClock_Hz
  The bus clock for SDRAMC.

38.4.2.0.0.33 Field Documentation

38.4.2.0.0.33.1 sdramc_refresh_time_t sdramc_refresh_config_t::refreshTime

38.4.2.0.0.33.2 uint32_t sdramc_refresh_config_t::sdramRefreshRow

38.4.2.0.0.33.3 uint32_t sdramc_refresh_config_t::busClock_Hz

38.4.3 struct sdramc_config_t

Defines a configure structure and uses the SDRAMC_Configure() function to make necessary initializations.

Data Fields

- sdramc_refresh_config_t ∗ refreshConfig
  Refresh timing configure structure pointer.
- sdramc_blockctl_config_t ∗ blockConfig
  Block configure structure pointer.
- uint8_t numBlockConfig
  SDRAM block numbers for configuration.

38.4.3.0.0.34 Field Documentation

38.4.3.0.0.34.1 sdramc_refresh_config_t* sdramc_config_t::refreshConfig

38.4.3.0.0.34.2 sdramc_blockctl_config_t* sdramc_config_t::blockConfig

If both SDRAM blocks are used, use the two continuous blockConfig.

38.4.3.0.0.34.3 uint8_t sdramc_config_t::numBlockConfig

Macro Definition Documentation

38.5.1 #define FSL_SDRAMC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

Enumeration Type Documentation

38.6.1 enum sdramc_refresh_time_t

Enumerator

kSDRAMC_RefreshThreeClocks The refresh timing with three bus clocks.
kSDRAMC_RefreshSixClocks The refresh timing with six bus clocks.
kSDRAMC_RefreshNineClocks The refresh timing with nine bus clocks.

38.6.2 enum sdramc_latency_t

The latency setting affects the following SDRAM timing specifications:

- trcd: SRAS assertion to SCAS assertion
- tcasl: SCAS assertion to data out
- tras: ACTV command to Precharge command
- trp: Precharge command to ACTV command
- trwl, trdl: Last data input to Precharge command
- tep: Last data out to Precharge command
  The details of the latency setting and timing specifications are shown in the following table list.
  latency trcd: tcasl tras trp trwl,trdl tep
  0 1 bus clock 1 bus clock 2 bus clocks 1 bus clock 1 bus clock 1 bus clock
  1 2 bus clock 2 bus clock 4 bus clocks 2 bus clock 1 bus clock 1 bus clock
  2 3 bus clock 3 bus clock 6 bus clocks 3 bus clock 1 bus clock 1 bus clock
  3 3 bus clock 3 bus clock 6 bus clocks 3 bus clock 1 bus clock 1 bus clock

Enumerator

kSDRAMC_LatencyZero Latency 0.
kSDRAMC_LatencyOne Latency 1.
kSDRAMC_LatencyTwo Latency 2.
kSDRAMC_LatencyThree Latency 3.

38.6.3 enum sdramc_command_bit_location_t

Enumerator

kSDRAMC_Commandbit17 Command bit location is bit 17.
kSDRAMC_Commandbit18 Command bit location is bit 18.
kSDRAMC_Commandbit19 Command bit location is bit 19.
kSDRAMC_Commandbit20 Command bit location is bit 20.
kSDRAMC_Commandbit21 Command bit location is bit 21.
kSDRAMC_Commandbit22 Command bit location is bit 22.
kSDRAMC_Commandbit23 Command bit location is bit 23.
kSDRAMC_Commandbit24 Command bit location is bit 24.

38.6.4 enum sdramc_command_t

Enumerator

kSDRAMC_ImrsCommand Initiate MRS command.
kSDRAMC_PrechargeCommand Initiate precharge command.
kSDRAMC_SelfrefreshEnterCommand Enter self-refresh command.
kSDRAMC_SelfrefreshExitCommand Exit self-refresh command.
kSDRAMC_AutoRefreshEnableCommand Enable Auto refresh command.
kSDRAMC_AutoRefreshDisableCommand Disable Auto refresh command.

38.6.5 enum sdramc_port_size_t

Enumerator

kSDRAMC_PortSize32Bit 32-Bit port size.
kSDRAMC_PortSize8Bit 8-Bit port size.
kSDRAMC_PortSize16Bit 16-Bit port size.

38.6.6 enum sdramc_block_selection_t

Enumerator

kSDRAMC_Block0 Select SDRAM block 0.
kSDRAMC_Block1 Select SDRAM block 1.

Function Documentation

38.7.1 void SDRAMC_Init (SDRAM_Type * *base*, sdramc_config_t * *configure*)

This function ungates the SDRAM controller clock and initializes the SDRAM controller. This function must be called before calling any other SDRAM controller driver functions. Example

```
sdramc_refresh_config_t refreshConfig;
sdramc_blockctl_config_t blockConfig;
sdramc_config_t config;

refreshConfig.refreshTime  = kSDRAM_RefreshThreeClocks;
refreshConfig.sdramRefreshRow = 15625;
refreshConfig.busClock = 60000000;

blockConfig.block = kSDRAMC_Block0;
blockConfig.portSize = kSDRAMC_PortSize16Bit;
blockConfig.location = kSDRAMC_Commandbit19;
blockConfig.latency = kSDRAMC_RefreshThreeClocks;
blockConfig.address = SDRAM_START_ADDRESS;
blockConfig.addressMask = 0x7c0000;

config.refreshConfig = &refreshConfig,
config.blockConfig = &blockConfig,
config.totalBlocks = 1;

SDRAMC_Init(SDRAM, &config);
```

Parameters

base	SDRAM controller peripheral base address.
configure	The SDRAM configuration structure pointer.

38.7.2 void SDRAMC_Deinit (SDRAM_Type * *base*)

This function gates the SDRAM controller clock. As a result, the SDRAM controller module doesn't work after calling this function.

Parameters

base	SDRAM controller peripheral base address.

38.7.3 void SDRAMC_SendCommand (SDRAM_Type * *base*, sdramc_block_selection_t *block*, sdramc_command_t *command*)

This function sends commands to SDRAM. The commands are precharge command, initialization MR-S command, auto-refresh enable/disable command, and self-refresh enter/exit commands. Note that the self-refresh enter/exit commands are all blocks setting and "block" is ignored. Ensure to set the correct "block" when send other commands.

Parameters

base	SDRAM controller peripheral base address.
block	The block selection.
command	The SDRAM command, see "sdramc_command_t". kSDRAMC_ImrsCommand - Initialize MRS command kSDRAMC_PrechargeCommand - Initialize precharge command kSDRAMC_SelfrefreshEnterCommand - Enter self-refresh command kSDRAMC_SelfrefreshExitCommand - Exit self-refresh command kSDRAMC_AutoRefreshEnableCommand - Enable auto refresh command kSDRAMC_AutoRefreshDisableCommand - Disable auto refresh command

38.7.4 static void SDRAMC_EnableWriteProtect (SDRAM_Type * *base*, sdramc_block_selection_t *block*, bool *enable*) [inline], [static]

Parameters

base	SDRAM peripheral base address.
block	The block which is selected.
enable	True enable write protection, false disable write protection.

38.7.5 static void SDRAMC_EnableOperateValid (SDRAM_Type * *base*, sdramc_block_selection_t *block*, bool *enable*) [inline], [static]

Parameters

base	SDRAM peripheral base address.
block	The block which is selected.
enable	True enable the valid operation; false disable the valid operation.

Chapter 39
SIM: System Integration Module Driver

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_usb_volt_reg_enable_mode {
  kSIM_UsbVoltRegEnable = (int)SIM_SOPT1_USBREGEN_MASK,
  kSIM_UsbVoltRegEnableInLowPower = SIM_SOPT1_USBVSTBY_MASK,
  kSIM_UsbVoltRegEnableInStop = SIM_SOPT1_USBSSTBY_MASK,
  kSIM_UsbVoltRegEnableInAllModes }
  USB voltage regulator enable setting.
- enum _sim_flash_mode {
  kSIM_FlashDisableInWait = SIM_FCFG1_FLASHDOZE_MASK,
  kSIM_FlashDisable = SIM_FCFG1_FLASHDIS_MASK }
  Flash enable mode.

Functions

- void SIM_SetUsbVoltRegulatorEnableMode (uint32_t mask)
  Sets the USB voltage regulator setting.
- 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, 2))

Data Structure Documentation

39.2.1 struct sim_uid_t

Data Fields

- uint32_t MH

UIDMH.

- uint32_t ML

  UIDML.
- uint32_t L

  UIDL.

39.2.1.0.0.35 Field Documentation

39.2.1.0.0.35.1 uint32_t sim_uid_t::MH

39.2.1.0.0.35.2 uint32_t sim_uid_t::ML

39.2.1.0.0.35.3 uint32_t sim_uid_t::L

Enumeration Type Documentation

39.3.1 enum _sim_usb_volt_reg_enable_mode

Enumerator

kSIM_UsbVoltRegEnable Enable voltage regulator.
kSIM_UsbVoltRegEnableInLowPower Enable voltage regulator in VLPR/VLPW modes.
kSIM_UsbVoltRegEnableInStop Enable voltage regulator in STOP/VLPS/LLS/VLLS modes.
kSIM_UsbVoltRegEnableInAllModes Enable voltage regulator in all power modes.

39.3.2 enum _sim_flash_mode

Enumerator

kSIM_FlashDisableInWait Disable flash in wait mode.
kSIM_FlashDisable Disable flash in normal mode.

Function Documentation

39.4.1 void SIM_SetUsbVoltRegulatorEnableMode (uint32_t *mask*)

This function configures whether the USB voltage regulator is enabled in normal RUN mode, STOP/-VLPS/LLS/VLLS modes, and VLPR/VLPW modes. The configurations are passed in as mask value of _sim_usb_volt_reg_enable_mode. For example, to enable USB voltage regulator in RUN/VLPR/VLPW modes and disable in STOP/VLPS/LLS/VLLS mode, use:

SIM_SetUsbVoltRegulatorEnableMode(kSIM_UsbVoltRegEnable | kSIM_UsbVoltRegEnableInLow-Power);

Parameters

mask	USB voltage regulator enable setting.

39.4.2 void SIM_GetUniqueId (sim_uid_t * *uid*)

Parameters

uid	Pointer to the structure to save the UID value.

39.4.3 static void SIM_SetFlashMode (uint8_t *mode*) [inline], [static]

Parameters

mode	The mode to set; see _sim_flash_mode for mode details.

Chapter 40
SMC: System Mode Controller Driver

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.

Typical use case

40.2.1 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_lls_config_t
  SMC Low-Leakage Stop power mode configuration. More...
- 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_AllowPowerModeLls = SMC_PMPROT_ALLS_MASK,
  kSMC_AllowPowerModeVlp = SMC_PMPROT_AVLP_MASK,
  kSMC_AllowPowerModeHsrun = SMC_PMPROT_AHSRUN_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_PowerStateLls = 0x01U << 5U,
  kSMC_PowerStateVlls = 0x01U << 6U,
  kSMC_PowerStateHsrun = 0x01U << 7U }
  Power Modes in PMSTAT.
- enum smc_run_mode_t {
  kSMC_RunNormal = 0U,
  kSMC_RunVlpr = 2U,
  kSMC_Hsrun = 3U }
  Run mode definition.
- enum smc_stop_mode_t {
  kSMC_StopNormal = 0U,
  kSMC_StopVlps = 2U,
  kSMC_StopLls = 3U,
  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_SetPowerModeHsrun (SMC_Type ∗base)
  Configures the system to HSRUN 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_SetPowerModeLls (SMC_Type ∗base, const smc_power_mode_lls_config_t ∗config)
  Configures the system to LLS power mode.
- status_t SMC_SetPowerModeVlls (SMC_Type ∗base, const smc_power_mode_vlls_config_t ∗config)
  Configures the system to VLLS power mode.

Data Structure Documentation

40.3.1 struct smc_power_mode_lls_config_t

Data Fields

- smc_stop_submode_t subMode
  Low-leakage Stop sub-mode.

40.3.2 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.
- bool enableRam2InVlls2
  Enable RAM2 power in VLLS2.

Enumeration Type Documentation

40.4.1 enum smc_power_mode_protection_t

Enumerator

kSMC_AllowPowerModeVlls Allow Very-low-leakage Stop Mode.
kSMC_AllowPowerModeLls Allow Low-leakage Stop Mode.
kSMC_AllowPowerModeVlp Allow Very-Low-power Mode.
kSMC_AllowPowerModeHsrun Allow High-speed Run mode.
kSMC_AllowPowerModeAll Allow all power mode.

40.4.2 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_PowerStateLls 0010_0000 - Current power mode is LLS
kSMC_PowerStateVlls 0100_0000 - Current power mode is VLLS
kSMC_PowerStateHsrun 1000_0000 - Current power mode is HSRUN

40.4.3 enum smc_run_mode_t

Enumerator

kSMC_RunNormal Normal RUN mode.
kSMC_RunVlpr Very-low-power RUN mode.
kSMC_Hsrun High-speed Run mode (HSRUN).

40.4.4 enum smc_stop_mode_t

Enumerator

kSMC_StopNormal Normal STOP mode.
kSMC_StopVlps Very-low-power STOP mode.
kSMC_StopLls Low-leakage Stop mode.
kSMC_StopVlls Very-low-leakage Stop mode.

40.4.5 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.

40.4.6 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.

40.4.7 anonymous enum

Enumerator

kStatus_SMC_StopAbort Entering Stop mode is abort.

Function Documentation

40.5.1 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.

40.5.2 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.

40.5.3 void SMC_PreEnterStopModes (void)

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

40.5.4 void SMC_PostExitStopModes (void)

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used with SMC-_PreEnterStopModes.

40.5.5 void SMC_PreEnterWaitModes (void)

This function should be called before entering WAIT/VLPW modes.

40.5.6 void SMC_PostExitWaitModes (void)

This function should be called after wake up from WAIT/VLPW modes. It is used with SMC_PreEnter-WaitModes.

40.5.7 status_t SMC_SetPowerModeRun (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

40.5.8 status_t SMC_SetPowerModeHsrun (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

40.5.9 status_t SMC_SetPowerModeWait (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

40.5.10 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.

40.5.11 status_t SMC_SetPowerModeVlpr (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

40.5.12 status_t SMC_SetPowerModeVlpw (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

40.5.13 status_t SMC_SetPowerModeVlps (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

40.5.14 status_t SMC_SetPowerModeLls (SMC_Type * *base*, const smc_power_mode_lls_config_t * *config*)

Parameters

base	SMC peripheral base address.
config	The LLS power mode configuration structure

Returns

SMC configuration error code.

40.5.15 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 41
SYSMPU: System Memory Protection Unit

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.

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

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.
- void SYSMPU_SetRegionRwMasterAccessRights (SYSMPU_Type *base, uint32_t regionNum, uint32_t masterNum, const sysmpu_rwrights_master_access_control_t *accessRights)
  Sets the SYSMPU region access rights for masters with read and write 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.

Data Structure Documentation

41.4.1 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.

41.4.1.0.0.36 Field Documentation

41.4.1.0.0.36.1 uint8_t sysmpu_hardware_info_t::hardwareRevisionLevel

41.4.1.0.0.36.2 uint8_t sysmpu_hardware_info_t::slavePortsNumbers

41.4.1.0.0.36.3 sysmpu_region_total_num_t sysmpu_hardware_info_t::regionsNumbers

41.4.2 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.

41.4.2.0.0.37 Field Documentation

41.4.2.0.0.37.1 uint32_t sysmpu_access_err_info_t::master

41.4.2.0.0.37.2 sysmpu_err_attributes_t sysmpu_access_err_info_t::attributes

41.4.2.0.0.37.3 sysmpu_err_access_type_t sysmpu_access_err_info_t::accessType

41.4.2.0.0.37.4 sysmpu_err_access_control_t sysmpu_access_err_info_t::accessControl

41.4.2.0.0.37.5 uint32_t sysmpu_access_err_info_t::address

41.4.2.0.0.37.6 uint8_t sysmpu_access_err_info_t::processorIdentification

41.4.3 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.

41.4.3.0.0.38 Field Documentation

41.4.3.0.0.38.1 sysmpu_supervisor_access_rights_t sysmpu_rwxrights_master_access_control_t::superAccessRights

41.4.3.0.0.38.2 sysmpu_user_access_rights_t sysmpu_rwxrights_master_access_control_t::userAccessRights

41.4.3.0.0.38.3 bool sysmpu_rwxrights_master_access_control_t::processIdentifierEnable

41.4.4 struct sysmpu_rwrights_master_access_control_t

Data Fields

- bool writeEnable
  Enables or disables write permission.
- bool readEnable
  Enables or disables read permission.

41.4.4.0.0.39 Field Documentation

41.4.4.0.0.39.1 bool sysmpu_rwrights_master_access_control_t::writeEnable

41.4.4.0.0.39.2 bool sysmpu_rwrights_master_access_control_t::readEnable

41.4.5 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.

41.4.5.0.0.40 Field Documentation

41.4.5.0.0.40.1 uint32_t sysmpu_region_config_t::regionNum

41.4.5.0.0.40.2 uint32_t sysmpu_region_config_t::startAddress

Note: bit0 $\sim$ bit4 always be marked as 0 by SYSMPU. The actual start address is 0-modulo-32 byte address.

41.4.5.0.0.40.3 uint32_t sysmpu_region_config_t::endAddress

Note: bit0 $\sim$ bit4 always be marked as 1 by SYSMPU. The actual end address is 31-modulo-32 byte address.

41.4.5.0.0.40.4 sysmpu_rwxrights_master_access_control_t sysmpu_region_config_t::accessRights1[4]

41.4.5.0.0.40.5 sysmpu_rwrights_master_access_control_t sysmpu_region_config_t::accessRights2[4]

41.4.5.0.0.40.6 uint8_t sysmpu_region_config_t::processIdentifier

41.4.5.0.0.40.7 uint8_t sysmpu_region_config_t::processIdMask

The setting bit will ignore the same bit in process identifier.

41.4.6 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.

41.4.6.0.0.41 Field Documentation

41.4.6.0.0.41.1 sysmpu_region_config_t sysmpu_config_t::regionConfig

41.4.6.0.0.41.2 struct _sysmpu_config* sysmpu_config_t::next

Macro Definition Documentation

41.5.1 #define FSL_SYSMPU_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))

41.5.2 #define SYSMPU_MASTER_RWATTRIBUTE_START_PORT (4U)

41.5.3 #define SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(*n*) ((n)*6U)

41.5.4 #define SYSMPU_REGION_RWXRIGHTS_MASTER_MASK(*n*) (0x1FUL << SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))

41.5.5 #define SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH 5U

41.5.6 #define SYSMPU_REGION_RWXRIGHTS_MASTER(*n,* *x*) (((uint32_t)(((uint32_t)(x)) << SYSMPU_REGION_RWXRIGHTS_MASTER_SHIFT(n))) & SYSMPU_REGION_RWXRIGHTS_MASTER_MASK(n))

41.5.7 #define SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(*n*) ((n)*6U + SYSMPU_REGION_RWXRIGHTS_MASTER_WIDTH)

41.5.8 #define SYSMPU_REGION_RWXRIGHTS_MASTER_PE_MASK(*n*) (0x1UL << SYSMPU_REGION_RWXRIGHTS_MASTER_PE_SHIFT(n))

41.5.9 #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))
```

41.5.10 #define SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(*n*) (((n)-SYSMPU_MASTER_RWATTRIBUTE_START_PORT) * 2U + 24U)

41.5.11 #define SYSMPU_REGION_RWRIGHTS_MASTER_MASK(*n*) (0x3UL << SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n))

41.5.12 #define SYSMPU_REGION_RWRIGHTS_MASTER(*n*, *x*) (((uint32_t)(((uint32_t)(x)) << SYSMPU_REGION_RWRIGHTS_MASTER_SHIFT(n))) & SYSMPU_REGION_RWRIGHTS_MASTER_MASK(n))

Enumeration Type Documentation

41.6.1 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.

41.6.2 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.

41.6.3 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.

41.6.4 enum sysmpu_err_access_type_t

Enumerator

kSYSMPU_ErrTypeRead SYSMPU error access type — read.
kSYSMPU_ErrTypeWrite SYSMPU error access type — write.

41.6.5 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.

41.6.6 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.

41.6.7 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.

Function Documentation

41.7.1 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.

41.7.2 void SYSMPU_Deinit (SYSMPU_Type ∗ *base*)

Parameters

base	SYSMPU peripheral base address.

41.7.3 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.

41.7.4 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.

41.7.5 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".

41.7.6 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".

41.7.7 void SYSMPU_SetRegionAddr (SYSMPU_Type ∗ *base*, uint32_t *regionNum*, uint32_t *startAddr*, uint32_t *endAddr*)

Memory region start address. Note: bit0 $\sim$ 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 $\sim$ 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.

41.7.8 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".

41.7.9 void SYSMPU_SetRegionRwMasterAccessRights (SYSMPU_Type * *base*, uint32_t *regionNum*, uint32_t *masterNum*, const sysmpu_rwrights_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 normal rights masters only have the read and write access permissions. The privilege rights access control see "SYSMPU_SetRegionRwxMasterAccessRights".

Parameters

base	SYSMPU peripheral base address.
regionNum	SYSMPU region number. The range is from 0 to FSL_FEATURE_SYSMPU_DESCRIPTOR_COUNT - 1.
masterNum	SYSMPU bus master number. Should range from SYSMPU_MASTER_RWATTRIBUTE_START_PORT to ~ FSL_FEATURE_SYSMPU_MASTER_COUNT - 1.

accessRights	The pointer to the SYSMPU access rights configuration. See "sysmpu_rwrights_-master_access_control_t".

41.7.10 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.

41.7.11 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 42
TPM: Timer PWM Module

Overview

The MCUXpresso SDK provides a driver for the Timer PWM Module (TPM) of MCUXpresso SDK devices.

The TPM driver supports the generation of PWM signals, input capture, and output compare modes. On some SoCs, the driver supports the generation of combined PWM signals, dual-edge capture, and quadrature decoder modes. The driver also supports configuring each of the TPM fault inputs. The fault input is available only on some SoCs.

Introduction of TPM

42.2.1 Initialization and deinitialization

The function TPM_Init() initializes the TPM with a specified configurations. The function TPM_GetDefaultConfig() gets the default configurations. On some SoCs, the initialization function issues a software reset to reset the TPM internal logic. The initialization function configures the TPM's behavior when it receives a trigger input and its operation in doze and debug modes.

The function TPM_Deinit() disables the TPM counter and turns off the module clock.

42.2.2 PWM Operations

The function TPM_SetupPwm() sets up TPM channels for the PWM output. The function can set up the PWM signal properties for multiple channels. Each channel has its own tpm_chnl_pwm_signal_param_t structure that is used to specify the output signals duty cycle and level-mode. However, the same PWM period and PWM mode is applied to all channels requesting a PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100 where 0=inactive signal (0% duty cycle) and 100=always active signal (100% duty cycle). When generating a combined PWM signal, the channel number passed refers to a channel pair number, for example 0 refers to channel 0 and 1, 1 refers to channels 2 and 3.

The function TPM_UpdatePwmDutycycle() updates the PWM signal duty cycle of a particular TPM channel.

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

42.2.3 Input capture operations

The function TPM_SetupInputCapture() sets up a TPM channel for input capture. The user can specify the capture edge.

The function TPM_SetupDualEdgeCapture() can be used to measure the pulse width of a signal. This is available only for certain SoCs. A channel pair is used during the capture with the input signal coming through a channel that can be configured. The user can specify the capture edge for each channel and any filter value to be used when processing the input signal.

42.2.4 Output compare operations

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

42.2.5 Quad decode

The function TPM_SetupQuadDecode() sets up TPM channels 0 and 1 for quad decode, which is available only for certain SoCs. The user can specify the quad decode mode, polarity, and filter properties for each input signal.

42.2.6 Fault operation

The function TPM_SetupFault() sets up the properties for each fault, which is available only for certain SoCs. The user can specify the fault polarity and whether to use a filter on a fault input. The overall fault filter value and fault control mode are set up during initialization.

42.2.7 Status

Provides functions to get and clear the TPM status.

42.2.8 Interrupt

Provides functions to enable/disable TPM interrupts and get current enabled interrupts.

Typical use case

42.3.1 PWM output

Output the PWM signal on 2 TPM channels with different duty cycles. Periodically update the PWM signal duty cycle. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/tpm

Data Structures

- struct tpm_chnl_pwm_signal_param_t
  Options to configure a TPM channel's PWM signal. More...
- struct tpm_dual_edge_capture_param_t
  TPM dual edge capture parameters. More...
- struct tpm_phase_params_t
  TPM quadrature decode phase parameters. More...
- struct tpm_config_t
  TPM config structure. More...

Enumerations

- enum tpm_chnl_t {
  kTPM_Chnl_0 = 0U,
  kTPM_Chnl_1,
  kTPM_Chnl_2,
  kTPM_Chnl_3,
  kTPM_Chnl_4,
  kTPM_Chnl_5,
  kTPM_Chnl_6,
  kTPM_Chnl_7 }
  List of TPM channels.
- enum tpm_pwm_mode_t {
  kTPM_EdgeAlignedPwm = 0U,
  kTPM_CenterAlignedPwm,
  kTPM_CombinedPwm }
  TPM PWM operation modes.
- enum tpm_pwm_level_select_t {
  kTPM_NoPwmSignal = 0U,
  kTPM_LowTrue,
  kTPM_HighTrue }
  TPM PWM output pulse mode: high-true, low-true or no output.
- enum tpm_trigger_select_t
  Trigger options available.
- enum tpm_trigger_source_t {
  kTPM_TriggerSource_External = 0U,
  kTPM_TriggerSource_Internal }
  Trigger source options available.

- enum tpm_output_compare_mode_t {
  kTPM_NoOutputSignal = (1U << TPM_CnSC_MSA_SHIFT),
  kTPM_ToggleOnMatch = ((1U << TPM_CnSC_MSA_SHIFT) | (1U << TPM_CnSC_ELSA_SHIFT)),
  kTPM_ClearOnMatch = ((1U << TPM_CnSC_MSA_SHIFT) | (2U << TPM_CnSC_ELSA_SHIFT)),
  kTPM_SetOnMatch = ((1U << TPM_CnSC_MSA_SHIFT) | (3U << TPM_CnSC_ELSA_SHIFT)),
  kTPM_HighPulseOutput = ((3U << TPM_CnSC_MSA_SHIFT) | (1U << TPM_CnSC_ELSA_SHIFT)),
  kTPM_LowPulseOutput = ((3U << TPM_CnSC_MSA_SHIFT) | (2U << TPM_CnSC_ELSA_SHIFT)) }
  TPM output compare modes.
- enum tpm_input_capture_edge_t {
  kTPM_RisingEdge = (1U << TPM_CnSC_ELSA_SHIFT),
  kTPM_FallingEdge = (2U << TPM_CnSC_ELSA_SHIFT),
  kTPM_RiseAndFallEdge = (3U << TPM_CnSC_ELSA_SHIFT) }
  TPM input capture edge.
- enum tpm_quad_decode_mode_t {
  kTPM_QuadPhaseEncode = 0U,
  kTPM_QuadCountAndDir }
  TPM quadrature decode modes.
- enum tpm_phase_polarity_t {
  kTPM_QuadPhaseNormal = 0U,
  kTPM_QuadPhaseInvert }
  TPM quadrature phase polarities.
- enum tpm_clock_source_t {
  kTPM_SystemClock = 1U,
  kTPM_ExternalClock }
  TPM clock source selection.
- enum tpm_clock_prescale_t {
  kTPM_Prescale_Divide_1 = 0U,
  kTPM_Prescale_Divide_2,
  kTPM_Prescale_Divide_4,
  kTPM_Prescale_Divide_8,
  kTPM_Prescale_Divide_16,
  kTPM_Prescale_Divide_32,
  kTPM_Prescale_Divide_64,
  kTPM_Prescale_Divide_128 }
  TPM prescale value selection for the clock source.
- enum tpm_interrupt_enable_t {

kTPM_Chnl0InterruptEnable = (1U << 0),
kTPM_Chnl1InterruptEnable = (1U << 1),
kTPM_Chnl2InterruptEnable = (1U << 2),
kTPM_Chnl3InterruptEnable = (1U << 3),
kTPM_Chnl4InterruptEnable = (1U << 4),
kTPM_Chnl5InterruptEnable = (1U << 5),
kTPM_Chnl6InterruptEnable = (1U << 6),
kTPM_Chnl7InterruptEnable = (1U << 7),
kTPM_TimeOverflowInterruptEnable = (1U << 8) }
List of TPM interrupts.
- enum tpm_status_flags_t {
kTPM_Chnl0Flag = (1U << 0),
kTPM_Chnl1Flag = (1U << 1),
kTPM_Chnl2Flag = (1U << 2),
kTPM_Chnl3Flag = (1U << 3),
kTPM_Chnl4Flag = (1U << 4),
kTPM_Chnl5Flag = (1U << 5),
kTPM_Chnl6Flag = (1U << 6),
kTPM_Chnl7Flag = (1U << 7),
kTPM_TimeOverflowFlag = (1U << 8) }
List of TPM flags.

Driver version

- #define FSL_TPM_DRIVER_VERSION (MAKE_VERSION(2, 0, 8))
Version 2.0.8.

Initialization and deinitialization

- void TPM_Init (TPM_Type ∗base, const tpm_config_t ∗config)
Ungates the TPM clock and configures the peripheral for basic operation.
- void TPM_Deinit (TPM_Type ∗base)
Stops the counter and gates the TPM clock.
- void TPM_GetDefaultConfig (tpm_config_t ∗config)
Fill in the TPM config struct with the default settings.

Channel mode operations

- status_t TPM_SetupPwm (TPM_Type ∗base, const tpm_chnl_pwm_signal_param_t ∗chnlParams, uint8_t numOfChnls, tpm_pwm_mode_t mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)
Configures the PWM signal parameters.
- void TPM_UpdatePwmDutycycle (TPM_Type ∗base, tpm_chnl_t chnlNumber, tpm_pwm_mode_t currentPwmMode, uint8_t dutyCyclePercent)
Update the duty cycle of an active PWM signal.
- void TPM_UpdateChnlEdgeLevelSelect (TPM_Type ∗base, tpm_chnl_t chnlNumber, uint8_t level)
Update the edge level selection for a channel.
- void TPM_SetupInputCapture (TPM_Type ∗base, tpm_chnl_t chnlNumber, tpm_input_capture_edge_t captureMode)

Enables capturing an input signal on the channel using the function parameters.
- void TPM_SetupOutputCompare (TPM_Type ∗base, tpm_chnl_t chnlNumber, tpm_output_compare_mode_t compareMode, uint32_t compareValue)
  Configures the TPM to generate timed pulses.
- void TPM_SetupDualEdgeCapture (TPM_Type ∗base, tpm_chnl_t chnlPairNumber, const tpm_dual_edge_capture_param_t ∗edgeParam, uint32_t filterValue)
  Configures the dual edge capture mode of the TPM.
- void TPM_SetupQuadDecode (TPM_Type ∗base, const tpm_phase_params_t ∗phaseAParams, const tpm_phase_params_t ∗phaseBParams, tpm_quad_decode_mode_t quadMode)
  Configures the parameters and activates the quadrature decode mode.

Interrupt Interface

- void TPM_EnableInterrupts (TPM_Type ∗base, uint32_t mask)
  Enables the selected TPM interrupts.
- void TPM_DisableInterrupts (TPM_Type ∗base, uint32_t mask)
  Disables the selected TPM interrupts.
- uint32_t TPM_GetEnabledInterrupts (TPM_Type ∗base)
  Gets the enabled TPM interrupts.

Status Interface

- static uint32_t TPM_GetStatusFlags (TPM_Type ∗base)
  Gets the TPM status flags.
- static void TPM_ClearStatusFlags (TPM_Type ∗base, uint32_t mask)
  Clears the TPM status flags.

Read and write the timer period

- static void TPM_SetTimerPeriod (TPM_Type ∗base, uint32_t ticks)
  Sets the timer period in units of ticks.
- static uint32_t TPM_GetCurrentTimerCount (TPM_Type ∗base)
  Reads the current timer counting value.

Timer Start and Stop

- static void TPM_StartTimer (TPM_Type ∗base, tpm_clock_source_t clockSource)
  Starts the TPM counter.
- static void TPM_StopTimer (TPM_Type ∗base)
  Stops the TPM counter.

Data Structure Documentation

42.4.1 struct tpm_chnl_pwm_signal_param_t

Data Fields

- tpm_chnl_t chnlNumber
  TPM channel to configure.

- tpm_pwm_level_select_t level
  PWM output active level select.
- uint8_t dutyCyclePercent
  PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...
- uint8_t firstEdgeDelayPercent
  Used only in combined PWM mode to generate asymmetrical PWM.

42.4.1.0.0.42 Field Documentation

42.4.1.0.0.42.1 tpm_chnl_t tpm_chnl_pwm_signal_param_t::chnlNumber

In combined mode (available in some SoC's, this represents the channel pair number

42.4.1.0.0.42.2 uint8_t tpm_chnl_pwm_signal_param_t::dutyCyclePercent

100=always active signal (100% duty cycle)

42.4.1.0.0.42.3 uint8_t tpm_chnl_pwm_signal_param_t::firstEdgeDelayPercent

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

42.4.2 struct tpm_dual_edge_capture_param_t

Note

This mode is available only on some SoC's.

Data Fields

- bool enableSwap
  true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored
- tpm_input_capture_edge_t currChanEdgeMode
  Input capture edge select for channel n.
- tpm_input_capture_edge_t nextChanEdgeMode
  Input capture edge select for channel n+1.

42.4.3 struct tpm_phase_params_t

Data Fields

- uint32_t phaseFilterVal
  Filter value, filter is disabled when the value is zero.
- tpm_phase_polarity_t phasePolarity
  Phase polarity.

42.4.4 struct tpm_config_t

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the TPM_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

- tpm_clock_prescale_t prescale
  Select TPM clock prescale value.
- bool useGlobalTimeBase
  true: Use of an external global time base is enabled; false: disabled
- tpm_trigger_select_t triggerSelect
  Input trigger to use for controlling the counter operation.
- tpm_trigger_source_t triggerSource
  Decides if we use external or internal trigger.
- bool enableDoze
  true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode
- bool enableDebugMode
  true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode
- bool enableReloadOnTrigger
  true: TPM counter is reloaded on trigger; false: TPM counter not reloaded
- bool enableStopOnOverflow
  true: TPM counter stops after overflow; false: TPM counter continues running after overflow
- bool enableStartOnTrigger
  true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately
- bool enablePauseOnTrigger
  true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running

42.4.4.0.0.43 Field Documentation

42.4.4.0.0.43.1 tpm_trigger_source_t tpm_config_t::triggerSource

Enumeration Type Documentation

42.5.1 enum tpm_chnl_t

Note

Actual number of available channels is SoC dependent

Enumerator

kTPM_Chnl_0 TPM channel number 0.
kTPM_Chnl_1 TPM channel number 1.
kTPM_Chnl_2 TPM channel number 2.
kTPM_Chnl_3 TPM channel number 3.

kTPM_Chnl_4 TPM channel number 4.
kTPM_Chnl_5 TPM channel number 5.
kTPM_Chnl_6 TPM channel number 6.
kTPM_Chnl_7 TPM channel number 7.

42.5.2 enum tpm_pwm_mode_t

Enumerator

kTPM_EdgeAlignedPwm Edge aligned PWM.
kTPM_CenterAlignedPwm Center aligned PWM.
kTPM_CombinedPwm Combined PWM.

42.5.3 enum tpm_pwm_level_select_t

Enumerator

kTPM_NoPwmSignal No PWM output on pin.
kTPM_LowTrue Low true pulses.
kTPM_HighTrue High true pulses.

42.5.4 enum tpm_trigger_select_t

This is used for both internal & external trigger sources (external option available in certain SoC's)

Note

The actual trigger options available is SoC-specific.

42.5.5 enum tpm_trigger_source_t

Note

This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal triger.

Enumerator

kTPM_TriggerSource_External Use external trigger input.
kTPM_TriggerSource_Internal Use internal trigger.

42.5.6 enum tpm_output_compare_mode_t

Enumerator

kTPM_NoOutputSignal No channel output when counter reaches CnV.
kTPM_ToggleOnMatch Toggle output.
kTPM_ClearOnMatch Clear output.
kTPM_SetOnMatch Set output.
kTPM_HighPulseOutput Pulse output high.
kTPM_LowPulseOutput Pulse output low.

42.5.7 enum tpm_input_capture_edge_t

Enumerator

kTPM_RisingEdge Capture on rising edge only.
kTPM_FallingEdge Capture on falling edge only.
kTPM_RiseAndFallEdge Capture on rising or falling edge.

42.5.8 enum tpm_quad_decode_mode_t

Note

This mode is available only on some SoC's.

Enumerator

kTPM_QuadPhaseEncode Phase A and Phase B encoding mode.
kTPM_QuadCountAndDir Count and direction encoding mode.

42.5.9 enum tpm_phase_polarity_t

Enumerator

kTPM_QuadPhaseNormal Phase input signal is not inverted.
kTPM_QuadPhaseInvert Phase input signal is inverted.

42.5.10 enum tpm_clock_source_t

Enumerator

kTPM_SystemClock System clock.
kTPM_ExternalClock External clock.

42.5.11 enum tpm_clock_prescale_t

Enumerator

kTPM_Prescale_Divide_1 Divide by 1.
kTPM_Prescale_Divide_2 Divide by 2.
kTPM_Prescale_Divide_4 Divide by 4.
kTPM_Prescale_Divide_8 Divide by 8.
kTPM_Prescale_Divide_16 Divide by 16.
kTPM_Prescale_Divide_32 Divide by 32.
kTPM_Prescale_Divide_64 Divide by 64.
kTPM_Prescale_Divide_128 Divide by 128.

42.5.12 enum tpm_interrupt_enable_t

Enumerator

kTPM_Chnl0InterruptEnable Channel 0 interrupt.
kTPM_Chnl1InterruptEnable Channel 1 interrupt.
kTPM_Chnl2InterruptEnable Channel 2 interrupt.
kTPM_Chnl3InterruptEnable Channel 3 interrupt.
kTPM_Chnl4InterruptEnable Channel 4 interrupt.
kTPM_Chnl5InterruptEnable Channel 5 interrupt.
kTPM_Chnl6InterruptEnable Channel 6 interrupt.
kTPM_Chnl7InterruptEnable Channel 7 interrupt.
kTPM_TimeOverflowInterruptEnable Time overflow interrupt.

42.5.13 enum tpm_status_flags_t

Enumerator

kTPM_Chnl0Flag Channel 0 flag.
kTPM_Chnl1Flag Channel 1 flag.
kTPM_Chnl2Flag Channel 2 flag.
kTPM_Chnl3Flag Channel 3 flag.
kTPM_Chnl4Flag Channel 4 flag.
kTPM_Chnl5Flag Channel 5 flag.
kTPM_Chnl6Flag Channel 6 flag.
kTPM_Chnl7Flag Channel 7 flag.
kTPM_TimeOverflowFlag Time overflow flag.

Function Documentation

42.6.1 void TPM_Init (TPM_Type ∗ *base,* const tpm_config_t ∗ *config*)

Note

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

Parameters

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

42.6.2 void TPM_Deinit (TPM_Type * *base*)

Parameters

base	TPM peripheral base address

42.6.3 void TPM_GetDefaultConfig (tpm_config_t * *config*)

The default values are:

```
*     config->prescale = kTPM_Prescale_Divide_1;
*     config->useGlobalTimeBase = false;
*     config->dozeEnable = false;
*     config->dbgMode = false;
*     config->enableReloadOnTrigger = false;
*     config->enableStopOnOverflow = false;
*     config->enableStartOnTrigger = false;
*#if FSL_FEATURE_TPM_HAS_PAUSE_COUNTER_ON_TRIGGER
*     config->enablePauseOnTrigger = false;
*#endif
*     config->triggerSelect = kTPM_Trigger_Select_0;
*#if FSL_FEATURE_TPM_HAS_EXTERNAL_TRIGGER_SELECTION
*     config->triggerSource = kTPM_TriggerSource_External;
*#endif
*
```

Parameters

config	Pointer to user's TPM config structure.

42.6.4 status_t TPM_SetupPwm (TPM_Type * *base*, const tpm_chnl_pwm_signal_param_t * *chnlParams*, uint8_t *numOfChnls*, tpm_pwm_mode_t *mode*, uint32_t *pwmFreq_Hz*, uint32_t *srcClock_Hz*)

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

Parameters

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

Returns

kStatus_Success if the PWM setup was successful, kStatus_Error on failure

42.6.5 void TPM_UpdatePwmDutycycle (TPM_Type ∗ *base*, tpm_chnl_t *chnlNumber*, tpm_pwm_mode_t *currentPwmMode*, uint8_t *dutyCyclePercent*)

Parameters

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

42.6.6 void TPM_UpdateChnlEdgeLevelSelect (TPM_Type ∗ *base*, tpm_chnl_t *chnlNumber*, uint8_t *level*)

Parameters

base	TPM peripheral base address

chnlNumber	The channel number
level	The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field.

42.6.7 void TPM_SetupInputCapture (TPM_Type ∗ *base*, tpm_chnl_t *chnlNumber*, tpm_input_capture_edge_t *captureMode*)

When the edge specified in the captureMode argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

Parameters

base	TPM peripheral base address
chnlNumber	The channel number
captureMode	Specifies which edge to capture

42.6.8 void TPM_SetupOutputCompare (TPM_Type ∗ *base*, tpm_chnl_t *chnlNumber*, tpm_output_compare_mode_t *compareMode*, uint32_t *compareValue*)

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

Parameters

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

42.6.9 void TPM_SetupDualEdgeCapture (TPM_Type ∗ *base*, tpm_chnl_t *chnlPairNumber*, const tpm_dual_edge_capture_param_t ∗ *edgeParam*, uint32_t *filterValue*)

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

Parameters

base	TPM peripheral base address
chnlPair-Number	The TPM channel pair number; options are 0, 1, 2, 3
edgeParam	Sets up the dual edge capture function
filterValue	Filter value, specify 0 to disable filter.

42.6.10 void TPM_SetupQuadDecode (TPM_Type * *base*, const tpm_phase_params_t * *phaseAParams*, const tpm_phase_params_t * *phaseBParams*, tpm_quad_decode_mode_t *quadMode*)

Parameters

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

42.6.11 void TPM_EnableInterrupts (TPM_Type * *base*, uint32_t *mask*)

Parameters

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

42.6.12 void TPM_DisableInterrupts (TPM_Type * *base*, uint32_t *mask*)

Parameters

base	TPM peripheral base address
mask	The interrupts to disable. This is a logical OR of members of the enumeration tpm_-interrupt_enable_t

42.6.13 uint32_t TPM_GetEnabledInterrupts (TPM_Type * *base*)

Parameters

base	TPM peripheral base address

Returns

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

42.6.14 static uint32_t TPM_GetStatusFlags (TPM_Type * *base*) [inline], [static]

Parameters

base	TPM peripheral base address

Returns

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

42.6.15 static void TPM_ClearStatusFlags (TPM_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	TPM peripheral base address

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

42.6.16 static void TPM_SetTimerPeriod (TPM_Type * *base*, uint32_t *ticks*) [inline], [static]

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

Note

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

Parameters

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

42.6.17 static uint32_t TPM_GetCurrentTimerCount (TPM_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	TPM peripheral base address

Returns

The current counter value in ticks

42.6.18 static void TPM_StartTimer (TPM_Type * *base*, tpm_clock_source_t *clockSource*) [inline], [static]

Parameters

base	TPM peripheral base address
clockSource	TPM clock source; once clock source is set the counter will start running

42.6.19 static void TPM_StopTimer (TPM_Type * *base*) [inline], [static]

Parameters

base	TPM peripheral base address

Chapter 43
TSI: Touch Sensing Input

Overview

The MCUXpresso SDK provides a driver for the Touch Sensing Input (TSI) module of MCUXpresso SDK devices.

Typical use case

43.2.1 TSI Operation

```
TSI_Init(TSI0);
TSI_Configure(TSI0, &user_config);
TSI_SetMeasuredChannelNumber(TSI0, channelMask);
TSI_EnableInterrupts(TSI0, kTSI_GlobalInterruptEnable |
      kTSI_EndOfScanInterruptEnable);

TSI_EnableSoftwareTriggerScan(TSI0);
TSI_EnableModule(TSI0);
while(1)
{
   TSI_StartSoftwareTrigger(TSI0);
   TSI_GetCounter(TSI0);
}
```

Data Structures

- struct tsi_calibration_data_t
  TSI calibration data storage. More...
- struct tsi_config_t
  TSI configuration structure. More...

Macros

- #define ALL_FLAGS_MASK (TSI_GENCS_EOSF_MASK | TSI_GENCS_OUTRGF_MASK)
  TSI status flags macro collection.
- #define TSI_V4_EXTCHRG_RESISTOR_BIT_SHIFT TSI_GENCS_EXTCHRG_SHIFT
  resistor bit shift in EXTCHRG bit-field
- #define TSI_V4_EXTCHRG_FILTER_BITS_SHIFT (1U + TSI_GENCS_EXTCHRG_SHIFT)
  filter bits shift in EXTCHRG bit-field
- #define TSI_V4_EXTCHRG_RESISTOR_BIT_CLEAR ((uint32_t)((~(ALL_FLAGS_MASK | TSI_GENCS_EXTCHRG_MASK)) | (3UL << TSI_V4_EXTCHRG_FILTER_BITS_SHIFT)))
  macro of clearing the resistor bit in EXTCHRG bit-field
- #define TSI_V4_EXTCHRG_FILTER_BITS_CLEAR ((uint32_t)((~(ALL_FLAGS_MASK | TSI_GENCS_EXTCHRG_MASK)) | (1UL << TSI_V4_EXTCHRG_RESISTOR_BIT_SHIFT)))
  macro of clearing the filter bits in EXTCHRG bit-field

Enumerations

- enum tsi_n_consecutive_scans_t {
  kTSI_ConsecutiveScansNumber_1time = 0U,
  kTSI_ConsecutiveScansNumber_2time = 1U,
  kTSI_ConsecutiveScansNumber_3time = 2U,
  kTSI_ConsecutiveScansNumber_4time = 3U,
  kTSI_ConsecutiveScansNumber_5time = 4U,
  kTSI_ConsecutiveScansNumber_6time = 5U,
  kTSI_ConsecutiveScansNumber_7time = 6U,
  kTSI_ConsecutiveScansNumber_8time = 7U,
  kTSI_ConsecutiveScansNumber_9time = 8U,
  kTSI_ConsecutiveScansNumber_10time = 9U,
  kTSI_ConsecutiveScansNumber_11time = 10U,
  kTSI_ConsecutiveScansNumber_12time = 11U,
  kTSI_ConsecutiveScansNumber_13time = 12U,
  kTSI_ConsecutiveScansNumber_14time = 13U,
  kTSI_ConsecutiveScansNumber_15time = 14U,
  kTSI_ConsecutiveScansNumber_16time = 15U,
  kTSI_ConsecutiveScansNumber_17time = 16U,
  kTSI_ConsecutiveScansNumber_18time = 17U,
  kTSI_ConsecutiveScansNumber_19time = 18U,
  kTSI_ConsecutiveScansNumber_20time = 19U,
  kTSI_ConsecutiveScansNumber_21time = 20U,
  kTSI_ConsecutiveScansNumber_22time = 21U,
  kTSI_ConsecutiveScansNumber_23time = 22U,
  kTSI_ConsecutiveScansNumber_24time = 23U,
  kTSI_ConsecutiveScansNumber_25time = 24U,
  kTSI_ConsecutiveScansNumber_26time = 25U,
  kTSI_ConsecutiveScansNumber_27time = 26U,
  kTSI_ConsecutiveScansNumber_28time = 27U,
  kTSI_ConsecutiveScansNumber_29time = 28U,
  kTSI_ConsecutiveScansNumber_30time = 29U,
  kTSI_ConsecutiveScansNumber_31time = 30U,
  kTSI_ConsecutiveScansNumber_32time = 31U }

  TSI number of scan intervals for each electrode.
- enum tsi_electrode_osc_prescaler_t {
  kTSI_ElecOscPrescaler_1div = 0U,
  kTSI_ElecOscPrescaler_2div = 1U,
  kTSI_ElecOscPrescaler_4div = 2U,
  kTSI_ElecOscPrescaler_8div = 3U,
  kTSI_ElecOscPrescaler_16div = 4U,
  kTSI_ElecOscPrescaler_32div = 5U,
  kTSI_ElecOscPrescaler_64div = 6U,
  kTSI_ElecOscPrescaler_128div = 7U }

  TSI electrode oscillator prescaler.

- enum tsi_analog_mode_t {
kTSI_AnalogModeSel_Capacitive = 0U,
kTSI_AnalogModeSel_NoiseNoFreqLim = 4U,
kTSI_AnalogModeSel_NoiseFreqLim = 8U,
kTSI_AnalogModeSel_AutoNoise = 12U }
TSI analog mode select.
- enum tsi_reference_osc_charge_current_t {
kTSI_RefOscChargeCurrent_500nA = 0U,
kTSI_RefOscChargeCurrent_1uA = 1U,
kTSI_RefOscChargeCurrent_2uA = 2U,
kTSI_RefOscChargeCurrent_4uA = 3U,
kTSI_RefOscChargeCurrent_8uA = 4U,
kTSI_RefOscChargeCurrent_16uA = 5U,
kTSI_RefOscChargeCurrent_32uA = 6U,
kTSI_RefOscChargeCurrent_64uA = 7U }
TSI Reference oscillator charge and discharge current select.
- enum tsi_osc_voltage_rails_t {
kTSI_OscVolRailsOption_0 = 0U,
kTSI_OscVolRailsOption_1 = 1U,
kTSI_OscVolRailsOption_2 = 2U,
kTSI_OscVolRailsOption_3 = 3U }
TSI oscillator's voltage rails.
- enum tsi_external_osc_charge_current_t {
kTSI_ExtOscChargeCurrent_500nA = 0U,
kTSI_ExtOscChargeCurrent_1uA = 1U,
kTSI_ExtOscChargeCurrent_2uA = 2U,
kTSI_ExtOscChargeCurrent_4uA = 3U,
kTSI_ExtOscChargeCurrent_8uA = 4U,
kTSI_ExtOscChargeCurrent_16uA = 5U,
kTSI_ExtOscChargeCurrent_32uA = 6U,
kTSI_ExtOscChargeCurrent_64uA = 7U }
TSI External oscillator charge and discharge current select.
- enum tsi_series_resistor_t {
kTSI_SeriesResistance_32k = 0U,
kTSI_SeriesResistance_187k = 1U }
TSI series resistance RS value select.
- enum tsi_filter_bits_t {
kTSI_FilterBits_3 = 0U,
kTSI_FilterBits_2 = 1U,
kTSI_FilterBits_1 = 2U,
kTSI_FilterBits_0 = 3U }
TSI series filter bits select.
- enum tsi_status_flags_t {
kTSI_EndOfScanFlag = TSI_GENCS_EOSF_MASK,
kTSI_OutOfRangeFlag = (int)TSI_GENCS_OUTRGF_MASK }
TSI status flags.

- enum tsi_interrupt_enable_t {
kTSI_GlobalInterruptEnable = 1U,
kTSI_OutOfRangeInterruptEnable = 2U,
kTSI_EndOfScanInterruptEnable = 4U }
TSI feature interrupt source.

Functions

- void TSI_Init (TSI_Type *base, const tsi_config_t *config)
Initializes hardware.
- void TSI_Deinit (TSI_Type *base)
De-initializes hardware.
- void TSI_GetNormalModeDefaultConfig (tsi_config_t *userConfig)
Gets the TSI normal mode user configuration structure.
- void TSI_GetLowPowerModeDefaultConfig (tsi_config_t *userConfig)
Gets the TSI low power mode default user configuration structure.
- void TSI_Calibrate (TSI_Type *base, tsi_calibration_data_t *calBuff)
Hardware calibration.
- void TSI_EnableInterrupts (TSI_Type *base, uint32_t mask)
Enables the TSI interrupt requests.
- void TSI_DisableInterrupts (TSI_Type *base, uint32_t mask)
Disables the TSI interrupt requests.
- static uint32_t TSI_GetStatusFlags (TSI_Type *base)
Gets an interrupt flag.
- void TSI_ClearStatusFlags (TSI_Type *base, uint32_t mask)
Clears the interrupt flag.
- static uint32_t TSI_GetScanTriggerMode (TSI_Type *base)
Gets the TSI scan trigger mode.
- static bool TSI_IsScanInProgress (TSI_Type *base)
Gets the scan in progress flag.
- static void TSI_SetElectrodeOSCPrescaler (TSI_Type *base, tsi_electrode_osc_prescaler_-t prescaler)
Sets the prescaler.
- static void TSI_SetNumberOfScans (TSI_Type *base, tsi_n_consecutive_scans_t number)
Sets the number of scans (NSCN).
- static void TSI_EnableModule (TSI_Type *base, bool enable)
Enables/disables the TSI module.
- static void TSI_EnableLowPower (TSI_Type *base, bool enable)
Sets the TSI low power STOP mode as enabled or disabled.
- static void TSI_EnableHardwareTriggerScan (TSI_Type *base, bool enable)
Enables/disables the hardware trigger scan.
- static void TSI_StartSoftwareTrigger (TSI_Type *base)
Starts a software trigger measurement (triggers a new measurement).
- static void TSI_SetMeasuredChannelNumber (TSI_Type *base, uint8_t channel)
Sets the measured channel number.
- static uint8_t TSI_GetMeasuredChannelNumber (TSI_Type *base)
Gets the current measured channel number.
- static void TSI_EnableDmaTransfer (TSI_Type *base, bool enable)
Enables/disables the DMA transfer.
- static void TSI_EnableEndOfScanDmaTransferOnly (TSI_Type *base, bool enable)
Decides whether to enable end of scan DMA transfer request only.

- static uint16_t TSI_GetCounter (TSI_Type *base)
  Gets the conversion counter value.
- static void TSI_SetLowThreshold (TSI_Type *base, uint16_t low_threshold)
  Sets the TSI wake-up channel low threshold.
- static void TSI_SetHighThreshold (TSI_Type *base, uint16_t high_threshold)
  Sets the TSI wake-up channel high threshold.
- static void TSI_SetAnalogMode (TSI_Type *base, tsi_analog_mode_t mode)
  Sets the analog mode of the TSI module.
- static uint8_t TSI_GetNoiseModeResult (TSI_Type *base)
  Gets the noise mode result of the TSI module.
- static void TSI_SetReferenceChargeCurrent (TSI_Type *base, tsi_reference_osc_charge_current_t current)
  Sets the reference oscillator charge current.
- static void TSI_SetElectrodeChargeCurrent (TSI_Type *base, tsi_external_osc_charge_current_-t current)
  Sets the external electrode charge current.
- static void TSI_SetOscVoltageRails (TSI_Type *base, tsi_osc_voltage_rails_t dvolt)
  Sets the oscillator's voltage rails.
- static void TSI_SetElectrodeSeriesResistor (TSI_Type *base, tsi_series_resistor_t resistor)
  Sets the electrode series resistance value in EXTCHRG[0] bit.
- static void TSI_SetFilterBits (TSI_Type *base, tsi_filter_bits_t filter)
  Sets the electrode filter bits value in EXTCHRG[2:1] bits.

Driver version

- #define FSL_TSI_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))
  TSI driver version.

Data Structure Documentation

43.3.1 struct tsi_calibration_data_t

Data Fields

- uint16_t calibratedData [FSL_FEATURE_TSI_CHANNEL_COUNT]
  TSI calibration data storage buffer.

43.3.2 struct tsi_config_t

This structure contains the settings for the most common TSI configurations including the TSI module charge currents, number of scans, thresholds, and so on.

Data Fields

- uint16_t thresh
  High threshold.

- uint16_t thresl
  Low threshold.
- tsi_electrode_osc_prescaler_t prescaler
  Prescaler.
- tsi_external_osc_charge_current_t extchrg
  Electrode charge current.
- tsi_reference_osc_charge_current_t refchrg
  Reference charge current.
- tsi_n_consecutive_scans_t nscn
  Number of scans.
- tsi_analog_mode_t mode
  TSI mode of operation.
- tsi_osc_voltage_rails_t dvolt
  Oscillator's voltage rails.
- tsi_series_resistor_t resistor
  Series resistance value.
- tsi_filter_bits_t filter
  Noise mode filter bits.

43.3.2.0.0.44 Field Documentation

43.3.2.0.0.44.1 uint16_t tsi_config_t::thresh

43.3.2.0.0.44.2 uint16_t tsi_config_t::thresl

43.3.2.0.0.44.3 tsi_n_consecutive_scans_t tsi_config_t::nscn

43.3.2.0.0.44.4 tsi_analog_mode_t tsi_config_t::mode

43.3.2.0.0.44.5 tsi_osc_voltage_rails_t tsi_config_t::dvolt

Macro Definition Documentation

43.4.1 #define FSL_TSI_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

Version 2.1.3

Enumeration Type Documentation

43.5.1 enum tsi_n_consecutive_scans_t

These constants define the tsi number of consecutive scans in a TSI instance for each electrode.

Enumerator

kTSI_ConsecutiveScansNumber_1time	Once per electrode.
kTSI_ConsecutiveScansNumber_2time	Twice per electrode.
kTSI_ConsecutiveScansNumber_3time	3 times consecutive scan
kTSI_ConsecutiveScansNumber_4time	4 times consecutive scan
kTSI_ConsecutiveScansNumber_5time	5 times consecutive scan

kTSI_ConsecutiveScansNumber_6time 6 times consecutive scan
kTSI_ConsecutiveScansNumber_7time 7 times consecutive scan
kTSI_ConsecutiveScansNumber_8time 8 times consecutive scan
kTSI_ConsecutiveScansNumber_9time 9 times consecutive scan
kTSI_ConsecutiveScansNumber_10time 10 times consecutive scan
kTSI_ConsecutiveScansNumber_11time 11 times consecutive scan
kTSI_ConsecutiveScansNumber_12time 12 times consecutive scan
kTSI_ConsecutiveScansNumber_13time 13 times consecutive scan
kTSI_ConsecutiveScansNumber_14time 14 times consecutive scan
kTSI_ConsecutiveScansNumber_15time 15 times consecutive scan
kTSI_ConsecutiveScansNumber_16time 16 times consecutive scan
kTSI_ConsecutiveScansNumber_17time 17 times consecutive scan
kTSI_ConsecutiveScansNumber_18time 18 times consecutive scan
kTSI_ConsecutiveScansNumber_19time 19 times consecutive scan
kTSI_ConsecutiveScansNumber_20time 20 times consecutive scan
kTSI_ConsecutiveScansNumber_21time 21 times consecutive scan
kTSI_ConsecutiveScansNumber_22time 22 times consecutive scan
kTSI_ConsecutiveScansNumber_23time 23 times consecutive scan
kTSI_ConsecutiveScansNumber_24time 24 times consecutive scan
kTSI_ConsecutiveScansNumber_25time 25 times consecutive scan
kTSI_ConsecutiveScansNumber_26time 26 times consecutive scan
kTSI_ConsecutiveScansNumber_27time 27 times consecutive scan
kTSI_ConsecutiveScansNumber_28time 28 times consecutive scan
kTSI_ConsecutiveScansNumber_29time 29 times consecutive scan
kTSI_ConsecutiveScansNumber_30time 30 times consecutive scan
kTSI_ConsecutiveScansNumber_31time 31 times consecutive scan
kTSI_ConsecutiveScansNumber_32time 32 times consecutive scan

43.5.2 enum tsi_electrode_osc_prescaler_t

These constants define the TSI electrode oscillator prescaler in a TSI instance.

Enumerator

kTSI_ElecOscPrescaler_1div Electrode oscillator frequency divided by 1.
kTSI_ElecOscPrescaler_2div Electrode oscillator frequency divided by 2.
kTSI_ElecOscPrescaler_4div Electrode oscillator frequency divided by 4.
kTSI_ElecOscPrescaler_8div Electrode oscillator frequency divided by 8.
kTSI_ElecOscPrescaler_16div Electrode oscillator frequency divided by 16.
kTSI_ElecOscPrescaler_32div Electrode oscillator frequency divided by 32.
kTSI_ElecOscPrescaler_64div Electrode oscillator frequency divided by 64.
kTSI_ElecOscPrescaler_128div Electrode oscillator frequency divided by 128.

43.5.3 enum tsi_analog_mode_t

Set up TSI analog modes in a TSI instance.

Enumerator

kTSI_AnalogModeSel_Capacitive Active TSI capacitive sensing mode.
kTSI_AnalogModeSel_NoiseNoFreqLim Single threshold noise detection mode with no freq. limitation.
kTSI_AnalogModeSel_NoiseFreqLim Single threshold noise detection mode with freq. limitation.

kTSI_AnalogModeSel_AutoNoise Active TSI analog in automatic noise detection mode.

43.5.4 enum tsi_reference_osc_charge_current_t

These constants define the TSI Reference oscillator charge current select in a TSI (REFCHRG) instance.

Enumerator

kTSI_RefOscChargeCurrent_500nA Reference oscillator charge current is 500 µA.
kTSI_RefOscChargeCurrent_1uA Reference oscillator charge current is 1 µA.
kTSI_RefOscChargeCurrent_2uA Reference oscillator charge current is 2 µA.
kTSI_RefOscChargeCurrent_4uA Reference oscillator charge current is 4 µA.
kTSI_RefOscChargeCurrent_8uA Reference oscillator charge current is 8 µA.
kTSI_RefOscChargeCurrent_16uA Reference oscillator charge current is 16 µA.
kTSI_RefOscChargeCurrent_32uA Reference oscillator charge current is 32 µA.
kTSI_RefOscChargeCurrent_64uA Reference oscillator charge current is 64 µA.

43.5.5 enum tsi_osc_voltage_rails_t

These bits indicate the oscillator's voltage rails.

Enumerator

kTSI_OscVolRailsOption_0 DVOLT value option 0, the value may differ on different platforms.
kTSI_OscVolRailsOption_1 DVOLT value option 1, the value may differ on different platforms.
kTSI_OscVolRailsOption_2 DVOLT value option 2, the value may differ on different platforms.
kTSI_OscVolRailsOption_3 DVOLT value option 3, the value may differ on different platforms.

43.5.6 enum tsi_external_osc_charge_current_t

These bits indicate the electrode oscillator charge and discharge current value in TSI (EXTCHRG) instance.

Enumerator

kTSI_ExtOscChargeCurrent_500nA External oscillator charge current is 500 µA.
kTSI_ExtOscChargeCurrent_1uA External oscillator charge current is 1 µA.
kTSI_ExtOscChargeCurrent_2uA External oscillator charge current is 2 µA.
kTSI_ExtOscChargeCurrent_4uA External oscillator charge current is 4 µA.
kTSI_ExtOscChargeCurrent_8uA External oscillator charge current is 8 µA.
kTSI_ExtOscChargeCurrent_16uA External oscillator charge current is 16 µA.
kTSI_ExtOscChargeCurrent_32uA External oscillator charge current is 32 µA.
kTSI_ExtOscChargeCurrent_64uA External oscillator charge current is 64 µA.

43.5.7 enum tsi_series_resistor_t

These bits indicate the electrode RS series resistance for the noise mode in TSI (EXTCHRG) instance.

Enumerator

kTSI_SeriesResistance_32k Series Resistance is 32 kilo ohms.
kTSI_SeriesResistance_187k Series Resistance is 18 7 kilo ohms.

43.5.8 enum tsi_filter_bits_t

These bits indicate the count of the filter bits in TSI noise mode EXTCHRG[2:1] bits

Enumerator

kTSI_FilterBits_3 3 filter bits, 8 peaks increments the cnt+1
kTSI_FilterBits_2 2 filter bits, 4 peaks increments the cnt+1
kTSI_FilterBits_1 1 filter bits, 2 peaks increments the cnt+1
kTSI_FilterBits_0 no filter bits,1 peak increments the cnt+1

43.5.9 enum tsi_status_flags_t

Enumerator

kTSI_EndOfScanFlag End-Of-Scan flag.
kTSI_OutOfRangeFlag Out-Of-Range flag.

43.5.10 enum tsi_interrupt_enable_t

Enumerator

kTSI_GlobalInterruptEnable TSI module global interrupt.

kTSI_OutOfRangeInterruptEnable Out-Of-Range interrupt.
kTSI_EndOfScanInterruptEnable End-Of-Scan interrupt.

Function Documentation

43.6.1 void TSI_Init (TSI_Type * *base*, const tsi_config_t * *config*)

Initializes the peripheral to the targeted state specified by parameter configuration, such as sets prescalers, number of scans, clocks, delta voltage series resistor, filter bits, reference, and electrode charge current and threshold.

Parameters

base	TSI peripheral base address.
config	Pointer to TSI module configuration structure.

Returns

none

43.6.2 void TSI_Deinit (TSI_Type * *base*)

De-initializes the peripheral to default state.

Parameters

base	TSI peripheral base address.

Returns

none

43.6.3 void TSI_GetNormalModeDefaultConfig (tsi_config_t * *userConfig*)

This interface sets userConfig structure to a default value. The configuration structure only includes the settings for the whole TSI. The user configure is set to these values:

```
userConfig->prescaler = kTSI_ElecOscPrescaler_2div;
userConfig->extchrg = kTSI_ExtOscChargeCurrent_500nA;
userConfig->refchrg = kTSI_RefOscChargeCurrent_4uA;
userConfig->nscn = kTSI_ConsecutiveScansNumber_10time;
userConfig->mode = kTSI_AnalogModeSel_Capacitive;
userConfig->dvolt = kTSI_OscVolRailsOption_0;
userConfig->thresh = 0U;
userConfig->thresl = 0U;
```

Parameters

userConfig	Pointer to the TSI user configuration structure.

43.6.4 void TSI_GetLowPowerModeDefaultConfig (tsi_config_t * *userConfig*)

This interface sets userConfig structure to a default value. The configuration structure only includes the settings for the whole TSI. The user configure is set to these values:

```
userConfig->prescaler = kTSI_ElecOscPrescaler_2div;
userConfig->extchrg = kTSI_ExtOscChargeCurrent_500nA;
userConfig->refchrg = kTSI_RefOscChargeCurrent_4uA;
userConfig->nscn = kTSI_ConsecutiveScansNumber_10time;
userConfig->mode = kTSI_AnalogModeSel_Capacitive;
userConfig->dvolt = kTSI_OscVolRailsOption_0;
userConfig->thresh = 400U;
userConfig->thresl = 0U;
```

Parameters

userConfig	Pointer to the TSI user configuration structure.

43.6.5 void TSI_Calibrate (TSI_Type * *base*, tsi_calibration_data_t * *calBuff*)

Calibrates the peripheral to fetch the initial counter value of the enabled electrodes. This API is mostly used at initial application setup. Call this function after the TSI_Init API and use the calibrated counter values to set up applications (such as to determine under which counter value we can confirm a touch event occurs).

Parameters

base	TSI peripheral base address.
calBuff	Data buffer that store the calibrated counter value.

Returns

none

43.6.6 void TSI_EnableInterrupts (TSI_Type * *base*, uint32_t *mask*)

Parameters

base	TSI peripheral base address.
mask	interrupt source The parameter can be combination of the following source if defined: • kTSI_GlobalInterruptEnable • kTSI_EndOfScanInterruptEnable • kTSI_OutOfRangeInterruptEnable

43.6.7 void TSI_DisableInterrupts (TSI_Type ∗ *base,* uint32_t *mask*)

Parameters

base	TSI peripheral base address.
mask	interrupt source The parameter can be combination of the following source if defined: • kTSI_GlobalInterruptEnable • kTSI_EndOfScanInterruptEnable • kTSI_OutOfRangeInterruptEnable

43.6.8 static uint32_t TSI_GetStatusFlags (TSI_Type ∗ *base*) [inline], [static]

This function gets the TSI interrupt flags.

Parameters

base	TSI peripheral base address.

Returns

The mask of these status flags combination.

43.6.9 void TSI_ClearStatusFlags (TSI_Type ∗ *base,* uint32_t *mask*)

This function clears the TSI interrupt flag, automatically cleared flags can't be cleared by this function.

Parameters

base	TSI peripheral base address.
mask	The status flags to clear.

43.6.10 static uint32_t TSI_GetScanTriggerMode (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

Scan trigger mode.

43.6.11 static bool TSI_IsScanInProgress (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

True - scan is in progress. False - scan is not in progress.

43.6.12 static void TSI_SetElectrodeOSCPrescaler (TSI_Type * *base*, tsi_electrode_osc_prescaler_t *prescaler*) [inline], [static]

Parameters

base	TSI peripheral base address.

prescaler	Prescaler value.

Returns

none.

43.6.13 static void TSI_SetNumberOfScans (TSI_Type * *base*, tsi_n_consecutive_scans_t *number*) [inline], [static]

Parameters

base	TSI peripheral base address.
number	Number of scans.

Returns

none.

43.6.14 static void TSI_EnableModule (TSI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	TSI peripheral base address.
enable	Choose whether to enable or disable module; • true Enable TSI module; • false Disable TSI module;

Returns

none.

43.6.15 static void TSI_EnableLowPower (TSI_Type * *base*, bool *enable*) [inline], [static]

```
This enables the TSI module function in low power modes.
```

Parameters

base	TSI peripheral base address.
enable	Choose to enable or disable STOP mode. • true Enable module in STOP mode; • false Disable module in STOP mode;

Returns

none.

43.6.16 static void TSI_EnableHardwareTriggerScan (TSI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	TSI peripheral base address.
enable	Choose to enable hardware trigger or software trigger scan. • true Enable hardware trigger scan; • false Enable software trigger scan;

Returns

none.

43.6.17 static void TSI_StartSoftwareTrigger (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

none.

43.6.18 static void TSI_SetMeasuredChannelNumber (TSI_Type * *base*, uint8_t *channel*) [inline], [static]

Parameters

base	TSI peripheral base address.
channel	Channel number 0 ... 15.

Returns

none.

43.6.19 static uint8_t TSI_GetMeasuredChannelNumber (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

uint8_t Channel number 0 ... 15.

43.6.20 static void TSI_EnableDmaTransfer (TSI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	TSI peripheral base address.
enable	Choose to enable DMA transfer or not. • true Enable DMA transfer; • false Disable DMA transfer;

Returns

none.

43.6.21 static void TSI_EnableEndOfScanDmaTransferOnly (TSI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	TSI peripheral base address.
enable	Choose whether to enable End of Scan DMA transfer request only. • true Enable End of Scan DMA transfer request only; • false Both End-of-Scan and Out-of-Range can generate DMA transfer request.

Returns

none.

43.6.22 static uint16_t TSI_GetCounter (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

Accumulated scan counter value ticked by the reference clock.

43.6.23 static void TSI_SetLowThreshold (TSI_Type * *base*, uint16_t *low_threshold*) [inline], [static]

Parameters

base	TSI peripheral base address.
low_threshold	Low counter threshold.

Returns

none.

43.6.24 static void TSI_SetHighThreshold (TSI_Type * *base*, uint16_t *high_threshold*) [inline], [static]

Parameters

base	TSI peripheral base address.
high_threshold	High counter threshold.

Returns

none.

43.6.25 static void TSI_SetAnalogMode (TSI_Type * *base*, tsi_analog_mode_t *mode*) [inline], [static]

Parameters

base	TSI peripheral base address.
mode	Mode value.

Returns

none.

43.6.26 static uint8_t TSI_GetNoiseModeResult (TSI_Type * *base*) [inline], [static]

Parameters

base	TSI peripheral base address.

Returns

Value of the GENCS[MODE] bit-fields.

43.6.27 static void TSI_SetReferenceChargeCurrent (TSI_Type * *base*, tsi_reference_osc_charge_current_t *current*) [inline], [static]

Parameters

base	TSI peripheral base address.
current	The reference oscillator charge current.

Returns

none.

43.6.28 static void TSI_SetElectrodeChargeCurrent (TSI_Type * *base*, tsi_external_osc_charge_current_t *current*) [inline], [static]

Parameters

base	TSI peripheral base address.
current	External electrode charge current.

Returns

none.

43.6.29 static void TSI_SetOscVoltageRails (TSI_Type * *base*, tsi_osc_voltage_rails_t *dvolt*) [inline], [static]

Parameters

base	TSI peripheral base address.
dvolt	The voltage rails.

Returns

none.

43.6.30 static void TSI_SetElectrodeSeriesResistor (TSI_Type * *base*, tsi_series_resistor_t *resistor*) [inline], [static]

Parameters

base	TSI peripheral base address.
resistor	Series resistance.

Returns

none.

43.6.31 static void TSI_SetFilterBits (TSI_Type * *base*, tsi_filter_bits_t *filter*) [inline], [static]

Parameters

base	TSI peripheral base address.
filter	Series resistance.

Returns

none.

Chapter 44
UART: Universal Asynchronous Receiver/Transmitter Driver

Overview

Modules

- UART Driver

UART Driver

44.2.1 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_Transfer-CreateHandle() API.

Transactional APIs support asynchronous transfer, which means that the functions UART_TransferSend-NonBlocking() 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.

44.2.2 Typical use case

44.2.2.1 UART Send/receive using a polling method

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

44.2.2.2 UART Send/receive using an interrupt method

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

44.2.2.3 UART Receive using the ringbuffer feature

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

44.2.2.4 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_LinBreakInterruptEnable = (UART_BDH_LBKDIE_MASK),
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_LinBreakFlag,
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.

Driver version

- #define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
UART driver version 2.4.0.

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.
- 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.
- void UART_TransferHandleIRQ (UART_Type *base, uart_handle_t *handle)
  UART IRQ handle function.
- void UART_TransferHandleErrorIRQ (UART_Type *base, uart_handle_t *handle)
  UART Error IRQ handle function.

44.2.3 Data Structure Documentation

44.2.3.1 struct uart_config_t

Data Fields

- uint32_t baudRate_Bps
  UART baud rate.
- uart_parity_mode_t parityMode
  Parity mode, disabled (default), even, odd.
- uart_stop_bit_count_t stopBitCount
  Number of stop bits, 1 stop bit (default) or 2 stop bits.
- uint8_t txFifoWatermark
  TX FIFO watermark.
- uint8_t rxFifoWatermark
  RX FIFO watermark.
- bool enableRxRTS
  RX RTS enable.
- bool enableTxCTS
  TX CTS enable.
- uart_idle_type_select_t idleType
  IDLE type select.
- bool enableTx
  Enable TX.
- bool enableRx
  Enable RX.

44.2.3.1.0.45 Field Documentation

44.2.3.1.0.45.1 uart_idle_type_select_t uart_config_t::idleType

44.2.3.2 struct uart_transfer_t

Data Fields

- uint8_t ∗ data
  The buffer of data to be transfer.
- size_t dataSize
  The byte count to be transfer.

44.2.3.2.0.46 Field Documentation

44.2.3.2.0.46.1 uint8_t* uart_transfer_t::data

44.2.3.2.0.46.2 size_t uart_transfer_t::dataSize

44.2.3.3 struct _uart_handle

Data Fields

- uint8_t *volatile txData
  Address of remaining data to send.
- volatile size_t txDataSize
  Size of the remaining data to send.
- 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.

44.2.3.3.0.47 Field Documentation

44.2.3.3.0.47.1 uint8_t* volatile uart_handle_t::txData

44.2.3.3.0.47.2 volatile size_t uart_handle_t::txDataSize

44.2.3.3.0.47.3 size_t uart_handle_t::txDataSizeAll

44.2.3.3.0.47.4 uint8_t* volatile uart_handle_t::rxData

44.2.3.3.0.47.5 volatile size_t uart_handle_t::rxDataSize

44.2.3.3.0.47.6 size_t uart_handle_t::rxDataSizeAll

44.2.3.3.0.47.7 uint8_t* uart_handle_t::rxRingBuffer

44.2.3.3.0.47.8 size_t uart_handle_t::rxRingBufferSize

44.2.3.3.0.47.9 volatile uint16_t uart_handle_t::rxRingBufferHead

44.2.3.3.0.47.10 volatile uint16_t uart_handle_t::rxRingBufferTail

44.2.3.3.0.47.11 uart_transfer_callback_t uart_handle_t::callback

44.2.3.3.0.47.12 void* uart_handle_t::userData

44.2.3.3.0.47.13 volatile uint8_t uart_handle_t::txState

44.2.4 Macro Definition Documentation

44.2.4.1 #define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

44.2.4.2 #define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */

44.2.5 Typedef Documentation

44.2.5.1 typedef void(* uart_transfer_callback_t)(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)

44.2.6 Enumeration Type Documentation

44.2.6.1 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.

44.2.6.2 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.

44.2.6.3 enum uart_stop_bit_count_t

Enumerator

kUART_OneStopBit One stop bit.
kUART_TwoStopBit Two stop bits.

44.2.6.4 enum uart_idle_type_select_t

Enumerator

kUART_IdleTypeStartBit Start counting after a valid start bit.
kUART_IdleTypeStopBit Start counting after a stop bit.

44.2.6.5 enum _uart_interrupt_enable

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

Enumerator

kUART_LinBreakInterruptEnable LIN break detect interrupt.
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.

44.2.6.6 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_LinBreakFlag LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.
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.

44.2.7 Function Documentation

44.2.7.1 uint32_t UART_GetInstance (UART_Type * *base*)

Parameters

base	UART peripheral base address.

Returns

UART instance.

44.2.7.2 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

44.2.7.3 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.

44.2.7.4 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.

44.2.7.5 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.

44.2.7.6 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.

44.2.7.7 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.

44.2.7.8 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.

44.2.7.9 static void UART_Set9thTransmitBit (UART_Type * *base*) [inline], [static]

Parameters

base	UART peripheral base address.

44.2.7.10 static void UART_Clear9thTransmitBit (UART_Type * *base*) [inline], [static]

Parameters

base	UART peripheral base address.

44.2.7.11 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.

44.2.7.12 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.

44.2.7.13 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.

44.2.7.14 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.

44.2.7.15 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.

44.2.7.16 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.

44.2.7.17 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.

44.2.7.18 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.

44.2.7.19 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.

44.2.7.20 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.

44.2.7.21 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.

44.2.7.22 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.

44.2.7.23 void UART_SendAddress (UART_Type ∗ *base*, uint8_t *address*)

Parameters

base	UART peripheral base address.
address	UART slave address.

44.2.7.24 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.

44.2.7.25 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.

44.2.7.26 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.

44.2.7.27 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.

44.2.7.28 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.

44.2.7.29 size_t UART_TransferGetRxRingBufferLength (uart_handle_t * *handle*)

Parameters

handle	UART handle pointer.

Returns

Length of received data in RX ring buffer.

44.2.7.30 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.

44.2.7.31 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.

44.2.7.32 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_NoTransferInProgress	No send in progress.
kStatus_InvalidArgument	The parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

44.2.7.33 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.

44.2.7.34 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.

44.2.7.35 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_NoTransferInProgress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

44.2.7.36 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.

44.2.7.37 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.

44.2.7.38 void UART_TransferHandleIRQ (UART_Type * *base*, uart_handle_t * *handle*)

This function handles the UART transmit and receive IRQ request.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.

44.2.7.39 void UART_TransferHandleErrorIRQ (UART_Type * *base*, uart_handle_t * *handle*)

This function handles the UART error IRQ request.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.

Chapter 45
VREF: Voltage Reference Driver

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.

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.

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.

Data Structure Documentation

45.4.1 struct vref_config_t

Data Fields

- vref_buffer_mode_t bufferMode
  Buffer mode selection.

Macro Definition Documentation

45.5.1 #define FSL_VREF_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

Enumeration Type Documentation

45.6.1 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.

Function Documentation

45.7.1 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.

45.7.2 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.

45.7.3 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.

45.7.4 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)).

45.7.5 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.

Chapter 46
WDOG: Watchdog Timer Driver

Overview

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

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.

Data Structure Documentation

46.3.1 struct wdog_work_mode_t

Data Fields

- bool enableWait
  Enables or disables WDOG in wait mode.
- bool enableStop
  Enables or disables WDOG in stop mode.
- bool enableDebug
  Enables or disables WDOG in debug mode.

46.3.2 struct wdog_config_t

Data Fields

- bool enableWdog
  Enables or disables WDOG.
- wdog_clock_source_t clockSource
  Clock source select.
- wdog_clock_prescaler_t prescaler
  Clock prescaler value.
- wdog_work_mode_t workMode
  Configures WDOG work mode in debug stop and wait mode.
- bool enableUpdate
  Update write-once register enable.
- bool enableInterrupt
  Enables or disables WDOG interrupt.
- bool enableWindowMode
  Enables or disables WDOG window mode.

- uint32_t windowValue
  Window value.
- uint32_t timeoutValue
  Timeout value.

46.3.3 struct wdog_test_config_t

Data Fields

- wdog_test_mode_t testMode
  Selects test mode.
- wdog_tested_byte_t testedByte
  Selects tested byte in byte test mode.
- uint32_t timeoutValue
  Timeout value.

Macro Definition Documentation

46.4.1 #define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Enumeration Type Documentation

46.5.1 enum wdog_clock_source_t

Enumerator

kWDOG_LpoClockSource WDOG clock sourced from LPO.
kWDOG_AlternateClockSource WDOG clock sourced from alternate clock source.

46.5.2 enum wdog_clock_prescaler_t

Enumerator

kWDOG_ClockPrescalerDivide1 Divided by 1.
kWDOG_ClockPrescalerDivide2 Divided by 2.
kWDOG_ClockPrescalerDivide3 Divided by 3.
kWDOG_ClockPrescalerDivide4 Divided by 4.
kWDOG_ClockPrescalerDivide5 Divided by 5.
kWDOG_ClockPrescalerDivide6 Divided by 6.
kWDOG_ClockPrescalerDivide7 Divided by 7.
kWDOG_ClockPrescalerDivide8 Divided by 8.

46.5.3 enum wdog_test_mode_t

Enumerator

kWDOG_QuickTest Selects quick test.
kWDOG_ByteTest Selects byte test.

46.5.4 enum wdog_tested_byte_t

Enumerator

kWDOG_TestByte0 Byte 0 selected in byte test mode.
kWDOG_TestByte1 Byte 1 selected in byte test mode.
kWDOG_TestByte2 Byte 2 selected in byte test mode.
kWDOG_TestByte3 Byte 3 selected in byte test mode.

46.5.5 enum _wdog_interrupt_enable_t

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

Enumerator

kWDOG_InterruptEnable WDOG timeout generates an interrupt before reset.

46.5.6 enum _wdog_status_flags_t

This structure contains the WDOG status flags for use in the WDOG functions.

Enumerator

kWDOG_RunningFlag Running flag, set when WDOG is enabled.
kWDOG_TimeoutFlag Interrupt flag, set when an exception occurs.

Function Documentation

46.6.1 void WDOG_GetDefaultConfig (wdog_config_t * *config*)

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

```
*   wdogConfig->enableWdog = true;
*   wdogConfig->clockSource = kWDOG_LpoClockSource;
*   wdogConfig->prescaler = kWDOG_ClockPrescalerDivide1;
*   wdogConfig->workMode.enableWait = true;
*   wdogConfig->workMode.enableStop = false;
*   wdogConfig->workMode.enableDebug = false;
*   wdogConfig->enableUpdate = true;
*   wdogConfig->enableInterrupt = false;
*   wdogConfig->enableWindowMode = false;
*   wdogConfig->windowValue = 0;
*   wdogConfig->timeoutValue = 0xFFFFU;
*
```

Parameters

config	Pointer to the WDOG configuration structure.

See Also

wdog_config_t

46.6.2 void WDOG_Init (WDOG_Type ∗ *base*, const wdog_config_t ∗ *config*)

This function initializes the WDOG. When called, the WDOG runs according to the configuration. To reconfigure WDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

This is an example.

```
*   wdog_config_t config;
*   WDOG_GetDefaultConfig(&config);
*   config.timeoutValue = 0x7ffU;
*   config.enableUpdate = true;
*   WDOG_Init(wdog_base,&config);
*
```

Parameters

base	WDOG peripheral base address
config	The configuration of WDOG

46.6.3 void WDOG_Deinit (WDOG_Type ∗ *base*)

This function shuts down the WDOG. Ensure that the WDOG_STCTRLH.ALLOWUPDATE is 1 which indicates that the register update is enabled.

46.6.4 void WDOG_SetTestModeConfig (WDOG_Type * *base*, wdog_test_config_t * *config*)

This function is used to configure the WDOG functional test. When called, the WDOG goes into test mode and runs according to the configuration. Ensure that the WDOG_STCTRLH.ALLOWUPDATE is 1 which means that the register update is enabled.

This is an example.

```
*   wdog_test_config_t test_config;
*   test_config.testMode = kWDOG_QuickTest;
*   test_config.timeoutValue = 0xfffffu;
*   WDOG_SetTestModeConfig(wdog_base, &test_config);
*
```

Parameters

base	WDOG peripheral base address
config	The functional test configuration of WDOG

46.6.5 static void WDOG_Enable (WDOG_Type * *base*) [inline], [static]

This function write value into WDOG_STCTRLH register to enable the WDOG, it is a write-once register, make sure that the WCT window is still open and this register has not been written in this WCT while this function is called.

Parameters

base	WDOG peripheral base address

46.6.6 static void WDOG_Disable (WDOG_Type * *base*) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to disable the WDOG. It is a write-once register. Ensure that the WCT window is still open and that register has not been written to in this WCT while the function is called.

Parameters

base	WDOG peripheral base address

46.6.7 static void WDOG_EnableInterrupts (WDOG_Type * *base*, uint32_t *mask*) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to enable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

base	WDOG peripheral base address
mask	The interrupts to enable The parameter can be combination of the following source if defined. • kWDOG_InterruptEnable

46.6.8 static void WDOG_DisableInterrupts (WDOG_Type * *base*, uint32_t *mask*) [inline], [static]

This function writes a value into the WDOG_STCTRLH register to disable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

base	WDOG peripheral base address
mask	The interrupts to disable The parameter can be combination of the following source if defined. • kWDOG_InterruptEnable

46.6.9 uint32_t WDOG_GetStatusFlags (WDOG_Type * *base*)

This function gets all status flags.

This is an example for getting the Running Flag.

```
*   uint32_t status;
*   status = WDOG_GetStatusFlags (wdog_base) &
      kWDOG_RunningFlag;
```

```
*
```

Parameters

base	WDOG peripheral base address

Returns

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

See Also

_wdog_status_flags_t

- true: a related status flag has been set.
- false: a related status flag is not set.

46.6.10 void WDOG_ClearStatusFlags (WDOG_Type * *base*, uint32_t *mask*)

This function clears the WDOG status flag.

This is an example for clearing the timeout (interrupt) flag.

```
*   WDOG_ClearStatusFlags(wdog_base,kWDOG_TimeoutFlag);
*
```

Parameters

base	WDOG peripheral base address
mask	The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

46.6.11 static void WDOG_SetTimeoutValue (WDOG_Type * *base*, uint32_t *timeoutCount*) [inline], [static]

This function sets the timeout value. It should be ensured that the time-out value for the WDOG is always greater than 2xWCT time + 20 bus clock cycles. This function writes a value into WDOG_TOVALH and WDOG_TOVALL registers which are wirte-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

base	WDOG peripheral base address
timeoutCount	WDOG timeout value; count of WDOG clock tick.

46.6.12 static void WDOG_SetWindowValue (WDOG_Type * *base,* uint32_t *windowValue*) [inline], [static]

This function sets the WDOG window value. This function writes a value into WDOG_WINH and 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.

46.6.13 static void WDOG_Unlock (WDOG_Type * *base*) [inline], [static]

This function unlocks the WDOG register written. Before starting the unlock sequence and following configuration, disable the global interrupts. Otherwise, an interrupt may invalidate the unlocking sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

base	WDOG peripheral base address

46.6.14 void WDOG_Refresh (WDOG_Type * *base*)

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

base	WDOG peripheral base address

46.6.15 static uint16_t WDOG_GetResetCount (WDOG_Type * *base*) [inline], [static]

This function gets the WDOG reset count value.

Parameters

base	WDOG peripheral base address

Returns

WDOG reset count value.

46.6.16 static void WDOG_ClearResetCount (WDOG_Type * *base*) [inline], [static]

This function clears the WDOG reset count value.

Parameters

base	WDOG peripheral base address

Chapter 47
Debug Console

Overview

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data. The below picture shows the laylout of debug console.

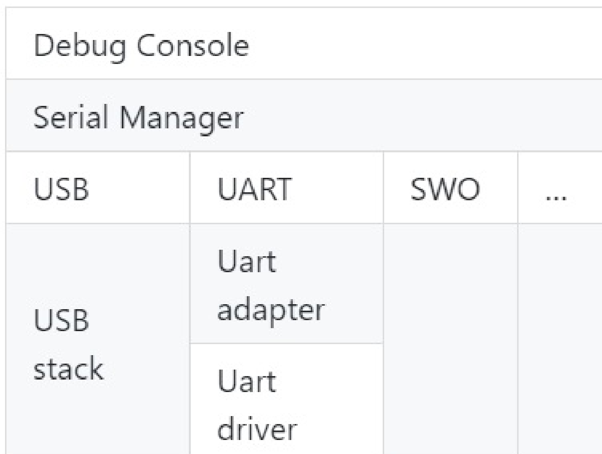

Figure 47.1.1: Debug console overview

Function groups

47.2.1 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);
```

47.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype " %[flags][width][.precision][length]specifier", which is explained below

flags	Description
-	Left-justified within the given field width. Right-justified is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is written, a blank space is inserted before the value.
#	Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

Width	Description
(number)	A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	Description
.number	For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

length	Description
Do not support	

specifier	Description
d or i	Signed decimal integer
f	Decimal floating point
F	Decimal floating point capital letters
x	Unsigned hexadecimal integer
X	Unsigned hexadecimal integer capital letters
o	Signed octal
b	Binary value
p	Pointer address
u	Unsigned decimal integer
c	Character
s	String of characters
n	Nothing printed

- Support a format specifier for SCANF following this prototype " %[*][width][length]specifier", which is explained below

*	Description
An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument.	

width	Description
This specifies the maximum number of characters to be read in the current reading operation.	

length	Description
hh	The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).
h	The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).
l	The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
ll	The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
L	The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).
j or z or t	Not supported

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char *

specifier	Qualifying Input	Type of argument
i	Integer: : Number optionally preceded with a + or - sign	int *
d	Decimal integer: Number optionally preceded with a + or - sign	int *
a, A, e, E, f, F, g, G	Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	float *
o	Octal Integer:	int *
s	String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).	char *
u	Unsigned decimal integer.	unsigned int *

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE == 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 */
```

47.2.3 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 MCUXpresso, if the macro SDK_DEBUGCONSOLE_UART is defined, __sys_write and __sys_readc will be used when __REDLIB__ is defined; _write and _read will be used in other cases.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_TOOLCHAIN	defined	Low level peripheral∗	Low level peripheral
DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	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.

Typical use case

Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalents 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

Print out failure messages using MCUXpresso SDK __assert_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
    PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file
       , line, func);
    for (;;)
    {}
}
```

Note:

To use 'printf' and 'scanf' for GNUC Base, add file **'fsl_sbrk.c'** in path: **..\{package}\devices\{subset}\utilities\fsl-_sbrk.c** to your project.

Macros

- #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U
  Definition select redirect toolchain printf, scanf to uart or not.
- #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U
  Select SDK version printf, scanf.
- #define DEBUGCONSOLE_DISABLE 2U
  Disable debugconsole function.
- #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK
  Definition to select sdk or toolchain printf, scanf.
- #define PRINTF DbgConsole_Printf
  Definition to select redirect toolchain printf, scanf to uart or not.

Typedefs

- typedef void(∗ printfCb)(char ∗buf, int32_t ∗indicator, char val, int len)
  A function pointer which is used when format printf log.

Functions

- int StrFormatPrintf (const char ∗fmt, va_list ap, char ∗buf, printfCb cb)
  This function outputs its parameters according to a formatted string.
- int StrFormatScanf (const char ∗line_ptr, char ∗format, va_list args_ptr)
  Converts an input line of ASCII characters based upon a provided string format.

Variables

- serial_handle_t g_serialHandle
  serial manager handle

Initialization

- status_t DbgConsole_Init (uint8_t instance, uint32_t baudRate, serial_port_type_t device, uint32_t clkSrcFreq)
  Initializes the peripheral used for debug messages.
- status_t DbgConsole_Deinit (void)
  De-initializes the peripheral used for debug messages.
- status_t DbgConsole_EnterLowpower (void)
  Prepares to enter low power consumption.
- status_t DbgConsole_ExitLowpower (void)
  Restores from low power consumption.
- int DbgConsole_Printf (const char ∗fmt_s,...)
  Writes formatted output to the standard output stream.
- int DbgConsole_Putchar (int ch)
  Writes a character to stdout.
- int DbgConsole_Scanf (char ∗formatString,...)
  Reads formatted data from the standard input stream.
- int DbgConsole_Getchar (void)
  Reads a character from standard input.
- int DbgConsole_BlockingPrintf (const char ∗formatString,...)
  Writes formatted output to the standard output stream with the blocking mode.
- status_t DbgConsole_Flush (void)
  Debug console flush.

Macro Definition Documentation

47.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

47.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

47.4.3 #define DEBUGCONSOLE_DISABLE 2U

47.4.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

The macro only support to be redefined in project setting.

47.4.5 #define PRINTF DbgConsole_Printf

if SDK_DEBUGCONSOLE defined to 0,it represents select toolchain printf, scanf. if SDK_DEBUGCO-NSOLE defined to 1,it represents select SDK version printf, scanf. if SDK_DEBUGCONSOLE defined to 2,it represents disable debugconsole function.

Function Documentation

47.5.1 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

47.5.2 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.

47.5.3 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.

47.5.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

47.5.5 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.

47.5.6 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.

47.5.7 int DbgConsole_Scanf (char * *formatString,* ...)

Call this function to read formatted data from the standard input stream.

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG_CONSOLE_TRANSFER_NON_B-LOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole_TryGetchar to get the input char.

Parameters

formatString	Format control string.

Returns

Returns the number of fields successfully converted and assigned.

47.5.8 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.

47.5.9 int DbgConsole_BlockingPrintf (const char * *formatString*, ...)

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

formatString	Format control string.

Returns

Returns the number of characters printed or a negative value if an error occurs.

47.5.10 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.

47.5.11 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

47.5.12 int StrFormatScanf (const char * *line_ptr*, char * *format*, va_list *args_ptr*)

Parameters

in	*line_ptr*	The input line of ASCII data.
in	*format*	Format first points to the format string.
in	*args_ptr*	The list of parameters.

Returns

Number of input items converted and assigned.

Return values

IO_EOF	When line_ptr is empty string "".

Chapter 48
Notification Framework

Overview

This section describes the programming interface of the Notifier driver.

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.

Data Structure Documentation

48.3.1 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.

48.3.1.0.0.48 Field Documentation

48.3.1.0.0.48.1 notifier_user_config_t* notifier_notification_block_t::targetConfig

48.3.1.0.0.48.2 notifier_policy_t notifier_notification_block_t::policy

48.3.1.0.0.48.3 notifier_notification_type_t notifier_notification_block_t::notifyType

48.3.2 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.

48.3.2.0.0.49 Field Documentation

48.3.2.0.0.49.1 notifier_callback_t notifier_callback_config_t::callback

48.3.2.0.0.49.2 notifier_callback_type_t notifier_callback_config_t::callbackType

48.3.2.0.0.49.3 void* notifier_callback_config_t::callbackData

48.3.3 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.

48.3.3.0.0.50 Field Documentation

48.3.3.0.0.50.1 notifier_user_config_t** notifier_handle_t::configsTable

48.3.3.0.0.50.2 uint8_t notifier_handle_t::configsNumber

48.3.3.0.0.50.3 notifier_callback_config_t* notifier_handle_t::callbacksTable

48.3.3.0.0.50.4 uint8_t notifier_handle_t::callbacksNumber

48.3.3.0.0.50.5 uint8_t notifier_handle_t::errorCallbackIndex

48.3.3.0.0.50.6 uint8_t notifier_handle_t::currentConfigIndex

48.3.3.0.0.50.7 notifier_user_function_t notifier_handle_t::userFunction

48.3.3.0.0.50.8 void* notifier_handle_t::userData

Typedef Documentation

48.4.1 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.

48.4.2 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.

48.4.3 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.

Enumeration Type Documentation

48.5.1 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.

48.5.2 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.

48.5.3 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.

48.5.4 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.

Function Documentation

48.6.1 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.

48.6.2 status_t NOTIFIER_SwitchConfig (notifier_handle_t * *notifierHandle*, uint8_t *configIndex*, notifier_policy_t *policy*)

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER_PolicyForcible) or exited (kNOTIFIER_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER_GetErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when NOTIFIER_SwitchConfig() exits.

Parameters

notifierHandle	pointer to notifier handle
configIndex	Index of the target configuration.
policy	Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible.

Returns

An error code or kStatus_Success.

48.6.3 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 49
Shell

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.

Function groups

49.2.1 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>");
```

49.2.2 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.

49.2.3 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 ∗shell-Command)
  Registers the shell command.
- shell_status_t SHELL_UnregisterCommand (shell_command_t ∗shellCommand)
  Unregisters the shell command.
- shell_status_t SHELL_Write (shell_handle_t shellHandle, char ∗buffer, uint32_t length)
  Sends data to the shell output stream.
- int SHELL_Printf (shell_handle_t shellHandle, const char ∗formatString,...)
  Writes formatted output to the shell output stream.
- void SHELL_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.

Data Structure Documentation

49.3.1 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

49.3.1.0.0.51 Field Documentation

49.3.1.0.0.51.1 const char∗ shell_command_t::pcCommand

For example "help". It must be all lower case.

49.3.1.0.0.51.2 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".

49.3.1.0.0.51.3 const cmd_function_t shell_command_t::pFuncCallBack

49.3.1.0.0.51.4 uint8_t shell_command_t::cExpectedNumberOfParameters

Macro Definition Documentation

49.4.1 #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE

49.4.2 #define SHELL_AUTO_COMPLETE (1U)

49.4.3 #define SHELL_BUFFER_SIZE (64U)

49.4.4 #define SHELL_MAX_ARGS (8U)

49.4.5 #define SHELL_HISTORY_COUNT (3U)

49.4.6 #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

49.4.7 #define SHELL_USE_COMMON_TASK (0U)

49.4.8 #define SHELL_TASK_PRIORITY (2U)

49.4.9 #define SHELL_TASK_STACK_SIZE (1000U)

49.4.10 #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.

49.4.11 #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.

49.4.12 #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".

Typedef Documentation

49.5.1 typedef shell_status_t(∗ cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char ∗∗argv)

Enumeration Type Documentation

49.6.1 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.

Function Documentation

49.7.1 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_Open-WriteHandleFailed	Open the write handle failed.
kStatus_SHELL_Open-ReadHandleFailed	Open the read handle failed.

49.7.2 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.

49.7.3 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.

49.7.4 shell_status_t SHELL_Write (shell_handle_t *shellHandle,* char ∗ *buffer,* uint32_t *length*)

This function is used to send data to the shell output stream.

Parameters

shellHandle	The shell module handle pointer.
buffer	Start address of the data to write.
length	Length of the data to write.

Return values

kStatus_SHELL_Success	Successfully send data.
kStatus_SHELL_Error	An error occurred.

49.7.5 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.

49.7.6 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.

49.7.7 void SHELL_PrintPrompt (shell_handle_t *shellHandle*)

Call this function to print shell prompt.

Parameters

shellHandle	The shell module handle pointer.

Returns

NULL.

49.7.8 void SHELL_Task (shell_handle_t *shellHandle*)

The task function for Shell; The function should be polled by upper layer. This function does not return until Shell command exit was called.

Parameters

shellHandle	The shell module handle pointer.

Chapter 50
Secure Digital Card/Embedded MultiMedia Card/SDIO card

The MCUXpresso SDK provides drivers to access the Secure Digital Card(up to v3.0) ,Embedded MultiMedia Card(up to v5.0) and sdio card(up to v3.0) based on the SDHC/USDHC/SDIF driver. Here is a simple block diagram about the driver,

```
---------------------------
|       Application       |
---------------------------
            |
            v
---------------------------
|       Card driver       |
---------------------------
            |
            v
------------- -------------
|host driver| |OSA adapter|
------------- -------------
            |
            v
----------------------------
|  host controller driver  |
----------------------------
            |
            v
----------------------------
|host controller peripheral|
----------------------------
            |
            v
----------------------------
|          card            |
----------------------------
```

Chapter 51
CODEC codec Driver

Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

Modules

- codec common Driver

codec common Driver

51.2.1 Overview

The codec common driver provide codec control abstraction interface.

Data Structures

- struct codec_config_t
  Initialize structure of the codec. More...
- struct codec_capability_t
  codec capability More...
- struct codec_handle_t
  Codec handle definition. More...

Macros

- #define CODEC_VOLUME_MAX_VALUE (0x80U)
  codec maximum volume range

Enumerations

- enum {
  kStatus_CODEC_NotSupport = MAKE_STATUS(kStatusGroup_CODEC, 0U),
  kStatus_CODEC_DeviceNotRegistered = MAKE_STATUS(kStatusGroup_CODEC, 1U),
  kStatus_CODEC_I2CBusInitialFailed,
  kStatus_CODEC_I2CCommandTransferFailed }
  CODEC status.
- enum codec_audio_protocol_t {
  kCODEC_BusI2S = 0U,
  kCODEC_BusLeftJustified = 1U,
  kCODEC_BusRightJustified = 2U,
  kCODEC_BusPCMA = 3U,
  kCODEC_BusPCMB = 4U,
  kCODEC_BusTDM = 5U }
  AUDIO format definition.
- enum {

kCODEC_AudioSampleRate8KHz = 8000U,
kCODEC_AudioSampleRate11025Hz = 11025U,
kCODEC_AudioSampleRate12KHz = 12000U,
kCODEC_AudioSampleRate16KHz = 16000U,
kCODEC_AudioSampleRate22050Hz = 22050U,
kCODEC_AudioSampleRate24KHz = 24000U,
kCODEC_AudioSampleRate32KHz = 32000U,
kCODEC_AudioSampleRate44100Hz = 44100U,
kCODEC_AudioSampleRate48KHz = 48000U,
kCODEC_AudioSampleRate96KHz = 96000U,
kCODEC_AudioSampleRate192KHz = 192000U,
kCODEC_AudioSampleRate384KHz = 384000U }
audio sample rate definition
- enum {
kCODEC_AudioBitWidth16bit = 16U,
kCODEC_AudioBitWidth20bit = 20U,
kCODEC_AudioBitWidth24bit = 24U,
kCODEC_AudioBitWidth32bit = 32U }
audio bit width
- enum codec_module_t {
kCODEC_ModuleADC = 0U,
kCODEC_ModuleDAC = 1U,
kCODEC_ModulePGA = 2U,
kCODEC_ModuleHeadphone = 3U,
kCODEC_ModuleSpeaker = 4U,
kCODEC_ModuleLinein = 5U,
kCODEC_ModuleLineout = 6U,
kCODEC_ModuleVref = 7U,
kCODEC_ModuleMicbias = 8U,
kCODEC_ModuleMic = 9U,
kCODEC_ModuleI2SIn = 10U,
kCODEC_ModuleI2SOut = 11U,
kCODEC_ModuleMxier = 12U }
audio codec module
- enum codec_module_ctrl_cmd_t { kCODEC_ModuleSwitchI2SInInterface = 0U }
audio codec module control cmd
- enum {
kCODEC_ModuleI2SInInterfacePCM = 0U,
kCODEC_ModuleI2SInInterfaceDSD = 1U }
audio codec module digital interface
- enum {
kCODEC_RecordSourceDifferentialLine = 1U,
kCODEC_RecordSourceLineInput = 2U,
kCODEC_RecordSourceDifferentialMic = 4U,
kCODEC_RecordSourceDigitalMic = 8U,
kCODEC_RecordSourceSingleEndMic = 16U }

audio codec module record source value

- enum {
kCODEC_RecordChannelLeft1 = 1U,
kCODEC_RecordChannelLeft2 = 2U,
kCODEC_RecordChannelLeft3 = 4U,
kCODEC_RecordChannelRight1 = 1U,
kCODEC_RecordChannelRight2 = 2U,
kCODEC_RecordChannelRight3 = 4U,
kCODEC_RecordChannelDifferentialPositive1 = 1U,
kCODEC_RecordChannelDifferentialPositive2 = 2U,
kCODEC_RecordChannelDifferentialPositive3 = 4U,
kCODEC_RecordChannelDifferentialNegative1 = 8U,
kCODEC_RecordChannelDifferentialNegative2 = 16U,
kCODEC_RecordChannelDifferentialNegative3 = 32U }
audio codec record channel
- enum {
kCODEC_PlaySourcePGA = 1U,
kCODEC_PlaySourceInput = 2U,
kCODEC_PlaySourceDAC = 4U,
kCODEC_PlaySourceMixerIn = 1U,
kCODEC_PlaySourceMixerInLeft = 2U,
kCODEC_PlaySourceMixerInRight = 4U,
kCODEC_PlaySourceAux = 8U }
audio codec module play source value
- enum {
kCODEC_PlayChannelHeadphoneLeft = 1U,
kCODEC_PlayChannelHeadphoneRight = 2U,
kCODEC_PlayChannelSpeakerLeft = 4U,
kCODEC_PlayChannelSpeakerRight = 8U,
kCODEC_PlayChannelLineOutLeft = 16U,
kCODEC_PlayChannelLineOutRight = 32U,
kCODEC_PlayChannelLeft0 = 1U,
kCODEC_PlayChannelRight0 = 2U,
kCODEC_PlayChannelLeft1 = 4U,
kCODEC_PlayChannelRight1 = 8U,
kCODEC_PlayChannelLeft2 = 16U,
kCODEC_PlayChannelRight2 = 32U,
kCODEC_PlayChannelLeft3 = 64U,
kCODEC_PlayChannelRight3 = 128U }
codec play channel
- enum {

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

audio codec capability

Functions

- status_t CODEC_Init (codec_handle_t ∗handle, codec_config_t ∗config)
    Codec initilization.
- status_t CODEC_Deinit (codec_handle_t ∗handle)
    Codec de-initilization.
- status_t CODEC_SetFormat (codec_handle_t ∗handle, uint32_t mclk, uint32_t sampleRate, uint32-_t bitWidth)
    set audio data format.
- status_t CODEC_ModuleControl (codec_handle_t ∗handle, codec_module_ctrl_cmd_t cmd, uint32_t data)
    codec module control.
- status_t CODEC_SetVolume (codec_handle_t ∗handle, uint32_t channel, uint32_t volume)
    set audio codec pl volume.
- status_t CODEC_SetMute (codec_handle_t ∗handle, uint32_t channel, bool mute)
    set audio codec module mute.
- status_t CODEC_SetPower (codec_handle_t ∗handle, codec_module_t module, bool powerOn)
    set audio codec power.
- status_t CODEC_SetRecord (codec_handle_t ∗handle, uint32_t recordSource)
    codec set record source.
- status_t CODEC_SetRecordChannel (codec_handle_t ∗handle, uint32_t leftRecordChannel, uint32-_t rightRecordChannel)
    codec set record channel.
- status_t CODEC_SetPlay (codec_handle_t ∗handle, uint32_t playSource)
    codec set play source.

Driver version

- #define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
    CLOCK driver version 2.2.1.

51.2.2 Data Structure Documentation

51.2.2.1 struct codec_config_t

Data Fields

- uint32_t codecDevType
    codec type
- void ∗ codecDevConfig
    Codec device specific configuration.

51.2.2.2 struct codec_capability_t

Data Fields

- uint32_t codecModuleCapability
  codec module capability
- uint32_t codecPlayCapability
  codec play capability
- uint32_t codecRecordCapability
  codec record capability

51.2.2.3 struct _codec_handle

codec handle declaration

- Application should allocate a buffer with CODEC_HANDLE_SIZE for handle definition, such as uint8_t codecHandleBuffer[CODEC_HANDLE_SIZE]; codec_handle_t ∗codecHandle = codecHandleBuffer;

Data Fields

- codec_config_t ∗ codecConfig
  codec configuration function pointer
- const codec_capability_t ∗ codecCapability
  codec capability
- uint8_t codecDevHandle [HAL_CODEC_HANDLER_SIZE]
  codec device handle

51.2.3 Macro Definition Documentation

51.2.3.1 #define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

51.2.4 Enumeration Type Documentation

51.2.4.1 anonymous enum

Enumerator

kStatus_CODEC_NotSupport CODEC not support status.
kStatus_CODEC_DeviceNotRegistered CODEC device register failed status.
kStatus_CODEC_I2CBusInitialFailed CODEC i2c bus initialization failed status.
kStatus_CODEC_I2CCommandTransferFailed CODEC i2c bus command transfer failed status.

51.2.4.2 enum codec_audio_protocol_t

Enumerator

kCODEC_BusI2S I2S type.
kCODEC_BusLeftJustified Left justified mode.
kCODEC_BusRightJustified Right justified mode.
kCODEC_BusPCMA DSP/PCM A mode.
kCODEC_BusPCMB DSP/PCM B mode.
kCODEC_BusTDM TDM mode.

51.2.4.3 anonymous enum

Enumerator

kCODEC_AudioSampleRate8KHz Sample rate 8000 Hz.
kCODEC_AudioSampleRate11025Hz Sample rate 11025 Hz.
kCODEC_AudioSampleRate12KHz Sample rate 12000 Hz.
kCODEC_AudioSampleRate16KHz Sample rate 16000 Hz.
kCODEC_AudioSampleRate22050Hz Sample rate 22050 Hz.
kCODEC_AudioSampleRate24KHz Sample rate 24000 Hz.
kCODEC_AudioSampleRate32KHz Sample rate 32000 Hz.
kCODEC_AudioSampleRate44100Hz Sample rate 44100 Hz.
kCODEC_AudioSampleRate48KHz Sample rate 48000 Hz.
kCODEC_AudioSampleRate96KHz Sample rate 96000 Hz.
kCODEC_AudioSampleRate192KHz Sample rate 192000 Hz.
kCODEC_AudioSampleRate384KHz Sample rate 384000 Hz.

51.2.4.4 anonymous enum

Enumerator

kCODEC_AudioBitWidth16bit audio bit width 16
kCODEC_AudioBitWidth20bit audio bit width 20
kCODEC_AudioBitWidth24bit audio bit width 24
kCODEC_AudioBitWidth32bit audio bit width 32

51.2.4.5 enum codec_module_t

Enumerator

kCODEC_ModuleADC codec module ADC
kCODEC_ModuleDAC codec module DAC
kCODEC_ModulePGA codec module PGA
kCODEC_ModuleHeadphone codec module headphone

kCODEC_ModuleSpeaker codec module speaker
kCODEC_ModuleLinein codec module linein
kCODEC_ModuleLineout codec module lineout
kCODEC_ModuleVref codec module VREF
kCODEC_ModuleMicbias codec module MIC BIAS
kCODEC_ModuleMic codec module MIC
kCODEC_ModuleI2SIn codec module I2S in
kCODEC_ModuleI2SOut codec module I2S out
kCODEC_ModuleMxier codec module mixer

51.2.4.6 enum codec_module_ctrl_cmd_t

Enumerator

kCODEC_ModuleSwitchI2SInInterface module digital interface siwtch.

51.2.4.7 anonymous enum

Enumerator

kCODEC_ModuleI2SInInterfacePCM Pcm interface.
kCODEC_ModuleI2SInInterfaceDSD DSD interface.

51.2.4.8 anonymous enum

Enumerator

kCODEC_RecordSourceDifferentialLine record source from differential line
kCODEC_RecordSourceLineInput record source from line input
kCODEC_RecordSourceDifferentialMic record source from differential mic
kCODEC_RecordSourceDigitalMic record source from digital microphone
kCODEC_RecordSourceSingleEndMic record source from single microphone

51.2.4.9 anonymous enum

Enumerator

kCODEC_RecordChannelLeft1 left record channel 1
kCODEC_RecordChannelLeft2 left record channel 2
kCODEC_RecordChannelLeft3 left record channel 3
kCODEC_RecordChannelRight1 right record channel 1
kCODEC_RecordChannelRight2 right record channel 2
kCODEC_RecordChannelRight3 right record channel 3
kCODEC_RecordChannelDifferentialPositive1 differential positive record channel 1

kCODEC_RecordChannelDifferentialPositive2 differential positive record channel 2
kCODEC_RecordChannelDifferentialPositive3 differential positive record channel 3
kCODEC_RecordChannelDifferentialNegative1 differential negative record channel 1
kCODEC_RecordChannelDifferentialNegative2 differential negative record channel 2
kCODEC_RecordChannelDifferentialNegative3 differential negative record channel 3

51.2.4.10 anonymous enum

Enumerator

kCODEC_PlaySourcePGA play source PGA, bypass ADC
kCODEC_PlaySourceInput play source Input3
kCODEC_PlaySourceDAC play source DAC
kCODEC_PlaySourceMixerIn play source mixer in
kCODEC_PlaySourceMixerInLeft play source mixer in left
kCODEC_PlaySourceMixerInRight play source mixer in right
kCODEC_PlaySourceAux play source mixer in AUx

51.2.4.11 anonymous enum

Enumerator

kCODEC_PlayChannelHeadphoneLeft play channel headphone left
kCODEC_PlayChannelHeadphoneRight play channel headphone right
kCODEC_PlayChannelSpeakerLeft play channel speaker left
kCODEC_PlayChannelSpeakerRight play channel speaker right
kCODEC_PlayChannelLineOutLeft play channel lineout left
kCODEC_PlayChannelLineOutRight play channel lineout right
kCODEC_PlayChannelLeft0 play channel left0
kCODEC_PlayChannelRight0 play channel right0
kCODEC_PlayChannelLeft1 play channel left1
kCODEC_PlayChannelRight1 play channel right1
kCODEC_PlayChannelLeft2 play channel left2
kCODEC_PlayChannelRight2 play channel right2
kCODEC_PlayChannelLeft3 play channel left3
kCODEC_PlayChannelRight3 play channel right3

51.2.4.12 anonymous enum

Enumerator

kCODEC_SupportModuleADC codec capability of module ADC
kCODEC_SupportModuleDAC codec capability of module DAC
kCODEC_SupportModulePGA codec capability of module PGA
kCODEC_SupportModuleHeadphone codec capability of module headphone

kCODEC_SupportModuleSpeaker codec capability of module speaker
kCODEC_SupportModuleLinein codec capability of module linein
kCODEC_SupportModuleLineout codec capability of module lineout
kCODEC_SupportModuleVref codec capability of module vref
kCODEC_SupportModuleMicbias codec capability of module mic bias
kCODEC_SupportModuleMic codec capability of module mic bias
kCODEC_SupportModuleI2SIn codec capability of module I2S in
kCODEC_SupportModuleI2SOut codec capability of module I2S out
kCODEC_SupportModuleMixer codec capability of module mixer
kCODEC_SupportModuleI2SInSwitchInterface codec capability of module I2S in switch interface

kCODEC_SupportPlayChannelLeft0 codec capability of play channel left 0
kCODEC_SupportPlayChannelRight0 codec capability of play channel right 0
kCODEC_SupportPlayChannelLeft1 codec capability of play channel left 1
kCODEC_SupportPlayChannelRight1 codec capability of play channel right 1
kCODEC_SupportPlayChannelLeft2 codec capability of play channel left 2
kCODEC_SupportPlayChannelRight2 codec capability of play channel right 2
kCODEC_SupportPlayChannelLeft3 codec capability of play channel left 3
kCODEC_SupportPlayChannelRight3 codec capability of play channel right 3
kCODEC_SupportPlaySourcePGA codec capability of set playback source PGA
kCODEC_SupportPlaySourceInput codec capability of set playback source INPUT
kCODEC_SupportPlaySourceDAC codec capability of set playback source DAC
kCODEC_SupportPlaySourceMixerIn codec capability of set play source Mixer in
kCODEC_SupportPlaySourceMixerInLeft codec capability of set play source Mixer in left
kCODEC_SupportPlaySourceMixerInRight codec capability of set play source Mixer in right
kCODEC_SupportPlaySourceAux codec capability of set play source aux
kCODEC_SupportRecordSourceDifferentialLine codec capability of record source differential line

kCODEC_SupportRecordSourceLineInput codec capability of record source line input
kCODEC_SupportRecordSourceDifferentialMic codec capability of record source differential mic

kCODEC_SupportRecordSourceDigitalMic codec capability of record digital mic
kCODEC_SupportRecordSourceSingleEndMic codec capability of single end mic
kCODEC_SupportRecordChannelLeft1 left record channel 1
kCODEC_SupportRecordChannelLeft2 left record channel 2
kCODEC_SupportRecordChannelLeft3 left record channel 3
kCODEC_SupportRecordChannelRight1 right record channel 1
kCODEC_SupportRecordChannelRight2 right record channel 2
kCODEC_SupportRecordChannelRight3 right record channel 3

51.2.5 Function Documentation

51.2.5.1 status_t CODEC_Init (codec_handle_t * *handle,* codec_config_t * *config*)

Parameters

handle	codec handle.
config	codec configurations.

Returns

kStatus_Success is success, else de-initial failed.

51.2.5.2 status_t CODEC_Deinit (codec_handle_t * *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

51.2.5.3 status_t CODEC_SetFormat (codec_handle_t * *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

51.2.5.4 status_t CODEC_ModuleControl (codec_handle_t * *handle,* codec_module_ctrl_cmd_t *cmd,* uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature.

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

51.2.5.5 status_t CODEC_SetVolume (codec_handle_t ∗ *handle,* uint32_t *channel,* uint32_t *volume*)

Parameters

handle	codec handle.
channel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 ∼ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

51.2.5.6 status_t CODEC_SetMute (codec_handle_t ∗ *handle,* uint32_t *channel,* bool *mute*)

Parameters

handle	codec handle.
channel	audio codec play channel, can be a value or combine value of _codec_play_channel.
mute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

51.2.5.7 status_t CODEC_SetPower (codec_handle_t * *handle,* codec_module_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

51.2.5.8 status_t CODEC_SetRecord (codec_handle_t ∗ *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

51.2.5.9 status_t CODEC_SetRecordChannel (codec_handle_t ∗ *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

51.2.5.10 status_t CODEC_SetPlay (codec_handle_t ∗ *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

Chapter 52
Serial_Manager

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.

Chapter 53
Dspi_cmsis_driver

This section describes the programming interface of the DSPI 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.

Function groups

53.1.1 DSPI CMSIS GetVersion Operation

This function group will return the DSPI CMSIS Driver version to user.

53.1.2 DSPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

53.1.3 DSPI 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.

53.1.4 DSPI CMSIS Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

53.1.5 DSPI CMSIS Status Operation

This function group gets the DSPI transfer status.

53.1.6 DSPI 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.

Typical use case

53.2.1 Master Operation

```
/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*DSPI master init*/
Driver_SPI0.Initialize(DSPI_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();
```

53.2.2 Slave Operation

```
/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*DSPI slave init*/
Driver_SPI1.Initialize(DSPI_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 54
Enet_cmsis_driver

This section describes the programming interface of the ENET 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 ENET 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.

Typical use case

```
void ENET_SignalEvent_t(uint32_t event)
{
    if (event == ARM_ETH_MAC_EVENT_RX_FRAME)
    {
        uint32_t size;
        uint32_t len;

        /* Get the Frame size */
        size =  EXAMPLE_ENET.GetRxFrameSize();
        /* Call ENET_ReadFrame when there is a received frame. */
        if (size != 0)
        {
            /* Received valid frame. Deliver the rx buffer with the size equal to length. */
            uint8_t *data = (uint8_t *)malloc(size);
            if (data)
            {
                len = EXAMPLE_ENET.ReadFrame(data, size);
                if (size == len)
                {
                    /* Increase the received frame numbers. */
                    if (g_rxIndex < ENET_EXAMPLE_LOOP_COUNT)
                    {
                        g_rxIndex++;
                    }
                }
                free(data);
            }
         }
    }
    if (event == ARM_ETH_MAC_EVENT_TX_FRAME)
    {
        g_testTxNum ++;
    }
}

    /* Initialize the ENET module. */
    EXAMPLE_ENET.Initialize(ENET_SignalEvent_t);
```

```
EXAMPLE_ENET.PowerControl(ARM_POWER_FULL);
EXAMPLE_ENET.SetMacAddress((ARM_ETH_MAC_ADDR *)g_macAddr);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONFIGURE, linkInfo.speed << ARM_ETH_MAC_SPEED_Pos  |
             linkInfo.duplex << ARM_ETH_MAC_DUPLEX_Pos | ARM_ETH_MAC_ADDRESS_BROADCAST);
EXAMPLE_ENET_PHY.PowerControl(ARM_POWER_FULL);
EXAMPLE_ENET_PHY.SetMode(ARM_ETH_PHY_AUTO_NEGOTIATE);
  EXAMPLE_ENET.Control(ARM_ETH_MAC_CONTROL_RX, 1);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONTROL_TX, 1);
if (EXAMPLE_ENET_PHY.GetLinkState() == ARM_ETH_LINK_UP)
{
    linkInfo = EXAMPLE_ENET_PHY.GetLinkInfo();
}
else
{
    PRINTF("\r\nPHY Link down, please check the cable connection and link partner setting.\r\n");
}

/* Build broadcast for sending. */
ENET_BuildBroadCastFrame();

while (1)
{
    /* Check the total number of received number. */
    if (g_rxCheckIdx != g_rxIndex)
    {
        PRINTF("The %d frame has been successfuly received!\r\n", g_rxIndex);
        g_rxCheckIdx = g_rxIndex;
    }
    if ( g_testTxNum && (g_txCheckIdx != g_testTxNum))
    {
        g_txCheckIdx = g_testTxNum;
        PRINTF("The %d frame transmitted success!\r\n", g_txCheckIdx);
    }
    /* Get the Frame size */
    if (txnumber < ENET_EXAMPLE_LOOP_COUNT)
    {
        txnumber ++;
        /* Send a multicast frame when the PHY is link up. */
        if (EXAMPLE_ENET.SendFrame(&g_frame[0], ENET_DATA_LENGTH, ARM_ETH_MAC_TX_FRAME_EVENT) ==
  ARM_DRIVER_OK)
        {
            for (uint32_t count = 0; count < 0x3FF; count++)
            {
                __ASM("nop");
            }
        }
        else
        {
            PRINTF(" \r\nTransmit frame failed!\r\n");
        }
     }
}
```

Chapter 55
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.

I2C CMSIS Driver

55.1.1 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;
```

55.1.2 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;
```

55.1.3 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 56
Lpuart_cmsis_driver

This section describes the programming interface of the LPUART 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`.

The LPUART 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.

Function groups

56.1.1 LPUART CMSIS GetVersion Operation

This function group will return the LPUART CMSIS Driver version to user.

56.1.2 LPUART CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

56.1.3 LPUART CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize the lpuart instance . And this API must be called before you configure a lpuart instance or after you Deinit a lpuart 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.

56.1.4 LPUART CMSIS Transfer Operation

This function group controls the transfer, send/receive data.

56.1.5 LPUART CMSIS Status Operation

This function group gets the LPUART transfer status.

56.1.6 LPUART CMSIS Control Operation

This function can configure an instance ,set baudrate for lpuart, get current baudrate ,set transfer data bits and other control command.

Chapter 57
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.

UART CMSIS Driver

57.1.1 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)
{
}
```

57.1.2 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 58
Cache

Overview

Macros

- #define L1CODEBUSCACHE_LINESIZE_BYTE FSL_FEATURE_L1ICACHE_LINESIZE_BYTE
  code bus cache line size is equal to system bus line size, so the unified I/D cache line size equals too.
- #define L1SYSTEMBUSCACHE_LINESIZE_BYTE L1CODEBUSCACHE_LINESIZE_BYTE
  The system bus CACHE line size is 16B = 128b.

Driver version

- #define FSL_CACHE_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))
  cache driver version.

cache control for L1 cache (local memory controller for code/system bus cache)

- void L1CACHE_EnableCodeCache (void)
  Enables the processor code bus cache.
- void L1CACHE_DisableCodeCache (void)
  Disables the processor code bus cache.
- void L1CACHE_InvalidateCodeCache (void)
  Invalidates the processor code bus cache.
- void L1CACHE_InvalidateCodeCacheByRange (uint32_t address, uint32_t size_byte)
  Invalidates processor code bus cache by range.
- void L1CACHE_CleanCodeCache (void)
  Cleans the processor code bus cache.
- void L1CACHE_CleanCodeCacheByRange (uint32_t address, uint32_t size_byte)
  Cleans processor code bus cache by range.
- void L1CACHE_CleanInvalidateCodeCache (void)
  Cleans and invalidates the processor code bus cache.
- void L1CACHE_CleanInvalidateCodeCacheByRange (uint32_t address, uint32_t size_byte)
  Cleans and invalidate processor code bus cache by range.
- static void L1CACHE_EnableCodeCacheWriteBuffer (bool enable)
  Enables/disables the processor code bus write buffer.

cache control for unified L1 cache driver

- void L1CACHE_InvalidateICacheByRange (uint32_t address, uint32_t size_byte)
  Invalidates cortex-m4 L1 instrument cache by range.
- static void L1CACHE_InvalidateDCacheByRange (uint32_t address, uint32_t size_byte)
  Invalidates cortex-m4 L1 data cache by range.
- void L1CACHE_CleanDCacheByRange (uint32_t address, uint32_t size_byte)
  Cleans cortex-m4 L1 data cache by range.

- void L1CACHE_CleanInvalidateDCacheByRange (uint32_t address, uint32_t size_byte)
  Cleans and Invalidates cortex-m4 L1 data cache by range.

Unified Cache Control for all caches

- static void ICACHE_InvalidateByRange (uint32_t address, uint32_t size_byte)
  Invalidates instruction cache by range.
- static void DCACHE_InvalidateByRange (uint32_t address, uint32_t size_byte)
  Invalidates data cache by range.
- static void DCACHE_CleanByRange (uint32_t address, uint32_t size_byte)
  Clean data cache by range.
- static void DCACHE_CleanInvalidateByRange (uint32_t address, uint32_t size_byte)
  Cleans and Invalidates data cache by range.

Macro Definition Documentation

58.2.1 #define FSL_CACHE_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

58.2.2 #define L1CODEBUSCACHE_LINESIZE_BYTE FSL_FEATURE_L1ICACHE_LINESIZE_BYTE

The code bus CACHE line size is 16B = 128b.

58.2.3 #define L1SYSTEMBUSCACHE_LINESIZE_BYTE L1CODEBUSCACHE_LINESIZE_BYTE

Function Documentation

58.3.1 void L1CACHE_InvalidateCodeCacheByRange (uint32_t *address*, uint32_t *size_byte*)

Parameters

address	The physical address of cache.
size_byte	size of the memory to be invalidated.

Note

Address and size should be aligned to "L1CODCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

58.3.2 void L1CACHE_CleanCodeCacheByRange (uint32_t *address*, uint32_t *size_byte*)

Parameters

address	The physical address of cache.
size_byte	size of the memory to be cleaned.

Note

Address and size should be aligned to "L1CODEBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

58.3.3 void L1CACHE_CleanInvalidateCodeCacheByRange (uint32_t *address*, uint32_t *size_byte*)

Parameters

address	The physical address of cache.
size_byte	size of the memory to be Cleaned and Invalidated.

Note

Address and size should be aligned to "L1CODEBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

58.3.4 static void L1CACHE_EnableCodeCacheWriteBuffer (bool *enable*) [inline], [static]

Parameters

enable	The enable or disable flag. true - enable the code bus write buffer. false - disable the code bus write buffer.

58.3.5 void L1CACHE_InvalidateICacheByRange (uint32_t *address*, uint32_t *size_byte*)

Parameters

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

Note

The start address and size_byte should be 16-Byte(FSL_FEATURE_L1ICACHE_LINESIZE_BYTE) aligned.

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

Parameters

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

Note

The start address and size_byte should be 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

58.3.7 void L1CACHE_CleanDCacheByRange (uint32_t *address*, uint32_t *size_byte*)

Parameters

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

Note

The start address and size_byte should be 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

58.3.8 void L1CACHE_CleanInvalidateDCacheByRange (uint32_t *address*, uint32_t *size_byte*)

Parameters

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

Note

The start address and size_byte should be 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

58.3.9 static void ICACHE_InvalidateByRange (uint32_t *address*, uint32_t *size_byte*) [inline], [static]

Parameters

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

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1ICACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

58.3.10 static void DCACHE_InvalidateByRange (uint32_t *address*, uint32_t *size_byte*) [inline], [static]

Parameters

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

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

58.3.11 static void DCACHE_CleanByRange (uint32_t *address,* uint32_t *size_byte*) [inline], [static]

Parameters

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

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_-L1DCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

58.3.12 static void DCACHE_CleanInvalidateByRange (uint32_t *address*, uint32_t *size_byte*) [inline], [static]

Parameters

address	The physical address.
size_byte	size of the memory to be Cleaned and Invalidated.

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_-L1DCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Chapter 59
Dspi_edma_driver

Overview

Data Structures

- struct dspi_master_edma_handle_t
  DSPI master eDMA transfer handle structure used for the transactional API. More...
- struct dspi_slave_edma_handle_t
  DSPI slave eDMA transfer handle structure used for the transactional API. More...

Macros

- #define DSPI_EDMA_MAX_TRANSFER_SIZE(base, width)
  DSPI EDMA max transfer data size calculate.

Typedefs

- typedef void(∗ dspi_master_edma_transfer_callback_t)(SPI_Type ∗base, dspi_master_edma_handle_t ∗handle, status_t status, void ∗userData)
  Completion callback function pointer type.
- typedef void(∗ dspi_slave_edma_transfer_callback_t)(SPI_Type ∗base, dspi_slave_edma_handle_t ∗handle, status_t status, void ∗userData)
  Completion callback function pointer type.

Driver version

- #define FSL_DSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))
  DSPI EDMA driver version 2.2.4.

Transactional APIs

- void DSPI_MasterTransferCreateHandleEDMA (SPI_Type ∗base, dspi_master_edma_handle_t ∗handle, dspi_master_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗edmaRxRegToRxDataHandle, edma_handle_t ∗edmaTxDataToIntermediaryHandle, edma_handle_t ∗edmaIntermediaryToTxRegHandle)
  Initializes the DSPI master eDMA handle.
- status_t DSPI_MasterTransferEDMA (SPI_Type ∗base, dspi_master_edma_handle_t ∗handle, dspi_transfer_t ∗transfer)
  DSPI master transfer data using eDMA.
- status_t DSPI_MasterHalfDuplexTransferEDMA (SPI_Type ∗base, dspi_master_edma_handle_t ∗handle, dspi_half_duplex_transfer_t ∗xfer)
  Transfers a block of data using a eDMA method.
- void DSPI_MasterTransferAbortEDMA (SPI_Type ∗base, dspi_master_edma_handle_t ∗handle)
  DSPI master aborts a transfer which is using eDMA.

- status_t DSPI_MasterTransferGetCountEDMA (SPI_Type *base, dspi_master_edma_handle_t *handle, size_t *count)
  Gets the master eDMA transfer count.
- void DSPI_SlaveTransferCreateHandleEDMA (SPI_Type *base, dspi_slave_edma_handle_t *handle, dspi_slave_edma_transfer_callback_t callback, void *userData, edma_handle_t *edmaRxRegToRxDataHandle, edma_handle_t *edmaTxDataToTxRegHandle)
  Initializes the DSPI slave eDMA handle.
- status_t DSPI_SlaveTransferEDMA (SPI_Type *base, dspi_slave_edma_handle_t *handle, dspi_transfer_t *transfer)
  DSPI slave transfer data using eDMA.
- void DSPI_SlaveTransferAbortEDMA (SPI_Type *base, dspi_slave_edma_handle_t *handle)
  DSPI slave aborts a transfer which is using eDMA.
- status_t DSPI_SlaveTransferGetCountEDMA (SPI_Type *base, dspi_slave_edma_handle_t *handle, size_t *count)
  Gets the slave eDMA transfer count.

Data Structure Documentation

59.2.1 struct _dspi_master_edma_handle

Forward declaration of the DSPI eDMA master handle typedefs.

Data Fields

- uint32_t bitsPerFrame
  The desired number of bits per frame.
- volatile uint32_t command
  The desired data command.
- volatile uint32_t lastCommand
  The desired last data command.
- uint8_t fifoSize
  FIFO dataSize.
- volatile bool isPcsActiveAfterTransfer
  Indicates whether the PCS signal keeps active after the last frame transfer.
- uint8_t nbytes
  eDMA minor byte transfer count initially configured.
- volatile uint8_t state
  DSPI transfer state, see _dspi_transfer_state.
- uint8_t *volatile txData
  Send buffer.
- uint8_t *volatile rxData
  Receive buffer.
- volatile size_t remainingSendByteCount
  A number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount
  A number of bytes remaining to receive.
- size_t totalByteCount
  A number of transfer bytes.

- uint32_t rxBuffIfNull
  Used if there is not rxData for DMA purpose.
- uint32_t txBuffIfNull
  Used if there is not txData for DMA purpose.
- dspi_master_edma_transfer_callback_t callback
  Completion callback.
- void * userData
  Callback user data.
- edma_handle_t * edmaRxRegToRxDataHandle
  edma_handle_t handle point used for RxReg to RxData buff
- edma_handle_t * edmaTxDataToIntermediaryHandle
  edma_handle_t handle point used for TxData to Intermediary
- edma_handle_t * edmaIntermediaryToTxRegHandle
  edma_handle_t handle point used for Intermediary to TxReg
- edma_tcd_t dspiSoftwareTCD [2]
  SoftwareTCD , internal used.

59.2.1.0.0.52 Field Documentation

59.2.1.0.0.52.1 uint32_t dspi_master_edma_handle_t::bitsPerFrame

59.2.1.0.0.52.2 volatile uint32_t dspi_master_edma_handle_t::command

59.2.1.0.0.52.3 volatile uint32_t dspi_master_edma_handle_t::lastCommand

59.2.1.0.0.52.4 uint8_t dspi_master_edma_handle_t::fifoSize

59.2.1.0.0.52.5 volatile bool dspi_master_edma_handle_t::isPcsActiveAfterTransfer

59.2.1.0.0.52.6 uint8_t dspi_master_edma_handle_t::nbytes

59.2.1.0.0.52.7 volatile uint8_t dspi_master_edma_handle_t::state

59.2.1.0.0.52.8 uint8_t∗ volatile dspi_master_edma_handle_t::txData

59.2.1.0.0.52.9 uint8_t∗ volatile dspi_master_edma_handle_t::rxData

59.2.1.0.0.52.10 volatile size_t dspi_master_edma_handle_t::remainingSendByteCount

59.2.1.0.0.52.11 volatile size_t dspi_master_edma_handle_t::remainingReceiveByteCount

59.2.1.0.0.52.12 uint32_t dspi_master_edma_handle_t::rxBuffIfNull

59.2.1.0.0.52.13 uint32_t dspi_master_edma_handle_t::txBuffIfNull

59.2.1.0.0.52.14 dspi_master_edma_transfer_callback_t dspi_master_edma_handle_t::callback

59.2.1.0.0.52.15 void∗ dspi_master_edma_handle_t::userData

59.2.2 struct _dspi_slave_edma_handle

Forward declaration of the DSPI eDMA slave handle typedefs.

Data Fields

- uint32_t bitsPerFrame
  The desired number of bits per frame.
- uint8_t ∗volatile txData
  Send buffer.
- uint8_t ∗volatile rxData
  Receive buffer.
- volatile size_t remainingSendByteCount
  A number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount
  A number of bytes remaining to receive.
- size_t totalByteCount

A number of transfer bytes.
- uint32_t rxBuffIfNull
  Used if there is not rxData for DMA purpose.
- uint32_t txBuffIfNull
  Used if there is not txData for DMA purpose.
- uint32_t txLastData
  Used if there is an extra byte when 16bits per frame for DMA purpose.
- uint8_t nbytes
  eDMA minor byte transfer count initially configured.
- volatile uint8_t state
  DSPI transfer state.
- dspi_slave_edma_transfer_callback_t callback
  Completion callback.
- void * userData
  Callback user data.
- edma_handle_t * edmaRxRegToRxDataHandle
  edma_handle_t handle point used for RxReg to RxData buff
- edma_handle_t * edmaTxDataToTxRegHandle
  edma_handle_t handle point used for TxData to TxReg

59.2.2.0.0.53 Field Documentation

59.2.2.0.0.53.1 uint32_t dspi_slave_edma_handle_t::bitsPerFrame

59.2.2.0.0.53.2 uint8_t* volatile dspi_slave_edma_handle_t::txData

59.2.2.0.0.53.3 uint8_t* volatile dspi_slave_edma_handle_t::rxData

59.2.2.0.0.53.4 volatile size_t dspi_slave_edma_handle_t::remainingSendByteCount

59.2.2.0.0.53.5 volatile size_t dspi_slave_edma_handle_t::remainingReceiveByteCount

59.2.2.0.0.53.6 uint32_t dspi_slave_edma_handle_t::rxBuffIfNull

59.2.2.0.0.53.7 uint32_t dspi_slave_edma_handle_t::txBuffIfNull

59.2.2.0.0.53.8 uint32_t dspi_slave_edma_handle_t::txLastData

59.2.2.0.0.53.9 uint8_t dspi_slave_edma_handle_t::nbytes

59.2.2.0.0.53.10 volatile uint8_t dspi_slave_edma_handle_t::state

59.2.2.0.0.53.11 dspi_slave_edma_transfer_callback_t dspi_slave_edma_handle_t::callback

59.2.2.0.0.53.12 void* dspi_slave_edma_handle_t::userData

Macro Definition Documentation

59.3.1 #define DSPI_EDMA_MAX_TRANSFER_SIZE(*base, width*)

Value:

```
((1 == FSL_FEATURE_DSPI_HAS_SEPARATE_DMA_RX_TX_REQn(base)) ? ((width > 8U) ? 65534U : 32767U) : \
                                                                         ((width > 8U) ? 1022U : 511U))
```

Parameters

base	DSPI peripheral base address.
width	Transfer width

Typedef Documentation

59.4.1 typedef void(∗ dspi_master_edma_transfer_callback_t)(SPI_Type ∗base, dspi_master_edma_handle_t ∗handle, status_t status, void ∗userData)

Parameters

base	DSPI peripheral base address.
handle	A pointer to the handle for the DSPI master.
status	Success or error code describing whether the transfer completed.
userData	An arbitrary pointer-dataSized value passed from the application.

59.4.2 typedef void(∗ dspi_slave_edma_transfer_callback_t)(SPI_Type ∗base, dspi_slave_edma_handle_t ∗handle, status_t status, void ∗userData)

Parameters

base	DSPI peripheral base address.
handle	A pointer to the handle for the DSPI slave.
status	Success or error code describing whether the transfer completed.
userData	An arbitrary pointer-dataSized value passed from the application.

Function Documentation

59.5.1 void DSPI_MasterTransferCreateHandleEDMA (SPI_Type ∗ *base,* dspi_master_edma_handle_t ∗ *handle,* dspi_master_edma_transfer_callback_t *callback,* void ∗ *userData,* edma_handle_t ∗ *edmaRxRegToRxDataHandle,* edma_handle_t ∗ *edmaTxDataToIntermediaryHandle,* edma_handle_t ∗ *edmaIntermediaryToTxRegHandle*)

This function initializes the DSPI eDMA handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Note

DSPI eDMA has separated (RX and TX as two sources) or shared (RX and TX are the same source) DMA request source.

- For the separated DMA request source, enable and set the RX DMAMUX source for edmaRxRegToRxDataHandle and TX DMAMUX source for edmaIntermediaryToTxRegHandle.
- For the shared DMA request source, enable and set the RX/RX DMAMUX source for the edmaRxRegToRxDataHandle.

Parameters

base	DSPI peripheral base address.
handle	DSPI handle pointer to _dspi_master_edma_handle.
callback	DSPI callback.
userData	A callback function parameter.
edmaRxRegToRxDataHandle	edmaRxRegToRxDataHandle pointer to edma_handle_t.
edmaTxDataToIntermediaryHandle	edmaTxDataToIntermediaryHandle pointer to edma_handle_t.
edmaIntermediaryToTxRegHandle	edmaIntermediaryToTxRegHandle pointer to edma_handle_t.

59.5.2 status_t DSPI_MasterTransferEDMA (SPI_Type * *base*, dspi_master_edma_handle_t * *handle*, dspi_transfer_t * *transfer*)

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note

The max transfer size of each transfer depends on whether the instance's Tx/Rx shares the same DMA request. If **FSL_FEATURE_DSPI_HAS_SEPARATE_DMA_RX_TX_REQn(x)** is true, then the max transfer size is 32767 datawidth of data, otherwise is 511.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the _dspi_master_edma_handle structure which stores the transfer state.
transfer	A pointer to the dspi_transfer_t structure.

Returns

status of status_t.

59.5.3 status_t DSPI_MasterHalfDuplexTransferEDMA (SPI_Type * *base*, dspi_master_edma_handle_t * *handle*, dspi_half_duplex_transfer_t * *xfer*)

This function transfers data using eDNA, the transfer mechanism is half-duplex. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

base	DSPI base pointer
handle	A pointer to the _dspi_master_edma_handle structure which stores the transfer state.
xfer	A pointer to the dspi_half_duplex_transfer_t structure.

Returns

status of status_t.

59.5.4 void DSPI_MasterTransferAbortEDMA (SPI_Type * *base*, dspi_master_edma_handle_t * *handle*)

This function aborts a transfer which is using eDMA.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the _dspi_master_edma_handle structure which stores the transfer state.

59.5.5 status_t DSPI_MasterTransferGetCountEDMA (SPI_Type * *base*, dspi_master_edma_handle_t * *handle*, size_t * *count*)

This function gets the master eDMA transfer count.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the _dspi_master_edma_handle structure which stores the transfer state.
count	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

59.5.6 void DSPI_SlaveTransferCreateHandleEDMA (SPI_Type * *base*, dspi_slave_edma_handle_t * *handle*, dspi_slave_edma_transfer_callback_t *callback*, void * *userData*, edma_handle_t * *edmaRxRegToRxDataHandle*, edma_handle_t * *edmaTxDataToTxRegHandle*)

This function initializes the DSPI eDMA handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Note

DSPI eDMA has separated (RN and TX in 2 sources) or shared (RX and TX are the same source) DMA request source.

- For the separated DMA request source, enable and set the RX DMAMUX source for edmaRxRegToRxDataHandle and TX DMAMUX source for edmaTxDataToTxRegHandle.
- For the shared DMA request source, enable and set the RX/RX DMAMUX source for the edmaRxRegToRxDataHandle.

Parameters

base	DSPI peripheral base address.
handle	DSPI handle pointer to _dspi_slave_edma_handle.
callback	DSPI callback.
userData	A callback function parameter.
edmaRxRegTo-RxDataHandle	edmaRxRegToRxDataHandle pointer to edma_handle_t.

edmaTxDataToTxRegHandle	edmaTxDataToTxRegHandle pointer to edma_handle_t.

59.5.7 status_t DSPI_SlaveTransferEDMA (SPI_Type * *base*, dspi_slave_edma_handle_t * *handle*, dspi_transfer_t * *transfer*)

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called. Note that the slave eDMA transfer doesn't support transfer_size is 1 when the bitsPerFrame is greater than eight.

Note

The max transfer size of each transfer depends on whether the instance's Tx/Rx shares the same DMA request. If **FSL_FEATURE_DSPI_HAS_SEPARATE_DMA_RX_TX_REQn(x)** is true, then the max transfer size is 32767 datawidth of data, otherwise is 511.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the _dspi_slave_edma_handle structure which stores the transfer state.
transfer	A pointer to the dspi_transfer_t structure.

Returns

status of status_t.

59.5.8 void DSPI_SlaveTransferAbortEDMA (SPI_Type * *base*, dspi_slave_edma_handle_t * *handle*)

This function aborts a transfer which is using eDMA.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the _dspi_slave_edma_handle structure which stores the transfer state.

59.5.9 status_t DSPI_SlaveTransferGetCountEDMA (SPI_Type * *base*, dspi_slave_edma_handle_t * *handle*, size_t * *count*)

This function gets the slave eDMA transfer count.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the _dspi_slave_edma_handle structure which stores the transfer state.
count	A number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

Chapter 60
Dspi_freertos_driver

Overview

Driver version

- #define FSL_DSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))
  DSPI FreeRTOS driver version 2.2.4.

DSPI RTOS Operation

- status_t DSPI_RTOS_Init (dspi_rtos_handle_t ∗handle, SPI_Type ∗base, const dspi_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes the DSPI.
- status_t DSPI_RTOS_Deinit (dspi_rtos_handle_t ∗handle)
  Deinitializes the DSPI.
- status_t DSPI_RTOS_Transfer (dspi_rtos_handle_t ∗handle, dspi_transfer_t ∗transfer)
  Performs the SPI transfer.

Macro Definition Documentation

60.2.1 #define FSL_DSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))

Function Documentation

60.3.1 status_t DSPI_RTOS_Init (dspi_rtos_handle_t ∗ *handle,* SPI_Type ∗ *base,* const dspi_master_config_t ∗ *masterConfig,* uint32_t *srcClock_Hz*)

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

Parameters

handle	The RTOS DSPI handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the DSPI instance to initialize.
masterConfig	A configuration structure to set-up the DSPI in master mode.

srcClock_Hz	A frequency of the input clock of the DSPI module.

Returns

status of the operation.

60.3.2 status_t DSPI_RTOS_Deinit (dspi_rtos_handle_t ∗ *handle*)

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

Parameters

handle	The RTOS DSPI handle.

60.3.3 status_t DSPI_RTOS_Transfer (dspi_rtos_handle_t ∗ *handle*, dspi_transfer_t ∗ *transfer*)

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

Parameters

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

Returns

status of the operation.

Chapter 61
I2c_edma_driver

Overview

Data Structures

- struct i2c_master_edma_handle_t
  I2C master eDMA transfer structure. More...

Typedefs

- typedef void(∗ i2c_master_edma_transfer_callback_t)(I2C_Type ∗base, i2c_master_edma_handle_t ∗handle, status_t status, void ∗userData)
  I2C master eDMA transfer callback typedef.

Driver version

- #define FSL_I2C_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 9))
  I2C EDMA driver version.

I2C Block eDMA Transfer Operation

- void I2C_MasterCreateEDMAHandle (I2C_Type ∗base, i2c_master_edma_handle_t ∗handle, i2c_master_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗edmaHandle)
  Initializes the I2C handle which is used in transactional functions.
- status_t I2C_MasterTransferEDMA (I2C_Type ∗base, i2c_master_edma_handle_t ∗handle, i2c_master_transfer_t ∗xfer)
  Performs a master eDMA non-blocking transfer on the I2C bus.
- status_t I2C_MasterTransferGetCountEDMA (I2C_Type ∗base, i2c_master_edma_handle_t ∗handle, size_t ∗count)
  Gets a master transfer status during the eDMA non-blocking transfer.
- void I2C_MasterTransferAbortEDMA (I2C_Type ∗base, i2c_master_edma_handle_t ∗handle)
  Aborts a master eDMA non-blocking transfer early.

Data Structure Documentation

61.2.1 struct _i2c_master_edma_handle

Retry times for waiting flag.

I2C master eDMA handle typedef.

Data Fields

- i2c_master_transfer_t transfer

I2C master transfer structure.
- size_t transferSize
  Total bytes to be transferred.
- uint8_t nbytes
  eDMA minor byte transfer count initially configured.
- uint8_t state
  I2C master transfer status.
- edma_handle_t * dmaHandle
  The eDMA handler used.
- i2c_master_edma_transfer_callback_t completionCallback
  A callback function called after the eDMA transfer is finished.
- void * userData
  A callback parameter passed to the callback function.

61.2.1.0.0.54 Field Documentation

61.2.1.0.0.54.1 i2c_master_transfer_t i2c_master_edma_handle_t::transfer

61.2.1.0.0.54.2 size_t i2c_master_edma_handle_t::transferSize

61.2.1.0.0.54.3 uint8_t i2c_master_edma_handle_t::nbytes

61.2.1.0.0.54.4 uint8_t i2c_master_edma_handle_t::state

61.2.1.0.0.54.5 edma_handle_t* i2c_master_edma_handle_t::dmaHandle

61.2.1.0.0.54.6 i2c_master_edma_transfer_callback_t i2c_master_edma_handle_t::completionCallback

61.2.1.0.0.54.7 void* i2c_master_edma_handle_t::userData

Macro Definition Documentation

61.3.1 #define FSL_I2C_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 9))

Typedef Documentation

61.4.1 typedef void(* i2c_master_edma_transfer_callback_t)(I2C_Type *base, i2c_master_edma_handle_t *handle, status_t status, void *userData)

Function Documentation

61.5.1 void I2C_MasterCreateEDMAHandle (I2C_Type * *base,* i2c_master_edma_handle_t * *handle,* i2c_master_edma_transfer_callback_t *callback,* void * *userData,* edma_handle_t * *edmaHandle*)

Parameters

base	I2C peripheral base address.
handle	A pointer to the i2c_master_edma_handle_t structure.
callback	A pointer to the user callback function.
userData	A user parameter passed to the callback function.
edmaHandle	eDMA handle pointer.

61.5.2 status_t I2C_MasterTransferEDMA (I2C_Type * *base*, i2c_master_edma_handle_t * *handle*, i2c_master_transfer_t * *xfer*)

Parameters

base	I2C peripheral base address.
handle	A pointer to the i2c_master_edma_handle_t structure.
xfer	A pointer to the transfer structure of i2c_master_transfer_t.

Return values

kStatus_Success	Successfully completed the data transmission.
kStatus_I2C_Busy	A previous transmission is still not finished.
kStatus_I2C_Timeout	Transfer error, waits for a signal timeout.
kStatus_I2C_ArbitrationLost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.

61.5.3 status_t I2C_MasterTransferGetCountEDMA (I2C_Type * *base*, i2c_master_edma_handle_t * *handle*, size_t * *count*)

Parameters

base	I2C peripheral base address.
handle	A pointer to the i2c_master_edma_handle_t structure.
count	A number of bytes transferred by the non-blocking transaction.

61.5.4 void I2C_MasterTransferAbortEDMA (I2C_Type ∗ *base*, i2c_master_edma_handle_t ∗ *handle*)

Parameters

base	I2C peripheral base address.
handle	A pointer to the i2c_master_edma_handle_t structure.

Chapter 62
I2C FreeRTOS Driver

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.

Macro Definition Documentation

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

Function Documentation

62.3.1 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.

62.3.2 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.

62.3.3 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.

Chapter 63
Lpuart_edma_driver

Overview

Data Structures

- struct lpuart_edma_handle_t
  LPUART eDMA handle. More...

Typedefs

- typedef void(∗ lpuart_edma_transfer_callback_t)(LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, status_t status, void ∗userData)
  LPUART transfer callback function.

Driver version

- #define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
  LPUART EDMA driver version 2.4.0.

eDMA transactional

- void LPUART_TransferCreateHandleEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, lpuart_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗txEdmaHandle, edma_handle_t ∗rxEdmaHandle)
  Initializes the LPUART handle which is used in transactional functions.
- status_t LPUART_SendEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  Sends data using eDMA.
- status_t LPUART_ReceiveEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  Receives data using eDMA.
- void LPUART_TransferAbortSendEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle)
  Aborts the sent data using eDMA.
- void LPUART_TransferAbortReceiveEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle)
  Aborts the received data using eDMA.
- status_t LPUART_TransferGetSendCountEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, uint32_t ∗count)
  Gets the number of bytes written to the LPUART TX register.
- status_t LPUART_TransferGetReceiveCountEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, uint32_t ∗count)
  Gets the number of received bytes.

Data Structure Documentation

63.2.1 struct _lpuart_edma_handle

Data Fields

- lpuart_edma_transfer_callback_t callback
  Callback function.
- void ∗ userData
  LPUART callback function parameter.
- size_t rxDataSizeAll
  Size of the data to receive.
- size_t txDataSizeAll
  Size of the data to send out.
- edma_handle_t ∗ txEdmaHandle
  The eDMA TX channel used.
- edma_handle_t ∗ rxEdmaHandle
  The eDMA RX channel used.
- uint8_t nbytes
  eDMA minor byte transfer count initially configured.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.

63.2.1.0.0.55 Field Documentation

63.2.1.0.0.55.1 lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback

63.2.1.0.0.55.2 void∗ lpuart_edma_handle_t::userData

63.2.1.0.0.55.3 size_t lpuart_edma_handle_t::rxDataSizeAll

63.2.1.0.0.55.4 size_t lpuart_edma_handle_t::txDataSizeAll

63.2.1.0.0.55.5 edma_handle_t∗ lpuart_edma_handle_t::txEdmaHandle

63.2.1.0.0.55.6 edma_handle_t∗ lpuart_edma_handle_t::rxEdmaHandle

63.2.1.0.0.55.7 uint8_t lpuart_edma_handle_t::nbytes

63.2.1.0.0.55.8 volatile uint8_t lpuart_edma_handle_t::txState

Macro Definition Documentation

63.3.1 #define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

Typedef Documentation

63.4.1 typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)

Function Documentation

63.5.1 void LPUART_TransferCreateHandleEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, lpuart_edma_transfer_callback_t *callback*, void * *userData*, edma_handle_t * *txEdmaHandle*, edma_handle_t * *rxEdmaHandle*)

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.
callback	Callback function.
userData	User data.
txEdmaHandle	User requested DMA handle for TX DMA transfer.
rxEdmaHandle	User requested DMA handle for RX DMA transfer.

63.5.2 status_t LPUART_SendEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, lpuart_transfer_t * *xfer*)

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART eDMA transfer structure. See lpuart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.

kStatus_LPUART_TxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

63.5.3 status_t LPUART_ReceiveEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, lpuart_transfer_t * *xfer*)

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.
xfer	LPUART eDMA transfer structure, see lpuart_transfer_t.

Return values

kStatus_Success	if succeed, others fail.
kStatus_LPUART_Rx-Busy	Previous transfer ongoing.
kStatus_InvalidArgument	Invalid argument.

63.5.4 void LPUART_TransferAbortSendEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*)

This function aborts the sent data using eDMA.

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.

63.5.5 void LPUART_TransferAbortReceiveEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*)

This function aborts the received data using eDMA.

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.

63.5.6 status_t LPUART_TransferGetSendCountEDMA (LPUART_Type ∗ *base*, lpuart_edma_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferInProgress	No send in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

63.5.7 status_t LPUART_TransferGetReceiveCountEDMA (LPUART_Type ∗ *base*, lpuart_edma_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of received bytes.

Parameters

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

Return values

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

Chapter 64
Lpuart_freertos_driver

Overview

Data Structures

- struct lpuart_rtos_config_t
  LPUART RTOS configuration structure. More...

Driver version

- #define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
  LPUART FreeRTOS driver version 2.4.0.

LPUART RTOS Operation

- int LPUART_RTOS_Init (lpuart_rtos_handle_t ∗handle, lpuart_handle_t ∗t_handle, const lpuart_-rtos_config_t ∗cfg)
  Initializes an LPUART instance for operation in RTOS.
- int LPUART_RTOS_Deinit (lpuart_rtos_handle_t ∗handle)
  Deinitializes an LPUART instance for operation.

LPUART transactional Operation

- int LPUART_RTOS_Send (lpuart_rtos_handle_t ∗handle, uint8_t ∗buffer, uint32_t length)
  Sends data in the background.
- int LPUART_RTOS_Receive (lpuart_rtos_handle_t ∗handle, uint8_t ∗buffer, uint32_t length, size_t ∗received)
  Receives data.

Data Structure Documentation

64.2.1 struct lpuart_rtos_config_t

Data Fields

- LPUART_Type ∗ base
  UART base address.
- uint32_t srcclk
  UART source clock in Hz.
- uint32_t baudrate
  Desired communication speed.
- lpuart_parity_mode_t parity
  Parity setting.

- lpuart_stop_bit_count_t stopbits
  Number of stop bits to use.
- uint8_t ∗ buffer
  Buffer for background reception.
- uint32_t buffer_size
  Size of buffer for background reception.
- bool enableRxRTS
  RX RTS enable.
- bool enableTxCTS
  TX CTS enable.
- lpuart_transmit_cts_source_t txCtsSource
  TX CTS source.
- lpuart_transmit_cts_config_t txCtsConfig
  TX CTS configure.

Macro Definition Documentation

64.3.1 #define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

Function Documentation

64.4.1 int LPUART_RTOS_Init (lpuart_rtos_handle_t ∗ *handle*, lpuart_handle_t ∗ *t_handle*, const lpuart_rtos_config_t ∗ *cfg*)

Parameters

handle	The RTOS LPUART handle, the pointer to an allocated space for RTOS context.
t_handle	The pointer to an allocated space to store the transactional layer internal state.
cfg	The pointer to the parameters required to configure the LPUART after initialization.

Returns

0 succeed, others failed

64.4.2 int LPUART_RTOS_Deinit (lpuart_rtos_handle_t ∗ *handle*)

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

Parameters

handle	The RTOS LPUART handle.

64.4.3 int LPUART_RTOS_Send (lpuart_rtos_handle_t * *handle*, uint8_t * *buffer*, uint32_t *length*)

This function sends data. It is an synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

handle	The RTOS LPUART handle.
buffer	The pointer to buffer to send.
length	The number of bytes to send.

64.4.4 int LPUART_RTOS_Receive (lpuart_rtos_handle_t * *handle*, uint8_t * *buffer*, uint32_t *length*, size_t * *received*)

This function receives data from LPUART. It is an synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

Parameters

handle	The RTOS LPUART handle.
buffer	The pointer to buffer where to write received data.
length	The number of bytes to receive.
received	The pointer to a variable of size_t where the number of received data is filled.

Chapter 65
Sai_edma

Overview

Data Structures

- struct sai_edma_handle_t
  SAI DMA transfer handle, users should not touch the content of the handle. More...

Typedefs

- typedef void(* sai_edma_callback_t)(I2S_Type *base, sai_edma_handle_t *handle, status_t status, void *userData)
  SAI eDMA transfer callback function for finish and error.

Driver version

- #define FSL_SAI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))
  Version 2.3.1.

eDMA Transactional

- void SAI_TransferTxCreateHandleEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_edma_callback_t callback, void *userData, edma_handle_t *txDmaHandle)
  Initializes the SAI eDMA handle.
- void SAI_TransferRxCreateHandleEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_edma_callback_t callback, void *userData, edma_handle_t *rxDmaHandle)
  Initializes the SAI Rx eDMA handle.
- void SAI_TransferTxSetFormatEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  Configures the SAI Tx audio format.
- void SAI_TransferRxSetFormatEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  Configures the SAI Rx audio format.
- void SAI_TransferTxSetConfigEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_transceiver_t *saiConfig)
  Configures the SAI Tx.
- void SAI_TransferRxSetConfigEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_transceiver_t *saiConfig)
  Configures the SAI Rx.
- status_t SAI_TransferSendEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer)
  Performs a non-blocking SAI transfer using DMA.
- status_t SAI_TransferReceiveEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer)

Performs a non-blocking SAI receive using eDMA.
- void SAI_TransferTerminateSendEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle)
  Terminate all SAI send.
- void SAI_TransferTerminateReceiveEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle)
  Terminate all SAI receive.
- void SAI_TransferAbortSendEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle)
  Aborts a SAI transfer using eDMA.
- void SAI_TransferAbortReceiveEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle)
  Aborts a SAI receive using eDMA.
- status_t SAI_TransferGetSendCountEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle, size_t ∗count)
  Gets byte count sent by SAI.
- status_t SAI_TransferGetReceiveCountEDMA (I2S_Type ∗base, sai_edma_handle_t ∗handle, size-_t ∗count)
  Gets byte count received by SAI.

Data Structure Documentation

65.2.1 struct sai_edma_handle

Data Fields

- edma_handle_t ∗ dmaHandle
  DMA handler for SAI send.
- uint8_t nbytes
  eDMA minor byte transfer count initially configured.
- uint8_t bytesPerFrame
  Bytes in a frame.
- uint8_t channel
  Which data channel.
- uint8_t count
  The transfer data count in a DMA request.
- uint32_t state
  Internal state for SAI eDMA transfer.
- sai_edma_callback_t callback
  Callback for users while transfer finish or error occurs.
- void ∗ userData
  User callback parameter.
- uint8_t tcd [(SAI_XFER_QUEUE_SIZE+1U)∗sizeof(edma_tcd_t)]
  TCD pool for eDMA transfer.
- sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]
  Transfer queue storing queued transfer.
- size_t transferSize [SAI_XFER_QUEUE_SIZE]
  Data bytes need to transfer.
- volatile uint8_t queueUser
  Index for user to queue transfer.
- volatile uint8_t queueDriver
  Index for driver to get the transfer data and size.

65.2.1.0.0.56 Field Documentation

65.2.1.0.0.56.1 uint8_t sai_edma_handle_t::nbytes

65.2.1.0.0.56.2 uint8_t sai_edma_handle_t::tcd[(SAI_XFER_QUEUE_SIZE+1U)∗sizeof(edma_-tcd_t)]

65.2.1.0.0.56.3 sai_transfer_t sai_edma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]

65.2.1.0.0.56.4 volatile uint8_t sai_edma_handle_t::queueUser

Function Documentation

65.3.1 void SAI_TransferTxCreateHandleEDMA (I2S_Type ∗ *base*, sai_edma_handle_t ∗ *handle*, sai_edma_callback_t *callback*, void ∗ *userData*, edma_handle_t ∗ *txDmaHandle*)

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.
base	SAI peripheral base address.
callback	Pointer to user callback function.
userData	User parameter passed to the callback function.
txDmaHandle	eDMA handle pointer, this handle shall be static allocated by users.

65.3.2 void SAI_TransferRxCreateHandleEDMA (I2S_Type ∗ *base*, sai_edma_handle_t ∗ *handle*, sai_edma_callback_t *callback*, void ∗ *userData*, edma_handle_t ∗ *rxDmaHandle*)

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.
base	SAI peripheral base address.
callback	Pointer to user callback function.
userData	User parameter passed to the callback function.
rxDmaHandle	eDMA handle pointer, this handle shall be static allocated by users.

65.3.3 void SAI_TransferTxSetFormatEDMA (I2S_Type ∗ *base*, sai_edma_handle_t ∗ *handle*, sai_transfer_format_t ∗ *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.
format	Pointer to SAI audio data format structure.
mclkSource-ClockHz	SAI master clock source frequency in Hz.
bclkSource-ClockHz	SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format.

Return values

kStatus_Success	Audio format set successfully.
kStatus_InvalidArgument	The input argument is invalid.

65.3.4 void SAI_TransferRxSetFormatEDMA (I2S_Type ∗ *base*, sai_edma_handle_t ∗ *handle*, sai_transfer_format_t ∗ *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.
format	Pointer to SAI audio data format structure.
mclkSource-ClockHz	SAI master clock source frequency in Hz.
bclkSource-ClockHz	SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format.

Return values

kStatus_Success	Audio format set successfully.
kStatus_InvalidArgument	The input argument is invalid.

65.3.5 void SAI_TransferTxSetConfigEDMA (I2S_Type ∗ *base*, sai_edma_handle_t ∗ *handle*, sai_transceiver_t ∗ *saiConfig*)

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.
saiConfig	sai configurations.

65.3.6 void SAI_TransferRxSetConfigEDMA (I2S_Type ∗ *base*, sai_edma_handle_t ∗ *handle*, sai_transceiver_t ∗ *saiConfig*)

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.
saiConfig	sai configurations.

65.3.7 status_t SAI_TransferSendEDMA (I2S_Type ∗ *base*, sai_edma_handle_t ∗ *handle*, sai_transfer_t ∗ *xfer*)

Note

This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.
xfer	Pointer to the DMA transfer structure.

Return values

kStatus_Success	Start a SAI eDMA send successfully.
kStatus_InvalidArgument	The input argument is invalid.
kStatus_TxBusy	SAI is busy sending data.

65.3.8 status_t SAI_TransferReceiveEDMA (I2S_Type ∗ *base*, sai_edma_handle_t ∗ *handle*, sai_transfer_t ∗ *xfer*)

Note

This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemaining-Bytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

base	SAI base pointer
handle	SAI eDMA handle pointer.
xfer	Pointer to DMA transfer structure.

Return values

kStatus_Success	Start a SAI eDMA receive successfully.
kStatus_InvalidArgument	The input argument is invalid.
kStatus_RxBusy	SAI is busy receiving data.

65.3.9 void SAI_TransferTerminateSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSendEDMA.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.

65.3.10 void SAI_TransferTerminateReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceiveEDMA.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.

65.3.11 void SAI_TransferAbortSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.

65.3.12 void SAI_TransferAbortReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

base	SAI base pointer
handle	SAI eDMA handle pointer.

65.3.13 status_t SAI_TransferGetSendCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.
count	Bytes count sent by SAI.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is no non-blocking transaction in progress.

65.3.14 status_t SAI_TransferGetReceiveCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

base	SAI base pointer
handle	SAI eDMA handle pointer.
count	Bytes count received by SAI.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is no non-blocking transaction in progress.

Chapter 66
Uart_edma_driver

Overview

Data Structures

- struct uart_edma_handle_t
  UART eDMA handle. More...

Typedefs

- typedef void(* uart_edma_transfer_callback_t)(UART_Type *base, uart_edma_handle_t *handle, status_t status, void *userData)
  UART transfer callback function.

Driver version

- #define FSL_UART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
  UART EDMA driver version 2.4.0.

eDMA transactional

- void UART_TransferCreateHandleEDMA (UART_Type *base, uart_edma_handle_t *handle, uart_edma_transfer_callback_t callback, void *userData, edma_handle_t *txEdmaHandle, edma_handle_t *rxEdmaHandle)
  Initializes the UART handle which is used in transactional functions.
- status_t UART_SendEDMA (UART_Type *base, uart_edma_handle_t *handle, uart_transfer_t *xfer)
  Sends data using eDMA.
- status_t UART_ReceiveEDMA (UART_Type *base, uart_edma_handle_t *handle, uart_transfer_t *xfer)
  Receives data using eDMA.
- void UART_TransferAbortSendEDMA (UART_Type *base, uart_edma_handle_t *handle)
  Aborts the sent data using eDMA.
- void UART_TransferAbortReceiveEDMA (UART_Type *base, uart_edma_handle_t *handle)
  Aborts the receive data using eDMA.
- status_t UART_TransferGetSendCountEDMA (UART_Type *base, uart_edma_handle_t *handle, uint32_t *count)
  Gets the number of bytes that have been written to UART TX register.
- status_t UART_TransferGetReceiveCountEDMA (UART_Type *base, uart_edma_handle_t *handle, uint32_t *count)
  Gets the number of received bytes.

Data Structure Documentation

66.2.1 struct _uart_edma_handle

Data Fields

- uart_edma_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.
- edma_handle_t ∗ txEdmaHandle
  The eDMA TX channel used.
- edma_handle_t ∗ rxEdmaHandle
  The eDMA RX channel used.
- uint8_t nbytes
  eDMA minor byte transfer count initially configured.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.

66.2.1.0.0.57 Field Documentation

66.2.1.0.0.57.1 uart_edma_transfer_callback_t uart_edma_handle_t::callback

66.2.1.0.0.57.2 void∗ uart_edma_handle_t::userData

66.2.1.0.0.57.3 size_t uart_edma_handle_t::rxDataSizeAll

66.2.1.0.0.57.4 size_t uart_edma_handle_t::txDataSizeAll

66.2.1.0.0.57.5 edma_handle_t∗ uart_edma_handle_t::txEdmaHandle

66.2.1.0.0.57.6 edma_handle_t∗ uart_edma_handle_t::rxEdmaHandle

66.2.1.0.0.57.7 uint8_t uart_edma_handle_t::nbytes

66.2.1.0.0.57.8 volatile uint8_t uart_edma_handle_t::txState

Macro Definition Documentation

66.3.1 #define FSL_UART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

Typedef Documentation

66.4.1 typedef void(∗ uart_edma_transfer_callback_t)(UART_Type ∗base, uart_edma_handle_t ∗handle, status_t status, void ∗userData)

Function Documentation

66.5.1 void UART_TransferCreateHandleEDMA (UART_Type ∗ *base*, uart_edma_handle_t ∗ *handle*, uart_edma_transfer_callback_t *callback*, void ∗ *userData*, edma_handle_t ∗ *txEdmaHandle*, edma_handle_t ∗ *rxEdmaHandle*)

Parameters

base	UART peripheral base address.
handle	Pointer to the uart_edma_handle_t structure.
callback	UART callback, NULL means no callback.
userData	User callback function data.
rxEdmaHandle	User-requested DMA handle for RX DMA transfer.
txEdmaHandle	User-requested DMA handle for TX DMA transfer.

66.5.2 status_t UART_SendEDMA (UART_Type ∗ *base*, uart_edma_handle_t ∗ *handle*, uart_transfer_t ∗ *xfer*)

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.
xfer	UART eDMA transfer structure. See uart_transfer_t.

Return values

kStatus_Success	if succeeded; otherwise failed.

kStatus_UART_TxBusy	Previous transfer ongoing.
kStatus_InvalidArgument	Invalid argument.

66.5.3 status_t UART_ReceiveEDMA (UART_Type * *base*, uart_edma_handle_t * *handle*, uart_transfer_t * *xfer*)

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

base	UART peripheral base address.
handle	Pointer to the uart_edma_handle_t structure.
xfer	UART eDMA transfer structure. See uart_transfer_t.

Return values

kStatus_Success	if succeeded; otherwise failed.
kStatus_UART_RxBusy	Previous transfer ongoing.
kStatus_InvalidArgument	Invalid argument.

66.5.4 void UART_TransferAbortSendEDMA (UART_Type * *base*, uart_edma_handle_t * *handle*)

This function aborts sent data using eDMA.

Parameters

base	UART peripheral base address.
handle	Pointer to the uart_edma_handle_t structure.

66.5.5 void UART_TransferAbortReceiveEDMA (UART_Type * *base*, uart_edma_handle_t * *handle*)

This function aborts receive data using eDMA.

Parameters

base	UART peripheral base address.
handle	Pointer to the uart_edma_handle_t structure.

66.5.6 status_t UART_TransferGetSendCountEDMA (UART_Type ∗ *base*, uart_edma_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of bytes that have been written to UART TX register by DMA.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferInProgress	No send in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

66.5.7 status_t UART_TransferGetReceiveCountEDMA (UART_Type ∗ *base*, uart_edma_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of received bytes.

Parameters

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

Return values

kStatus_NoTransferInProgress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

Chapter 67
Uart_freertos_driver

Overview

Data Structures

- struct uart_rtos_config_t
  UART configuration structure. More...

Driver version

- #define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
  UART FreeRTOS driver version 2.4.0.

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.

Data Structure Documentation

67.2.1 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.

Macro Definition Documentation

67.3.1 #define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

Function Documentation

67.4.1 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.

67.4.2 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.

67.4.3 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.

67.4.4 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.

Chapter 68 SDMMC_COMMON

Overview

Data Structures

- struct sd_detect_card_t
  sd card detect More...
- struct sd_io_voltage_t
  io voltage control configuration More...
- struct sd_usr_param_t
  sdcard user parameter More...
- struct sdio_card_int_t
  card interrupt application callback More...
- struct sdio_usr_param_t
  sdio user parameter More...
- struct sdio_fbr_t
  sdio card FBR register More...
- struct sdio_common_cis_t
  sdio card common CIS More...
- struct sdio_func_cis_t
  sdio card function CIS More...
- struct sd_status_t
  SD card status. More...
- struct sd_cid_t
  SD card CID register. More...
- struct sd_csd_t
  SD card CSD register. More...
- struct sd_scr_t
  SD card SCR register. More...
- struct mmc_cid_t
  MMC card CID register. More...
- struct mmc_csd_t
  MMC card CSD register. More...
- struct mmc_extended_csd_t
  MMC card Extended CSD register (unit: byte). More...
- struct mmc_extended_csd_config_t
  MMC Extended CSD configuration. More...
- struct mmc_boot_config_t
  MMC card boot configuration definition. More...

Macros

- #define SWAP_WORD_BYTE_SEQUENCE(x) (__REV(x))
  Reverse byte sequence in uint32_t.
- #define SWAP_HALF_WROD_BYTE_SEQUENCE(x) (__REV16(x))
  Reverse byte sequence for each half word in uint32_t.

- #define FSL_SDMMC_MAX_VOLTAGE_RETRIES (1000U)
  Maximum loop count to check the card operation voltage range.
- #define FSL_SDMMC_MAX_CMD_RETRIES (10U)
  Maximum loop count to send the cmd.
- #define FSL_SDMMC_DEFAULT_BLOCK_SIZE (512U)
  Default block size.
- #define SDMMC_DATA_BUFFER_ALIGN_CACHE FSL_FEATURE_L1DCACHE_LINESIZE_BYTE
  make sure the internal buffer address is cache align
- #define FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE (FSL_SDMMC_DEFAULT_BLOCK_SIZE + SDMMC_DATA_BUFFER_ALIGN_CACHE)
  sdmmc card internal buffer size
- #define FSL_SDMMC_CARD_MAX_BUS_FREQ(max, target) ((max) == 0U ? (target) : ((max) > (target) ? (target) : (max)))
  get maximum freq
- #define SDMMC_LOG(format,...)
  SD/MMC error log.
- #define SDMMC_CLOCK_400KHZ (400000U)
  SD/MMC card initialization clock frequency.
- #define SD_CLOCK_25MHZ (25000000U)
  SD card bus frequency 1 in high-speed mode.
- #define SD_CLOCK_50MHZ (50000000U)
  SD card bus frequency 2 in high-speed mode.
- #define SD_CLOCK_100MHZ (100000000U)
  SD card bus frequency in SDR50 mode.
- #define SD_CLOCK_208MHZ (208000000U)
  SD card bus frequency in SDR104 mode.
- #define MMC_CLOCK_26MHZ (26000000U)
  MMC card bus frequency 1 in high-speed mode.
- #define MMC_CLOCK_52MHZ (52000000U)
  MMC card bus frequency 2 in high-speed mode.
- #define MMC_CLOCK_DDR52 (52000000U)
  MMC card bus frequency in high-speed DDR52 mode.
- #define MMC_CLOCK_HS200 (200000000U)
  MMC card bus frequency in high-speed HS200 mode.
- #define MMC_CLOCK_HS400 (400000000U)
  MMC card bus frequency in high-speed HS400 mode.
- #define SDMMC_MASK(bit) (1U << (bit))
  mask convert
- #define SDMMC_R1_ALL_ERROR_FLAG
  R1 all the error flag.
- #define SDMMC_R1_CURRENT_STATE(x) (((x)&0x00001E00U) >> 9U)
  R1: current state.
- #define SDSPI_R7_VERSION_SHIFT (28U)
  The bit mask for COMMAND VERSION field in R7.
- #define SDSPI_R7_VERSION_MASK (0xFU)
  The bit mask for COMMAND VERSION field in R7.
- #define SDSPI_R7_VOLTAGE_SHIFT (8U)
  The bit shift for VOLTAGE ACCEPTED field in R7.
- #define SDSPI_R7_VOLTAGE_MASK (0xFU)
  The bit mask for VOLTAGE ACCEPTED field in R7.

- #define SDSPI_R7_VOLTAGE_27_36_MASK (0x1U << SDSPI_R7_VOLTAGE_SHIFT)
  The bit mask for VOLTAGE 2.7V to 3.6V field in R7.
- #define SDSPI_R7_ECHO_SHIFT (0U)
  The bit shift for ECHO field in R7.
- #define SDSPI_R7_ECHO_MASK (0xFFU)
  The bit mask for ECHO field in R7.
- #define SDSPI_DATA_ERROR_TOKEN_MASK (0xFU)
  Data error token mask.
- #define SDSPI_DATA_RESPONSE_TOKEN_MASK (0x1FU)
  Mask for data response bits.
- #define SDIO_CCCR_REG_NUMBER (0x16U)
  sdio card cccr register number
- #define SDIO_IO_READY_TIMEOUT_UNIT (10U)
  sdio IO ready timeout steps
- #define SDIO_CMD_ARGUMENT_RW_POS (31U)
  read/write flag position
- #define SDIO_CMD_ARGUMENT_FUNC_NUM_POS (28U)
  function number position
- #define SDIO_DIRECT_CMD_ARGUMENT_RAW_POS (27U)
  direct raw flag position
- #define SDIO_CMD_ARGUMENT_REG_ADDR_POS (9U)
  direct reg addr position
- #define SDIO_CMD_ARGUMENT_REG_ADDR_MASK (0x1FFFFU)
  direct reg addr mask
- #define SDIO_DIRECT_CMD_DATA_MASK (0xFFU)
  data mask
- #define SDIO_EXTEND_CMD_ARGUMENT_BLOCK_MODE_POS (27U)
  extended command argument block mode bit position
- #define SDIO_EXTEND_CMD_ARGUMENT_OP_CODE_POS (26U)
  extended command argument OP Code bit position
- #define SDIO_EXTEND_CMD_BLOCK_MODE_MASK (0x08000000U)
  block mode mask
- #define SDIO_EXTEND_CMD_OP_CODE_MASK (0x04000000U)
  op code mask
- #define SDIO_EXTEND_CMD_COUNT_MASK (0x1FFU)
  byte/block count mask
- #define SDIO_MAX_BLOCK_SIZE (2048U)
  max block size
- #define SDIO_FBR_BASE(x) ((x)*0x100U)
  function basic register
- #define SDIO_TPL_CODE_END (0xFFU)
  tuple end
- #define SDIO_TPL_CODE_MANIFID (0x20U)
  manufacturer ID
- #define SDIO_TPL_CODE_FUNCID (0x21U)
  function ID
- #define SDIO_TPL_CODE_FUNCE (0x22U)
  function extension tuple
- #define SDIO_OCR_VOLTAGE_WINDOW_MASK (0xFFFFU << 8U)
  sdio ocr voltage window mask
- #define SDIO_OCR_IO_NUM_MASK (7U << kSDIO_OcrIONumber)

sdio ocr reigster IO NUMBER mask
- #define SDIO_CCCR_SUPPORT_HIGHSPEED (1u << 9U)
  UHS timing mode flag.
- #define SDIO_CCCR_DRIVER_TYPE_MASK (3U << 4U)
  Driver type flag.
- #define SDIO_CCCR_ASYNC_INT_MASK (1U)
  aync interrupt flag
- #define SDIO_CCCR_SUPPORT_8BIT_BUS (1UL << 18U)
  8 bit data bus flag
- #define MMC_OCR_V170TO195_SHIFT (7U)
  The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.
- #define MMC_OCR_V170TO195_MASK (0x00000080U)
  The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.
- #define MMC_OCR_V200TO260_SHIFT (8U)
  The bit shift for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.
- #define MMC_OCR_V200TO260_MASK (0x00007F00U)
  The bit mask for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.
- #define MMC_OCR_V270TO360_SHIFT (15U)
  The bit shift for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.
- #define MMC_OCR_V270TO360_MASK (0x00FF8000U)
  The bit mask for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.
- #define MMC_OCR_ACCESS_MODE_SHIFT (29U)
  The bit shift for ACCESS MODE field in OCR.
- #define MMC_OCR_ACCESS_MODE_MASK (0x60000000U)
  The bit mask for ACCESS MODE field in OCR.
- #define MMC_OCR_BUSY_SHIFT (31U)
  The bit shift for BUSY field in OCR.
- #define MMC_OCR_BUSY_MASK (1U << MMC_OCR_BUSY_SHIFT)
  The bit mask for BUSY field in OCR.
- #define MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT (0U)
  The bit shift for FREQUENCY UNIT field in TRANSFER SPEED(TRAN-SPEED in Extended CSD)
- #define MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK (0x07U)
  The bit mask for FRQEUENCY UNIT in TRANSFER SPEED.
- #define MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT (3U)
  The bit shift for MULTIPLIER field in TRANSFER SPEED.
- #define MMC_TRANSFER_SPEED_MULTIPLIER_MASK (0x78U)
  The bit mask for MULTIPLIER field in TRANSFER SPEED.
- #define READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT(CSD) ((((CSD).transferSpeed) & MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK) >> MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT)
  Read the value of FREQUENCY UNIT in TRANSFER SPEED.
- #define READ_MMC_TRANSFER_SPEED_MULTIPLIER(CSD) ((((CSD).transferSpeed) & MMC_TRANSFER_SPEED_MULTIPLIER_MASK) >> MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT)
  Read the value of MULTIPLER filed in TRANSFER SPEED.
- #define MMC_POWER_CLASS_4BIT_MASK (0x0FU)
  The power class value bit mask when bus in 4 bit mode.
- #define MMC_POWER_CLASS_8BIT_MASK (0xF0U)
  The power class current value bit mask when bus in 8 bit mode.
- #define MMC_CACHE_CONTROL_ENABLE (1U)
  mmc cache control enable

- #define MMC_CACHE_TRIGGER_FLUSH (1U)
  mmc cache flush
- #define MMC_DATA_BUS_WIDTH_TYPE_NUMBER (3U)
  The number of data bus width type.
- #define MMC_PARTITION_CONFIG_PARTITION_ACCESS_SHIFT (0U)
  The bit shift for PARTITION ACCESS filed in BOOT CONFIG (BOOT_CONFIG in Extend CSD)
- #define MMC_PARTITION_CONFIG_PARTITION_ACCESS_MASK (0x00000007U)
  The bit mask for PARTITION ACCESS field in BOOT CONFIG.
- #define MMC_PARTITION_CONFIG_PARTITION_ENABLE_SHIFT (3U)
  The bit shift for PARTITION ENABLE field in BOOT CONFIG.
- #define MMC_PARTITION_CONFIG_PARTITION_ENABLE_MASK (0x00000038U)
  The bit mask for PARTITION ENABLE field in BOOT CONFIG.
- #define MMC_PARTITION_CONFIG_BOOT_ACK_SHIFT (6U)
  The bit shift for ACK field in BOOT CONFIG.
- #define MMC_PARTITION_CONFIG_BOOT_ACK_MASK (0x00000040U)
  The bit mask for ACK field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_BUS_WIDTH_SHIFT (0U)
  The bit shift for BOOT BUS WIDTH field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_BUS_WIDTH_MASK (3U)
  The bit mask for BOOT BUS WIDTH field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_RESET_BUS_CONDITION_SHIFT (2U)
  The bit shift for BOOT BUS WIDTH RESET field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_RESET_BUS_CONDITION_MASK (4U)
  The bit mask for BOOT BUS WIDTH RESET field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_BOOT_MODE_SHIFT (3U)
  The bit shift for BOOT MODE field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_BOOT_MODE_MASK (0x18U)
  The bit mask for BOOT MODE field in BOOT CONFIG.
- #define MMC_EXTENDED_CSD_BYTES (512U)
  The length of Extended CSD register, unit as bytes.
- #define MMC_DEFAULT_RELATIVE_ADDRESS (2U)
  MMC card default relative address.
- #define SD_PRODUCT_NAME_BYTES (5U)
  SD card product name length united as bytes.
- #define SD_AU_START_VALUE (1U)
  SD AU start value.
- #define SD_UHS_AU_START_VALUE (7U)
  SD UHS AU start value.
- #define SD_TRANSFER_SPEED_RATE_UNIT_SHIFT (0U)
  The bit shift for RATE UNIT field in TRANSFER SPEED.
- #define SD_TRANSFER_SPEED_RATE_UNIT_MASK (0x07U)
  The bit mask for RATE UNIT field in TRANSFER SPEED.
- #define SD_TRANSFER_SPEED_TIME_VALUE_SHIFT (2U)
  The bit shift for TIME VALUE field in TRANSFER SPEED.
- #define SD_TRANSFER_SPEED_TIME_VALUE_MASK (0x78U)
  The bit mask for TIME VALUE field in TRANSFER SPEED.
- #define SD_RD_TRANSFER_SPEED_RATE_UNIT(x) (((x.transferSpeed) & SD_TRANSFER_SPEED_RATE_UNIT_MASK) >> SD_TRANSFER_SPEED_RATE_UNIT_SHIFT)
  Read the value of FREQUENCY UNIT in TRANSFER SPEED field.
- #define SD_RD_TRANSFER_SPEED_TIME_VALUE(x) (((x.transferSpeed) & SD_TRANSFER_SPEED_TIME_VALUE_MASK) >> SD_TRANSFER_SPEED_TIME_VALUE_SHIFT)

Read the value of TIME VALUE in TRANSFER SPEED field.
- #define MMC_PRODUCT_NAME_BYTES (6U)
  MMC card product name length united as bytes.
- #define MMC_SWITCH_COMMAND_SET_SHIFT (0U)
  The bit shift for COMMAND SET field in SWITCH command.
- #define MMC_SWITCH_COMMAND_SET_MASK (0x00000007U)
  The bit mask for COMMAND set field in SWITCH command.
- #define MMC_SWITCH_VALUE_SHIFT (8U)
  The bit shift for VALUE field in SWITCH command.
- #define MMC_SWITCH_VALUE_MASK (0x0000FF00U)
  The bit mask for VALUE field in SWITCH command.
- #define MMC_SWITCH_BYTE_INDEX_SHIFT (16U)
  The bit shift for BYTE INDEX field in SWITCH command.
- #define MMC_SWITCH_BYTE_INDEX_MASK (0x00FF0000U)
  The bit mask for BYTE INDEX field in SWITCH command.
- #define MMC_SWITCH_ACCESS_MODE_SHIFT (24U)
  The bit shift for ACCESS MODE field in SWITCH command.
- #define MMC_SWTICH_ACCESS_MODE_MASK (0x03000000U)
  The bit mask for ACCESS MODE field in SWITCH command.

Typedefs

- typedef void(∗ sd_cd_t)(bool isInserted, void ∗userData)
  card detect aoolication callback definition
- typedef bool(∗ sd_cd_status_t)(void)
  card detect status
- typedef void(∗ sd_io_voltage_func_t)(sdmmc_operation_voltage_t voltage)
  card switch voltage function pointer
- typedef void(∗ sd_pwr_t)(bool enable)
  card power control function pointer
- typedef void(∗ sd_io_strength_t)(uint32_t busFreq)
  card io strength control
- typedef void(∗ sdio_int_t)(void ∗userData)
  card interrupt function pointer

Enumerations

- enum {
kStatus_SDMMC_NotSupportYet = MAKE_STATUS(kStatusGroup_SDMMC, 0U),
kStatus_SDMMC_TransferFailed = MAKE_STATUS(kStatusGroup_SDMMC, 1U),
kStatus_SDMMC_SetCardBlockSizeFailed = MAKE_STATUS(kStatusGroup_SDMMC, 2U),
kStatus_SDMMC_HostNotSupport = MAKE_STATUS(kStatusGroup_SDMMC, 3U),
kStatus_SDMMC_CardNotSupport = MAKE_STATUS(kStatusGroup_SDMMC, 4U),
kStatus_SDMMC_AllSendCidFailed = MAKE_STATUS(kStatusGroup_SDMMC, 5U),
kStatus_SDMMC_SendRelativeAddressFailed = MAKE_STATUS(kStatusGroup_SDMMC, 6U),
kStatus_SDMMC_SendCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 7U),
kStatus_SDMMC_SelectCardFailed = MAKE_STATUS(kStatusGroup_SDMMC, 8U),
kStatus_SDMMC_SendScrFailed = MAKE_STATUS(kStatusGroup_SDMMC, 9U),
kStatus_SDMMC_SetDataBusWidthFailed = MAKE_STATUS(kStatusGroup_SDMMC, 10U),
kStatus_SDMMC_GoIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 11U),
kStatus_SDMMC_HandShakeOperationConditionFailed,
kStatus_SDMMC_SendApplicationCommandFailed,
kStatus_SDMMC_SwitchFailed = MAKE_STATUS(kStatusGroup_SDMMC, 14U),
kStatus_SDMMC_StopTransmissionFailed = MAKE_STATUS(kStatusGroup_SDMMC, 15U),
kStatus_SDMMC_WaitWriteCompleteFailed = MAKE_STATUS(kStatusGroup_SDMMC, 16U),
kStatus_SDMMC_SetBlockCountFailed = MAKE_STATUS(kStatusGroup_SDMMC, 17U),
kStatus_SDMMC_SetRelativeAddressFailed = MAKE_STATUS(kStatusGroup_SDMMC, 18U),
kStatus_SDMMC_SwitchBusTimingFailed = MAKE_STATUS(kStatusGroup_SDMMC, 19U),
kStatus_SDMMC_SendExtendedCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 20U),
kStatus_SDMMC_ConfigureBootFailed = MAKE_STATUS(kStatusGroup_SDMMC, 21U),
kStatus_SDMMC_ConfigureExtendedCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 22-U),
kStatus_SDMMC_EnableHighCapacityEraseFailed,
kStatus_SDMMC_SendTestPatternFailed = MAKE_STATUS(kStatusGroup_SDMMC, 24U),
kStatus_SDMMC_ReceiveTestPatternFailed = MAKE_STATUS(kStatusGroup_SDMMC, 25U),
kStatus_SDMMC_SDIO_ResponseError = MAKE_STATUS(kStatusGroup_SDMMC, 26U),
kStatus_SDMMC_SDIO_InvalidArgument,
kStatus_SDMMC_SDIO_SendOperationConditionFail,
kStatus_SDMMC_InvalidVoltage = MAKE_STATUS(kStatusGroup_SDMMC, 29U),
kStatus_SDMMC_SDIO_SwitchHighSpeedFail = MAKE_STATUS(kStatusGroup_SDMMC, 30-U),
kStatus_SDMMC_SDIO_ReadCISFail = MAKE_STATUS(kStatusGroup_SDMMC, 31U),
kStatus_SDMMC_SDIO_InvalidCard = MAKE_STATUS(kStatusGroup_SDMMC, 32U),
kStatus_SDMMC_TuningFail = MAKE_STATUS(kStatusGroup_SDMMC, 33U),
kStatus_SDMMC_SwitchVoltageFail = MAKE_STATUS(kStatusGroup_SDMMC, 34U),
kStatus_SDMMC_SwitchVoltage18VFail33VSuccess = MAKE_STATUS(kStatusGroup_SDMM-

C, 35U),
kStatus_SDMMC_ReTuningRequest = MAKE_STATUS(kStatusGroup_SDMMC, 36U),
kStatus_SDMMC_SetDriverStrengthFail = MAKE_STATUS(kStatusGroup_SDMMC, 37U),
kStatus_SDMMC_SetPowerClassFail = MAKE_STATUS(kStatusGroup_SDMMC, 38U),
kStatus_SDMMC_HostNotReady = MAKE_STATUS(kStatusGroup_SDMMC, 39U),
kStatus_SDMMC_CardDetectFailed = MAKE_STATUS(kStatusGroup_SDMMC, 40U),
kStatus_SDMMC_AuSizeNotSetProperly = MAKE_STATUS(kStatusGroup_SDMMC, 41U),
kStatus_SDMMC_PollingCardIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 42U) }
SD/MMC card API's running status.
- enum {
kSDMMC_SignalLineCmd = 1U,
kSDMMC_SignalLineData0 = 2U,
kSDMMC_SignalLineData1 = 4U,
kSDMMC_SignalLineData2 = 8U,
kSDMMC_SignalLineData3 = 16U,
kSDMMC_SignalLineData4 = 32U,
kSDMMC_SignalLineData5 = 64U,
kSDMMC_SignalLineData6 = 128U,
kSDMMC_SignalLineData7 = 256U }
sdmmc signal line
- enum sdmmc_operation_voltage_t {
kSDMMC_OperationVoltageNone = 0U,
kSDMMC_OperationVoltage330V = 1U,
kSDMMC_OperationVoltage300V = 2U,
kSDMMC_OperationVoltage180V = 3U }
card operation voltage
- enum {
kSDMMC_BusWdith1Bit = 0U,
kSDMMC_BusWdith4Bit = 1U,
kSDMMC_BusWdith8Bit = 2U }
card bus width
- enum { kSDMMC_Support8BitWidth = 1U }
sdmmc capability flag
- enum sd_detect_card_type_t {
kSD_DetectCardByGpioCD,
kSD_DetectCardByHostCD,
kSD_DetectCardByHostDATA3 }
sd card detect type
- enum {
kSD_Inserted = 1U,
kSD_Removed = 0U }
@ brief SD card detect status
- enum {
kSD_DAT3PullDown = 0U,
kSD_DAT3PullUp = 1U }
@ brief SD card detect status
- enum sd_io_voltage_ctrl_type_t {

kSD_IOVoltageCtrlNotSupport = 0U,
kSD_IOVoltageCtrlByHost = 1U,
kSD_IOVoltageCtrlByGpio = 2U }

io voltage control type

- enum {
kSDMMC_R1OutOfRangeFlag = 31,
kSDMMC_R1AddressErrorFlag = 30,
kSDMMC_R1BlockLengthErrorFlag = 29,
kSDMMC_R1EraseSequenceErrorFlag = 28,
kSDMMC_R1EraseParameterErrorFlag = 27,
kSDMMC_R1WriteProtectViolationFlag = 26,
kSDMMC_R1CardIsLockedFlag = 25,
kSDMMC_R1LockUnlockFailedFlag = 24,
kSDMMC_R1CommandCrcErrorFlag = 23,
kSDMMC_R1IllegalCommandFlag = 22,
kSDMMC_R1CardEccFailedFlag = 21,
kSDMMC_R1CardControllerErrorFlag = 20,
kSDMMC_R1ErrorFlag = 19,
kSDMMC_R1CidCsdOverwriteFlag = 16,
kSDMMC_R1WriteProtectEraseSkipFlag = 15,
kSDMMC_R1CardEccDisabledFlag = 14,
kSDMMC_R1EraseResetFlag = 13,
kSDMMC_R1ReadyForDataFlag = 8,
kSDMMC_R1SwitchErrorFlag = 7,
kSDMMC_R1ApplicationCommandFlag = 5,
kSDMMC_R1AuthenticationSequenceErrorFlag = 3 }

Card status bit in R1.

- enum sdmmc_r1_current_state_t {
kSDMMC_R1StateIdle = 0U,
kSDMMC_R1StateReady = 1U,
kSDMMC_R1StateIdentify = 2U,
kSDMMC_R1StateStandby = 3U,
kSDMMC_R1StateTransfer = 4U,
kSDMMC_R1StateSendData = 5U,
kSDMMC_R1StateReceiveData = 6U,
kSDMMC_R1StateProgram = 7U,
kSDMMC_R1StateDisconnect = 8U }

CURRENT_STATE filed in R1.

- enum {
kSDSPI_R1InIdleStateFlag = (1U << 0U),
kSDSPI_R1EraseResetFlag = (1U << 1U),
kSDSPI_R1IllegalCommandFlag = (1U << 2U),
kSDSPI_R1CommandCrcErrorFlag = (1U << 3U),
kSDSPI_R1EraseSequenceErrorFlag = (1U << 4U),
kSDSPI_R1AddressErrorFlag = (1U << 5U),

kSDSPI_R1ParameterErrorFlag = (1U << 6U) }
  Error bit in SPI mode R1.
- enum {
kSDSPI_R2CardLockedFlag = (1U << 0U),
kSDSPI_R2WriteProtectEraseSkip = (1U << 1U),
kSDSPI_R2LockUnlockFailed = (1U << 1U),
kSDSPI_R2ErrorFlag = (1U << 2U),
kSDSPI_R2CardControllerErrorFlag = (1U << 3U),
kSDSPI_R2CardEccFailedFlag = (1U << 4U),
kSDSPI_R2WriteProtectViolationFlag = (1U << 5U),
kSDSPI_R2EraseParameterErrorFlag = (1U << 6U),
kSDSPI_R2OutOfRangeFlag = (1U << 7U),
kSDSPI_R2CsdOverwriteFlag = (1U << 7U) }
  Error bit in SPI mode R2.
- enum {
kSDSPI_DataErrorTokenError = (1U << 0U),
kSDSPI_DataErrorTokenCardControllerError = (1U << 1U),
kSDSPI_DataErrorTokenCardEccFailed = (1U << 2U),
kSDSPI_DataErrorTokenOutOfRange = (1U << 3U) }
  Data Error Token mask bit.
- enum sdspi_data_token_t {
kSDSPI_DataTokenBlockRead = 0xFEU,
kSDSPI_DataTokenSingleBlockWrite = 0xFEU,
kSDSPI_DataTokenMultipleBlockWrite = 0xFCU,
kSDSPI_DataTokenStopTransfer = 0xFDU }
  Data Token.
- enum sdspi_data_response_token_t {
kSDSPI_DataResponseTokenAccepted = 0x05U,
kSDSPI_DataResponseTokenCrcError = 0x0BU,
kSDSPI_DataResponseTokenWriteError = 0x0DU }
  Data Response Token.
- enum sd_command_t {
kSD_SendRelativeAddress = 3U,
kSD_Switch = 6U,
kSD_SendInterfaceCondition = 8U,
kSD_VoltageSwitch = 11U,
kSD_SpeedClassControl = 20U,
kSD_EraseWriteBlockStart = 32U,
kSD_EraseWriteBlockEnd = 33U,
kSD_SendTuningBlock = 19U }
  SD card individual commands.
- enum sdspi_command_t { kSDSPI_CommandCrc = 59U }
  SDSPI individual commands.
- enum sd_application_command_t {

kSD_ApplicationSetBusWdith = 6U,
kSD_ApplicationStatus = 13U,
kSD_ApplicationSendNumberWriteBlocks = 22U,
kSD_ApplicationSetWriteBlockEraseCount = 23U,
kSD_ApplicationSendOperationCondition = 41U,
kSD_ApplicationSetClearCardDetect = 42U,
kSD_ApplicationSendScr = 51U }
SD card individual application commands.
- enum {
kSDMMC_CommandClassBasic = (1U << 0U),
kSDMMC_CommandClassBlockRead = (1U << 2U),
kSDMMC_CommandClassBlockWrite = (1U << 4U),
kSDMMC_CommandClassErase = (1U << 5U),
kSDMMC_CommandClassWriteProtect = (1U << 6U),
kSDMMC_CommandClassLockCard = (1U << 7U),
kSDMMC_CommandClassApplicationSpecific = (1U << 8U),
kSDMMC_CommandClassInputOutputMode = (1U << 9U),
kSDMMC_CommandClassSwitch = (1U << 10U) }
SD card command class.
- enum {
kSD_OcrPowerUpBusyFlag = 31,
kSD_OcrHostCapacitySupportFlag = 30,
kSD_OcrCardCapacitySupportFlag = kSD_OcrHostCapacitySupportFlag,
kSD_OcrSwitch18RequestFlag = 24,
kSD_OcrSwitch18AcceptFlag = kSD_OcrSwitch18RequestFlag,
kSD_OcrVdd27_28Flag = 15,
kSD_OcrVdd28_29Flag = 16,
kSD_OcrVdd29_30Flag = 17,
kSD_OcrVdd30_31Flag = 18,
kSD_OcrVdd31_32Flag = 19,
kSD_OcrVdd32_33Flag = 20,
kSD_OcrVdd33_34Flag = 21,
kSD_OcrVdd34_35Flag = 22,
kSD_OcrVdd35_36Flag = 23 }
OCR register in SD card.
- enum {
kSD_SpecificationVersion1_0 = (1U << 0U),
kSD_SpecificationVersion1_1 = (1U << 1U),
kSD_SpecificationVersion2_0 = (1U << 2U),
kSD_SpecificationVersion3_0 = (1U << 3U) }
SD card specification version number.
- enum sd_switch_mode_t {
kSD_SwitchCheck = 0U,
kSD_SwitchSet = 1U }
SD card switch mode.
- enum {

kSD_CsdReadBlockPartialFlag = (1U << 0U),
kSD_CsdWriteBlockMisalignFlag = (1U << 1U),
kSD_CsdReadBlockMisalignFlag = (1U << 2U),
kSD_CsdDsrImplementedFlag = (1U << 3U),
kSD_CsdEraseBlockEnabledFlag = (1U << 4U),
kSD_CsdWriteProtectGroupEnabledFlag = (1U << 5U),
kSD_CsdWriteBlockPartialFlag = (1U << 6U),
kSD_CsdFileFormatGroupFlag = (1U << 7U),
kSD_CsdCopyFlag = (1U << 8U),
kSD_CsdPermanentWriteProtectFlag = (1U << 9U),
kSD_CsdTemporaryWriteProtectFlag = (1U << 10U) }
SD card CSD register flags.

- enum {
kSD_ScrDataStatusAfterErase = (1U << 0U),
kSD_ScrSdSpecification3 = (1U << 1U) }
SD card SCR register flags.
- enum {
kSD_FunctionSDR12Deafult = 0U,
kSD_FunctionSDR25HighSpeed = 1U,
kSD_FunctionSDR50 = 2U,
kSD_FunctionSDR104 = 3U,
kSD_FunctionDDR50 = 4U }
SD timing function number.
- enum {
kSD_GroupTimingMode = 0U,
kSD_GroupCommandSystem = 1U,
kSD_GroupDriverStrength = 2U,
kSD_GroupCurrentLimit = 3U }
SD group number.
- enum sd_timing_mode_t {
kSD_TimingSDR12DefaultMode = 0U,
kSD_TimingSDR25HighSpeedMode = 1U,
kSD_TimingSDR50Mode = 2U,
kSD_TimingSDR104Mode = 3U,
kSD_TimingDDR50Mode = 4U }
SD card timing mode flags.
- enum sd_driver_strength_t {
kSD_DriverStrengthTypeB = 0U,
kSD_DriverStrengthTypeA = 1U,
kSD_DriverStrengthTypeC = 2U,
kSD_DriverStrengthTypeD = 3U }
SD card driver strength.
- enum sd_max_current_t {
kSD_CurrentLimit200MA = 0U,
kSD_CurrentLimit400MA = 1U,
kSD_CurrentLimit600MA = 2U,

kSD_CurrentLimit800MA = 3U }
SD card current limit.
- enum sdmmc_command_t {
kSDMMC_GoIdleState = 0U,
kSDMMC_AllSendCid = 2U,
kSDMMC_SetDsr = 4U,
kSDMMC_SelectCard = 7U,
kSDMMC_SendCsd = 9U,
kSDMMC_SendCid = 10U,
kSDMMC_StopTransmission = 12U,
kSDMMC_SendStatus = 13U,
kSDMMC_GoInactiveState = 15U,
kSDMMC_SetBlockLength = 16U,
kSDMMC_ReadSingleBlock = 17U,
kSDMMC_ReadMultipleBlock = 18U,
kSDMMC_SetBlockCount = 23U,
kSDMMC_WriteSingleBlock = 24U,
kSDMMC_WriteMultipleBlock = 25U,
kSDMMC_ProgramCsd = 27U,
kSDMMC_SetWriteProtect = 28U,
kSDMMC_ClearWriteProtect = 29U,
kSDMMC_SendWriteProtect = 30U,
kSDMMC_Erase = 38U,
kSDMMC_LockUnlock = 42U,
kSDMMC_ApplicationCommand = 55U,
kSDMMC_GeneralCommand = 56U,
kSDMMC_ReadOcr = 58U }
SD/MMC card common commands.
- enum {

kSDIO_RegCCCRSdioVer = 0x00U,
kSDIO_RegSDVersion = 0x01U,
kSDIO_RegIOEnable = 0x02U,
kSDIO_RegIOReady = 0x03U,
kSDIO_RegIOIntEnable = 0x04U,
kSDIO_RegIOIntPending = 0x05U,
kSDIO_RegIOAbort = 0x06U,
kSDIO_RegBusInterface = 0x07U,
kSDIO_RegCardCapability = 0x08U,
kSDIO_RegCommonCISPointer = 0x09U,
kSDIO_RegBusSuspend = 0x0C,
kSDIO_RegFunctionSelect = 0x0DU,
kSDIO_RegExecutionFlag = 0x0EU,
kSDIO_RegReadyFlag = 0x0FU,
kSDIO_RegFN0BlockSizeLow = 0x10U,
kSDIO_RegFN0BlockSizeHigh = 0x11U,
kSDIO_RegPowerControl = 0x12U,
kSDIO_RegBusSpeed = 0x13U,
kSDIO_RegUHSITimingSupport = 0x14U,
kSDIO_RegDriverStrength = 0x15U,
kSDIO_RegInterruptExtension = 0x16U }
  sdio card cccr register addr
- enum sdio_command_t {
  kSDIO_SendRelativeAddress = 3U,
  kSDIO_SendOperationCondition = 5U,
  kSDIO_SendInterfaceCondition = 8U,
  kSDIO_RWIODirect = 52U,
  kSDIO_RWIOExtended = 53U }
  sdio card individual commands
- enum sdio_func_num_t {
  kSDIO_FunctionNum0,
  kSDIO_FunctionNum1,
  kSDIO_FunctionNum2,
  kSDIO_FunctionNum3,
  kSDIO_FunctionNum4,
  kSDIO_FunctionNum5,
  kSDIO_FunctionNum6,
  kSDIO_FunctionNum7,
  kSDIO_FunctionMemory }
  sdio card individual commands
- enum {

kSDIO_StatusCmdCRCError = 0x8000U,
kSDIO_StatusIllegalCmd = 0x4000U,
kSDIO_StatusR6Error = 0x2000U,
kSDIO_StatusError = 0x0800U,
kSDIO_StatusFunctionNumError = 0x0200U,
kSDIO_StatusOutofRange = 0x0100U }
sdio command response flag
- enum {
kSDIO_OcrPowerUpBusyFlag = 31,
kSDIO_OcrIONumber = 28,
kSDIO_OcrMemPresent = 27,
kSDIO_OcrVdd20_21Flag = 8,
kSDIO_OcrVdd21_22Flag = 9,
kSDIO_OcrVdd22_23Flag = 10,
kSDIO_OcrVdd23_24Flag = 11,
kSDIO_OcrVdd24_25Flag = 12,
kSDIO_OcrVdd25_26Flag = 13,
kSDIO_OcrVdd26_27Flag = 14,
kSDIO_OcrVdd27_28Flag = 15,
kSDIO_OcrVdd28_29Flag = 16,
kSDIO_OcrVdd29_30Flag = 17,
kSDIO_OcrVdd30_31Flag = 18,
kSDIO_OcrVdd31_32Flag = 19,
kSDIO_OcrVdd32_33Flag = 20,
kSDIO_OcrVdd33_34Flag = 21,
kSDIO_OcrVdd34_35Flag = 22,
kSDIO_OcrVdd35_36Flag = 23 }
sdio operation condition flag
- enum {
kSDIO_CCCRSupportDirectCmdDuringDataTrans = (1U << 0U),
kSDIO_CCCRSupportMultiBlock = (1U << 1U),
kSDIO_CCCRSupportReadWait = (1U << 2U),
kSDIO_CCCRSupportSuspendResume = (1U << 3U),
kSDIO_CCCRSupportIntDuring4BitDataTrans = (1U << 4U),
kSDIO_CCCRSupportLowSpeed1Bit = (1U << 6U),
kSDIO_CCCRSupportLowSpeed4Bit = (1U << 7U),
kSDIO_CCCRSupportMasterPowerControl = (1U << 8U),
kSDIO_CCCRSupportHighSpeed = (1U << 9U),
kSDIO_CCCRSupportContinuousSPIInt = (1U << 10U) }
sdio capability flag
- enum {
kSDIO_FBRSupportCSA = (1U << 0U),
kSDIO_FBRSupportPowerSelection = (1U << 1U) }
sdio fbr flag
- enum sdio_bus_width_t {

kSDIO_DataBus1Bit = 0x00U,
kSDIO_DataBus4Bit = 0X02U,
kSDIO_DataBus8Bit = 0X03U }
sdio bus width
- enum mmc_command_t {
kMMC_SendOperationCondition = 1U,
kMMC_SetRelativeAddress = 3U,
kMMC_SleepAwake = 5U,
kMMC_Switch = 6U,
kMMC_SendExtendedCsd = 8U,
kMMC_ReadDataUntilStop = 11U,
kMMC_BusTestRead = 14U,
kMMC_SendingBusTest = 19U,
kMMC_WriteDataUntilStop = 20U,
kMMC_SendTuningBlock = 21U,
kMMC_ProgramCid = 26U,
kMMC_EraseGroupStart = 35U,
kMMC_EraseGroupEnd = 36U,
kMMC_FastInputOutput = 39U,
kMMC_GoInterruptState = 40U }
MMC card individual commands.
- enum mmc_classified_voltage_t {
kMMC_ClassifiedVoltageHigh = 0U,
kMMC_ClassifiedVoltageDual = 1U }
MMC card classified as voltage range.
- enum mmc_classified_density_t { kMMC_ClassifiedDensityWithin2GB = 0U }
MMC card classified as density level.
- enum mmc_access_mode_t {
kMMC_AccessModeByte = 0U,
kMMC_AccessModeSector = 2U }
MMC card access mode(Access mode in OCR).
- enum mmc_voltage_window_t {
kMMC_VoltageWindowNone = 0U,
kMMC_VoltageWindow120 = 0x01U,
kMMC_VoltageWindow170to195 = 0x02U,
kMMC_VoltageWindows270to360 = 0x1FFU }
MMC card voltage window(VDD voltage window in OCR).
- enum mmc_csd_structure_version_t {
kMMC_CsdStrucureVersion10 = 0U,
kMMC_CsdStrucureVersion11 = 1U,
kMMC_CsdStrucureVersion12 = 2U,
kMMC_CsdStrucureVersionInExtcsd = 3U }
CSD structure version(CSD_STRUCTURE in CSD).
- enum mmc_specification_version_t {

kMMC_SpecificationVersion0 = 0U,
kMMC_SpecificationVersion1 = 1U,
kMMC_SpecificationVersion2 = 2U,
kMMC_SpecificationVersion3 = 3U,
kMMC_SpecificationVersion4 = 4U }
MMC card specification version(SPEC_VERS in CSD).
- enum {
kMMC_ExtendedCsdRevision10 = 0U,
kMMC_ExtendedCsdRevision11 = 1U,
kMMC_ExtendedCsdRevision12 = 2U,
kMMC_ExtendedCsdRevision13 = 3U,
kMMC_ExtendedCsdRevision14 = 4U,
kMMC_ExtendedCsdRevision15 = 5U,
kMMC_ExtendedCsdRevision16 = 6U,
kMMC_ExtendedCsdRevision17 = 7U }
MMC card Extended CSD fix version(EXT_CSD_REV in Extended CSD)
- enum mmc_command_set_t {
kMMC_CommandSetStandard = 0U,
kMMC_CommandSet1 = 1U,
kMMC_CommandSet2 = 2U,
kMMC_CommandSet3 = 3U,
kMMC_CommandSet4 = 4U }
MMC card command set(COMMAND_SET in Extended CSD)
- enum {
kMMC_SupportAlternateBoot = 1U,
kMMC_SupportDDRBoot = 2U,
kMMC_SupportHighSpeedBoot = 4U }
boot support(BOOT_INFO in Extended CSD)
- enum mmc_high_speed_timing_t {
kMMC_HighSpeedTimingNone = 0U,
kMMC_HighSpeedTiming = 1U,
kMMC_HighSpeed200Timing = 2U,
kMMC_HighSpeed400Timing = 3U,
kMMC_EnhanceHighSpeed400Timing = 4U }
MMC card high-speed timing(HS_TIMING in Extended CSD)
- enum mmc_data_bus_width_t {
kMMC_DataBusWidth1bit = 0U,
kMMC_DataBusWidth4bit = 1U,
kMMC_DataBusWidth8bit = 2U,
kMMC_DataBusWidth4bitDDR = 5U,
kMMC_DataBusWidth8bitDDR = 6U,
kMMC_DataBusWidth8bitDDRSTROBE = 0x86U }
MMC card data bus width(BUS_WIDTH in Extended CSD)
- enum mmc_boot_partition_enable_t {

kMMC_BootPartitionEnableNot = 0U,
kMMC_BootPartitionEnablePartition1 = 1U,
kMMC_BootPartitionEnablePartition2 = 2U,
kMMC_BootPartitionEnableUserAera = 7U }
MMC card boot partition enabled(BOOT_PARTITION_ENABLE in Extended CSD)
- enum mmc_boot_timing_mode_t {
kMMC_BootModeSDRWithDefaultTiming = 0U,
kMMC_BootModeSDRWithHighSpeedTiming = 1U,
kMMC_BootModeDDRTiming = 2U }
boot mode configuration Note: HS200 & HS400 is not support during BOOT operation.
- enum mmc_boot_partition_wp_t {
kMMC_BootPartitionWPDisable = 0x50U,
kMMC_BootPartitionPwrWPToBothPartition,
kMMC_BootPartitionPermWPToBothPartition = 0x04U,
kMMC_BootPartitionPwrWPToPartition1 = (1U << 7U) | 1U,
kMMC_BootPartitionPwrWPToPartition2 = (1U << 7U) | 3U,
kMMC_BootPartitionPermWPToPartition1,
kMMC_BootPartitionPermWPToPartition2,
kMMC_BootPartitionPermWPToPartition1PwrWPToPartition2,
kMMC_BootPartitionPermWPToPartition2PwrWPToPartition1 }
MMC card boot partition write protect configurations All the bits in BOOT_WP register, except the two R/W bits B_PERM_WP_DIS and B_PERM_WP_EN, shall only be written once per power cycle.The protection mdde intended for both boot areas will be set with a single write.
- enum {
kMMC_BootPartitionNotProtected = 0U,
kMMC_BootPartitionPwrProtected = 1U,
kMMC_BootPartitionPermProtected = 2U }
MMC card boot partition write protect status.
- enum mmc_access_partition_t {
kMMC_AccessPartitionUserAera = 0U,
kMMC_AccessPartitionBoot1 = 1U,
kMMC_AccessPartitionBoot2 = 2U,
kMMC_AccessRPMB = 3U,
kMMC_AccessGeneralPurposePartition1 = 4U,
kMMC_AccessGeneralPurposePartition2 = 5U,
kMMC_AccessGeneralPurposePartition3 = 6U,
kMMC_AccessGeneralPurposePartition4 = 7U }
MMC card partition to be accessed(BOOT_PARTITION_ACCESS in Extended CSD)
- enum {

kMMC_CsdReadBlockPartialFlag = (1U << 0U),
kMMC_CsdWriteBlockMisalignFlag = (1U << 1U),
kMMC_CsdReadBlockMisalignFlag = (1U << 2U),
kMMC_CsdDsrImplementedFlag = (1U << 3U),
kMMC_CsdWriteProtectGroupEnabledFlag = (1U << 4U),
kMMC_CsdWriteBlockPartialFlag = (1U << 5U),
kMMC_ContentProtectApplicationFlag = (1U << 6U),
kMMC_CsdFileFormatGroupFlag = (1U << 7U),
kMMC_CsdCopyFlag = (1U << 8U),
kMMC_CsdPermanentWriteProtectFlag = (1U << 9U),
kMMC_CsdTemporaryWriteProtectFlag = (1U << 10U) }
MMC card CSD register flags.

- enum mmc_extended_csd_access_mode_t {
kMMC_ExtendedCsdAccessModeCommandSet = 0U,
kMMC_ExtendedCsdAccessModeSetBits = 1U,
kMMC_ExtendedCsdAccessModeClearBits = 2U,
kMMC_ExtendedCsdAccessModeWriteBits = 3U }
Extended CSD register access mode(Access mode in CMD6).
- enum mmc_extended_csd_index_t {
kMMC_ExtendedCsdIndexFlushCache = 32U,
kMMC_ExtendedCsdIndexCacheControl = 33U,
kMMC_ExtendedCsdIndexBootPartitionWP = 173U,
kMMC_ExtendedCsdIndexEraseGroupDefinition = 175U,
kMMC_ExtendedCsdIndexBootBusConditions = 177U,
kMMC_ExtendedCsdIndexBootConfigWP = 178U,
kMMC_ExtendedCsdIndexPartitionConfig = 179U,
kMMC_ExtendedCsdIndexBusWidth = 183U,
kMMC_ExtendedCsdIndexHighSpeedTiming = 185U,
kMMC_ExtendedCsdIndexPowerClass = 187U,
kMMC_ExtendedCsdIndexCommandSet = 191U }
EXT CSD byte index.
- enum {
kMMC_DriverStrength0 = 0U,
kMMC_DriverStrength1 = 1U,
kMMC_DriverStrength2 = 2U,
kMMC_DriverStrength3 = 3U,
kMMC_DriverStrength4 = 4U }
mmc driver strength
- enum mmc_extended_csd_flags_t {
kMMC_ExtCsdExtPartitionSupport = (1 << 0U),
kMMC_ExtCsdEnhancePartitionSupport = (1 << 1U),
kMMC_ExtCsdPartitioningSupport = (1 << 2U),
kMMC_ExtCsdPrgCIDCSDInDDRModeSupport = (1 << 3U),
kMMC_ExtCsdBKOpsSupport = (1 << 4U),
kMMC_ExtCsdDataTagSupport = (1 << 5U),
kMMC_ExtCsdModeOperationCodeSupport = (1 << 6U) }

mmc extended csd flags
- enum mmc_boot_mode_t {
kMMC_BootModeNormal = 0U,
kMMC_BootModeAlternative = 1U }
  MMC card boot mode.

common function

tuning pattern

- status_t SDMMC_SelectCard (sdmmchost_t ∗host, uint32_t relativeAddress, bool isSelected)
  Selects the card to put it into transfer state.
- status_t SDMMC_SendApplicationCommand (sdmmchost_t ∗host, uint32_t relativeAddress)
  Sends an application command.
- status_t SDMMC_SetBlockCount (sdmmchost_t ∗host, uint32_t blockCount)
  Sets the block count.
- status_t SDMMC_GoIdle (sdmmchost_t ∗host)
  Sets the card to be idle state.
- status_t SDMMC_SetBlockSize (sdmmchost_t ∗host, uint32_t blockSize)
  Sets data block size.
- status_t SDMMC_SetCardInactive (sdmmchost_t ∗host)
  Sets card to inactive status.

Data Structure Documentation

68.2.1 struct sd_detect_card_t

Data Fields

- sd_detect_card_type_t type
  card detect type
- uint32_t cdDebounce_ms
  card detect debounce delay ms
- sd_cd_t callback
  card inserted callback which is meaningful for interrupt case
- sd_cd_status_t cardDetected
  used to check sd cd status when card detect through GPIO
- sd_dat3_pull_t dat3PullFunc
  function pointer of DATA3 pull up/down
- void ∗ userData
  user data

68.2.2 struct sd_io_voltage_t

Data Fields

- sd_io_voltage_ctrl_type_t type
  io voltage switch type

- sd_io_voltage_func_t func
  io voltage switch function

68.2.3 struct sd_usr_param_t

Data Fields

- sd_pwr_t pwr
  power control configuration pointer
- sd_io_strength_t ioStrength
  swicth sd io strength
- sd_io_voltage_t * ioVoltage
  switch io voltage
- sd_detect_card_t * cd
  card detect
- uint32_t maxFreq
  board support maximum frequency
- uint32_t capability
  board capability flag

68.2.4 struct sdio_card_int_t

Data Fields

- void * userData
  user data
- sdio_int_t cardInterrupt
  card int call back

68.2.5 struct sdio_usr_param_t

Data Fields

- sd_pwr_t pwr
  power control configuration pointer
- sd_io_strength_t ioStrength
  swicth sd io strength
- sd_io_voltage_t * ioVoltage
  switch io voltage
- sd_detect_card_t * cd
  card detect
- sdio_card_int_t * sdioInt
  card int
- uint32_t maxFreq
  board support maximum frequency

- uint32_t capability
  board capability flag

68.2.6 struct sdio_fbr_t

Data Fields

- uint8_t flags
  current io flags
- uint8_t ioStdFunctionCode
  current io standard function code
- uint8_t ioExtFunctionCode
  current io extended function code
- uint32_t ioPointerToCIS
  current io pointer to CIS
- uint32_t ioPointerToCSA
  current io pointer to CSA
- uint16_t ioBlockSize
  current io block size

68.2.7 struct sdio_common_cis_t

Data Fields

- uint16_t mID
  manufacturer code
- uint16_t mInfo
  manufacturer information
- uint8_t funcID
  function ID
- uint16_t fn0MaxBlkSize
  function 0 max block size
- uint8_t maxTransSpeed
  max data transfer speed for all function

68.2.8 struct sdio_func_cis_t

Data Fields

- uint8_t funcID
  function ID
- uint8_t funcInfo
  function info
- uint8_t ioVersion
  level of application specification this io support

- uint32_t cardPSN
  product serial number
- uint32_t ioCSASize
  avaliable CSA size for io
- uint8_t ioCSAProperty
  CSA property.
- uint16_t ioMaxBlockSize
  io max transfer data size
- uint32_t ioOCR
  io ioeration condition
- uint8_t ioOPMinPwr
  min current in operation mode
- uint8_t ioOPAvgPwr
  average current in operation mode
- uint8_t ioOPMaxPwr
  max current in operation mode
- uint8_t ioSBMinPwr
  min current in standby mode
- uint8_t ioSBAvgPwr
  average current in standby mode
- uint8_t ioSBMaxPwr
  max current in standby mode
- uint16_t ioMinBandWidth
  io min transfer bandwidth
- uint16_t ioOptimumBandWidth
  io optimum transfer bandwidth
- uint16_t ioReadyTimeout
  timeout value from enalbe to ready
- uint16_t ioHighCurrentAvgCurrent
  `the average peak current (mA)`
  when IO operating in high current mode
- uint16_t ioHighCurrentMaxCurrent
  `the max peak current (mA)`
  when IO operating in high current mode
- uint16_t ioLowCurrentAvgCurrent
  `the average peak current (mA)`
  when IO operating in lower current mode
- uint16_t ioLowCurrentMaxCurrent
  `the max peak current (mA)`
  when IO operating in lower current mode

68.2.9 struct sd_status_t

Data Fields

- uint8_t busWidth
  current buswidth
- uint8_t secureMode
  secured mode

- uint16_t cardType
  sdcard type
- uint32_t protectedSize
  size of protected area
- uint8_t speedClass
  speed class of card
- uint8_t performanceMove
  Performance of move indicated by 1[MB/S]step.
- uint8_t auSize
  size of AU
- uint16_t eraseSize
  number of AUs to be erased at a time
- uint8_t eraseTimeout
  timeout value for erasing areas specified by UNIT OF ERASE AU
- uint8_t eraseOffset
  fixed offset value added to erase time
- uint8_t uhsSpeedGrade
  speed grade for UHS mode
- uint8_t uhsAuSize
  size of AU for UHS mode

68.2.10 struct sd_cid_t

Data Fields

- uint8_t manufacturerID
  Manufacturer ID [127:120].
- uint16_t applicationID
  OEM/Application ID [119:104].
- uint8_t productName [SD_PRODUCT_NAME_BYTES]
  Product name [103:64].
- uint8_t productVersion
  Product revision [63:56].
- uint32_t productSerialNumber
  Product serial number [55:24].
- uint16_t manufacturerData
  Manufacturing date [19:8].

68.2.11 struct sd_csd_t

Data Fields

- uint8_t csdStructure
  CSD structure [127:126].
- uint8_t dataReadAccessTime1
  Data read access-time-1 [119:112].
- uint8_t dataReadAccessTime2

Data read access-time-2 in clock cycles (NSAC∗100) [111:104].
- uint8_t transferSpeed
  Maximum data transfer rate [103:96].
- uint16_t cardCommandClass
  Card command classes [95:84].
- uint8_t readBlockLength
  Maximum read data block length [83:80].
- uint16_t flags
  Flags in _sd_csd_flag.
- uint32_t deviceSize
  Device size [73:62].
- uint8_t readCurrentVddMin
  Maximum read current at VDD min [61:59].
- uint8_t readCurrentVddMax
  Maximum read current at VDD max [58:56].
- uint8_t writeCurrentVddMin
  Maximum write current at VDD min [55:53].
- uint8_t writeCurrentVddMax
  Maximum write current at VDD max [52:50].
- uint8_t deviceSizeMultiplier
  Device size multiplier [49:47].
- uint8_t eraseSectorSize
  Erase sector size [45:39].
- uint8_t writeProtectGroupSize
  Write protect group size [38:32].
- uint8_t writeSpeedFactor
  Write speed factor [28:26].
- uint8_t writeBlockLength
  Maximum write data block length [25:22].
- uint8_t fileFormat
  File format [11:10].

68.2.12 struct sd_scr_t

Data Fields

- uint8_t scrStructure
  SCR Structure [63:60].
- uint8_t sdSpecification
  SD memory card specification version [59:56].
- uint16_t flags
  SCR flags in _sd_scr_flag.
- uint8_t sdSecurity
  Security specification supported [54:52].
- uint8_t sdBusWidths
  Data bus widths supported [51:48].
- uint8_t extendedSecurity
  Extended security support [46:43].
- uint8_t commandSupport

Command support bits [33:32] 33-support CMD23, 32-support cmd20.
- uint32_t reservedForManufacturer
  reserved for manufacturer usage [31:0]

68.2.13 struct mmc_cid_t

Data Fields

- uint8_t manufacturerID
  Manufacturer ID.
- uint16_t applicationID
  OEM/Application ID.
- uint8_t productName [MMC_PRODUCT_NAME_BYTES]
  Product name.
- uint8_t productVersion
  Product revision.
- uint32_t productSerialNumber
  Product serial number.
- uint8_t manufacturerData
  Manufacturing date.

68.2.14 struct mmc_csd_t

Data Fields

- uint8_t csdStructureVersion
  CSD structure [127:126].
- uint8_t systemSpecificationVersion
  System specification version [125:122].
- uint8_t dataReadAccessTime1
  Data read access-time 1 [119:112].
- uint8_t dataReadAccessTime2
  *Data read access-time 2 in CLOCK cycles (NSAC*100) [111:104].*
- uint8_t transferSpeed
  Max.
- uint16_t cardCommandClass
  card command classes [95:84]
- uint8_t readBlockLength
  Max.
- uint16_t flags
  Contain flags in _mmc_csd_flag.
- uint16_t deviceSize
  Device size [73:62].
- uint8_t readCurrentVddMin
  Max.
- uint8_t readCurrentVddMax
  Max.

- uint8_t writeCurrentVddMin
  Max.
- uint8_t writeCurrentVddMax
  Max.
- uint8_t deviceSizeMultiplier
  Device size multiplier [49:47].
- uint8_t eraseGroupSize
  Erase group size [46:42].
- uint8_t eraseGroupSizeMultiplier
  Erase group size multiplier [41:37].
- uint8_t writeProtectGroupSize
  Write protect group size [36:32].
- uint8_t defaultEcc
  Manufacturer default ECC [30:29].
- uint8_t writeSpeedFactor
  Write speed factor [28:26].
- uint8_t maxWriteBlockLength
  Max.
- uint8_t fileFormat
  File format [11:10].
- uint8_t eccCode
  ECC code [9:8].

68.2.14.0.0.58 Field Documentation

68.2.14.0.0.58.1 uint8_t mmc_csd_t::transferSpeed

bus clock frequency [103:96]

68.2.14.0.0.58.2 uint8_t mmc_csd_t::readBlockLength

read data block length [83:80]

68.2.14.0.0.58.3 uint8_t mmc_csd_t::readCurrentVddMin

read current @ VDD min [61:59]

68.2.14.0.0.58.4 uint8_t mmc_csd_t::readCurrentVddMax

read current @ VDD max [58:56]

68.2.14.0.0.58.5 uint8_t mmc_csd_t::writeCurrentVddMin

write current @ VDD min [55:53]

68.2.14.0.0.58.6 uint8_t mmc_csd_t::writeCurrentVddMax

write current @ VDD max [52:50]

68.2.14.0.0.58.7 uint8_t mmc_csd_t::maxWriteBlockLength

write data block length [25:22]

68.2.15 struct mmc_extended_csd_t

Data Fields

- uint8_t cacheCtrl
  < secure removal type[16]
- uint8_t partitionAttribute
  < power off notification[34]
- uint8_t userWP
  < max enhance area size [159-157]
- uint8_t bootPartitionWP
  boot write protect register[173]
- uint8_t bootWPStatus
  boot write protect status register[174]
- uint8_t highDensityEraseGroupDefinition
  High-density erase group definition [175].
- uint8_t bootDataBusConditions
  Boot bus conditions [177].
- uint8_t bootConfigProtect
  Boot config protection [178].
- uint8_t partitionConfig
  Boot configuration [179].
- uint8_t eraseMemoryContent
  Erased memory content [181].
- uint8_t dataBusWidth
  Data bus width mode [183].
- uint8_t highSpeedTiming
  High-speed interface timing [185].
- uint8_t powerClass
  Power class [187].
- uint8_t commandSetRevision
  Command set revision [189].
- uint8_t commandSet
  Command set [191].
- uint8_t extendecCsdVersion
  Extended CSD revision [192].
- uint8_t csdStructureVersion
  CSD structure version [194].
- uint8_t cardType
  Card Type [196].
- uint8_t ioDriverStrength
  IO driver strength [197].
- uint8_t powerClass52MHz195V
  < out of interrupt busy timing [198]
- uint8_t powerClass26MHz195V

Power Class for 26MHz @ 1.95V [201].
- uint8_t powerClass52MHz360V
  Power Class for 52MHz @ 3.6V [202].
- uint8_t powerClass26MHz360V
  Power Class for 26MHz @ 3.6V [203].
- uint8_t minimumReadPerformance4Bit26MHz
  Minimum Read Performance for 4bit at 26MHz [205].
- uint8_t minimumWritePerformance4Bit26MHz
  Minimum Write Performance for 4bit at 26MHz [206].
- uint8_t minimumReadPerformance8Bit26MHz4Bit52MHz
  Minimum read Performance for 8bit at 26MHz/4bit @52MHz [207].
- uint8_t minimumWritePerformance8Bit26MHz4Bit52MHz
  Minimum Write Performance for 8bit at 26MHz/4bit @52MHz [208].
- uint8_t minimumReadPerformance8Bit52MHz
  Minimum Read Performance for 8bit at 52MHz [209].
- uint8_t minimumWritePerformance8Bit52MHz
  Minimum Write Performance for 8bit at 52MHz [210].
- uint32_t sectorCount
  Sector Count [215:212].
- uint8_t sleepAwakeTimeout
  < sleep notification timeout [216]
- uint8_t sleepCurrentVCCQ
  < Production state awareness timeout [218]
- uint8_t sleepCurrentVCC
  Sleep current (VCC) [220].
- uint8_t highCapacityWriteProtectGroupSize
  High-capacity write protect group size [221].
- uint8_t reliableWriteSectorCount
  Reliable write sector count [222].
- uint8_t highCapacityEraseTimeout
  High-capacity erase timeout [223].
- uint8_t highCapacityEraseUnitSize
  High-capacity erase unit size [224].
- uint8_t accessSize
  Access size [225].
- uint8_t minReadPerformance8bitAt52MHZDDR
  < secure trim multiplier[229]
- uint8_t minWritePerformance8bitAt52MHZDDR
  Minimum write performance for 8bit at DDR 52MHZ[235].
- uint8_t powerClass200MHZVCCQ130VVCC360V
  power class for 200MHZ, at VCCQ= 1.3V,VCC=3.6V[236]
- uint8_t powerClass200MHZVCCQ195VVCC360V
  power class for 200MHZ, at VCCQ= 1.95V,VCC=3.6V[237]
- uint8_t powerClass52MHZDDR195V
  power class for 52MHZ,DDR at Vcc 1.95V[238]
- uint8_t powerClass52MHZDDR360V
  power class for 52MHZ,DDR at Vcc 3.6V[239]
- uint32_t genericCMD6Timeout
  < 1st initialization time after partitioning[241]
- uint32_t cacheSize
  cache size[252-249]

- uint8_t powerClass200MHZDDR360V
  power class for 200MHZ, DDR at VCC=2.6V[253]
- uint8_t extPartitionSupport
  < fw VERSION [261-254]
- uint8_t supportedCommandSet
  < large unit size[495]

68.2.15.0.0.59 Field Documentation

68.2.15.0.0.59.1 uint8_t mmc_extended_csd_t::cacheCtrl

< product state awareness enablement[17]

< max preload data size[21-18]

< pre-load data size[25-22]

< FFU status [26]

< mode operation code[29]

< mode config [30] control to turn on/off cache[33]

68.2.15.0.0.59.2 uint8_t mmc_extended_csd_t::partitionAttribute

< packed cmd fail index [35]

< packed cmd status[36]

< context configuration[51-37]

< extended partitions attribut[53-52]

< exception events status[55-54]

< exception events control[57-56]

< number of group to be released[58]

< class 6 command control[59]

< 1st initiallization after disabling sector size emu[60]

< sector size[61]

< sector size emulation[62]

< native sector size[63]

< period wakeup [131]

< package case temperature is controlled[132]

< production state awareness[133]

< enhanced user data start addr [139-136]

< enhanced user data area size[142-140]

< general purpose partition size[154-143] partition attribute [156]

68.2.15.0.0.59.3 uint8_t mmc_extended_csd_t::userWP

< HPI management [161]

< write reliability parameter register[166]

< write reliability setting register[167]

< RPMB size multi [168]

< FW configuration[169] user write protect register[171]

68.2.15.0.0.59.4 uint8_t mmc_extended_csd_t::powerClass52MHz195V

< partition switch timing [199] Power Class for 52MHz @ 1.95V [200]

68.2.15.0.0.59.5 uint8_t mmc_extended_csd_t::sleepAwakeTimeout

Sleep/awake timeout [217]

68.2.15.0.0.59.6 uint8_t mmc_extended_csd_t::sleepCurrentVCCQ

Sleep current (VCCQ) [219]

68.2.15.0.0.59.7 uint8_t mmc_extended_csd_t::minReadPerformance8bitAt52MHZDDR

< secure erase multiplier[230]

< secure feature support[231]

< trim multiplier[232] Minimum read performance for 8bit at DDR 52MHZ[234]

68.2.15.0.0.59.8 uint32_t mmc_extended_csd_t::genericCMD6Timeout

< correct prg sectors number[245-242]

< background operations status[246]

< power off notification timeout[247] generic CMD6 timeout[248]

68.2.15.0.0.59.9 uint8_t mmc_extended_csd_t::extPartitionSupport

< device version[263-262]

< optimal trim size[264]

< optimal write size[265]

< optimal read size[266]

< pre EOL information[267]

< device life time estimation typeA[268]

< device life time estimation typeB[269]

< number of FW sectors correctly programmed[305-302]

< FFU argument[490-487]

< operation code timeout[491]

< support mode [493] extended partition attribute support[494]

68.2.15.0.0.59.10 uint8_t mmc_extended_csd_t::supportedCommandSet

< context management capability[496]

< tag resource size[497]

< tag unit size[498]

< max packed write cmd[500]

< max packed read cmd[501]

< HPI feature[503] Supported Command Sets [504]

68.2.16 struct mmc_extended_csd_config_t

Data Fields

- mmc_command_set_t commandSet
  Command set.
- uint8_t ByteValue
  The value to set.
- uint8_t ByteIndex
  The byte index in Extended CSD(mmc_extended_csd_index_t)
- mmc_extended_csd_access_mode_t accessMode
  Access mode.

68.2.17 struct mmc_boot_config_t

Data Fields

- mmc_boot_mode_t bootMode
  mmc boot mode
- bool enableBootAck
  Enable boot ACK.
- mmc_boot_partition_enable_t bootPartition
  Boot partition.
- mmc_boot_timing_mode_t bootTimingMode
  boot mode
- mmc_data_bus_width_t bootDataBusWidth
  Boot data bus width.
- bool retainBootbusCondition
  If retain boot bus width and boot mode conditions.
- bool pwrBootConfigProtection

`Disable the change of boot configuration register bits from at this point`
until next power cycle or next H/W reset operation

- bool premBootConfigProtection
  Disable the change of boot configuration register bits permanently.
- mmc_boot_partition_wp_t bootPartitionWP
  boot partition write protect configurations

Macro Definition Documentation

68.3.1 #define SDMMC_LOG(*format, ...*)

68.3.2 #define READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT(*CSD*) ((((CSD).transferSpeed) & MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK) >> MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT)

68.3.3 #define READ_MMC_TRANSFER_SPEED_MULTIPLIER(*CSD*) ((((CSD).transferSpeed) & MMC_TRANSFER_SPEED_MULTIPLIER_MASK) >> MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT)

68.3.4 #define MMC_EXTENDED_CSD_BYTES (512U)

68.3.5 #define SD_PRODUCT_NAME_BYTES (5U)

68.3.6 #define MMC_PRODUCT_NAME_BYTES (6U)

68.3.7 #define MMC_SWITCH_COMMAND_SET_SHIFT (0U)

68.3.8 #define MMC_SWITCH_COMMAND_SET_MASK (0x00000007U)

Enumeration Type Documentation

68.4.1 anonymous enum

Enumerator

kStatus_SDMMC_NotSupportYet Haven't supported.
kStatus_SDMMC_TransferFailed Send command failed.
kStatus_SDMMC_SetCardBlockSizeFailed Set block size failed.
kStatus_SDMMC_HostNotSupport Host doesn't support.
kStatus_SDMMC_CardNotSupport Card doesn't support.
kStatus_SDMMC_AllSendCidFailed Send CID failed.
kStatus_SDMMC_SendRelativeAddressFailed Send relative address failed.
kStatus_SDMMC_SendCsdFailed Send CSD failed.

kStatus_SDMMC_SelectCardFailed Select card failed.
kStatus_SDMMC_SendScrFailed Send SCR failed.
kStatus_SDMMC_SetDataBusWidthFailed Set bus width failed.
kStatus_SDMMC_GoIdleFailed Go idle failed.
kStatus_SDMMC_HandShakeOperationConditionFailed Send Operation Condition failed.
kStatus_SDMMC_SendApplicationCommandFailed Send application command failed.
kStatus_SDMMC_SwitchFailed Switch command failed.
kStatus_SDMMC_StopTransmissionFailed Stop transmission failed.
kStatus_SDMMC_WaitWriteCompleteFailed Wait write complete failed.
kStatus_SDMMC_SetBlockCountFailed Set block count failed.
kStatus_SDMMC_SetRelativeAddressFailed Set relative address failed.
kStatus_SDMMC_SwitchBusTimingFailed Switch high speed failed.
kStatus_SDMMC_SendExtendedCsdFailed Send EXT_CSD failed.
kStatus_SDMMC_ConfigureBootFailed Configure boot failed.
kStatus_SDMMC_ConfigureExtendedCsdFailed Configure EXT_CSD failed.
kStatus_SDMMC_EnableHighCapacityEraseFailed Enable high capacity erase failed.
kStatus_SDMMC_SendTestPatternFailed Send test pattern failed.
kStatus_SDMMC_ReceiveTestPatternFailed Receive test pattern failed.
kStatus_SDMMC_SDIO_ResponseError sdio response error
kStatus_SDMMC_SDIO_InvalidArgument sdio invalid argument response error
kStatus_SDMMC_SDIO_SendOperationConditionFail sdio send operation condition fail
kStatus_SDMMC_InvalidVoltage invaild voltage
kStatus_SDMMC_SDIO_SwitchHighSpeedFail switch to high speed fail
kStatus_SDMMC_SDIO_ReadCISFail read CIS fail
kStatus_SDMMC_SDIO_InvalidCard invaild SDIO card
kStatus_SDMMC_TuningFail tuning fail
kStatus_SDMMC_SwitchVoltageFail switch voltage fail
kStatus_SDMMC_SwitchVoltage18VFail33VSuccess switch voltage fail
kStatus_SDMMC_ReTuningRequest retuning request
kStatus_SDMMC_SetDriverStrengthFail set driver strength fail
kStatus_SDMMC_SetPowerClassFail set power class fail
kStatus_SDMMC_HostNotReady host controller not ready
kStatus_SDMMC_CardDetectFailed card detect failed
kStatus_SDMMC_AuSizeNotSetProperly AU size not set properly.
kStatus_SDMMC_PollingCardIdleFailed polling card idle status failed

68.4.2 anonymous enum

Enumerator

kSDMMC_SignalLineCmd cmd line
kSDMMC_SignalLineData0 data line
kSDMMC_SignalLineData1 data line
kSDMMC_SignalLineData2 data line

kSDMMC_SignalLineData3 data line
kSDMMC_SignalLineData4 data line
kSDMMC_SignalLineData5 data line
kSDMMC_SignalLineData6 data line
kSDMMC_SignalLineData7 data line

68.4.3 enum sdmmc_operation_voltage_t

Enumerator

kSDMMC_OperationVoltageNone indicate current voltage setting is not setting by suser
kSDMMC_OperationVoltage330V card operation voltage around 3.3v
kSDMMC_OperationVoltage300V card operation voltage around 3.0v
kSDMMC_OperationVoltage180V card operation voltage around 1.8v

68.4.4 anonymous enum

Enumerator

kSDMMC_BusWdith1Bit card bus 1 width
kSDMMC_BusWdith4Bit card bus 4 width
kSDMMC_BusWdith8Bit card bus 8 width

68.4.5 anonymous enum

Enumerator

kSDMMC_Support8BitWidth 8 bit data width capability

68.4.6 enum sd_detect_card_type_t

Enumerator

kSD_DetectCardByGpioCD sd card detect by CD pin through GPIO
kSD_DetectCardByHostCD sd card detect by CD pin through host
kSD_DetectCardByHostDATA3 sd card detect by DAT3 pin through host

68.4.7 anonymous enum

Enumerator

kSD_Inserted card is inserted

kSD_Removed card is removed

68.4.8 anonymous enum

Enumerator

kSD_DAT3PullDown data3 pull down
kSD_DAT3PullUp data3 pull up

68.4.9 enum sd_io_voltage_ctrl_type_t

Enumerator

kSD_IOVoltageCtrlNotSupport io voltage control not support
kSD_IOVoltageCtrlByHost io voltage control by host
kSD_IOVoltageCtrlByGpio io voltage control by gpio

68.4.10 anonymous enum

Enumerator

kSDMMC_R1OutOfRangeFlag Out of range status bit.
kSDMMC_R1AddressErrorFlag Address error status bit.
kSDMMC_R1BlockLengthErrorFlag Block length error status bit.
kSDMMC_R1EraseSequenceErrorFlag Erase sequence error status bit.
kSDMMC_R1EraseParameterErrorFlag Erase parameter error status bit.
kSDMMC_R1WriteProtectViolationFlag Write protection violation status bit.
kSDMMC_R1CardIsLockedFlag Card locked status bit.
kSDMMC_R1LockUnlockFailedFlag lock/unlock error status bit
kSDMMC_R1CommandCrcErrorFlag CRC error status bit.
kSDMMC_R1IllegalCommandFlag Illegal command status bit.
kSDMMC_R1CardEccFailedFlag Card ecc error status bit.
kSDMMC_R1CardControllerErrorFlag Internal card controller error status bit.
kSDMMC_R1ErrorFlag A general or an unknown error status bit.
kSDMMC_R1CidCsdOverwriteFlag Cid/csd overwrite status bit.
kSDMMC_R1WriteProtectEraseSkipFlag Write protection erase skip status bit.
kSDMMC_R1CardEccDisabledFlag Card ecc disabled status bit.
kSDMMC_R1EraseResetFlag Erase reset status bit.
kSDMMC_R1ReadyForDataFlag Ready for data status bit.
kSDMMC_R1SwitchErrorFlag Switch error status bit.
kSDMMC_R1ApplicationCommandFlag Application command enabled status bit.
kSDMMC_R1AuthenticationSequenceErrorFlag error in the sequence of authentication process

68.4.11 enum sdmmc_r1_current_state_t

Enumerator

kSDMMC_R1StateIdle R1: current state: idle.
kSDMMC_R1StateReady R1: current state: ready.
kSDMMC_R1StateIdentify R1: current state: identification.
kSDMMC_R1StateStandby R1: current state: standby.
kSDMMC_R1StateTransfer R1: current state: transfer.
kSDMMC_R1StateSendData R1: current state: sending data.
kSDMMC_R1StateReceiveData R1: current state: receiving data.
kSDMMC_R1StateProgram R1: current state: programming.
kSDMMC_R1StateDisconnect R1: current state: disconnect.

68.4.12 anonymous enum

Enumerator

kSDSPI_R1InIdleStateFlag In idle state.
kSDSPI_R1EraseResetFlag Erase reset.
kSDSPI_R1IllegalCommandFlag Illegal command.
kSDSPI_R1CommandCrcErrorFlag Com crc error.
kSDSPI_R1EraseSequenceErrorFlag Erase sequence error.
kSDSPI_R1AddressErrorFlag Address error.
kSDSPI_R1ParameterErrorFlag Parameter error.

68.4.13 anonymous enum

Enumerator

kSDSPI_R2CardLockedFlag Card is locked.
kSDSPI_R2WriteProtectEraseSkip Write protect erase skip.
kSDSPI_R2LockUnlockFailed Lock/unlock command failed.
kSDSPI_R2ErrorFlag Unknown error.
kSDSPI_R2CardControllerErrorFlag Card controller error.
kSDSPI_R2CardEccFailedFlag Card ecc failed.
kSDSPI_R2WriteProtectViolationFlag Write protect violation.
kSDSPI_R2EraseParameterErrorFlag Erase parameter error.
kSDSPI_R2OutOfRangeFlag Out of range.
kSDSPI_R2CsdOverwriteFlag CSD overwrite.

68.4.14 anonymous enum

Enumerator

kSDSPI_DataErrorTokenError Data error.
kSDSPI_DataErrorTokenCardControllerError Card controller error.
kSDSPI_DataErrorTokenCardEccFailed Card ecc error.
kSDSPI_DataErrorTokenOutOfRange Out of range.

68.4.15 enum sdspi_data_token_t

Enumerator

kSDSPI_DataTokenBlockRead Single block read, multiple block read.
kSDSPI_DataTokenSingleBlockWrite Single block write.
kSDSPI_DataTokenMultipleBlockWrite Multiple block write.
kSDSPI_DataTokenStopTransfer Stop transmission.

68.4.16 enum sdspi_data_response_token_t

Enumerator

kSDSPI_DataResponseTokenAccepted Data accepted.
kSDSPI_DataResponseTokenCrcError Data rejected due to CRC error.
kSDSPI_DataResponseTokenWriteError Data rejected due to write error.

68.4.17 enum sd_command_t

Enumerator

kSD_SendRelativeAddress Send Relative Address.
kSD_Switch Switch Function.
kSD_SendInterfaceCondition Send Interface Condition.
kSD_VoltageSwitch Voltage Switch.
kSD_SpeedClassControl Speed Class control.
kSD_EraseWriteBlockStart Write Block Start.
kSD_EraseWriteBlockEnd Write Block End.
kSD_SendTuningBlock Send Tuning Block.

68.4.18 enum sdspi_command_t

Enumerator

kSDSPI_CommandCrc Command crc protection on/off.

68.4.19 enum sd_application_command_t

Enumerator

kSD_ApplicationSetBusWdith Set Bus Width.
kSD_ApplicationStatus Send SD status.
kSD_ApplicationSendNumberWriteBlocks Send Number Of Written Blocks.
kSD_ApplicationSetWriteBlockEraseCount Set Write Block Erase Count.
kSD_ApplicationSendOperationCondition Send Operation Condition.
kSD_ApplicationSetClearCardDetect Set Connnect/Disconnect pull up on detect pin.
kSD_ApplicationSendScr Send Scr.

68.4.20 anonymous enum

Enumerator

kSDMMC_CommandClassBasic Card command class 0.
kSDMMC_CommandClassBlockRead Card command class 2.
kSDMMC_CommandClassBlockWrite Card command class 4.
kSDMMC_CommandClassErase Card command class 5.
kSDMMC_CommandClassWriteProtect Card command class 6.
kSDMMC_CommandClassLockCard Card command class 7.
kSDMMC_CommandClassApplicationSpecific Card command class 8.
kSDMMC_CommandClassInputOutputMode Card command class 9.
kSDMMC_CommandClassSwitch Card command class 10.

68.4.21 anonymous enum

Enumerator

kSD_OcrPowerUpBusyFlag Power up busy status.
kSD_OcrHostCapacitySupportFlag Card capacity status.
kSD_OcrCardCapacitySupportFlag Card capacity status.
kSD_OcrSwitch18RequestFlag Switch to 1.8V request.
kSD_OcrSwitch18AcceptFlag Switch to 1.8V accepted.
kSD_OcrVdd27_28Flag VDD 2.7-2.8.
kSD_OcrVdd28_29Flag VDD 2.8-2.9.

kSD_OcrVdd29_30Flag VDD 2.9-3.0.
kSD_OcrVdd30_31Flag VDD 2.9-3.0.
kSD_OcrVdd31_32Flag VDD 3.0-3.1.
kSD_OcrVdd32_33Flag VDD 3.1-3.2.
kSD_OcrVdd33_34Flag VDD 3.2-3.3.
kSD_OcrVdd34_35Flag VDD 3.3-3.4.
kSD_OcrVdd35_36Flag VDD 3.4-3.5.

68.4.22 anonymous enum

Enumerator

kSD_SpecificationVersion1_0 SD card version 1.0-1.01.
kSD_SpecificationVersion1_1 SD card version 1.10.
kSD_SpecificationVersion2_0 SD card version 2.00.
kSD_SpecificationVersion3_0 SD card version 3.0.

68.4.23 enum sd_switch_mode_t

Enumerator

kSD_SwitchCheck SD switch mode 0: check function.
kSD_SwitchSet SD switch mode 1: set function.

68.4.24 anonymous enum

Enumerator

kSD_CsdReadBlockPartialFlag Partial blocks for read allowed [79:79].
kSD_CsdWriteBlockMisalignFlag Write block misalignment [78:78].
kSD_CsdReadBlockMisalignFlag Read block misalignment [77:77].
kSD_CsdDsrImplementedFlag DSR implemented [76:76].
kSD_CsdEraseBlockEnabledFlag Erase single block enabled [46:46].
kSD_CsdWriteProtectGroupEnabledFlag Write protect group enabled [31:31].
kSD_CsdWriteBlockPartialFlag Partial blocks for write allowed [21:21].
kSD_CsdFileFormatGroupFlag File format group [15:15].
kSD_CsdCopyFlag Copy flag [14:14].
kSD_CsdPermanentWriteProtectFlag Permanent write protection [13:13].
kSD_CsdTemporaryWriteProtectFlag Temporary write protection [12:12].

68.4.25 anonymous enum

Enumerator

kSD_ScrDataStatusAfterErase Data status after erases [55:55].
kSD_ScrSdSpecification3 Specification version 3.00 or higher [47:47].

68.4.26 anonymous enum

Enumerator

kSD_FunctionSDR12Deafult SDR12 mode & default.
kSD_FunctionSDR25HighSpeed SDR25 & high speed.
kSD_FunctionSDR50 SDR50 mode.
kSD_FunctionSDR104 SDR104 mode.
kSD_FunctionDDR50 DDR50 mode.

68.4.27 anonymous enum

Enumerator

kSD_GroupTimingMode acess mode group
kSD_GroupCommandSystem command system group
kSD_GroupDriverStrength driver strength group
kSD_GroupCurrentLimit current limit group

68.4.28 enum sd_timing_mode_t

Enumerator

kSD_TimingSDR12DefaultMode Identification mode & SDR12.
kSD_TimingSDR25HighSpeedMode High speed mode & SDR25.
kSD_TimingSDR50Mode SDR50 mode.
kSD_TimingSDR104Mode SDR104 mode.
kSD_TimingDDR50Mode DDR50 mode.

68.4.29 enum sd_driver_strength_t

Enumerator

kSD_DriverStrengthTypeB default driver strength
kSD_DriverStrengthTypeA driver strength TYPE A

kSD_DriverStrengthTypeC driver strength TYPE C
kSD_DriverStrengthTypeD driver strength TYPE D

68.4.30 enum sd_max_current_t

Enumerator

kSD_CurrentLimit200MA default current limit
kSD_CurrentLimit400MA current limit to 400MA
kSD_CurrentLimit600MA current limit to 600MA
kSD_CurrentLimit800MA current limit to 800MA

68.4.31 enum sdmmc_command_t

Enumerator

kSDMMC_GoIdleState Go Idle State.
kSDMMC_AllSendCid All Send CID.
kSDMMC_SetDsr Set DSR.
kSDMMC_SelectCard Select Card.
kSDMMC_SendCsd Send CSD.
kSDMMC_SendCid Send CID.
kSDMMC_StopTransmission Stop Transmission.
kSDMMC_SendStatus Send Status.
kSDMMC_GoInactiveState Go Inactive State.
kSDMMC_SetBlockLength Set Block Length.
kSDMMC_ReadSingleBlock Read Single Block.
kSDMMC_ReadMultipleBlock Read Multiple Block.
kSDMMC_SetBlockCount Set Block Count.
kSDMMC_WriteSingleBlock Write Single Block.
kSDMMC_WriteMultipleBlock Write Multiple Block.
kSDMMC_ProgramCsd Program CSD.
kSDMMC_SetWriteProtect Set Write Protect.
kSDMMC_ClearWriteProtect Clear Write Protect.
kSDMMC_SendWriteProtect Send Write Protect.
kSDMMC_Erase Erase.
kSDMMC_LockUnlock Lock Unlock.
kSDMMC_ApplicationCommand Send Application Command.
kSDMMC_GeneralCommand General Purpose Command.
kSDMMC_ReadOcr Read OCR.

68.4.32 anonymous enum

Enumerator

kSDIO_RegCCCRSdioVer CCCR & SDIO version.
kSDIO_RegSDVersion SD version.
kSDIO_RegIOEnable io enable register
kSDIO_RegIOReady io ready register
kSDIO_RegIOIntEnable io interrupt enable register
kSDIO_RegIOIntPending io interrupt pending register
kSDIO_RegIOAbort io abort register
kSDIO_RegBusInterface bus interface register
kSDIO_RegCardCapability card capability register
kSDIO_RegCommonCISPointer common CIS pointer register
kSDIO_RegBusSuspend bus suspend register
kSDIO_RegFunctionSelect function select register
kSDIO_RegExecutionFlag execution flag register
kSDIO_RegReadyFlag ready flag register
kSDIO_RegFN0BlockSizeLow FN0 block size register.
kSDIO_RegFN0BlockSizeHigh FN0 block size register.
kSDIO_RegPowerControl power control register
kSDIO_RegBusSpeed bus speed register
kSDIO_RegUHSITimingSupport UHS-I timing support register.
kSDIO_RegDriverStrength Driver strength register.
kSDIO_RegInterruptExtension Interrupt extension register.

68.4.33 enum sdio_command_t

Enumerator

kSDIO_SendRelativeAddress send relative address
kSDIO_SendOperationCondition send operation condition
kSDIO_SendInterfaceCondition send interface condition
kSDIO_RWIODirect read/write IO direct command
kSDIO_RWIOExtended read/write IO extended command

68.4.34 enum sdio_func_num_t

Enumerator

kSDIO_FunctionNum0 sdio function0
kSDIO_FunctionNum1 sdio function1
kSDIO_FunctionNum2 sdio function2

kSDIO_FunctionNum3 sdio function3
kSDIO_FunctionNum4 sdio function4
kSDIO_FunctionNum5 sdio function5
kSDIO_FunctionNum6 sdio function6
kSDIO_FunctionNum7 sdio function7
kSDIO_FunctionMemory for combo card

68.4.35 anonymous enum

Enumerator

kSDIO_StatusCmdCRCError the CRC check of the previous cmd fail
kSDIO_StatusIllegalCmd cmd illegal for the card state
kSDIO_StatusR6Error special for R6 error status
kSDIO_StatusError A general or an unknown error occurred.
kSDIO_StatusFunctionNumError invail function error
kSDIO_StatusOutofRange cmd argument was out of the allowed range

68.4.36 anonymous enum

Enumerator

kSDIO_OcrPowerUpBusyFlag Power up busy status.
kSDIO_OcrIONumber number of IO function
kSDIO_OcrMemPresent memory present flag
kSDIO_OcrVdd20_21Flag VDD 2.0-2.1.
kSDIO_OcrVdd21_22Flag VDD 2.1-2.2.
kSDIO_OcrVdd22_23Flag VDD 2.2-2.3.
kSDIO_OcrVdd23_24Flag VDD 2.3-2.4.
kSDIO_OcrVdd24_25Flag VDD 2.4-2.5.
kSDIO_OcrVdd25_26Flag VDD 2.5-2.6.
kSDIO_OcrVdd26_27Flag VDD 2.6-2.7.
kSDIO_OcrVdd27_28Flag VDD 2.7-2.8.
kSDIO_OcrVdd28_29Flag VDD 2.8-2.9.
kSDIO_OcrVdd29_30Flag VDD 2.9-3.0.
kSDIO_OcrVdd30_31Flag VDD 2.9-3.0.
kSDIO_OcrVdd31_32Flag VDD 3.0-3.1.
kSDIO_OcrVdd32_33Flag VDD 3.1-3.2.
kSDIO_OcrVdd33_34Flag VDD 3.2-3.3.
kSDIO_OcrVdd34_35Flag VDD 3.3-3.4.
kSDIO_OcrVdd35_36Flag VDD 3.4-3.5.

68.4.37 anonymous enum

Enumerator

kSDIO_CCCRSupportDirectCmdDuringDataTrans support direct cmd during data transfer
kSDIO_CCCRSupportMultiBlock support multi block mode
kSDIO_CCCRSupportReadWait support read wait
kSDIO_CCCRSupportSuspendResume support suspend resume
kSDIO_CCCRSupportIntDuring4BitDataTrans support interrupt during 4-bit data transfer
kSDIO_CCCRSupportLowSpeed1Bit support low speed 1bit mode
kSDIO_CCCRSupportLowSpeed4Bit support low speed 4bit mode
kSDIO_CCCRSupportMasterPowerControl support master power control
kSDIO_CCCRSupportHighSpeed support high speed
kSDIO_CCCRSupportContinuousSPIInt support continuous SPI interrupt

68.4.38 anonymous enum

Enumerator

kSDIO_FBRSupportCSA function support CSA
kSDIO_FBRSupportPowerSelection function support power selection

68.4.39 enum sdio_bus_width_t

Enumerator

kSDIO_DataBus1Bit 1 bit bus mode
kSDIO_DataBus4Bit 4 bit bus mode
kSDIO_DataBus8Bit 8 bit bus mode

68.4.40 enum mmc_command_t

Enumerator

kMMC_SendOperationCondition Send Operation Condition.
kMMC_SetRelativeAddress Set Relative Address.
kMMC_SleepAwake Sleep Awake.
kMMC_Switch Switch.
kMMC_SendExtendedCsd Send EXT_CSD.
kMMC_ReadDataUntilStop Read Data Until Stop.
kMMC_BusTestRead Test Read.
kMMC_SendingBusTest test bus width cmd
kMMC_WriteDataUntilStop Write Data Until Stop.

kMMC_SendTuningBlock MMC sending tuning block.
kMMC_ProgramCid Program CID.
kMMC_EraseGroupStart Erase Group Start.
kMMC_EraseGroupEnd Erase Group End.
kMMC_FastInputOutput Fast IO.
kMMC_GoInterruptState Go interrupt State.

68.4.41 enum mmc_classified_voltage_t

Enumerator

kMMC_ClassifiedVoltageHigh High-voltage MMC card.
kMMC_ClassifiedVoltageDual Dual-voltage MMC card.

68.4.42 enum mmc_classified_density_t

Enumerator

kMMC_ClassifiedDensityWithin2GB Density byte is less than or equal 2GB.

68.4.43 enum mmc_access_mode_t

Enumerator

kMMC_AccessModeByte The card should be accessed as byte.
kMMC_AccessModeSector The card should be accessed as sector.

68.4.44 enum mmc_voltage_window_t

Enumerator

kMMC_VoltageWindowNone voltage window is not define by user
kMMC_VoltageWindow120 Voltage window is 1.20V.
kMMC_VoltageWindow170to195 Voltage window is 1.70V to 1.95V.
kMMC_VoltageWindows270to360 Voltage window is 2.70V to 3.60V.

68.4.45 enum mmc_csd_structure_version_t

Enumerator

kMMC_CsdStrucureVersion10 CSD version No. 1.0
kMMC_CsdStrucureVersion11 CSD version No. 1.1
kMMC_CsdStrucureVersion12 CSD version No. 1.2
kMMC_CsdStrucureVersionInExtcsd Version coded in Extended CSD.

68.4.46 enum mmc_specification_version_t

Enumerator

kMMC_SpecificationVersion0 Allocated by MMCA.
kMMC_SpecificationVersion1 Allocated by MMCA.
kMMC_SpecificationVersion2 Allocated by MMCA.
kMMC_SpecificationVersion3 Allocated by MMCA.
kMMC_SpecificationVersion4 Version 4.1/4.2/4.3/4.41-4.5-4.51-5.0.

68.4.47 anonymous enum

Enumerator

kMMC_ExtendedCsdRevision10 Revision 1.0.
kMMC_ExtendedCsdRevision11 Revision 1.1.
kMMC_ExtendedCsdRevision12 Revision 1.2.
kMMC_ExtendedCsdRevision13 Revision 1.3 MMC4.3.
kMMC_ExtendedCsdRevision14 Revision 1.4 obsolete.
kMMC_ExtendedCsdRevision15 Revision 1.5 MMC4.41.
kMMC_ExtendedCsdRevision16 Revision 1.6 MMC4.5.
kMMC_ExtendedCsdRevision17 Revision 1.7 MMC5.0.

68.4.48 enum mmc_command_set_t

Enumerator

kMMC_CommandSetStandard Standard MMC.
kMMC_CommandSet1 Command set 1.
kMMC_CommandSet2 Command set 2.
kMMC_CommandSet3 Command set 3.
kMMC_CommandSet4 Command set 4.

68.4.49 anonymous enum

Enumerator

kMMC_SupportAlternateBoot support alternative boot mode
kMMC_SupportDDRBoot support DDR boot mode
kMMC_SupportHighSpeedBoot support high speed boot mode

68.4.50 enum mmc_high_speed_timing_t

Enumerator

kMMC_HighSpeedTimingNone MMC card using none high-speed timing.
kMMC_HighSpeedTiming MMC card using high-speed timing.
kMMC_HighSpeed200Timing MMC card high speed 200 timing.
kMMC_HighSpeed400Timing MMC card high speed 400 timing.
kMMC_EnhanceHighSpeed400Timing MMC card high speed 400 timing.

68.4.51 enum mmc_data_bus_width_t

Enumerator

kMMC_DataBusWidth1bit MMC data bus width is 1 bit.
kMMC_DataBusWidth4bit MMC data bus width is 4 bits.
kMMC_DataBusWidth8bit MMC data bus width is 8 bits.
kMMC_DataBusWidth4bitDDR MMC data bus width is 4 bits ddr.
kMMC_DataBusWidth8bitDDR MMC data bus width is 8 bits ddr.
kMMC_DataBusWidth8bitDDRSTROBE MMC data bus width is 8 bits ddr strobe mode.

68.4.52 enum mmc_boot_partition_enable_t

Enumerator

kMMC_BootPartitionEnableNot Device not boot enabled (default)
kMMC_BootPartitionEnablePartition1 Boot partition 1 enabled for boot.
kMMC_BootPartitionEnablePartition2 Boot partition 2 enabled for boot.
kMMC_BootPartitionEnableUserAera User area enabled for boot.

68.4.53 enum mmc_boot_timing_mode_t

Enumerator

kMMC_BootModeSDRWithDefaultTiming boot mode single data rate with backward compatiable timings
kMMC_BootModeSDRWithHighSpeedTiming boot mode single data rate with high speed timing
kMMC_BootModeDDRTiming boot mode dual date rate

68.4.54 enum mmc_boot_partition_wp_t

Enumerator

kMMC_BootPartitionWPDisable boot partition write protection disable
kMMC_BootPartitionPwrWPToBothPartition power on period write protection apply to both boot partitions
kMMC_BootPartitionPermWPToBothPartition permanent write protection apply to both boot partitions
kMMC_BootPartitionPwrWPToPartition1 power on period write protection apply to partition1
kMMC_BootPartitionPwrWPToPartition2 power on period write protection apply to partition2
kMMC_BootPartitionPermWPToPartition1 permanent write protection apply to partition1
kMMC_BootPartitionPermWPToPartition2 permanent write protection apply to partition2
kMMC_BootPartitionPermWPToPartition1PwrWPToPartition2 permanent write protection apply to partition1, power on period write protection apply to partition2
kMMC_BootPartitionPermWPToPartition2PwrWPToPartition1 permanent write protection apply to partition2, power on period write protection apply to partition1

68.4.55 anonymous enum

Enumerator

kMMC_BootPartitionNotProtected boot partition not protected
kMMC_BootPartitionPwrProtected boot partition is power on period write protected
kMMC_BootPartitionPermProtected boot partition is permanently protected

68.4.56 enum mmc_access_partition_t

Enumerator

kMMC_AccessPartitionUserAera No access to boot partition (default), normal partition.
kMMC_AccessPartitionBoot1 Read/Write boot partition 1.
kMMC_AccessPartitionBoot2 Read/Write boot partition 2.

kMMC_AccessRPMB Replay protected mem block.
kMMC_AccessGeneralPurposePartition1 access to general purpose partition 1
kMMC_AccessGeneralPurposePartition2 access to general purpose partition 2
kMMC_AccessGeneralPurposePartition3 access to general purpose partition 3
kMMC_AccessGeneralPurposePartition4 access to general purpose partition 4

68.4.57 anonymous enum

Enumerator

kMMC_CsdReadBlockPartialFlag Partial blocks for read allowed.
kMMC_CsdWriteBlockMisalignFlag Write block misalignment.
kMMC_CsdReadBlockMisalignFlag Read block misalignment.
kMMC_CsdDsrImplementedFlag DSR implemented.
kMMC_CsdWriteProtectGroupEnabledFlag Write protect group enabled.
kMMC_CsdWriteBlockPartialFlag Partial blocks for write allowed.
kMMC_ContentProtectApplicationFlag Content protect application.
kMMC_CsdFileFormatGroupFlag File format group.
kMMC_CsdCopyFlag Copy flag.
kMMC_CsdPermanentWriteProtectFlag Permanent write protection.
kMMC_CsdTemporaryWriteProtectFlag Temporary write protection.

68.4.58 enum mmc_extended_csd_access_mode_t

Enumerator

kMMC_ExtendedCsdAccessModeCommandSet Command set related setting.
kMMC_ExtendedCsdAccessModeSetBits Set bits in specific byte in Extended CSD.
kMMC_ExtendedCsdAccessModeClearBits Clear bits in specific byte in Extended CSD.
kMMC_ExtendedCsdAccessModeWriteBits Write a value to specific byte in Extended CSD.

68.4.59 enum mmc_extended_csd_index_t

Enumerator

kMMC_ExtendedCsdIndexFlushCache flush cache
kMMC_ExtendedCsdIndexCacheControl cache control
kMMC_ExtendedCsdIndexBootPartitionWP Boot partition write protect.
kMMC_ExtendedCsdIndexEraseGroupDefinition Erase Group Def.
kMMC_ExtendedCsdIndexBootBusConditions Boot Bus conditions.
kMMC_ExtendedCsdIndexBootConfigWP Boot config write protect.
kMMC_ExtendedCsdIndexPartitionConfig Partition Config, before BOOT_CONFIG.

kMMC_ExtendedCsdIndexBusWidth Bus Width.
kMMC_ExtendedCsdIndexHighSpeedTiming High-speed Timing.
kMMC_ExtendedCsdIndexPowerClass Power Class.
kMMC_ExtendedCsdIndexCommandSet Command Set.

68.4.60 anonymous enum

Enumerator

kMMC_DriverStrength0 Driver type0 ,nominal impedance 50ohm.
kMMC_DriverStrength1 Driver type1 ,nominal impedance 33ohm.
kMMC_DriverStrength2 Driver type2 ,nominal impedance 66ohm.
kMMC_DriverStrength3 Driver type3 ,nominal impedance 100ohm.
kMMC_DriverStrength4 Driver type4 ,nominal impedance 40ohm.

68.4.61 enum mmc_extended_csd_flags_t

Enumerator

kMMC_ExtCsdExtPartitionSupport partitioning support[160]
kMMC_ExtCsdEnhancePartitionSupport partitioning support[160]
kMMC_ExtCsdPartitioningSupport partitioning support[160]
kMMC_ExtCsdPrgCIDCSDInDDRModeSupport CMD26 and CMD27 are support dual data rate [130].
kMMC_ExtCsdBKOpsSupport background operation feature support [502]
kMMC_ExtCsdDataTagSupport data tag support[499]
kMMC_ExtCsdModeOperationCodeSupport mode operation code support[493]

68.4.62 enum mmc_boot_mode_t

Enumerator

kMMC_BootModeNormal Normal boot.
kMMC_BootModeAlternative Alternative boot.

Function Documentation

68.5.1 status_t SDMMC_SelectCard (sdmmchost_t ∗ *host,* uint32_t *relativeAddress,* bool *isSelected*)

Parameters

host	host handler.
relativeAddress	Relative address.
isSelected	True to put card into transfer state.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

68.5.2 status_t SDMMC_SendApplicationCommand (sdmmchost_t ∗ *host,* uint32_t *relativeAddress*)

Parameters

host	host handler.
relativeAddress	Card relative address.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_SDMMC_Card-NotSupport	Card doesn't support.
kStatus_Success	Operate successfully.

68.5.3 status_t SDMMC_SetBlockCount (sdmmchost_t ∗ *host,* uint32_t *blockCount*)

Parameters

host	host handler.

blockCount	Block count.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

68.5.4 status_t SDMMC_GoIdle (sdmmchost_t * *host*)

Parameters

host	host handler.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

68.5.5 status_t SDMMC_SetBlockSize (sdmmchost_t * *host,* uint32_t *blockSize*)

Parameters

host	host handler.
blockSize	Block size.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

68.5.6 status_t SDMMC_SetCardInactive (sdmmchost_t * *host*)

Parameters

host	host handler.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

Chapter 69
SDMMCHOST_SDHC

Overview

Data Structures

- struct sdmmchost_pwr_card_t
  card power control More...
- struct sdmmchost_t
  sdmmc host handler More...

Macros

- #define FSL_SDMMC_HOST_ADAPTER_VERSION (MAKE_VERSION(2U, 3U, 0U)) /*2.3.-0*/
  Middleware adapter version.
- #define SDMMCHOST_SUPPORT_HIGH_SPEED (1U)
  host capability
- #define SDMMCHOST_INSTANCE_SUPPORT_8_BIT_WIDTH(host) 1U
  sdmmc host instance capability
- #define SDMMCHOST_DMA_DESCRIPTOR_BUFFER_ALIGN_SIZE (4U)
  SDMMC host dma descriptor buffer address align size.

Typedefs

- typedef sdhc_transfer_t sdmmchost_transfer_t
  sdmmc host transfer function
- typedef void(* sdmmchost_pwr_t)(void)
  card power control function pointer

Enumerations

- enum _sdmmchost_endian_mode {
  kSDMMCHOST_EndianModeBig = 0U,
  kSDMMCHOST_EndianModeHalfWordBig = 1U,
  kSDMMCHOST_EndianModeLittle = 2U }
  host Endian mode corresponding to driver define

SDHC host controller function

- void SDMMCHOST_SetCardBusWidth (sdmmchost_t *host, uint32_t dataBusWidth)
  set data bus width.
- static void SDMMCHOST_SendCardActive (sdmmchost_t *host)
  Send initilization active 80 clocks to card.
- static uint32_t SDMMCHOST_SetCardClock (sdmmchost_t *host, uint32_t targetClock)

Set card bus clock.
- static bool SDMMCHOST_IsCardBusy (sdmmchost_t *host)
  check card status by DATA0.
- status_t SDMMCHOST_StartBoot (sdmmchost_t *host, sdmmchost_boot_config_t *hostConfig, sdmmchost_cmd_t *cmd, uint8_t *buffer)
  start read boot data.
- status_t SDMMCHOST_ReadBootData (sdmmchost_t *host, sdmmchost_boot_config_t *hostConfig, uint8_t *buffer)
  read boot data.
- static void SDMMCHOST_EnableBoot (sdmmchost_t *host, bool enable)
  enable boot mode.
- static void SDMMCHOST_EnableCardInt (sdmmchost_t *host, bool enable)
  enable card interrupt.
- status_t SDMMCHOST_CardIntInit (sdmmchost_t *host, void *sdioInt)
  card interrupt function.
- status_t SDMMCHOST_CardDetectInit (sdmmchost_t *host, void *cd)
  card detect init function.
- status_t SDMMCHOST_PollingCardDetectStatus (sdmmchost_t *host, uint32_t waitCardStatus, uint32_t timeout)
  Detect card insert, only need for SD cases.
- uint32_t SDMMCHOST_CardDetectStatus (sdmmchost_t *host)
  card detect status.
- status_t SDMMCHOST_Init (sdmmchost_t *host)
  Init host controller.
- void SDMMCHOST_Deinit (sdmmchost_t *host)
  Deinit host controller.
- void SDMMCHOST_SetCardPower (sdmmchost_t *host, bool enable)
  host power off card function.
- status_t SDMMCHOST_TransferFunction (sdmmchost_t *host, sdmmchost_transfer_t *content)
  host transfer function.
- void SDMMCHOST_Reset (sdmmchost_t *host)
  host reset function.
- status_t SDMMCHOST_WaitCardDetectStatus (SDMMCHOST_TYPE *hostBase, const sdmmchost_detect_card_t *cd, bool waitCardStatus)
  wait card detect status
- void SDMMCHOST_PowerOffCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)
  host power off card function.
- void SDMMCHOST_PowerOnCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)
  host power on card function.

Data Structure Documentation

69.2.1 struct sdmmchost_pwr_card_t

Deprecated Do not use this structure anymore.

Data Fields

- sdmmchost_pwr_t powerOn
  power on function pointer
- uint32_t powerOnDelay_ms
  power on delay
- sdmmchost_pwr_t powerOff
  power off function pointer
- uint32_t powerOffDelay_ms
  power off delay

69.2.2 struct sdmmchost_t

Data Fields

- sdhc_host_t hostController
  host configuration
- void * dmaDesBuffer
  DMA descriptor buffer address.
- uint32_t dmaDesBufferWordsNum
  DMA descriptor buffer size in byte.
- sdhc_handle_t handle
  host controller handler
- sdmmc_osa_event_t hostEvent
  host event handler
- void * cd
  card detect
- void * cardInt
  call back function for card interrupt

Macro Definition Documentation

69.3.1 #define FSL_SDMMC_HOST_ADAPTER_VERSION (MAKE_VERSION(2U, 3U, 0U)) /*2.3.0*/

Enumeration Type Documentation

69.4.1 enum _sdmmchost_endian_mode

Enumerator

kSDMMCHOST_EndianModeBig Big endian mode.
kSDMMCHOST_EndianModeHalfWordBig Half word big endian mode.
kSDMMCHOST_EndianModeLittle Little endian mode.

Function Documentation

69.5.1 void SDMMCHOST_SetCardBusWidth (sdmmchost_t * *host,* uint32_t *dataBusWidth*)

Parameters

host	host handler
dataBusWidth	data bus width

69.5.2 static void SDMMCHOST_SendCardActive (sdmmchost_t ∗ *host*) [inline], [static]

Parameters

host	host handler

69.5.3 static uint32_t SDMMCHOST_SetCardClock (sdmmchost_t ∗ *host,* uint32_t *targetClock*) [inline], [static]

Parameters

host	host handler
targetClock	target clock frequency

Return values

actual	clock frequency can be reach.

69.5.4 static bool SDMMCHOST_IsCardBusy (sdmmchost_t ∗ *host*) [inline], [static]

Parameters

host	host handler

Return values

true	is busy, false is idle.

69.5.5 status_t SDMMCHOST_StartBoot (sdmmchost_t * *host,* sdmmchost_boot_config_t * *hostConfig,* sdmmchost_cmd_t * *cmd,* uint8_t * *buffer*)

Parameters

host	host handler
hostConfig	boot configuration
cmd	boot command
buffer	buffer address

69.5.6 status_t SDMMCHOST_ReadBootData (sdmmchost_t * *host,* sdmmchost_boot_config_t * *hostConfig,* uint8_t * *buffer*)

Parameters

host	host handler
hostConfig	boot configuration
buffer	buffer address

69.5.7 static void SDMMCHOST_EnableBoot (sdmmchost_t * *host,* bool *enable*) [inline], [static]

Parameters

host	host handler
enable	true is enable, false is disable

69.5.8 static void SDMMCHOST_EnableCardInt (sdmmchost_t * *host,* bool *enable*) [inline], [static]

Parameters

host	host handler
enable	true is enable, false is disable.

69.5.9 status_t SDMMCHOST_CardIntInit (sdmmchost_t * *host*, void * *sdioInt*)

Parameters

host	host handler
sdioInt	card interrupt configuration

69.5.10 status_t SDMMCHOST_CardDetectInit (sdmmchost_t * *host*, void * *cd*)

Parameters

host	host handler
cd	card detect configuration

69.5.11 status_t SDMMCHOST_PollingCardDetectStatus (sdmmchost_t * *host*, uint32_t *waitCardStatus*, uint32_t *timeout*)

Parameters

host	host handler
waitCardStatus	status which user want to wait
timeout	wait time out.

Return values

kStatus_Success	detect card insert
kStatus_Fail	card insert event fail

69.5.12 uint32_t SDMMCHOST_CardDetectStatus (sdmmchost_t * *host*)

Parameters

host	host handler

Return values

kSD_Inserted,kSD_-Removed	

69.5.13 status_t SDMMCHOST_Init (sdmmchost_t * *host*)

Parameters

host	host handler

Return values

kStatus_Success	host init success
kStatus_Fail	event fail

69.5.14 void SDMMCHOST_Deinit (sdmmchost_t * *host*)

Parameters

host	host handler

69.5.15 void SDMMCHOST_SetCardPower (sdmmchost_t * *host,* bool *enable*)

Parameters

host	host handler
enable	true is power on, false is power down.

69.5.16 status_t SDMMCHOST_TransferFunction (sdmmchost_t * *host,* sdmmchost_transfer_t * *content*)

Parameters

host	host handler
content	transfer content.

69.5.17 void SDMMCHOST_Reset (sdmmchost_t * *host*)

Deprecated Do not use this function. Application should not call this function, driver is responsible for the host reset..

Parameters

host	host handler

69.5.18 status_t SDMMCHOST_WaitCardDetectStatus (SDMMCHOST_TYPE * *hostBase*, const sdmmchost_detect_card_t * *cd*, bool *waitCardStatus*)

Deprecated Do not use this function.It has been superceded by SDMMCHOST_PollingCardDetect-Status..

Parameters

hostBase	host handler
cd	card detect configuration.
waitCardStatus	status to wait.

69.5.19 void SDMMCHOST_PowerOffCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

Deprecated Do not use this function.It has been superceded by SDMMCHOST_SetCardPower..

Parameters

base	host base address.
pwr	depend on user define power configuration.

69.5.20 void SDMMCHOST_PowerOnCard (SDMMCHOST_TYPE ∗ *base*, const sdmmchost_pwr_card_t ∗ *pwr*)

Deprecated Do not use this function.It has been superceded by SDMMCHOST_SetCardPower..

Parameters

base	host base address.
pwr	depend on user define power configuration.

Chapter 70
MMCCARD

Overview

Data Structures

- struct mmc_usr_param_t
  card user parameter More...
- struct mmc_card_t
  mmc card state More...

Macros

- #define FSL_MMC_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 1U)) /*2.4.1*/
  Middleware mmc version.

Typedefs

- typedef void(* mmc_io_strength_t)(uint32_t busFreq)
  card io strength control

Enumerations

- enum {
  kMMC_SupportHighSpeed26MHZFlag = (1U << 0U),
  kMMC_SupportHighSpeed52MHZFlag = (1U << 1U),
  kMMC_SupportHighSpeedDDR52MHZ180V300VFlag = (1 << 2U),
  kMMC_SupportHighSpeedDDR52MHZ120VFlag = (1 << 3U),
  kMMC_SupportHS200200MHZ180VFlag = (1 << 4U),
  kMMC_SupportHS200200MHZ120VFlag = (1 << 5U),
  kMMC_SupportHS400DDR200MHZ180VFlag = (1 << 6U),
  kMMC_SupportHS400DDR200MHZ120VFlag = (1 << 7U),
  kMMC_SupportHighCapacityFlag = (1U << 8U),
  kMMC_SupportAlternateBootFlag = (1U << 9U),
  kMMC_SupportDDRBootFlag = (1U << 10U),
  kMMC_SupportHighSpeedBootFlag = (1U << 11U),
  kMMC_SupportEnhanceHS400StrobeFlag = (1U << 12U) }
  MMC card flags.

MMCCARD Function

- status_t MMC_Init (mmc_card_t *card)
  Initializes the MMC card and host.
- void MMC_Deinit (mmc_card_t *card)

Deinitializes the card and host.
- status_t MMC_CardInit (mmc_card_t ∗card)
  intialize the card.
- void MMC_CardDeinit (mmc_card_t ∗card)
  Deinitializes the card.
- status_t MMC_HostInit (mmc_card_t ∗card)
  initialize the host.
- void MMC_HostDeinit (mmc_card_t ∗card)
  Deinitializes the host.
- void MMC_HostReset (SDMMCHOST_CONFIG ∗host)
  reset the host.
- void MMC_PowerOnCard (SDMMCHOST_TYPE ∗base, const sdmmchost_pwr_card_t ∗pwr)
  power on card.
- void MMC_PowerOffCard (SDMMCHOST_TYPE ∗base, const sdmmchost_pwr_card_t ∗pwr)
  power off card.
- void MMC_SetCardPower (mmc_card_t ∗card, bool enable)
  set card power.
- bool MMC_CheckReadOnly (mmc_card_t ∗card)
  Checks if the card is read-only.
- status_t MMC_ReadBlocks (mmc_card_t ∗card, uint8_t ∗buffer, uint32_t startBlock, uint32_t blockCount)
  Reads data blocks from the card.
- status_t MMC_WriteBlocks (mmc_card_t ∗card, const uint8_t ∗buffer, uint32_t startBlock, uint32_t blockCount)
  Writes data blocks to the card.
- status_t MMC_EraseGroups (mmc_card_t ∗card, uint32_t startGroup, uint32_t endGroup)
  Erases groups of the card.
- status_t MMC_SelectPartition (mmc_card_t ∗card, mmc_access_partition_t partitionNumber)
  Selects the partition to access.
- status_t MMC_SetBootConfig (mmc_card_t ∗card, const mmc_boot_config_t ∗config)
  Configures the boot activity of the card.
- status_t MMC_StartBoot (mmc_card_t ∗card, const mmc_boot_config_t ∗mmcConfig, uint8_t ∗buffer, sdmmchost_boot_config_t ∗hostConfig)
  MMC card start boot.
- status_t MMC_SetBootConfigWP (mmc_card_t ∗card, uint8_t wp)
  MMC card set boot configuration write protect.
- status_t MMC_ReadBootData (mmc_card_t ∗card, uint8_t ∗buffer, sdmmchost_boot_config_t ∗hostConfig)
  MMC card continous read boot data.
- status_t MMC_StopBoot (mmc_card_t ∗card, uint32_t bootMode)
  MMC card stop boot mode.
- status_t MMC_SetBootPartitionWP (mmc_card_t ∗card, mmc_boot_partition_wp_t bootPartitionWP)
  MMC card set boot partition write protect.
- status_t MMC_EnableCacheControl (mmc_card_t ∗card, bool enable)
  MMC card cache control function.
- status_t MMC_FlushCache (mmc_card_t ∗card)
  MMC card cache flush function.

Data Structure Documentation

70.2.1 struct mmc_usr_param_t

Data Fields

- mmc_io_strength_t ioStrength
  swicth sd io strength
- uint32_t maxFreq
  board support maximum frequency
- uint32_t capability
  board capability flag

70.2.2 struct mmc_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- sdmmchost_t ∗ host
  Host information.
- mmc_usr_param_t usrParam
  user parameter
- bool isHostReady
  Use this flag to indicate if need host re-init or not.
- bool noInteralAlign
  use this flag to disable sdmmc align.
- uint32_t busClock_Hz
  MMC bus clock united in Hz.
- uint32_t relativeAddress
  Relative address of the card.
- bool enablePreDefinedBlockCount
  Enable PRE-DEFINED block count when read/write.
- uint32_t flags
  Capability flag in _mmc_card_flag.
- uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]
  raw buffer used for mmc driver internal
- uint32_t ocr
  Raw OCR content.
- mmc_cid_t cid
  CID.
- mmc_csd_t csd
  CSD.
- mmc_extended_csd_t extendedCsd
  Extended CSD.
- uint32_t blockSize
  Card block size.
- uint32_t userPartitionBlocks

Card total block number in user partition.
- uint32_t bootPartitionBlocks
  Boot partition size united as block size.
- uint32_t eraseGroupBlocks
  Erase group size united as block size.
- mmc_access_partition_t currentPartition
  Current access partition.
- mmc_voltage_window_t hostVoltageWindowVCCQ
  application must set this value according to board specific
- mmc_voltage_window_t hostVoltageWindowVCC
  application must set this value according to board specific
- mmc_high_speed_timing_t busTiming
  indicate the current work timing mode
- mmc_data_bus_width_t busWidth
  indicate the current work bus width

70.2.2.0.0.60 Field Documentation

70.2.2.0.0.60.1 bool mmc_card_t::noInteralAlign

If disable, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are align to low level driver

Macro Definition Documentation

70.3.1 #define FSL_MMC_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 1U)) /*2.4.1*/

Enumeration Type Documentation

70.4.1 anonymous enum

Enumerator

kMMC_SupportHighSpeed26MHZFlag Support high speed 26MHZ.
kMMC_SupportHighSpeed52MHZFlag Support high speed 52MHZ.
kMMC_SupportHighSpeedDDR52MHZ180V300VFlag ddr 52MHZ 1.8V or 3.0V
kMMC_SupportHighSpeedDDR52MHZ120VFlag DDR 52MHZ 1.2V.
kMMC_SupportHS200200MHZ180VFlag HS200 ,200MHZ,1.8V.
kMMC_SupportHS200200MHZ120VFlag HS200, 200MHZ, 1.2V.
kMMC_SupportHS400DDR200MHZ180VFlag HS400, DDR, 200MHZ,1.8V.
kMMC_SupportHS400DDR200MHZ120VFlag HS400, DDR, 200MHZ,1.2V.
kMMC_SupportHighCapacityFlag Support high capacity.
kMMC_SupportAlternateBootFlag Support alternate boot.
kMMC_SupportDDRBootFlag support DDR boot flag
kMMC_SupportHighSpeedBootFlag support high speed boot flag
kMMC_SupportEnhanceHS400StrobeFlag support enhance HS400 strobe

Function Documentation

70.5.1 status_t MMC_Init (mmc_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_HostNotReady	host is not ready.
kStatus_SDMMC_GoIdleFailed	Go idle failed.
kStatus_SDMMC_SendOperationConditionFailed	Send operation condition failed.
kStatus_SDMMC_AllSendCidFailed	Send CID failed.
kStatus_SDMMC_SetRelativeAddressFailed	Set relative address failed.
kStatus_SDMMC_SendCsdFailed	Send CSD failed.
kStatus_SDMMC_CardNotSupport	Card not support.
kStatus_SDMMC_SelectCardFailed	Send SELECT_CARD command failed.
kStatus_SDMMC_SendExtendedCsdFailed	Send EXT_CSD failed.
kStatus_SDMMC_SetBusWidthFailed	Set bus width failed.
kStatus_SDMMC_SwitchHighSpeedFailed	Switch high speed failed.
kStatus_SDMMC_SetCardBlockSizeFailed	Set card block size failed.
kStatus_Success	Operate successfully.

70.5.2 void MMC_Deinit (mmc_card_t * *card*)

Parameters

card	Card descriptor.

70.5.3 status_t MMC_CardInit (mmc_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_Host-NotReady	host is not ready.
kStatus_SDMMC_Go-IdleFailed	Go idle failed.
kStatus_SDMMC_Send-OperationCondition-Failed	Send operation condition failed.
kStatus_SDMMC_All-SendCidFailed	Send CID failed.
kStatus_SDMMC_Set-RelativeAddressFailed	Set relative address failed.
kStatus_SDMMC_Send-CsdFailed	Send CSD failed.
kStatus_SDMMC_Card-NotSupport	Card not support.
kStatus_SDMMC_Select-CardFailed	Send SELECT_CARD command failed.
kStatus_SDMMC_Send-ExtendedCsdFailed	Send EXT_CSD failed.
kStatus_SDMMC_SetBus-WidthFailed	Set bus width failed.

kStatus_SDMMC_Switch-HighSpeedFailed	Switch high speed failed.
kStatus_SDMMC_Set-CardBlockSizeFailed	Set card block size failed.
kStatus_Success	Operate successfully.

70.5.4 void MMC_CardDeinit (mmc_card_t ∗ *card*)

Parameters

card	Card descriptor.

70.5.5 status_t MMC_HostInit (mmc_card_t ∗ *card*)

This function deinitializes the specific host.

Parameters

card	Card descriptor.

70.5.6 void MMC_HostDeinit (mmc_card_t ∗ *card*)

This function deinitializes the host.

Parameters

card	Card descriptor.

70.5.7 void MMC_HostReset (SDMMCHOST_CONFIG ∗ *host*)

This function reset the specific host.

Parameters

host	host descriptor.

70.5.8 void MMC_PowerOnCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

The power on operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by MMC_SetCardPower.

Parameters

base	host base address.
pwr	user define power control configuration

70.5.9 void MMC_PowerOffCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

The power off operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by MMC_SetCardPower.

Parameters

base	host base address.
pwr	user define power control configuration

70.5.10 void MMC_SetCardPower (mmc_card_t * *card*, bool *enable*)

Parameters

card	card descriptor.

enable	true is power on, false is power off.

70.5.11 bool MMC_CheckReadOnly (mmc_card_t * *card*)

Parameters

card	Card descriptor.

Return values

true	Card is read only.
false	Card isn't read only.

70.5.12 status_t MMC_ReadBlocks (mmc_card_t * *card,* uint8_t * *buffer,* uint32_t *startBlock,* uint32_t *blockCount*)

Parameters

card	Card descriptor.
buffer	The buffer to save data.
startBlock	The start block index.
blockCount	The number of blocks to read.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_Card-NotSupport	Card not support.
kStatus_SDMMC_Set-BlockCountFailed	Set block count failed.
kStatus_SDMMC_-TransferFailed	Transfer failed.

kStatus_SDMMC_Stop-TransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

70.5.13 status_t MMC_WriteBlocks (mmc_card_t ∗ *card,* const uint8_t ∗ *buffer,* uint32_t *startBlock,* uint32_t *blockCount*)

Parameters

card	Card descriptor.
buffer	The buffer to save data blocks.
startBlock	Start block number to write.
blockCount	Block count.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_Not-SupportYet	Not support now.
kStatus_SDMMC_Set-BlockCountFailed	Set block count failed.
kStatus_SDMMC_Wait-WriteCompleteFailed	Send status failed.
kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_SDMMC_Stop-TransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

70.5.14 status_t MMC_EraseGroups (mmc_card_t ∗ *card,* uint32_t *startGroup,* uint32_t *endGroup*)

Erase group is the smallest erase unit in MMC card. The erase range is [startGroup, endGroup].

Parameters

card	Card descriptor.
startGroup	Start group number.
endGroup	End group number.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_Wait-WriteCompleteFailed	Send status failed.
kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

70.5.15 status_t MMC_SelectPartition (mmc_card_t ∗ *card*, mmc_access_partition_t *partitionNumber*)

Parameters

card	Card descriptor.
partition-Number	The partition number.

Return values

kStatus_SDMMC_-ConfigureExtendedCsd-Failed	Configure EXT_CSD failed.
kStatus_Success	Operate successfully.

70.5.16 status_t MMC_SetBootConfig (mmc_card_t ∗ *card*, const mmc_boot_config_t ∗ *config*)

Parameters

card	Card descriptor.
config	Boot configuration structure.

Return values

kStatus_SDMMC_NotSupportYet	Not support now.
kStatus_SDMMC_ConfigureExtendedCsdFailed	Configure EXT_CSD failed.
kStatus_SDMMC_ConfigureBootFailed	Configure boot failed.
kStatus_Success	Operate successfully.

70.5.17 status_t MMC_StartBoot (mmc_card_t * *card*, const mmc_boot_config_t * *mmcConfig*, uint8_t * *buffer*, sdmmchost_boot_config_t * *hostConfig*)

Parameters

card	Card descriptor.
mmcConfig	mmc Boot configuration structure.
buffer	address to recieve data.
hostConfig	host boot configurations.

Return values

kStatus_Fail	fail.
kStatus_SDMMC_TransferFailed	transfer fail.
kStatus_SDMMC_GoIdleFailed	reset card fail.

kStatus_Success	Operate successfully.

70.5.18 status_t MMC_SetBootConfigWP (mmc_card_t ∗ *card,* uint8_t *wp*)

Parameters

card	Card descriptor.
wp	write protect value.

70.5.19 status_t MMC_ReadBootData (mmc_card_t ∗ *card,* uint8_t ∗ *buffer,* sdmmchost_boot_config_t ∗ *hostConfig*)

Parameters

card	Card descriptor.
buffer	buffer address.
hostConfig	host boot configurations.

70.5.20 status_t MMC_StopBoot (mmc_card_t ∗ *card,* uint32_t *bootMode*)

Parameters

card	Card descriptor.
bootMode	boot mode.

70.5.21 status_t MMC_SetBootPartitionWP (mmc_card_t ∗ *card,* mmc_boot_partition_wp_t *bootPartitionWP*)

Parameters

card	Card descriptor.
bootPartition-WP	boot partition write protect value.

70.5.22 status_t MMC_EnableCacheControl (mmc_card_t * *card,* bool *enable*)

The mmc device's cache is enabled by the driver by default. The cache should in typical case reduce the access time (compared to an access to the main nonvolatile storage) for both write and read.

Parameters

card	Card descriptor.
enable	true is enable the cache, false is disable the cache.

70.5.23 status_t MMC_FlushCache (mmc_card_t * *card*)

A Flush operation refers to the requirement, from the host to the device, to write the cached data to the nonvolatile memory. Prior to a flush, the device may autonomously write data to the nonvolatile memory, but after the flush operation all data in the volatile area must be written to nonvolatile memory There is no requirement for flush due to switching between the partitions. (Note: This also implies that the cache data shall not be lost when switching between partitions). Cached data may be lost in SLEEP state, so host should flush the cache before placing the device into SLEEP state.

Parameters

card	Card descriptor.

Chapter 71
SDMMC_OSA

Overview

Data Structures

- struct sdmmc_osa_event_t
  sdmmc osa event More...

Macros

- #define SDMMC_OSA_EVENT_TRANSFER_CMD_SUCCESS (1UL << 0U)
  transfer event
- #define SDMMC_OSA_EVENT_CARD_INSERTED (1UL << 8U)
  card detect event, start from index 8
- #define SDMMC_OSA_POLLING_EVENT_BY_SEMPHORE 1
  enable semphore by default

sdmmc osa Function

- void SDMMC_OSAInit (void)
  Initialize OSA.
- status_t SDMMC_OSAEventCreate (void ∗eventHandle)
  OSA Create event.
- status_t SDMMC_OSAEventWait (void ∗eventHandle, uint32_t eventType, uint32_t timeoutMilliseconds, uint32_t ∗event)
  Wait event.
- status_t SDMMC_OSAEventSet (void ∗eventHandle, uint32_t eventType)
  set event.
- status_t SDMMC_OSAEventGet (void ∗eventHandle, uint32_t eventType, uint32_t ∗flag)
  Get event flag.
- status_t SDMMC_OSAEventClear (void ∗eventHandle, uint32_t eventType)
  clear event flag.
- status_t SDMMC_OSAEventDestroy (void ∗eventHandle)
  Delete event.
- void SDMMC_OSADelay (uint32_t milliseconds)
  sdmmc delay.

Data Structure Documentation

71.2.1 struct sdmmc_osa_event_t

Function Documentation

71.3.1 status_t SDMMC_OSAEventCreate (void ∗ *eventHandle*)

Parameters

eventHandle	event handle.

Return values

kStatus_Fail	or kStatus_Success.

71.3.2 status_t SDMMC_OSAEventWait (void ∗ *eventHandle*, uint32_t *eventType*, uint32_t *timeoutMilliseconds*, uint32_t ∗ *event*)

Parameters

eventHandle	The event type
eventType	Timeout time in milliseconds.
timeout-Milliseconds	timeout value in ms.
event	event flags.

Return values

kStatus_Fail	or kStatus_Success.

71.3.3 status_t SDMMC_OSAEventSet (void ∗ *eventHandle*, uint32_t *eventType*)

Parameters

eventHandle	event handle.
eventType	The event type

Return values

kStatus_Fail	or kStatus_Success.

71.3.4 status_t SDMMC_OSAEventGet (void ∗ *eventHandle*, uint32_t *eventType*, uint32_t ∗ *flag*)

Parameters

eventHandle	event handle.
eventType	event type.
flag	pointer to store event value.

Return values

kStatus_Fail	or kStatus_Success.

71.3.5 status_t SDMMC_OSAEventClear (void * *eventHandle*, uint32_t *eventType*)

Parameters

eventHandle	event handle.
eventType	The event type

Return values

kStatus_Fail	or kStatus_Success.

71.3.6 status_t SDMMC_OSAEventDestroy (void * *eventHandle*)

Parameters

eventHandle	The event handle.

71.3.7 void SDMMC_OSADelay (uint32_t *milliseconds*)

Parameters

milliseconds	time to delay

Chapter 72
SDCARD

Overview

Data Structures

- struct sd_card_t
  SD card state. More...

Macros

- #define FSL_SD_DRIVER_VERSION (MAKE_VERSION(2U, 3U, 3U)) /*2.3.3*/
  Driver version.

Enumerations

- enum {
  kSD_SupportHighCapacityFlag = (1U << 1U),
  kSD_Support4BitWidthFlag = (1U << 2U),
  kSD_SupportSdhcFlag = (1U << 3U),
  kSD_SupportSdxcFlag = (1U << 4U),
  kSD_SupportVoltage180v = (1U << 5U),
  kSD_SupportSetBlockCountCmd = (1U << 6U),
  kSD_SupportSpeedClassControlCmd = (1U << 7U) }
  SD card flags.

SDCARD Function

- status_t SD_Init (sd_card_t *card)
  Initializes the card on a specific host controller.
- void SD_Deinit (sd_card_t *card)
  Deinitializes the card.
- status_t SD_CardInit (sd_card_t *card)
  Initializes the card.
- void SD_CardDeinit (sd_card_t *card)
  Deinitializes the card.
- status_t SD_HostInit (sd_card_t *card)
  initialize the host.
- void SD_HostDeinit (sd_card_t *card)
  Deinitializes the host.
- void SD_HostDoReset (sd_card_t *card)
  reset the host.
- void SD_HostReset (SDMMCHOST_CONFIG *host)
  reset the host.
- void SD_PowerOnCard (SDMMCHOST_TYPE *base, const sdmmchost_pwr_card_t *pwr)

power on card.
- void SD_PowerOffCard (SDMMCHOST_TYPE ∗base, const sdmmchost_pwr_card_t ∗pwr)
  power off card.
- void SD_SetCardPower (sd_card_t ∗card, bool enable)
  set card power.
- status_t SD_WaitCardDetectStatus (SDMMCHOST_TYPE ∗hostBase, const sdmmchost_detect_card_t ∗cd, bool waitCardStatus)
  sd wait card detect function.
- status_t SD_PollingCardInsert (sd_card_t ∗card, uint32_t status)
  sd wait card detect function.
- bool SD_IsCardPresent (sd_card_t ∗card)
  sd card present check function.
- bool SD_CheckReadOnly (sd_card_t ∗card)
  Checks whether the card is write-protected.
- status_t SD_SelectCard (sd_card_t ∗card, bool isSelected)
  Send SELECT_CARD command to set the card to be transfer state or not.
- status_t SD_ReadStatus (sd_card_t ∗card)
  Send ACMD13 to get the card current status.
- status_t SD_ReadBlocks (sd_card_t ∗card, uint8_t ∗buffer, uint32_t startBlock, uint32_t blockCount)
  Reads blocks from the specific card.
- status_t SD_WriteBlocks (sd_card_t ∗card, const uint8_t ∗buffer, uint32_t startBlock, uint32_t blockCount)
  Writes blocks of data to the specific card.
- status_t SD_EraseBlocks (sd_card_t ∗card, uint32_t startBlock, uint32_t blockCount)
  Erases blocks of the specific card.
- status_t SD_SetDriverStrength (sd_card_t ∗card, sd_driver_strength_t driverStrength)
  select card driver strength select card driver strength
- status_t SD_SetMaxCurrent (sd_card_t ∗card, sd_max_current_t maxCurrent)
  select max current select max operation current

Data Structure Documentation

72.2.1 struct sd_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- sdmmchost_t ∗ host
  Host configuration.
- sd_usr_param_t usrParam
  user parameter
- bool isHostReady
  use this flag to indicate if need host re-init or not
- bool noInteralAlign
  used to enable/disable the functionality of the exchange buffer
- uint32_t busClock_Hz
  SD bus clock frequency united in Hz.

- uint32_t relativeAddress
  Relative address of the card.
- uint32_t version
  Card version.
- uint32_t flags
  Flags in _sd_card_flag.
- uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]
  internal buffer
- uint32_t ocr
  Raw OCR content.
- sd_cid_t cid
  CID.
- sd_csd_t csd
  CSD.
- sd_scr_t scr
  SCR.
- sd_status_t stat
  sd 512 bit status
- uint32_t blockCount
  Card total block number.
- uint32_t blockSize
  Card block size.
- sd_timing_mode_t currentTiming
  current timing mode
- sd_driver_strength_t driverStrength
  driver strength
- sd_max_current_t maxCurrent
  card current limit
- sdmmc_operation_voltage_t operationVoltage
  card operation voltage

Macro Definition Documentation

72.3.1 #define FSL_SD_DRIVER_VERSION (MAKE_VERSION(2U, 3U, 3U)) /*2.3.3*/

Enumeration Type Documentation

72.4.1 anonymous enum

Enumerator

kSD_SupportHighCapacityFlag Support high capacity.
kSD_Support4BitWidthFlag Support 4-bit data width.
kSD_SupportSdhcFlag Card is SDHC.
kSD_SupportSdxcFlag Card is SDXC.
kSD_SupportVoltage180v card support 1.8v voltage
kSD_SupportSetBlockCountCmd card support cmd23 flag
kSD_SupportSpeedClassControlCmd card support speed class control flag

Function Documentation

72.5.1 status_t SD_Init (sd_card_t * *card*)

Deprecated Do not use this function. It has been superceded by SD_HostInit,SD_CardInit.

This function initializes the card on a specific host controller, it is consist of host init, card detect, card init function, however user can ignore this high level function, instead of use the low level function, such as SD_CardInit, SD_HostInit, SD_CardDetect.

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_HostNotReady	host is not ready.
kStatus_SDMMC_GoIdleFailed	Go idle failed.
kStatus_SDMMC_NotSupportYet	Card not support.
kStatus_SDMMC_SendOperationConditionFailed	Send operation condition failed.
kStatus_SDMMC_AllSendCidFailed	Send CID failed.
kStatus_SDMMC_SendRelativeAddressFailed	Send relative address failed.
kStatus_SDMMC_SendCsdFailed	Send CSD failed.
kStatus_SDMMC_SelectCardFailed	Send SELECT_CARD command failed.
kStatus_SDMMC_SendScrFailed	Send SCR failed.

kStatus_SDMMC_SetBus-WidthFailed	Set bus width failed.
kStatus_SDMMC_Switch-HighSpeedFailed	Switch high speed failed.
kStatus_SDMMC_Set-CardBlockSizeFailed	Set card block size failed.
kStatus_Success	Operate successfully.

72.5.2 void SD_Deinit (sd_card_t * *card*)

Deprecated Do not use this function. It has been superceded by SD_HostDeinit,SD_CardDeinit. This function deinitializes the specific card and host.

Parameters

card	Card descriptor.

72.5.3 status_t SD_CardInit (sd_card_t * *card*)

This function initializes the card only, make sure the host is ready when call this function, otherwise it will return kStatus_SDMMC_HostNotReady.

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_Host-NotReady	host is not ready.
kStatus_SDMMC_Go-IdleFailed	Go idle failed.
kStatus_SDMMC_Not-SupportYet	Card not support.

kStatus_SDMMC_SendOperationConditionFailed	Send operation condition failed.
kStatus_SDMMC_AllSendCidFailed	Send CID failed.
kStatus_SDMMC_SendRelativeAddressFailed	Send relative address failed.
kStatus_SDMMC_SendCsdFailed	Send CSD failed.
kStatus_SDMMC_SelectCardFailed	Send SELECT_CARD command failed.
kStatus_SDMMC_SendScrFailed	Send SCR failed.
kStatus_SDMMC_SetBusWidthFailed	Set bus width failed.
kStatus_SDMMC_SwitchHighSpeedFailed	Switch high speed failed.
kStatus_SDMMC_SetCardBlockSizeFailed	Set card block size failed.
kStatus_Success	Operate successfully.

72.5.4 void SD_CardDeinit (sd_card_t * *card*)

This function deinitializes the specific card.

Parameters

card	Card descriptor.

72.5.5 status_t SD_HostInit (sd_card_t * *card*)

This function deinitializes the specific host.

Parameters

card	Card descriptor.

72.5.6 void SD_HostDeinit (sd_card_t * *card*)

This function deinitializes the host.

Parameters

card	Card descriptor.

72.5.7 void SD_HostDoReset (sd_card_t * *card*)

This function reset the specific host.

Parameters

card	Card descriptor.

72.5.8 void SD_HostReset (SDMMCHOST_CONFIG * *host*)

This function reset the specific host.

Deprecated Do not use this function. It has been superceded by SD_HostDoReset.

Parameters

host	host descriptor.

72.5.9 void SD_PowerOnCard (SDMMCHOST_TYPE * *base,* const sdmmchost_pwr_card_t * *pwr*)

The power on operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by SD_SetCardPower.

Parameters

base	host base address.
pwr	user define power control configuration

72.5.10 void SD_PowerOffCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

The power off operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by SD_SetCardPower.

Parameters

base	host base address.
pwr	user define power control configuration

72.5.11 void SD_SetCardPower (sd_card_t * *card*, bool *enable*)

The power off operation depend on host or the user define power on function.

Parameters

card	card descriptor.
enable	true is power on, false is power off.

72.5.12 status_t SD_WaitCardDetectStatus (SDMMCHOST_TYPE * *hostBase*, const sdmmchost_detect_card_t * *cd*, bool *waitCardStatus*)

Detect card through GPIO, CD, DATA3.

Deprecated Do not use this function. It has been superceded by SD_PollingCardInsert.

Parameters

hostBase	host base address.
cd	card detect configuration
waitCardStatus	wait card detect status

72.5.13 status_t SD_PollingCardInsert (sd_card_t * *card*, uint32_t *status*)

Detect card through GPIO, CD, DATA3.

Parameters

card	card descriptor.
status	detect status, kSD_Inserted or kSD_Removed.

72.5.14 bool SD_IsCardPresent (sd_card_t * *card*)

Parameters

card	card descriptor.

72.5.15 bool SD_CheckReadOnly (sd_card_t * *card*)

This function checks if the card is write-protected via the CSD register.

Parameters

card	The specific card.

Return values

true	Card is read only.
false	Card isn't read only.

72.5.16 status_t SD_SelectCard (sd_card_t * *card*, bool *isSelected*)

Parameters

card	Card descriptor.
isSelected	True to set the card into transfer state, false to disselect.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

72.5.17 status_t SD_ReadStatus (sd_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_SDMMC_Send-ApplicationCommand-Failed	send application command failed.
kStatus_Success	Operate successfully.

72.5.18 status_t SD_ReadBlocks (sd_card_t * *card,* uint8_t * *buffer,* uint32_t *startBlock,* uint32_t *blockCount*)

This function reads blocks from the specific card with default block size defined by the SDHC_CARD_-DEFAULT_BLOCK_SIZE.

Parameters

card	Card descriptor.

buffer	The buffer to save the data read from card.
startBlock	The start block index.
blockCount	The number of blocks to read.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_Card-NotSupport	Card not support.
kStatus_SDMMC_Not-SupportYet	Not support now.
kStatus_SDMMC_Wait-WriteCompleteFailed	Send status failed.
kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_SDMMC_Stop-TransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

72.5.19 status_t SD_WriteBlocks (sd_card_t ∗ *card,* const uint8_t ∗ *buffer,* uint32_t *startBlock,* uint32_t *blockCount*)

This function writes blocks to the specific card with default block size 512 bytes.

Parameters

card	Card descriptor.
buffer	The buffer holding the data to be written to the card.
startBlock	The start block index.
blockCount	The number of blocks to write.

Return values

kStatus_InvalidArgument	Invalid argument.

kStatus_SDMMC_NotSupportYet	Not support now.
kStatus_SDMMC_CardNotSupport	Card not support.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_SDMMC_TransferFailed	Transfer failed.
kStatus_SDMMC_StopTransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

72.5.20 status_t SD_EraseBlocks (sd_card_t * *card*, uint32_t *startBlock*, uint32_t *blockCount*)

This function erases blocks of the specific card with default block size 512 bytes.

Parameters

card	Card descriptor.
startBlock	The start block index.
blockCount	The number of blocks to erase.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_SDMMC_TransferFailed	Transfer failed.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.

kStatus_Success	Operate successfully.

72.5.21 status_t SD_SetDriverStrength (sd_card_t * *card*, sd_driver_strength_t *driverStrength*)

Parameters

card	Card descriptor.
driverStrength	Driver strength

72.5.22 status_t SD_SetMaxCurrent (sd_card_t * *card*, sd_max_current_t *maxCurrent*)

Parameters

card	Card descriptor.
maxCurrent	Max current

Chapter 73
SDIOCARD

Overview

Data Structures

- struct sdio_card_t
  SDIO card state. More...

Macros

- #define FSL_SDIO_DRIVER_VERSION (MAKE_VERSION(2U, 3U, 2U)) /*2.3.2*/
  Middleware version.
- #define FSL_SDIO_MAX_IO_NUMS (7U)
  sdio device support maximum IO number

Typedefs

- typedef void(* sdio_io_irq_handler_t)(sdio_card_t *card, uint32_t func)
  sdio io handler

Enumerations

- enum sdio_io_direction_t {
  kSDIO_IORead = 0U,
  kSDIO_IOWrite = 1U }
  sdio io read/write direction

Initialization and deinitialization

- status_t SDIO_Init (sdio_card_t *card)
  SDIO card init function.
- void SDIO_Deinit (sdio_card_t *card)
  SDIO card deinit, include card and host deinit.
- status_t SDIO_CardInit (sdio_card_t *card)
  Initializes the card.
- void SDIO_CardDeinit (sdio_card_t *card)
  Deinitializes the card.
- status_t SDIO_HostInit (sdio_card_t *card)
  initialize the host.
- void SDIO_HostDeinit (sdio_card_t *card)
  Deinitializes the host.
- void SDIO_HostReset (SDMMCHOST_CONFIG *host)
  reset the host.
- void SDIO_HostDoReset (sdio_card_t *card)
  reset the host.

- void SDIO_PowerOnCard (SDMMCHOST_TYPE ∗base, const sdmmchost_pwr_card_t ∗pwr)
  power on card.
- void SDIO_PowerOffCard (SDMMCHOST_TYPE ∗base, const sdmmchost_pwr_card_t ∗pwr)
  power on card.
- void SDIO_SetCardPower (sdio_card_t ∗card, bool enable)
  set card power.
- status_t SDIO_CardInActive (sdio_card_t ∗card)
  set SDIO card to inactive state
- status_t SDIO_GetCardCapability (sdio_card_t ∗card, sdio_func_num_t func)
  get SDIO card capability
- status_t SDIO_SetBlockSize (sdio_card_t ∗card, sdio_func_num_t func, uint32_t blockSize)
  set SDIO card block size
- status_t SDIO_CardReset (sdio_card_t ∗card)
  set SDIO card reset
- status_t SDIO_SetDataBusWidth (sdio_card_t ∗card, sdio_bus_width_t busWidth)
  set SDIO card data bus width
- status_t SDIO_SwitchToHighSpeed (sdio_card_t ∗card)
  switch the card to high speed
- status_t SDIO_ReadCIS (sdio_card_t ∗card, sdio_func_num_t func, const uint32_t ∗tupleList, uint32_t tupleNum)
  read SDIO card CIS for each function
- status_t SDIO_WaitCardDetectStatus (SDMMCHOST_TYPE ∗hostBase, const sdmmchost_detect-_card_t ∗cd, bool waitCardStatus)
  sdio wait card detect function.
- status_t SDIO_PollingCardInsert (sdio_card_t ∗card, uint32_t status)
  sdio wait card detect function.
- bool SDIO_IsCardPresent (sdio_card_t ∗card)
  sdio card present check function.

IO operations

- status_t SDIO_IO_Write_Direct (sdio_card_t ∗card, sdio_func_num_t func, uint32_t regAddr, uint8_t ∗data, bool raw)
  IO direct write transfer function.
- status_t SDIO_IO_Read_Direct (sdio_card_t ∗card, sdio_func_num_t func, uint32_t regAddr, uint8_t ∗data)
  IO direct read transfer function.
- status_t SDIO_IO_RW_Direct (sdio_card_t ∗card, sdio_io_direction_t direction, sdio_func_num_t func, uint32_t regAddr, uint8_t dataIn, uint8_t ∗dataOut)
  IO direct read/write transfer function.
- status_t SDIO_IO_Write_Extended (sdio_card_t ∗card, sdio_func_num_t func, uint32_t regAddr, uint8_t ∗buffer, uint32_t count, uint32_t flags)
  IO extended write transfer function.
- status_t SDIO_IO_Read_Extended (sdio_card_t ∗card, sdio_func_num_t func, uint32_t regAddr, uint8_t ∗buffer, uint32_t count, uint32_t flags)
  IO extended read transfer function.
- status_t SDIO_EnableIOInterrupt (sdio_card_t ∗card, sdio_func_num_t func, bool enable)
  enable IO interrupt
- status_t SDIO_EnableIO (sdio_card_t ∗card, sdio_func_num_t func, bool enable)
  enable IO and wait IO ready

- status_t SDIO_SelectIO (sdio_card_t ∗card, sdio_func_num_t func)
  select IO
- status_t SDIO_AbortIO (sdio_card_t ∗card, sdio_func_num_t func)
  Abort IO transfer.
- status_t SDIO_SetDriverStrength (sdio_card_t ∗card, sd_driver_strength_t driverStrength)
  Set driver strength.
- status_t SDIO_EnableAsyncInterrupt (sdio_card_t ∗card, bool enable)
  Enable/Disable Async interrupt.
- status_t SDIO_GetPendingInterrupt (sdio_card_t ∗card, uint8_t ∗pendingInt)
  Get pending interrupt.
- status_t SDIO_IO_Transfer (sdio_card_t ∗card, sdio_command_t cmd, uint32_t argument, uint32_t blockSize, uint8_t ∗txData, uint8_t ∗rxData, uint16_t dataSize, uint32_t ∗response)
  sdio card io transfer function.
- void SDIO_SetIOIRQHandler (sdio_card_t ∗card, sdio_func_num_t func, sdio_io_irq_handler_t handler)
  sdio set io IRQ handler.
- status_t SDIO_HandlePendingIOInterrupt (sdio_card_t ∗card)
  sdio card io pending interrupt handle function.

Data Structure Documentation

73.2.1 struct _sdio_card

sdio card descriptor

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- sdmmchost_t ∗ host
  Host information.
- sdio_usr_param_t usrParam
  user parameter
- bool noInternalAlign
  use this flag to disable sdmmc align.
- uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]
  internal buffer
- bool isHostReady
  use this flag to indicate if need host re-init or not
- bool memPresentFlag
  indicate if memory present
- uint32_t busClock_Hz
  SD bus clock frequency united in Hz.
- uint32_t relativeAddress
  Relative address of the card.
- uint8_t sdVersion
  SD version.
- sd_timing_mode_t currentTiming
  current timing mode

- sd_driver_strength_t driverStrength
  driver strength
- sd_max_current_t maxCurrent
  card current limit
- sdmmc_operation_voltage_t operationVoltage
  card operation voltage
- uint8_t sdioVersion
  SDIO version.
- uint8_t cccrVersioin
  CCCR version.
- uint8_t ioTotalNumber
  total number of IO function
- uint32_t cccrflags
  Flags in _sd_card_flag.
- uint32_t io0blockSize
  record the io0 block size
- uint32_t ocr
  Raw OCR content, only 24bit avalible for SDIO card.
- uint32_t commonCISPointer
  point to common CIS
- sdio_common_cis_t commonCIS
  CIS table.
- sdio_fbr_t ioFBR [FSL_SDIO_MAX_IO_NUMS]
  FBR table.
- sdio_func_cis_t funcCIS [FSL_SDIO_MAX_IO_NUMS]
  function CIS table
- sdio_io_irq_handler_t ioIRQHandler [FSL_SDIO_MAX_IO_NUMS]
  io IRQ handler
- uint8_t ioIntIndex
  used to record current enabled io interrupt index
- uint8_t ioIntNums
  used to record total enabled io interrupt numbers

73.2.1.0.0.61 Field Documentation

73.2.1.0.0.61.1 bool sdio_card_t::noInternalAlign

If disable, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are align to low level driver

Macro Definition Documentation

73.3.1 #define FSL_SDIO_DRIVER_VERSION (MAKE_VERSION(2U, 3U, 2U)) /*2.3.2*/

Enumeration Type Documentation

73.4.1 enum sdio_io_direction_t

Enumerator

kSDIO_IORead io read
kSDIO_IOWrite io write

Function Documentation

73.5.1 status_t SDIO_Init (sdio_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_GoIdleFailed	
kStatus_SDMMC_HandShakeOperationConditionFailed	
kStatus_SDMMC_SDIO_InvalidCard	
kStatus_SDMMC_SDIO_InvalidVoltage	
kStatus_SDMMC_SendRelativeAddressFailed	
kStatus_SDMMC_SelectCardFailed	
kStatus_SDMMC_SDIO_SwitchHighSpeedFail	
kStatus_SDMMC_SDIO_ReadCISFail	

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.2 void SDIO_Deinit (sdio_card_t * *card*)

Parameters

card	Card descriptor.

73.5.3 status_t SDIO_CardInit (sdio_card_t * *card*)

This function initializes the card only, make sure the host is ready when call this function, otherwise it will return kStatus_SDMMC_HostNotReady.

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_Host-NotReady	host is not ready.
kStatus_SDMMC_Go-IdleFailed	Go idle failed.
kStatus_SDMMC_Not-SupportYet	Card not support.
kStatus_SDMMC_Send-OperationCondition-Failed	Send operation condition failed.
kStatus_SDMMC_All-SendCidFailed	Send CID failed.

kStatus_SDMMC_SendRelativeAddressFailed	Send relative address failed.
kStatus_SDMMC_SendCsdFailed	Send CSD failed.
kStatus_SDMMC_SelectCardFailed	Send SELECT_CARD command failed.
kStatus_SDMMC_SendScrFailed	Send SCR failed.
kStatus_SDMMC_SetBusWidthFailed	Set bus width failed.
kStatus_SDMMC_SwitchHighSpeedFailed	Switch high speed failed.
kStatus_SDMMC_SetCardBlockSizeFailed	Set card block size failed.
kStatus_Success	Operate successfully.

73.5.4 void SDIO_CardDeinit (sdio_card_t * *card*)

This function deinitializes the specific card.

Parameters

card	Card descriptor.

73.5.5 status_t SDIO_HostInit (sdio_card_t * *card*)

This function deinitializes the specific host.

Parameters

card	Card descriptor.

73.5.6 void SDIO_HostDeinit (sdio_card_t * *card*)

This function deinitializes the host.

Parameters

card	Card descriptor.

73.5.7 void SDIO_HostReset (SDMMCHOST_CONFIG * *host*)

This function reset the specific host.

Deprecated Do not use this function. It has been superceded by SDIO_HostDoReset.

Parameters

host	host descriptor.

73.5.8 void SDIO_HostDoReset (sdio_card_t * *card*)

This function reset the specific host.

Parameters

card	Card descriptor.

73.5.9 void SDIO_PowerOnCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

The power on operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by SDIO_SetCardPower.

Parameters

base	host base address.
pwr	user define power control configuration

73.5.10 void SDIO_PowerOffCard (SDMMCHOST_TYPE * *base*, const sdmmchost_pwr_card_t * *pwr*)

The power off operation depend on host or the user define power on function.

Deprecated Do not use this function. It has been superceded by SDIO_SetCardPower.

Parameters

base	host base address.
pwr	user define power control configuration

73.5.11 void SDIO_SetCardPower (sdio_card_t * *card*, bool *enable*)

The power off operation depend on host or the user define power on function.

Parameters

card	card descriptor.
enable	true is power on, false is power off.

73.5.12 status_t SDIO_CardInActive (sdio_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.13 status_t SDIO_GetCardCapability (sdio_card_t * *card*, sdio_func_num_t *func*)

Parameters

card	Card descriptor.
func	IO number

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.14 status_t SDIO_SetBlockSize (sdio_card_t ∗ *card,* sdio_func_num_t *func,* uint32_t *blockSize*)

Parameters

card	Card descriptor.
func	io number
blockSize	block size

Return values

kStatus_SDMMC_Set-CardBlockSizeFailed	
kStatus_SDMMC_SDIO-_InvalidArgument	
kStatus_Success	

73.5.15 status_t SDIO_CardReset (sdio_card_t ∗ *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.16 status_t SDIO_SetDataBusWidth (sdio_card_t ∗ *card*, sdio_bus_width_t *busWidth*)

Parameters

card	Card descriptor.
busWidth	bus width

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.17 status_t SDIO_SwitchToHighSpeed (sdio_card_t ∗ *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_SDMMC_SDIO-_SwitchHighSpeedFail	
kStatus_Success	

73.5.18 status_t SDIO_ReadCIS (sdio_card_t ∗ *card*, sdio_func_num_t *func*, const uint32_t ∗ *tupleList*, uint32_t *tupleNum*)

Parameters

card	Card descriptor.
func	io number
tupleList	code list
tupleNum	code number

Return values

kStatus_SDMMC_SDIO-_ReadCISFail	
kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.19 status_t SDIO_WaitCardDetectStatus (SDMMCHOST_TYPE ∗ *hostBase*, const sdmmchost_detect_card_t ∗ *cd*, bool *waitCardStatus*)

Detect card through GPIO, CD, DATA3.

Deprecated Do not use this function. It has been superceded by SDIO_PollingCardInsert.

Parameters

hostBase	card descriptor.
cd	detect configuration
waitCardStatus	wait card detect status

73.5.20 status_t SDIO_PollingCardInsert (sdio_card_t ∗ *card*, uint32_t *status*)

Detect card through GPIO, CD, DATA3.

Parameters

card	card descriptor.
status	detect status, kSD_Inserted or kSD_Removed.

73.5.21 bool SDIO_IsCardPresent (sdio_card_t * card)

Parameters

card	card descriptor.

73.5.22 status_t SDIO_IO_Write_Direct (sdio_card_t * *card*, sdio_func_num_t *func*, uint32_t *regAddr*, uint8_t * *data*, bool *raw*)

Parameters

card	Card descriptor.
func	IO numner
regAddr	register address
data	the data pinter to write
raw	flag, indicate read after write or write only

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.23 status_t SDIO_IO_Read_Direct (sdio_card_t * *card*, sdio_func_num_t *func*, uint32_t *regAddr*, uint8_t * *data*)

Parameters

card	Card descriptor.
func	IO number
regAddr	register address
data	pointer to read

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.24 status_t SDIO_IO_RW_Direct (sdio_card_t * *card*, sdio_io_direction_t *direction*, sdio_func_num_t *func*, uint32_t *regAddr*, uint8_t *dataIn*, uint8_t * *dataOut*)

Parameters

card	Card descriptor.
direction	io access direction, please reference sdio_io_direction_t.
func	IO number
regAddr	register address
dataIn	data to write
dataOut	data pointer for readback data, support both for read and write, when application want readback the data after write command, dataOut should not be NULL.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.25 status_t SDIO_IO_Write_Extended (sdio_card_t * *card*, sdio_func_num_t *func*, uint32_t *regAddr*, uint8_t * *buffer*, uint32_t *count*, uint32_t *flags*)

Parameters

card	Card descriptor.
func	IO number
regAddr	register address
buffer	data buffer to write
count	data count
flags	write flags

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_SDMMC_SDIO-_InvalidArgument	
kStatus_Success	

73.5.26 status_t SDIO_IO_Read_Extended (sdio_card_t * *card*, sdio_func_num_t *func*, uint32_t *regAddr*, uint8_t * *buffer*, uint32_t *count*, uint32_t *flags*)

Parameters

card	Card descriptor.
func	IO number
regAddr	register address
buffer	data buffer to read
count	data count
flags	write flags

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_SDMMC_SDIO-_InvalidArgument	
kStatus_Success	

73.5.27 status_t SDIO_EnableIOInterrupt (sdio_card_t * *card*, sdio_func_num_t *func*, bool *enable*)

Parameters

card	Card descriptor.
func	IO number
enable	enable/disable flag

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.28 status_t SDIO_EnableIO (sdio_card_t * *card*, sdio_func_num_t *func*, bool *enable*)

Parameters

card	Card descriptor.
func	IO number
enable	enable/disable flag

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.29 status_t SDIO_SelectIO (sdio_card_t * *card*, sdio_func_num_t *func*)

Parameters

card	Card descriptor.
func	IO number

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.30 status_t SDIO_AbortIO (sdio_card_t * *card*, sdio_func_num_t *func*)

Parameters

card	Card descriptor.
func	IO number

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.31 status_t SDIO_SetDriverStrength (sdio_card_t * *card*, sd_driver_strength_t *driverStrength*)

Parameters

card	Card descriptor.
driverStrength	target driver strength.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.32 status_t SDIO_EnableAsyncInterrupt (sdio_card_t * *card*, bool *enable*)

Parameters

card	Card descriptor.
enable	true is enable, false is disable.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.33 status_t SDIO_GetPendingInterrupt (sdio_card_t * *card*, uint8_t * *pendingInt*)

Parameters

card	Card descriptor.
pendingInt	pointer store pending interrupt

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

73.5.34 status_t SDIO_IO_Transfer (sdio_card_t ∗ *card,* sdio_command_t *cmd,* uint32_t *argument,* uint32_t *blockSize,* uint8_t ∗ *txData,* uint8_t ∗ *rxData,* uint16_t *dataSize,* uint32_t ∗ *response*)

This function can be used for trnansfer direct/extend command. Please pay attention to the non-align data buffer address transfer, if data buffer address can not meet host controller internal DMA requirement, sdio driver will try to use internal align buffer if data size is not bigger than internal buffer size, Align address transfer always can get a better performance, so if application want sdio driver make sure buffer address align, please redefine the SDMMC_GLOBAL_BUFFER_SIZE macro to a value which is big enough for your application.

Parameters

card	card descriptor.
cmd	command to transfer
argument	argument to transfer
blockSize	used for block mode.
txData	tx buffer pointer or NULL
rxData	rx buffer pointer or NULL
dataSize	transfer data size
response	reponse pointer, if application want read response back, please set it to a NON-NULL pointer.

73.5.35 void SDIO_SetIOIRQHandler (sdio_card_t ∗ *card,* sdio_func_num_t *func,* sdio_io_irq_handler_t *handler*)

Parameters

card	card descriptor.
func	function io number.
handler	io IRQ handler.

73.5.36 status_t SDIO_HandlePendingIOInterrupt (sdio_card_t * *card*)

This function is used to handle the pending io interrupt. To reigster a IO IRQ handler,

```
* //initialization
* SDIO_EnableIOInterrupt(card, 0, true);
* SDIO_SetIOIRQHandler(card, 0, func0_handler);
* //call it in interrupt callback
* SDIO_HandlePendingIOInterrupt(card);
*
```

To releae a IO IRQ handler,

```
* SDIO_EnableIOInterrupt(card, 0, false);
* SDIO_SetIOIRQHandler(card, 0, NULL);
*
```

Parameters

card	card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

Chapter 74
SDSPI

Overview

Data Structures

- struct sdspi_host_t
  SDSPI host state. More...
- struct sdspi_card_t
  SD Card Structure. More...

Macros

- #define FSL_SDSPI_DRIVER_VERSION (MAKE_VERSION(2U, 2U, 1U)) /*2.2.1*/
  Driver version.
- #define FSL_SDSPI_DEFAULT_BLOCK_SIZE (512U)
  Default block size.
- #define DSPI_DUMMY_DATA (0xFFU)
  Dummy byte define, OxFF should be defined as the dummy data.
- #define SDSPI_CARD_CRC_PROTECTION_ENABLE 0U
  This macro is used to enable or disable the CRC protection for SD card command.

Enumerations

- enum {
  kStatus_SDSPI_SetFrequencyFailed = MAKE_STATUS(kStatusGroup_SDSPI, 0U),
  kStatus_SDSPI_ExchangeFailed = MAKE_STATUS(kStatusGroup_SDSPI, 1U),
  kStatus_SDSPI_WaitReadyFailed = MAKE_STATUS(kStatusGroup_SDSPI, 2U),
  kStatus_SDSPI_ResponseError = MAKE_STATUS(kStatusGroup_SDSPI, 3U),
  kStatus_SDSPI_WriteProtected = MAKE_STATUS(kStatusGroup_SDSPI, 4U),
  kStatus_SDSPI_GoIdleFailed = MAKE_STATUS(kStatusGroup_SDSPI, 5U),
  kStatus_SDSPI_SendCommandFailed = MAKE_STATUS(kStatusGroup_SDSPI, 6U),
  kStatus_SDSPI_ReadFailed = MAKE_STATUS(kStatusGroup_SDSPI, 7U),
  kStatus_SDSPI_WriteFailed = MAKE_STATUS(kStatusGroup_SDSPI, 8U),
  kStatus_SDSPI_SendInterfaceConditionFailed,
  kStatus_SDSPI_SendOperationConditionFailed,
  kStatus_SDSPI_ReadOcrFailed = MAKE_STATUS(kStatusGroup_SDSPI, 11U),
  kStatus_SDSPI_SetBlockSizeFailed = MAKE_STATUS(kStatusGroup_SDSPI, 12U),
  kStatus_SDSPI_SendCsdFailed = MAKE_STATUS(kStatusGroup_SDSPI, 13U),
  kStatus_SDSPI_SendCidFailed = MAKE_STATUS(kStatusGroup_SDSPI, 14U),
  kStatus_SDSPI_StopTransmissionFailed = MAKE_STATUS(kStatusGroup_SDSPI, 15U),
  kStatus_SDSPI_SendApplicationCommandFailed,
  kStatus_SDSPI_InvalidVoltage = MAKE_STATUS(kStatusGroup_SDSPI, 17U),
  kStatus_SDSPI_SwitchCmdFail = MAKE_STATUS(kStatusGroup_SDSPI, 18U),

kStatus_SDSPI_NotSupportYet = MAKE_STATUS(kStatusGroup_SDSPI, 19U) }
SDSPI API status.
- enum {
kSDSPI_SupportHighCapacityFlag = (1U << 0U),
kSDSPI_SupportSdhcFlag = (1U << 1U),
kSDSPI_SupportSdxcFlag = (1U << 2U),
kSDSPI_SupportSdscFlag = (1U << 3U) }
SDSPI card flag.
- enum {
kSDSPI_ResponseTypeR1 = 0U,
kSDSPI_ResponseTypeR1b = 1U,
kSDSPI_ResponseTypeR2 = 2U,
kSDSPI_ResponseTypeR3 = 3U,
kSDSPI_ResponseTypeR7 = 4U }
SDSPI response type.
- enum {
kSDSPI_CmdGoIdle = kSDMMC_GoIdleState << 8U | kSDSPI_ResponseTypeR1,
kSDSPI_CmdCrc = kSDSPI_CommandCrc << 8U | kSDSPI_ResponseTypeR1,
kSDSPI_CmdSendInterfaceCondition }
SDSPI command type.
- enum sdspi_cs_active_polarity_t {
kSDSPI_CsActivePolarityHigh = 0U,
kSDSPI_CsActivePolarityLow }
cs active polarity

SDSPI Function

- status_t SDSPI_Init (sdspi_card_t ∗card)
Initializes the card on a specific SPI instance.
- void SDSPI_Deinit (sdspi_card_t ∗card)
Deinitializes the card.
- bool SDSPI_CheckReadOnly (sdspi_card_t ∗card)
Checks whether the card is write-protected.
- status_t SDSPI_ReadBlocks (sdspi_card_t ∗card, uint8_t ∗buffer, uint32_t startBlock, uint32_t blockCount)
Reads blocks from the specific card.
- status_t SDSPI_WriteBlocks (sdspi_card_t ∗card, uint8_t ∗buffer, uint32_t startBlock, uint32_t blockCount)
Writes blocks of data to the specific card.
- status_t SDSPI_SendCid (sdspi_card_t ∗card)
Send GET-CID command In our sdspi init function, this function is removed for better code size, if id information is needed, you can call it after the init function directly.
- status_t SDSPI_SendPreErase (sdspi_card_t ∗card, uint32_t blockCount)
Multiple blocks write pre-erase function.
- status_t SDSPI_EraseBlocks (sdspi_card_t ∗card, uint32_t startBlock, uint32_t blockCount)
Block erase function.
- status_t SDSPI_SwitchToHighSpeed (sdspi_card_t ∗card)
Switch to high speed function.

Data Structure Documentation

74.2.1 struct sdspi_host_t

Data Fields

- uint32_t busBaudRate
  Bus baud rate.
- status_t(∗ setFrequency)(uint32_t frequency)
  Set frequency of SPI.
- status_t(∗ exchange)(uint8_t ∗in, uint8_t ∗out, uint32_t size)
  Exchange data over SPI.
- void(∗ init)(void)
  SPI initialization.
- void(∗ deinit)(void)
  SPI de-initialization.
- void(∗ csActivePolarity)(sdspi_cs_active_polarity_t polarity)
  SPI CS active polarity.

74.2.2 struct sdspi_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- sdspi_host_t ∗ host
  Host state information.
- uint32_t relativeAddress
  Relative address of the card.
- uint32_t flags
  Flags defined in _sdspi_card_flag.
- uint8_t internalBuffer [16U]
  internal buffer for card raw register content
- uint32_t ocr
  Raw OCR content.
- sd_cid_t cid
  CID.
- sd_csd_t csd
  CSD.
- sd_scr_t scr
  SCR.
- uint32_t blockCount
  Card total block number.
- uint32_t blockSize
  Card block size.

74.2.2.0.0.62 Field Documentation

74.2.2.0.0.62.1 uint32_t sdspi_card_t::flags

Macro Definition Documentation

74.3.1 #define FSL_SDSPI_DRIVER_VERSION (MAKE_VERSION(2U, 2U, 1U)) /∗2.2.1∗/

74.3.2 #define DSPI_DUMMY_DATA (0xFFU)

Dummy data used for Tx if there is no txData.

74.3.3 #define SDSPI_CARD_CRC_PROTECTION_ENABLE 0U

The SPI interface is intialized in the CRC off mode by default.However, the RESET command(cmd0) that is used to switch the card to SPI mode, is recieved by by the card while in SD mode and therefore, shall have a valid CRC filed, after the card put into SPI mode , CRC check for all command include CMD0 will be done according to CMD59 setting, host can turn CRC option on and off using the CMD59, this command should be call before ACMD41. CMD8 CRC verification is always enabled. The host shall set correct CRC in the argument of CMD8. If CRC check is enabled, then sdspi code size and read/write performance will be lower than CRC off. CRC check is off by default.

Enumeration Type Documentation

74.4.1 anonymous enum

Enumerator

kStatus_SDSPI_SetFrequencyFailed Set frequency failed.
kStatus_SDSPI_ExchangeFailed Exchange data on SPI bus failed.
kStatus_SDSPI_WaitReadyFailed Wait card ready failed.
kStatus_SDSPI_ResponseError Response is error.
kStatus_SDSPI_WriteProtected Write protected.
kStatus_SDSPI_GoIdleFailed Go idle failed.
kStatus_SDSPI_SendCommandFailed Send command failed.
kStatus_SDSPI_ReadFailed Read data failed.
kStatus_SDSPI_WriteFailed Write data failed.
kStatus_SDSPI_SendInterfaceConditionFailed Send interface condition failed.
kStatus_SDSPI_SendOperationConditionFailed Send operation condition failed.
kStatus_SDSPI_ReadOcrFailed Read OCR failed.
kStatus_SDSPI_SetBlockSizeFailed Set block size failed.
kStatus_SDSPI_SendCsdFailed Send CSD failed.
kStatus_SDSPI_SendCidFailed Send CID failed.
kStatus_SDSPI_StopTransmissionFailed Stop transmission failed.

kStatus_SDSPI_SendApplicationCommandFailed Send application command failed.
kStatus_SDSPI_InvalidVoltage invaild supply voltage
kStatus_SDSPI_SwitchCmdFail switch command crc protection on/off
kStatus_SDSPI_NotSupportYet not support

74.4.2 anonymous enum

Enumerator

kSDSPI_SupportHighCapacityFlag Card is high capacity.
kSDSPI_SupportSdhcFlag Card is SDHC.
kSDSPI_SupportSdxcFlag Card is SDXC.
kSDSPI_SupportSdscFlag Card is SDSC.

74.4.3 anonymous enum

Enumerator

kSDSPI_ResponseTypeR1 Response 1.
kSDSPI_ResponseTypeR1b Response 1 with busy.
kSDSPI_ResponseTypeR2 Response 2.
kSDSPI_ResponseTypeR3 Response 3.
kSDSPI_ResponseTypeR7 Response 7.

74.4.4 anonymous enum

Enumerator

kSDSPI_CmdGoIdle command go idle
kSDSPI_CmdCrc command crc protection
kSDSPI_CmdSendInterfaceCondition command send interface condition

74.4.5 enum sdspi_cs_active_polarity_t

Enumerator

kSDSPI_CsActivePolarityHigh CS active polarity high.
kSDSPI_CsActivePolarityLow CS active polarity low.

Function Documentation

74.5.1 status_t SDSPI_Init (sdspi_card_t * *card*)

This function initializes the card on a specific SPI instance.

Parameters

card	Card descriptor

Return values

kStatus_SDSPI_SetFrequencyFailed	Set frequency failed.
kStatus_SDSPI_GoIdleFailed	Go idle failed.
kStatus_SDSPI_SendInterfaceConditionFailed	Send interface condition failed.
kStatus_SDSPI_SendOperationConditionFailed	Send operation condition failed.
kStatus_Timeout	Send command timeout.
kStatus_SDSPI_NotSupportYet	Not support yet.
kStatus_SDSPI_ReadOcrFailed	Read OCR failed.
kStatus_SDSPI_SetBlockSizeFailed	Set block size failed.
kStatus_SDSPI_SendCsdFailed	Send CSD failed.
kStatus_SDSPI_SendCidFailed	Send CID failed.
kStatus_Success	Operate successfully.

74.5.2 void SDSPI_Deinit (sdspi_card_t * *card*)

This function deinitializes the specific card.

Parameters

card	Card descriptor

74.5.3 bool SDSPI_CheckReadOnly (sdspi_card_t * *card*)

This function checks if the card is write-protected via CSD register.

Parameters

card	Card descriptor.

Return values

true	Card is read only.
false	Card isn't read only.

74.5.4 status_t SDSPI_ReadBlocks (sdspi_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

This function reads blocks from specific card.

Parameters

card	Card descriptor.
buffer	the buffer to hold the data read from card
startBlock	the start block index
blockCount	the number of blocks to read

Return values

kStatus_SDSPI_SendCommandFailed	Send command failed.
kStatus_SDSPI_ReadFailed	Read data failed.
kStatus_SDSPI_StopTransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

74.5.5 status_t SDSPI_WriteBlocks (sdspi_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

This function writes blocks to specific card

Parameters

card	Card descriptor.
buffer	the buffer holding the data to be written to the card
startBlock	the start block index
blockCount	the number of blocks to write

Return values

kStatus_SDSPI_Write-Protected	Card is write protected.
kStatus_SDSPI_Send-CommandFailed	Send command failed.
kStatus_SDSPI_-ResponseError	Response is error.
kStatus_SDSPI_Write-Failed	Write data failed.
kStatus_SDSPI_-ExchangeFailed	Exchange data over SPI failed.
kStatus_SDSPI_Wait-ReadyFailed	Wait card to be ready status failed.
kStatus_Success	Operate successfully.

74.5.6 status_t SDSPI_SendCid (sdspi_card_t ∗ *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDSPI_Send-CommandFailed	Send command failed.

kStatus_SDSPI_Read-Failed	Read data blocks failed.
kStatus_Success	Operate successfully.

74.5.7 status_t SDSPI_SendPreErase (sdspi_card_t ∗ *card*, uint32_t *blockCount*)

This function should be called before SDSPI_WriteBlocks, it is used to set the number of the write blocks to be pre-erased before writing.

Parameters

card	Card descriptor.
blockCount	the block counts to be write.

Return values

kStatus_SDSPI_Send-CommandFailed	Send command failed.
kStatus_SDSPI_Send-ApplicationCommand-Failed	
kStatus_SDSPI_-ResponseError	
kStatus_Success	Operate successfully.

74.5.8 status_t SDSPI_EraseBlocks (sdspi_card_t ∗ *card*, uint32_t *startBlock*, uint32_t *blockCount*)

Parameters

card	Card descriptor.
startBlock	start block address to be erase.
blockCount	the block counts to be erase.

Return values

kStatus_SDSPI_WaitReadyFailed	Wait ready failed.
kStatus_SDSPI_SendCommandFailed	Send command failed.
kStatus_Success	Operate successfully.

74.5.9 status_t SDSPI_SwitchToHighSpeed (sdspi_card_t * *card*)

This function can be called after SDSPI_Init function if target board's layout support >25MHZ spi baudrate, otherwise this function is useless.Be careful with call this function, code size and stack usage will be enlarge.

Parameters

card	Card descriptor.

Return values

kStatus_Fail	switch failed.
kStatus_Success	Operate successfully.

Chapter 75
Cs42888

Overview

Data Structures

- struct cs42888_audio_format_t
  cs42888 audio format More...
- struct cs42888_config_t
  Initialize structure of CS42888. More...
- struct cs42888_handle_t
  cs42888 handler More...

Macros

- #define CS42888_I2C_HANDLER_SIZE CODEC_I2C_MASTER_HANDLER_SIZE
  CS42888 handle size.
- #define CS42888_ID 0x01U
  Define the register address of CS42888.
- #define CS42888_AOUT_MAX_VOLUME_VALUE 0xFFU
  CS42888 volume setting range.
- #define CS42888_CACHEREGNUM 28U
  Cache register number.
- #define CS42888_I2C_ADDR 0x48U
  CS42888 I2C address.
- #define CS42888_I2C_BITRATE (100000U)
  CS42888 I2C baudrate.

Typedefs

- typedef void(∗ cs42888_reset)(bool state)
  cs42888 reset function pointer

Enumerations

- enum cs42888_func_mode {
  kCS42888_ModeMasterSSM = 0x0,
  kCS42888_ModeMasterDSM = 0x1,
  kCS42888_ModeMasterQSM = 0x2,
  kCS42888_ModeSlave = 0x3 }
  CS42888 support modes.
- enum cs42888_module_t {

kCS42888_ModuleDACPair1 = 0x2,
kCS42888_ModuleDACPair2 = 0x4,
kCS42888_ModuleDACPair3 = 0x8,
kCS42888_ModuleDACPair4 = 0x10,
kCS42888_ModuleADCPair1 = 0x20,
kCS42888_ModuleADCPair2 = 0x40 }
Modules in CS42888 board.
- enum cs42888_bus_t {
kCS42888_BusLeftJustified = 0x0,
kCS42888_BusI2S = 0x1,
kCS42888_BusRightJustified = 0x2,
kCS42888_BusOL1 = 0x4,
kCS42888_BusOL2 = 0x5,
kCS42888_BusTDM = 0x6 }
CS42888 supported audio bus type.
- enum {
kCS42888_AOUT1 = 1U,
kCS42888_AOUT2 = 2U,
kCS42888_AOUT3 = 3U,
kCS42888_AOUT4 = 4U,
kCS42888_AOUT5 = 5U,
kCS42888_AOUT6 = 6U,
kCS42888_AOUT7 = 7U,
kCS42888_AOUT8 = 8U }
CS428888 play channel.

Functions

- status_t CS42888_Init (cs42888_handle_t ∗handle, cs42888_config_t ∗config)
CS42888 initialize function.
- status_t CS42888_Deinit (cs42888_handle_t ∗handle)
Deinit the CS42888 codec.
- status_t CS42888_SetProtocol (cs42888_handle_t ∗handle, cs42888_bus_t protocol, uint32_t bitWidth)
Set the audio transfer protocol.
- void CS42888_SetFuncMode (cs42888_handle_t ∗handle, cs42888_func_mode mode)
Set CS42888 to differernt working mode.
- status_t CS42888_SelectFunctionalMode (cs42888_handle_t ∗handle, cs42888_func_mode adcMode, cs42888_func_mode dacMode)
Set CS42888 to differernt functional mode.
- status_t CS42888_SetAOUTVolume (cs42888_handle_t ∗handle, uint8_t channel, uint8_t volume)
Set the volume of different modules in CS42888.
- status_t CS42888_SetAINVolume (cs42888_handle_t ∗handle, uint8_t channel, uint8_t volume)
Set the volume of different modules in CS42888.
- uint8_t CS42888_GetAOUTVolume (cs42888_handle_t ∗handle, uint8_t channel)
Get the volume of different AOUT channel in CS42888.
- uint8_t CS42888_GetAINVolume (cs42888_handle_t ∗handle, uint8_t channel)
Get the volume of different AIN channel in CS42888.

- status_t CS42888_SetMute (cs42888_handle_t *handle, uint8_t channelMask)
  Mute modules in CS42888.
- status_t CS42888_SetChannelMute (cs42888_handle_t *handle, uint8_t channel, bool isMute)
  Mute channel modules in CS42888.
- status_t CS42888_SetModule (cs42888_handle_t *handle, cs42888_module_t module, bool isEnabled)
  Enable/disable expected devices.
- status_t CS42888_ConfigDataFormat (cs42888_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)
  Configure the data format of audio data.
- status_t CS42888_WriteReg (cs42888_handle_t *handle, uint8_t reg, uint8_t val)
  Write register to CS42888 using I2C.
- status_t CS42888_ReadReg (cs42888_handle_t *handle, uint8_t reg, uint8_t *val)
  Read register from CS42888 using I2C.
- status_t CS42888_ModifyReg (cs42888_handle_t *handle, uint8_t reg, uint8_t mask, uint8_t val)
  Modify some bits in the register using I2C.

Driver version

- #define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))
  cs42888 driver version 2.1.2.

Data Structure Documentation

75.2.1 struct cs42888_audio_format_t

Data Fields

- uint32_t mclk_HZ
  master clock frequency
- uint32_t sampleRate
  sample rate
- uint32_t bitWidth
  bit width

75.2.2 struct cs42888_config_t

Data Fields

- cs42888_bus_t bus
  Audio transfer protocol.
- cs42888_audio_format_t format
  cs42888 audio format
- cs42888_func_mode ADCMode
  CS42888 ADC function mode.
- cs42888_func_mode DACMode
  CS42888 DAC function mode.
- bool master

true is master, false is slave
- codec_i2c_config_t i2cConfig
  i2c bus configuration
- uint8_t slaveAddress
  slave address
- cs42888_reset reset
  reset function pointer

75.2.2.0.0.63 Field Documentation

75.2.2.0.0.63.1 cs42888_func_mode cs42888_config_t::ADCMode

75.2.2.0.0.63.2 cs42888_func_mode cs42888_config_t::DACMode

75.2.3 struct cs42888_handle_t

Data Fields

- cs42888_config_t ∗ config
  cs42888 config pointer
- uint8_t i2cHandle [CS42888_I2C_HANDLER_SIZE]
  i2c handle pointer

Macro Definition Documentation

75.3.1 #define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

75.3.2 #define CS42888_ID 0x01U

75.3.3 #define CS42888_I2C_ADDR 0x48U

Enumeration Type Documentation

75.4.1 enum cs42888_func_mode

Enumerator

kCS42888_ModeMasterSSM master single speed mode
kCS42888_ModeMasterDSM master dual speed mode
kCS42888_ModeMasterQSM master quad speed mode
kCS42888_ModeSlave master single speed mode

75.4.2 enum cs42888_module_t

Enumerator

kCS42888_ModuleDACPair1 DAC pair1 (AOUT1 and AOUT2) module in CS42888.

kCS42888_ModuleDACPair2 DAC pair2 (AOUT3 and AOUT4) module in CS42888.
kCS42888_ModuleDACPair3 DAC pair3 (AOUT5 and AOUT6) module in CS42888.
kCS42888_ModuleDACPair4 DAC pair4 (AOUT7 and AOUT8) module in CS42888.
kCS42888_ModuleADCPair1 ADC pair1 (AIN1 and AIN2) module in CS42888.
kCS42888_ModuleADCPair2 ADC pair2 (AIN3 and AIN4) module in CS42888.

75.4.3 enum cs42888_bus_t

Enumerator

kCS42888_BusLeftJustified Left justified format, up to 24 bits.
kCS42888_BusI2S I2S format, up to 24 bits.
kCS42888_BusRightJustified Right justified, can support 16bits and 24 bits.
kCS42888_BusOL1 One-Line #1 mode.
kCS42888_BusOL2 One-Line #2 mode.
kCS42888_BusTDM TDM mode.

75.4.4 anonymous enum

Enumerator

kCS42888_AOUT1 aout1
kCS42888_AOUT2 aout2
kCS42888_AOUT3 aout3
kCS42888_AOUT4 aout4
kCS42888_AOUT5 aout5
kCS42888_AOUT6 aout6
kCS42888_AOUT7 aout7
kCS42888_AOUT8 aout8

Function Documentation

75.5.1 status_t CS42888_Init (cs42888_handle_t * *handle*, cs42888_config_t * *config*)

The second parameter is NULL to CS42888 in this version. If users want to change the settings, they have to use cs42888_write_reg() or cs42888_modify_reg() to set the register value of CS42888. Note-: If the codec_config is NULL, it would initialize CS42888 using default settings. The default setting: codec_config->bus = kCS42888_BusI2S codec_config->ADCmode = kCS42888_ModeSlave codec_-config->DACmode = kCS42888_ModeSlave

Parameters

handle	CS42888 handle structure.
config	CS42888 configuration structure.

75.5.2 status_t CS42888_Deinit (cs42888_handle_t * *handle*)

This function close all modules in CS42888 to save power.

Parameters

handle	CS42888 handle structure pointer.

75.5.3 status_t CS42888_SetProtocol (cs42888_handle_t * *handle*, cs42888_bus_t *protocol*, uint32_t *bitWidth*)

CS42888 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

handle	CS42888 handle structure.
protocol	Audio data transfer protocol.
bitWidth	bit width

75.5.4 void CS42888_SetFuncMode (cs42888_handle_t * *handle*, cs42888_func_mode *mode*)

Deprecated api, Do not use it anymore. It has been superceded by CS42888_SelectFunctionalMode.

Parameters

handle	CS42888 handle structure.
mode	differenht working mode of CS42888.

75.5.5 status_t CS42888_SelectFunctionalMode (cs42888_handle_t * *handle*, cs42888_func_mode *adcMode*, cs42888_func_mode *dacMode*)

Parameters

handle	CS42888 handle structure.
adcMode	differenht working mode of CS42888.
dacMode	differenht working mode of CS42888.

75.5.6 status_t CS42888_SetAOUTVolume (cs42888_handle_t ∗ *handle,* uint8_t *channel,* uint8_t *volume*)

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AOUT channel, it shall be 1∼8.
volume	Volume value need to be set.

75.5.7 status_t CS42888_SetAINVolume (cs42888_handle_t ∗ *handle,* uint8_t *channel,* uint8_t *volume*)

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AIN channel, it shall be 1∼4.
volume	Volume value need to be set.

75.5.8 uint8_t CS42888_GetAOUTVolume (cs42888_handle_t ∗ *handle,* uint8_t *channel*)

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AOUT channel, it shall be 1∼8.

75.5.9 uint8_t CS42888_GetAINVolume (cs42888_handle_t ∗ *handle,* uint8_t *channel*)

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AIN channel, it shall be 1∼4.

75.5.10 status_t CS42888_SetMute (cs42888_handle_t ∗ *handle,* uint8_t *channelMask*)

Parameters

handle	CS42888 handle structure.
channelMask	Channel mask for mute. Mute channel 0, it shall be 0x1, while mute channel 0 and 1, it shall be 0x3. Mute all channel, it shall be 0xFF. Each bit represent one channel, 1 means mute, 0 means unmute.

75.5.11 status_t CS42888_SetChannelMute (cs42888_handle_t ∗ *handle,* uint8_t *channel,* bool *isMute*)

Parameters

handle	CS42888 handle structure.
channel	reference _cs42888_play_channel.

isMute	true is mute, falase is unmute.

75.5.12 status_t CS42888_SetModule (cs42888_handle_t * *handle*, cs42888_module_t *module*, bool *isEnabled*)

Parameters

handle	CS42888 handle structure.
module	Module expected to enable.
isEnabled	Enable or disable moudles.

75.5.13 status_t CS42888_ConfigDataFormat (cs42888_handle_t * *handle*, uint32_t *mclk*, uint32_t *sample_rate*, uint32_t *bits*)

This function would configure the registers about the sample rate, bit depths.

Parameters

handle	CS42888 handle structure pointer.
mclk	Master clock frequency of I2S.
sample_rate	Sample rate of audio file running in CS42888. CS42888 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
bits	Bit depth of audio file (CS42888 only supports 16bit, 20bit, 24bit and 32 bit in HW).

75.5.14 status_t CS42888_WriteReg (cs42888_handle_t * *handle*, uint8_t *reg*, uint8_t *val*)

Parameters

handle	CS42888 handle structure.
reg	The register address in CS42888.
val	Value needs to write into the register.

75.5.15 status_t CS42888_ReadReg (cs42888_handle_t * *handle*, uint8_t *reg*, uint8_t * *val*)

Parameters

handle	CS42888 handle structure.
reg	The register address in CS42888.
val	Value written to.

75.5.16 status_t CS42888_ModifyReg (cs42888_handle_t * *handle*, uint8_t *reg*, uint8_t *mask*, uint8_t *val*)

Parameters

handle	CS42888 handle structure.
reg	The register address in CS42888.
mask	The mask code for the bits want to write. The bit you want to write should be 0.
val	Value needs to write into the register.

Chapter 76
Da7212

Overview

Data Structures

- struct da7212_pll_config_t
  da7212 pll configuration More...
- struct da7212_audio_format_t
  da7212 audio format More...
- struct da7212_config_t
  DA7212 configure structure. More...
- struct da7212_handle_t
  da7212 codec handler More...

Macros

- #define DA7212_I2C_HANDLER_SIZE CODEC_I2C_MASTER_HANDLER_SIZE
  da7212 handle size
- #define DA7212_ADDRESS (0x1A)
  DA7212 I2C address.
- #define DA7212_HEADPHONE_MAX_VOLUME_VALUE 0x3FU
  da7212 volume setting range

Enumerations

- enum da7212_Input_t {
  kDA7212_Input_AUX = 0x0,
  kDA7212_Input_MIC1_Dig,
  kDA7212_Input_MIC1_An,
  kDA7212_Input_MIC2 }
  DA7212 input source select.
- enum _da7212_play_channel {
  kDA7212_HeadphoneLeft = 1U,
  kDA7212_HeadphoneRight = 2U,
  kDA7212_Speaker = 4U }
  da7212 play channel
- enum da7212_Output_t {
  kDA7212_Output_HP = 0x0,
  kDA7212_Output_SP }
  DA7212 output device select.
- enum _da7212_module {
  kDA7212_ModuleADC,
  kDA7212_ModuleDAC,
  kDA7212_ModuleHeadphone,

kDA7212_ModuleSpeaker }
  DA7212 module.
- enum da7212_dac_source_t {
  kDA7212_DACSourceADC = 0x0U,
  kDA7212_DACSourceInputStream = 0x3U }
  DA7212 functionality.
- enum da7212_volume_t {
  kDA7212_DACGainMute = 0x7,
  kDA7212_DACGainM72DB = 0x17,
  kDA7212_DACGainM60DB = 0x1F,
  kDA7212_DACGainM54DB = 0x27,
  kDA7212_DACGainM48DB = 0x2F,
  kDA7212_DACGainM42DB = 0x37,
  kDA7212_DACGainM36DB = 0x3F,
  kDA7212_DACGainM30DB = 0x47,
  kDA7212_DACGainM24DB = 0x4F,
  kDA7212_DACGainM18DB = 0x57,
  kDA7212_DACGainM12DB = 0x5F,
  kDA7212_DACGainM6DB = 0x67,
  kDA7212_DACGain0DB = 0x6F,
  kDA7212_DACGain6DB = 0x77,
  kDA7212_DACGain12DB = 0x7F }
  DA7212 volume.
- enum da7212_protocol_t {
  kDA7212_BusI2S = 0x0,
  kDA7212_BusLeftJustified,
  kDA7212_BusRightJustified,
  kDA7212_BusDSPMode }
  The audio data transfer protocol choice.
- enum da7212_sys_clk_source_t {
  kDA7212_SysClkSourceMCLK = 0U,
  kDA7212_SysClkSourcePLL = 1U << 14 }
  da7212 system clock source
- enum da7212_pll_clk_source_t { kDA7212_PLLClkSourceMCLK = 0U }
  DA7212 pll clock source.
- enum da7212_pll_out_clk_t {
  kDA7212_PLLOutputClk11289600 = 11289600U,
  kDA7212_PLLOutputClk12288000 = 12288000U }
  DA7212 output clock frequency.
- enum da7212_master_bits_t {
  kDA7212_MasterBits32PerFrame = 0U,
  kDA7212_MasterBits64PerFrame = 1U,
  kDA7212_MasterBits128PerFrame = 2U,
  kDA7212_MasterBits256PerFrame = 3U }
  master mode bits per frame

Functions

- status_t DA7212_Init (da7212_handle_t *handle, da7212_config_t *codecConfig)
  DA7212 initialize function.
- status_t DA7212_ConfigAudioFormat (da7212_handle_t *handle, uint32_t masterClock_Hz, uint32_t sampleRate_Hz, uint32_t dataBits)
  Set DA7212 audio format.
- status_t DA7212_SetPLLConfig (da7212_handle_t *handle, da7212_pll_config_t *config)
  DA7212 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.
- void DA7212_ChangeHPVolume (da7212_handle_t *handle, da7212_volume_t volume)
  Set DA7212 playback volume.
- void DA7212_Mute (da7212_handle_t *handle, bool isMuted)
  Mute or unmute DA7212.
- void DA7212_ChangeInput (da7212_handle_t *handle, da7212_Input_t DA7212_Input)
  Set the input data source of DA7212.
- void DA7212_ChangeOutput (da7212_handle_t *handle, da7212_Output_t DA7212_Output)
  Set the output device of DA7212.
- status_t DA7212_SetChannelVolume (da7212_handle_t *handle, uint32_t channel, uint32_t volume)
  Set module volume.
- status_t DA7212_SetChannelMute (da7212_handle_t *handle, uint32_t channel, bool isMute)
  Set module mute.
- status_t DA7212_SetProtocol (da7212_handle_t *handle, da7212_protocol_t protocol)
  Set protocol for DA7212.
- status_t DA7212_SetMasterModeBits (da7212_handle_t *handle, uint32_t bitWidth)
  Set master mode bits per frame for DA7212.
- status_t DA7212_WriteRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t u8RegisterData)
  Write a register for DA7212.
- status_t DA7212_ReadRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t *pu8RegisterData)
  Get a register value of DA7212.
- status_t DA7212_Deinit (da7212_handle_t *handle)
  Deinit DA7212.

Driver version

- #define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))
  CLOCK driver version 2.2.2.

Data Structure Documentation

76.2.1 struct da7212_pll_config_t

Data Fields

- da7212_pll_clk_source_t source
  pll reference clock source
- uint32_t refClock_HZ

pll reference clock frequency
- da7212_pll_out_clk_t outputClock_HZ
  pll output clock frequency

76.2.2 struct da7212_audio_format_t

Data Fields

- uint32_t mclk_HZ
  master clock frequency
- uint32_t sampleRate
  sample rate
- uint32_t bitWidth
  bit width
- bool isBclkInvert
  bit clock intervet

76.2.3 struct da7212_config_t

Data Fields

- bool isMaster
  If DA7212 is master, true means master, false means slave.
- da7212_protocol_t protocol
  Audio bus format, can be I2S, LJ, RJ or DSP mode.
- da7212_dac_source_t dacSource
  DA7212 data source.
- da7212_audio_format_t format
  audio format
- uint8_t slaveAddress
  device address
- codec_i2c_config_t i2cConfig
  i2c configuration
- da7212_sys_clk_source_t sysClkSource
  system clock source
- da7212_pll_config_t ∗ pll
  pll configuration

76.2.3.0.0.64 Field Documentation

76.2.3.0.0.64.1 bool da7212_config_t::isMaster

76.2.3.0.0.64.2 da7212_protocol_t da7212_config_t::protocol

76.2.3.0.0.64.3 da7212_dac_source_t da7212_config_t::dacSource

76.2.4 struct da7212_handle_t

Data Fields

- da7212_config_t * config
  da7212 config pointer
- uint8_t i2cHandle [DA7212_I2C_HANDLER_SIZE]
  i2c handle

Macro Definition Documentation

76.3.1 #define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))

Enumeration Type Documentation

76.4.1 enum da7212_Input_t

Enumerator

kDA7212_Input_AUX Input from AUX.
kDA7212_Input_MIC1_Dig Input from MIC1 Digital.
kDA7212_Input_MIC1_An Input from Mic1 Analog.
kDA7212_Input_MIC2 Input from MIC2.

76.4.2 enum _da7212_play_channel

Enumerator

kDA7212_HeadphoneLeft headphone left
kDA7212_HeadphoneRight headphone right
kDA7212_Speaker speaker channel

76.4.3 enum da7212_Output_t

Enumerator

kDA7212_Output_HP Output to headphone.

kDA7212_Output_SP Output to speaker.

76.4.4 enum _da7212_module

Enumerator

kDA7212_ModuleADC module ADC
kDA7212_ModuleDAC module DAC
kDA7212_ModuleHeadphone module headphone
kDA7212_ModuleSpeaker module speaker

76.4.5 enum da7212_dac_source_t

Enumerator

kDA7212_DACSourceADC DAC source from ADC.
kDA7212_DACSourceInputStream DAC source from.

76.4.6 enum da7212_volume_t

Enumerator

kDA7212_DACGainMute Mute DAC.
kDA7212_DACGainM72DB DAC volume -72db.
kDA7212_DACGainM60DB DAC volume -60db.
kDA7212_DACGainM54DB DAC volume -54db.
kDA7212_DACGainM48DB DAC volume -48db.
kDA7212_DACGainM42DB DAC volume -42db.
kDA7212_DACGainM36DB DAC volume -36db.
kDA7212_DACGainM30DB DAC volume -30db.
kDA7212_DACGainM24DB DAC volume -24db.
kDA7212_DACGainM18DB DAC volume -18db.
kDA7212_DACGainM12DB DAC volume -12db.
kDA7212_DACGainM6DB DAC volume -6bb.
kDA7212_DACGain0DB DAC volume +0db.
kDA7212_DACGain6DB DAC volume +6db.
kDA7212_DACGain12DB DAC volume +12db.

76.4.7 enum da7212_protocol_t

Enumerator

kDA7212_BusI2S I2S Type.
kDA7212_BusLeftJustified Left justified.
kDA7212_BusRightJustified Right Justified.
kDA7212_BusDSPMode DSP mode.

76.4.8 enum da7212_sys_clk_source_t

Enumerator

kDA7212_SysClkSourceMCLK da7212 system clock soure from MCLK
kDA7212_SysClkSourcePLL da7212 system clock soure from pLL

76.4.9 enum da7212_pll_clk_source_t

Enumerator

kDA7212_PLLClkSourceMCLK DA7212 PLL clock source from MCLK.

76.4.10 enum da7212_pll_out_clk_t

Enumerator

kDA7212_PLLOutputClk11289600 output 112896000U
kDA7212_PLLOutputClk12288000 output 12288000U

76.4.11 enum da7212_master_bits_t

Enumerator

kDA7212_MasterBits32PerFrame master mode bits32 per frame
kDA7212_MasterBits64PerFrame master mode bits64 per frame
kDA7212_MasterBits128PerFrame master mode bits128 per frame
kDA7212_MasterBits256PerFrame master mode bits256 per frame

Function Documentation

76.5.1 status_t DA7212_Init (da7212_handle_t * *handle*, da7212_config_t * *codecConfig*)

Parameters

handle	DA7212 handle pointer.

codecConfig	Codec configure structure. This parameter can be NULL, if NULL, set as default settings. The default setting:
	`* sgtl_init_t codec_config`
	`* codec_config.route = kDA7212_RoutePlayback`
	`* codec_config.bus = kDA7212_BusI2S`
	`* codec_config.isMaster = false`
	`*`

76.5.2 status_t DA7212_ConfigAudioFormat (da7212_handle_t ∗ *handle*, uint32_t *masterClock_Hz*, uint32_t *sampleRate_Hz*, uint32_t *dataBits*)

Parameters

handle	DA7212 handle pointer.
masterClock_-Hz	Master clock frequency in Hz. If DA7212 is slave, use the frequency of master, if DA7212 as master, it should be 1228000 while sample rate frequency is 8k/12K/16-K/24K/32K/48K/96K, 11289600 whie sample rate is 11.025K/22.05K/44.1K
sampleRate_Hz	Sample rate frequency in Hz.
dataBits	How many bits in a word of a audio frame, DA7212 only supports 16/20/24/32 bits.

76.5.3 status_t DA7212_SetPLLConfig (da7212_handle_t ∗ *handle*, da7212_pll_config_t ∗ *config*)

Parameters

handle	DA7212 handler pointer.
config	PLL configuration pointer.

76.5.4 void DA7212_ChangeHPVolume (da7212_handle_t ∗ *handle*, da7212_volume_t *volume*)

Parameters

handle	DA7212 handle pointer.
volume	The volume of playback.

76.5.5 void DA7212_Mute (da7212_handle_t * *handle*, bool *isMuted*)

Parameters

handle	DA7212 handle pointer.
isMuted	True means mute, false means unmute.

76.5.6 void DA7212_ChangeInput (da7212_handle_t * *handle*, da7212_Input_t *DA7212_Input*)

Parameters

handle	DA7212 handle pointer.
DA7212_Input	Input data source.

76.5.7 void DA7212_ChangeOutput (da7212_handle_t * *handle*, da7212_Output_t *DA7212_Output*)

Parameters

handle	DA7212 handle pointer.
DA7212_Output	Output device of DA7212.

76.5.8 status_t DA7212_SetChannelVolume (da7212_handle_t * *handle*, uint32_t *channel*, uint32_t *volume*)

Parameters

handle	DA7212 handle pointer.
channel	shoule be a value of _da7212_channel.
volume	volume range 0 - 0x3F mapped to range -57dB - 6dB.

76.5.9 status_t DA7212_SetChannelMute (da7212_handle_t * *handle*, uint32_t *channel*, bool *isMute*)

Parameters

handle	DA7212 handle pointer.
channel	shoule be a value of _da7212_channel.
isMute	true is mute, false is unmute.

76.5.10 status_t DA7212_SetProtocol (da7212_handle_t * *handle*, da7212_protocol_t *protocol*)

Parameters

handle	DA7212 handle pointer.
protocol	da7212_protocol_t.

76.5.11 status_t DA7212_SetMasterModeBits (da7212_handle_t * *handle*, uint32_t *bitWidth*)

Parameters

handle	DA7212 handle pointer.
bitWidth	audio data bitwidth.

76.5.12 status_t DA7212_WriteRegister (da7212_handle_t * *handle*, uint8_t *u8Register*, uint8_t *u8RegisterData*)

Parameters

handle	DA7212 handle pointer.
u8Register	DA7212 register address to be written.
u8RegisterData	Data to be written into regsiter

76.5.13 status_t DA7212_ReadRegister (da7212_handle_t * *handle*, uint8_t *u8Register*, uint8_t * *pu8RegisterData*)

Parameters

handle	DA7212 handle pointer.
u8Register	DA7212 register address to be read.
pu8Register-Data	Pointer where the read out value to be stored.

76.5.14 status_t DA7212_Deinit (da7212_handle_t * *handle*)

Parameters

handle	DA7212 handle pointer.

Chapter 77
Cs42888_adapter

Overview

Macros

- #define HAL_CODEC_HANDLER_SIZE (CS42888_I2C_HANDLER_SIZE + 4)
  codec handler size

Enumerations

- enum _codec_type {
  kCODEC_CS42888,
  kCODEC_DA7212,
  kCODEC_SGTL5000,
  kCODEC_WM8904 }
  codec type

Functions

- status_t HAL_CODEC_Init (void ∗handle, void ∗config)
  Codec initilization.
- status_t HAL_CODEC_Deinit (void ∗handle)
  Codec de-initilization.
- status_t HAL_CODEC_SetFormat (void ∗handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_SetVolume (void ∗handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_SetMute (void ∗handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_SetPower (void ∗handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

Enumeration Type Documentation

77.2.1 enum _codec_type

Enumerator

kCODEC_CS42888 CS42888.
kCODEC_DA7212 da7212
kCODEC_SGTL5000 sgtl5000
kCODEC_WM8904 wm8904

Function Documentation

77.3.1 status_t HAL_CODEC_Init (void * *handle,* void * *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

77.3.2 status_t HAL_CODEC_Deinit (void * *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

77.3.3 status_t HAL_CODEC_SetFormat (void * *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

77.3.4 status_t HAL_CODEC_SetVolume (void * *handle*, uint32_t *playChannel*, uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

77.3.5 status_t HAL_CODEC_SetMute (void * *handle*, uint32_t *playChannel*, bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

77.3.6 status_t HAL_CODEC_SetPower (void * *handle*, uint32_t *module*, bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

77.3.7 status_t HAL_CODEC_SetRecord (void * *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

77.3.8 status_t HAL_CODEC_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

77.3.9 status_t HAL_CODEC_SetPlay (void * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

77.3.10 status_t HAL_CODEC_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

Chapter 78
Da7212_adapter

Overview

Macros

- #define HAL_CODEC_HANDLER_SIZE (DA7212_I2C_HANDLER_SIZE + 4)
  codec handler size

Enumerations

- enum _codec_type {
  kCODEC_CS42888,
  kCODEC_DA7212,
  kCODEC_SGTL5000,
  kCODEC_WM8904 }
  codec type

Functions

- status_t HAL_CODEC_Init (void ∗handle, void ∗config)
  Codec initilization.
- status_t HAL_CODEC_Deinit (void ∗handle)
  Codec de-initilization.
- status_t HAL_CODEC_SetFormat (void ∗handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_SetVolume (void ∗handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_SetMute (void ∗handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_SetPower (void ∗handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

Enumeration Type Documentation

78.2.1 enum _codec_type

Enumerator

kCODEC_CS42888 CS42888.
kCODEC_DA7212 da7212
kCODEC_SGTL5000 sgtl5000
kCODEC_WM8904 wm8904

Function Documentation

78.3.1 status_t HAL_CODEC_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

78.3.2 status_t HAL_CODEC_Deinit (void ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

78.3.3 status_t HAL_CODEC_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

78.3.4 status_t HAL_CODEC_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

78.3.5 status_t HAL_CODEC_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

78.3.6 status_t HAL_CODEC_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

78.3.7 status_t HAL_CODEC_SetRecord (void * *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

78.3.8 status_t HAL_CODEC_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

78.3.9 status_t HAL_CODEC_SetPlay (void * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

78.3.10 status_t HAL_CODEC_ModuleControl (void ∗ *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

Chapter 79
Sgtl5000_adapter

Overview

Macros

- #define HAL_CODEC_HANDLER_SIZE (SGTL_I2C_HANDLER_SIZE + 4)
  codec handler size

Enumerations

- enum _codec_type {
  kCODEC_CS42888,
  kCODEC_DA7212,
  kCODEC_SGTL5000,
  kCODEC_WM8904 }
  codec type

Functions

- status_t HAL_CODEC_Init (void ∗handle, void ∗config)
  Codec initilization.
- status_t HAL_CODEC_Deinit (void ∗handle)
  Codec de-initilization.
- status_t HAL_CODEC_SetFormat (void ∗handle, uint32_t mclk, uint32_t sampleRate, uint32_t bit-Width)
  set audio data format.
- status_t HAL_CODEC_SetVolume (void ∗handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_SetMute (void ∗handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_SetPower (void ∗handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

Enumeration Type Documentation

79.2.1 enum _codec_type

Enumerator

kCODEC_CS42888 CS42888.
kCODEC_DA7212 da7212
kCODEC_SGTL5000 sgtl5000
kCODEC_WM8904 wm8904

Function Documentation

79.3.1 status_t HAL_CODEC_Init (void * *handle,* void * *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

79.3.2 status_t HAL_CODEC_Deinit (void * *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

79.3.3 status_t HAL_CODEC_SetFormat (void * *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

79.3.4 status_t HAL_CODEC_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 ∼ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

79.3.5 status_t HAL_CODEC_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

79.3.6 status_t HAL_CODEC_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

79.3.7 status_t HAL_CODEC_SetRecord (void * *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

79.3.8 status_t HAL_CODEC_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

79.3.9 status_t HAL_CODEC_SetPlay (void * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

79.3.10 status_t HAL_CODEC_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

Chapter 80
Wm8904_adapter

Overview

Macros

- #define HAL_CODEC_HANDLER_SIZE (WM8904_I2C_HANDLER_SIZE + 4)
  codec handler size

Enumerations

- enum _codec_type {
  kCODEC_CS42888,
  kCODEC_DA7212,
  kCODEC_SGTL5000,
  kCODEC_WM8904 }
  codec type

Functions

- status_t HAL_CODEC_Init (void ∗handle, void ∗config)
  Codec initilization.
- status_t HAL_CODEC_Deinit (void ∗handle)
  Codec de-initilization.
- status_t HAL_CODEC_SetFormat (void ∗handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_SetVolume (void ∗handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_SetMute (void ∗handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_SetPower (void ∗handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

Enumeration Type Documentation

80.2.1 enum _codec_type

Enumerator

kCODEC_CS42888 CS42888.
kCODEC_DA7212 da7212
kCODEC_SGTL5000 sgtl5000
kCODEC_WM8904 wm8904

Function Documentation

80.3.1 status_t HAL_CODEC_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

80.3.2 status_t HAL_CODEC_Deinit (void ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

80.3.3 status_t HAL_CODEC_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

80.3.4 status_t HAL_CODEC_SetVolume (void ∗ *handle*, uint32_t *playChannel*, uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

80.3.5 status_t HAL_CODEC_SetMute (void ∗ *handle*, uint32_t *playChannel*, bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

80.3.6 status_t HAL_CODEC_SetPower (void ∗ *handle*, uint32_t *module*, bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

80.3.7 status_t HAL_CODEC_SetRecord (void ∗ *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

80.3.8 status_t HAL_CODEC_SetRecordChannel (void ∗ *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

80.3.9 status_t HAL_CODEC_SetPlay (void ∗ *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

80.3.10 status_t HAL_CODEC_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

Chapter 81
Wm8960_adapter

Overview

Macros

- #define HAL_CODEC_HANDLER_SIZE (WM8960_I2C_HANDLER_SIZE + 4U)
  codec handler size

Enumerations

- enum { kCODEC_WM8960 }
  codec type

Functions

- status_t HAL_CODEC_Init (void ∗handle, void ∗config)
  Codec initilization.
- status_t HAL_CODEC_Deinit (void ∗handle)
  Codec de-initilization.
- status_t HAL_CODEC_SetFormat (void ∗handle, uint32_t mclk, uint32_t sampleRate, uint32_t bit-Width)
  set audio data format.
- status_t HAL_CODEC_SetVolume (void ∗handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_SetMute (void ∗handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_SetPower (void ∗handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

Enumeration Type Documentation

81.2.1 anonymous enum

Enumerator

kCODEC_WM8960 wm8960

Function Documentation

81.3.1 status_t HAL_CODEC_Init (void * *handle,* void * *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

81.3.2 status_t HAL_CODEC_Deinit (void * *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

81.3.3 status_t HAL_CODEC_SetFormat (void * *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

81.3.4 status_t HAL_CODEC_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

81.3.5 status_t HAL_CODEC_SetMute (void ∗ *handle*, uint32_t *playChannel*, bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

81.3.6 status_t HAL_CODEC_SetPower (void ∗ *handle*, uint32_t *module*, bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

81.3.7 status_t HAL_CODEC_SetRecord (void ∗ *handle*, uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

81.3.8 status_t HAL_CODEC_SetRecordChannel (void ∗ *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

81.3.9 status_t HAL_CODEC_SetPlay (void ∗ *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

81.3.10 status_t HAL_CODEC_ModuleControl (void ∗ *handle,* uint32_t *cmd,* uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

Chapter 82
Sgtl5000

Overview

Data Structures

- struct sgtl_audio_format_t
  Audio format configuration. More...
- struct sgtl_config_t
  Initailize structure of sgtl5000. More...
- struct sgtl_handle_t
  SGTL codec handler. More...

Macros

- #define CHIP_ID 0x0000U
  Define the register address of sgtl5000.
- #define SGTL5000_HEADPHONE_MAX_VOLUME_VALUE 0x7FU
  SGTL5000 volume setting range.
- #define SGTL5000_I2C_ADDR 0x0A
  SGTL5000 I2C address.
- #define SGTL_I2C_HANDLER_SIZE CODEC_I2C_MASTER_HANDLER_SIZE
  sgtl handle size
- #define SGTL_I2C_BITRATE 100000U
  sgtl i2c baudrate

Enumerations

- enum sgtl_module_t {
  kSGTL_ModuleADC = 0x0,
  kSGTL_ModuleDAC,
  kSGTL_ModuleDAP,
  kSGTL_ModuleHP,
  kSGTL_ModuleI2SIN,
  kSGTL_ModuleI2SOUT,
  kSGTL_ModuleLineIn,
  kSGTL_ModuleLineOut,
  kSGTL_ModuleMicin }
  Modules in Sgtl5000 board.
- enum sgtl_route_t {
  kSGTL_RouteBypass = 0x0,
  kSGTL_RoutePlayback,
  kSGTL_RoutePlaybackandRecord,
  kSGTL_RoutePlaybackwithDAP,
  kSGTL_RoutePlaybackwithDAPandRecord,

kSGTL_RouteRecord }
Sgtl5000 data route.
- enum sgtl_protocol_t {
kSGTL_BusI2S = 0x0,
kSGTL_BusLeftJustified,
kSGTL_BusRightJustified,
kSGTL_BusPCMA,
kSGTL_BusPCMB }
The audio data transfer protocol choice.
- enum {
kSGTL_HeadphoneLeft = 0,
kSGTL_HeadphoneRight = 1,
kSGTL_LineoutLeft = 2,
kSGTL_LineoutRight = 3 }
sgtl play channel
- enum {
kSGTL_RecordSourceLineIn = 0U,
kSGTL_RecordSourceMic = 1U }
sgtl record source _sgtl_record_source
- enum {
kSGTL_PlaySourceLineIn = 0U,
kSGTL_PlaySourceDAC = 1U }
sgtl play source _stgl_play_source
- enum sgtl_sclk_edge_t {
kSGTL_SclkValidEdgeRising = 0U,
kSGTL_SclkValidEdgeFailling = 1U }
SGTL SCLK valid edge.

Functions

- status_t SGTL_Init (sgtl_handle_t ∗handle, sgtl_config_t ∗config)
sgtl5000 initialize function.
- status_t SGTL_SetDataRoute (sgtl_handle_t ∗handle, sgtl_route_t route)
Set audio data route in sgtl5000.
- status_t SGTL_SetProtocol (sgtl_handle_t ∗handle, sgtl_protocol_t protocol)
Set the audio transfer protocol.
- void SGTL_SetMasterSlave (sgtl_handle_t ∗handle, bool master)
Set sgtl5000 as master or slave.
- status_t SGTL_SetVolume (sgtl_handle_t ∗handle, sgtl_module_t module, uint32_t volume)
Set the volume of different modules in sgtl5000.
- uint32_t SGTL_GetVolume (sgtl_handle_t ∗handle, sgtl_module_t module)
Get the volume of different modules in sgtl5000.
- status_t SGTL_SetMute (sgtl_handle_t ∗handle, sgtl_module_t module, bool mute)
Mute/unmute modules in sgtl5000.
- status_t SGTL_EnableModule (sgtl_handle_t ∗handle, sgtl_module_t module)
Enable expected devices.
- status_t SGTL_DisableModule (sgtl_handle_t ∗handle, sgtl_module_t module)
Disable expected devices.
- status_t SGTL_Deinit (sgtl_handle_t ∗handle)

Deinit the sgtl5000 codec.
- status_t SGTL_ConfigDataFormat (sgtl_handle_t ∗handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)
  Configure the data format of audio data.
- status_t SGTL_SetPlay (sgtl_handle_t ∗handle, uint32_t playSource)
  select SGTL codec play source.
- status_t SGTL_SetRecord (sgtl_handle_t ∗handle, uint32_t recordSource)
  select SGTL codec record source.
- status_t SGTL_WriteReg (sgtl_handle_t ∗handle, uint16_t reg, uint16_t val)
  Write register to sgtl using I2C.
- status_t SGTL_ReadReg (sgtl_handle_t ∗handle, uint16_t reg, uint16_t ∗val)
  Read register from sgtl using I2C.
- status_t SGTL_ModifyReg (sgtl_handle_t ∗handle, uint16_t reg, uint16_t clr_mask, uint16_t val)
  Modify some bits in the register using I2C.

Driver version

- #define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  CLOCK driver version 2.1.1.

Data Structure Documentation

82.2.1 struct sgtl_audio_format_t

Data Fields

- uint32_t mclk_HZ
  master clock
- uint32_t sampleRate
  Sample rate.
- uint32_t bitWidth
  Bit width.
- sgtl_sclk_edge_t sclkEdge
  sclk valid edge

82.2.2 struct sgtl_config_t

Data Fields

- sgtl_route_t route
  Audio data route.
- sgtl_protocol_t bus
  Audio transfer protocol.
- bool master_slave
  Master or slave.
- sgtl_audio_format_t format
  audio format
- uint8_t slaveAddress

code device slave address
- codec_i2c_config_t i2cConfig
  i2c bus configuration

82.2.2.0.0.65 Field Documentation

82.2.2.0.0.65.1 sgtl_route_t sgtl_config_t::route

82.2.2.0.0.65.2 bool sgtl_config_t::master_slave

True means master, false means slave.

82.2.3 struct sgtl_handle_t

Data Fields

- sgtl_config_t ∗ config
  sgtl config pointer
- uint8_t i2cHandle [SGTL_I2C_HANDLER_SIZE]
  i2c handle

Macro Definition Documentation

82.3.1 #define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

82.3.2 #define CHIP_ID 0x0000U

82.3.3 #define SGTL5000_I2C_ADDR 0x0A

Enumeration Type Documentation

82.4.1 enum sgtl_module_t

Enumerator

kSGTL_ModuleADC ADC module in SGTL5000.
kSGTL_ModuleDAC DAC module in SGTL5000.
kSGTL_ModuleDAP DAP module in SGTL5000.
kSGTL_ModuleHP Headphone module in SGTL5000.
kSGTL_ModuleI2SIN I2S-IN module in SGTL5000.
kSGTL_ModuleI2SOUT I2S-OUT module in SGTL5000.
kSGTL_ModuleLineIn Line-in moudle in SGTL5000.
kSGTL_ModuleLineOut Line-out module in SGTL5000.
kSGTL_ModuleMicin Micphone module in SGTL5000.

82.4.2 enum sgtl_route_t

Note

Only provide some typical data route, not all route listed. Users cannot combine any routes, once a new route is set, the precios one would be replaced.

Enumerator

kSGTL_RouteBypass LINEIN->Headphone.
kSGTL_RoutePlayback I2SIN->DAC->Headphone.
kSGTL_RoutePlaybackandRecord I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.
kSGTL_RoutePlaybackwithDAP I2SIN->DAP->DAC->Headphone.
kSGTL_RoutePlaybackwithDAPandRecord I2SIN->DAP->DAC->HP, LINEIN->ADC->I2SO-UT.
kSGTL_RouteRecord LINEIN->ADC->I2SOUT.

82.4.3 enum sgtl_protocol_t

Sgtl5000 only supports I2S format and PCM format.

Enumerator

kSGTL_BusI2S I2S Type.
kSGTL_BusLeftJustified Left justified.
kSGTL_BusRightJustified Right Justified.
kSGTL_BusPCMA PCMA.
kSGTL_BusPCMB PCMB.

82.4.4 anonymous enum

Enumerator

kSGTL_HeadphoneLeft headphone left channel
kSGTL_HeadphoneRight headphone right channel
kSGTL_LineoutLeft lineout left channel
kSGTL_LineoutRight lineout right channel

82.4.5 anonymous enum

Enumerator

kSGTL_RecordSourceLineIn record source line in
kSGTL_RecordSourceMic record source single end

82.4.6 anonymous enum

Enumerator

kSGTL_PlaySourceLineIn play source line in
kSGTL_PlaySourceDAC play source line in

82.4.7 enum sgtl_sclk_edge_t

Enumerator

kSGTL_SclkValidEdgeRising SCLK valid edge.
kSGTL_SclkValidEdgeFailling SCLK failling edge.

Function Documentation

82.5.1 status_t SGTL_Init (sgtl_handle_t ∗ *handle,* sgtl_config_t ∗ *config*)

This function calls SGTL_I2CInit(), and in this function, some configurations are fixed. The second parameter can be NULL. If users want to change the SGTL5000 settings, a configure structure should be prepared.

Note

If the codec_config is NULL, it would initialize sgtl5000 using default settings. The default setting:

```
* sgtl_init_t codec_config
* codec_config.route = kSGTL_RoutePlaybackandRecord
* codec_config.bus = kSGTL_BusI2S
* codec_config.master = slave
*
```

Parameters

handle	Sgtl5000 handle structure.
config	sgtl5000 configuration structure. If this pointer equals to NULL, it means using the default configuration.

Returns

Initialization status

82.5.2 status_t SGTL_SetDataRoute (sgtl_handle_t * *handle*, sgtl_route_t *route*)

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules.

Note

If a new route is set, the previous route would not work.

Parameters

handle	Sgtl5000 handle structure.
route	Audio data route in sgtl5000.

82.5.3 status_t SGTL_SetProtocol (sgtl_handle_t * *handle*, sgtl_protocol_t *protocol*)

Sgtl5000 only supports I2S, I2S left, I2S right, PCM A, PCM B format.

Parameters

handle	Sgtl5000 handle structure.
protocol	Audio data transfer protocol.

82.5.4 void SGTL_SetMasterSlave (sgtl_handle_t * *handle*, bool *master*)

Parameters

handle	Sgtl5000 handle structure.
master	1 represent master, 0 represent slave.

82.5.5 status_t SGTL_SetVolume (sgtl_handle_t * *handle*, sgtl_module_t *module*, uint32_t *volume*)

This function would set the volume of sgtl5000 modules. This interface set module volume. The function assume that left channel and right channel has the same volume.

kSGTL_ModuleADC volume range: 0 - 0xF, 0dB - 22.5dB kSGTL_ModuleDAC volume range: 0x3C - 0xF0, 0dB - -90dB kSGTL_ModuleHP volume range: 0 - 0x7F, 12dB - -51.5dB kSGTL_ModuleLineOut volume range: 0 - 0x1F, 0.5dB steps

Parameters

handle	Sgtl5000 handle structure.
module	Sgtl5000 module, such as DAC, ADC and etc.
volume	Volume value need to be set. The value is the exact value in register.

82.5.6 uint32_t SGTL_GetVolume (sgtl_handle_t ∗ *handle,* sgtl_module_t *module*)

This function gets the volume of sgtl5000 modules. This interface get DAC module volume. The function assume that left channel and right channel has the same volume.

Parameters

handle	Sgtl5000 handle structure.
module	Sgtl5000 module, such as DAC, ADC and etc.

Returns

Module value, the value is exact value in register.

82.5.7 status_t SGTL_SetMute (sgtl_handle_t ∗ *handle,* sgtl_module_t *module,* bool *mute*)

Parameters

handle	Sgtl5000 handle structure.
module	Sgtl5000 module, such as DAC, ADC and etc.
mute	True means mute, and false means unmute.

82.5.8 status_t SGTL_EnableModule (sgtl_handle_t ∗ *handle,* sgtl_module_t *module*)

Parameters

handle	Sgtl5000 handle structure.
module	Module expected to enable.

82.5.9 status_t SGTL_DisableModule (sgtl_handle_t ∗ *handle,* sgtl_module_t *module*)

Parameters

handle	Sgtl5000 handle structure.
module	Module expected to enable.

82.5.10 status_t SGTL_Deinit (sgtl_handle_t ∗ *handle*)

Shut down Sgtl5000 modules.

Parameters

handle	Sgtl5000 handle structure pointer.

82.5.11 status_t SGTL_ConfigDataFormat (sgtl_handle_t ∗ *handle,* uint32_t *mclk,* uint32_t *sample_rate,* uint32_t *bits*)

This function would configure the registers about the sample rate, bit depths.

Parameters

handle	Sgtl5000 handle structure pointer.
mclk	Master clock frequency of I2S.
sample_rate	Sample rate of audio file running in sgtl5000. Sgtl5000 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
bits	Bit depth of audio file (Sgtl5000 only supports 16bit, 20bit, 24bit and 32 bit in HW).

82.5.12 status_t SGTL_SetPlay (sgtl_handle_t ∗ *handle,* uint32_t *playSource*)

Parameters

handle	Sgtl5000 handle structure pointer.
playSource	play source value, reference _sgtl_play_source.

Returns

kStatus_Success, else failed.

82.5.13 status_t SGTL_SetRecord (sgtl_handle_t ∗ *handle,* uint32_t *recordSource*)

Parameters

handle	Sgtl5000 handle structure pointer.
recordSource	record source value, reference _sgtl_record_source.

Returns

kStatus_Success, else failed.

82.5.14 status_t SGTL_WriteReg (sgtl_handle_t ∗ *handle,* uint16_t *reg,* uint16_t *val*)

Parameters

handle	Sgtl5000 handle structure.
reg	The register address in sgtl.
val	Value needs to write into the register.

82.5.15 status_t SGTL_ReadReg (sgtl_handle_t ∗ *handle,* uint16_t *reg,* uint16_t ∗ *val*)

Parameters

handle	Sgtl5000 handle structure.
reg	The register address in sgtl.
val	Value written to.

82.5.16 status_t SGTL_ModifyReg (sgtl_handle_t * *handle*, uint16_t *reg*, uint16_t *clr_mask*, uint16_t *val*)

Parameters

handle	Sgtl5000 handle structure.
reg	The register address in sgtl.
clr_mask	The mask code for the bits want to write. The bit you want to write should be 0.
val	Value needs to write into the register.

Chapter 83
Wm8904

Overview

Data Structures

- struct wm8904_fll_config_t
  wm8904 fll configuration More...
- struct wm8904_audio_format_t
  Audio format configuration. More...
- struct wm8904_config_t
  Configuration structure of WM8904. More...
- struct wm8904_handle_t
  wm8904 codec handler More...

Macros

- #define WM8904_I2C_HANDLER_SIZE (CODEC_I2C_MASTER_HANDLER_SIZE)
  wm8904 handle size
- #define WM8904_DEBUG_REGISTER 0
  wm8904 debug macro
- #define WM8904_RESET (0x00)
  WM8904 register map.
- #define WM8904_I2C_ADDRESS (0x1A)
  WM8904 I2C address.
- #define WM8904_I2C_BITRATE (400000U)
  WM8904 I2C bit rate.
- #define WM8904_MAP_HEADPHONE_LINEOUT_MAX_VOLUME 0x3FU
  WM8904 maximum headphone/lineout volume.

Enumerations

- enum {
  kStatus_WM8904_Success = 0x0,
  kStatus_WM8904_Fail = 0x1 }
  WM8904 status return codes.
- enum {
  kWM8904_LRCPolarityNormal = 0U,
  kWM8904_LRCPolarityInverted = 1U << 4U }
  WM8904 lrc polarity.
- enum wm8904_module_t {
  kWM8904_ModuleADC = 0,
  kWM8904_ModuleDAC = 1,
  kWM8904_ModulePGA = 2,
  kWM8904_ModuleHeadphone = 3,
  kWM8904_ModuleLineout = 4 }

wm8904 module value
- enum
  wm8904 play channel
- enum wm8904_timeslot_t {
  kWM8904_TimeSlot0 = 0U,
  kWM8904_TimeSlot1 = 1U }
  WM8904 time slot.
- enum wm8904_protocol_t {
  kWM8904_ProtocolI2S = 0x2,
  kWM8904_ProtocolLeftJustified = 0x1,
  kWM8904_ProtocolRightJustified = 0x0,
  kWM8904_ProtocolPCMA = 0x3,
  kWM8904_ProtocolPCMB = 0x3 | (1 << 4) }
  The audio data transfer protocol.
- enum wm8904_fs_ratio_t {
  kWM8904_FsRatio64X = 0x0,
  kWM8904_FsRatio128X = 0x1,
  kWM8904_FsRatio192X = 0x2,
  kWM8904_FsRatio256X = 0x3,
  kWM8904_FsRatio384X = 0x4,
  kWM8904_FsRatio512X = 0x5,
  kWM8904_FsRatio768X = 0x6,
  kWM8904_FsRatio1024X = 0x7,
  kWM8904_FsRatio1408X = 0x8,
  kWM8904_FsRatio1536X = 0x9 }
  The SYSCLK / fs ratio.
- enum wm8904_sample_rate_t {
  kWM8904_SampleRate8kHz = 0x0,
  kWM8904_SampleRate12kHz = 0x1,
  kWM8904_SampleRate16kHz = 0x2,
  kWM8904_SampleRate24kHz = 0x3,
  kWM8904_SampleRate32kHz = 0x4,
  kWM8904_SampleRate48kHz = 0x5 }
  Sample rate.
- enum wm8904_bit_width_t {
  kWM8904_BitWidth16 = 0x0,
  kWM8904_BitWidth20 = 0x1,
  kWM8904_BitWidth24 = 0x2,
  kWM8904_BitWidth32 = 0x3 }
  Bit width.
- enum {
  kWM8904_RecordSourceDifferentialLine = 1U,
  kWM8904_RecordSourceLineInput = 2U,
  kWM8904_RecordSourceDifferentialMic = 4U,
  kWM8904_RecordSourceDigitalMic = 8U }
  wm8904 record source
- enum {

kWM8904_RecordChannelLeft1 = 1U,
kWM8904_RecordChannelLeft2 = 2U,
kWM8904_RecordChannelLeft3 = 4U,
kWM8904_RecordChannelRight1 = 1U,
kWM8904_RecordChannelRight2 = 2U,
kWM8904_RecordChannelRight3 = 4U,
kWM8904_RecordChannelDifferentialPositive1 = 1U,
kWM8904_RecordChannelDifferentialPositive2 = 2U,
kWM8904_RecordChannelDifferentialPositive3 = 4U,
kWM8904_RecordChannelDifferentialNegative1 = 8U,
kWM8904_RecordChannelDifferentialNegative2 = 16U,
kWM8904_RecordChannelDifferentialNegative3 = 32U }
  wm8904 record channel
- enum {
  kWM8904_PlaySourcePGA = 1U,
  kWM8904_PlaySourceDAC = 4U }
  wm8904 play source
- enum wm8904_sys_clk_source_t {
  kWM8904_SysClkSourceMCLK = 0U,
  kWM8904_SysClkSourceFLL = 1U << 14 }
  wm8904 system clock source
- enum wm8904_fll_clk_source_t { kWM8904_FLLClkSourceMCLK = 0U }
  wm8904 fll clock source

Functions

- status_t WM8904_WriteRegister (wm8904_handle_t ∗handle, uint8_t reg, uint16_t value)
  WM8904 write register.
- status_t WM8904_ReadRegister (wm8904_handle_t ∗handle, uint8_t reg, uint16_t ∗value)
  WM8904 write register.
- status_t WM8904_ModifyRegister (wm8904_handle_t ∗handle, uint8_t reg, uint16_t mask, uint16-_t value)
  WM8904 modify register.
- status_t WM8904_Init (wm8904_handle_t ∗handle, wm8904_config_t ∗wm8904Config)
  Initializes WM8904.
- status_t WM8904_Deinit (wm8904_handle_t ∗handle)
  Deinitializes the WM8904 codec.
- void WM8904_GetDefaultConfig (wm8904_config_t ∗config)
  Fills the configuration structure with default values.
- status_t WM8904_SetMasterSlave (wm8904_handle_t ∗handle, bool master)
  Sets WM8904 as master or slave.
- status_t WM8904_SeMasterClock (wm8904_handle_t ∗handle, uint32_t sysclk, uint32_t sample-Rate, uint32_t bitWidth)
  Sets WM8904 master clock configuration.
- status_t WM8904_SetFLLConfig (wm8904_handle_t ∗handle, wm8904_fll_config_t ∗config)
  WM8904 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.
- status_t WM8904_SetProtocol (wm8904_handle_t ∗handle, wm8904_protocol_t protocol)
  Sets the audio data transfer protocol.

- status_t WM8904_SetAudioFormat (wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth)
  Sets the audio data format.
- status_t WM8904_CheckAudioFormat (wm8904_handle_t *handle, wm8904_audio_format_t *format, uint32_t mclkFreq)
  check and update the audio data format.
- status_t WM8904_SetVolume (wm8904_handle_t *handle, uint16_t volumeLeft, uint16_t volumeRight)
  Sets the module output volume.
- status_t WM8904_SetMute (wm8904_handle_t *handle, bool muteLeft, bool muteRight)
  Sets the headphone output mute.
- status_t WM8904_SelectLRCPolarity (wm8904_handle_t *handle, uint32_t polarity)
  Select LRC polarity.
- status_t WM8904_EnableDACTDMMode (wm8904_handle_t *handle, wm8904_timeslot_t timeSlot)
  Enable WM8904 DAC time slot.
- status_t WM8904_EnableADCTDMMode (wm8904_handle_t *handle, wm8904_timeslot_t timeSlot)
  Enable WM8904 ADC time slot.
- status_t WM8904_SetModulePower (wm8904_handle_t *handle, wm8904_module_t module, bool isEnabled)
  brief SET the module output power.
- status_t WM8904_SetChannelVolume (wm8904_handle_t *handle, uint32_t channel, uint32_t volume)
  Sets the channel output volume.
- status_t WM8904_SetRecord (wm8904_handle_t *handle, uint32_t recordSource)
  SET the WM8904 record source.
- status_t WM8904_SetRecordChannel (wm8904_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  SET the WM8904 record source.
- status_t WM8904_SetPlay (wm8904_handle_t *handle, uint32_t playSource)
  SET the WM8904 play source.
- status_t WM8904_SetChannelMute (wm8904_handle_t *handle, uint32_t channel, bool isMute)
  Sets the channel mute.

Driver version

- #define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 4, 3))
  WM8904 driver version 2.4.3.

Data Structure Documentation

83.2.1 struct wm8904_fll_config_t

Data Fields

- wm8904_fll_clk_source_t source
  fll reference clock source
- uint32_t refClock_HZ

fll reference clock frequency
- uint32_t outputClock_HZ
  fll output clock frequency

83.2.2 struct wm8904_audio_format_t

Data Fields

- wm8904_fs_ratio_t fsRatio
  SYSCLK / fs ratio.
- wm8904_sample_rate_t sampleRate
  Sample rate.
- wm8904_bit_width_t bitWidth
  Bit width.

83.2.3 struct wm8904_config_t

Data Fields

- bool master
  Master or slave.
- wm8904_sys_clk_source_t sysClkSource
  system clock source
- wm8904_fll_config_t ∗ fll
  fll configuration
- wm8904_protocol_t protocol
  Audio transfer protocol.
- wm8904_audio_format_t format
  Audio format.
- uint32_t mclk_HZ
  MCLK frequency value.
- uint16_t recordSource
  record source
- uint16_t recordChannelLeft
  record channel
- uint16_t recordChannelRight
  record channel
- uint16_t playSource
  play source
- uint8_t slaveAddress
  code device slave address
- codec_i2c_config_t i2cConfig
  i2c bus configuration

83.2.4 struct wm8904_handle_t

Data Fields

- wm8904_config_t * config
  wm8904 config pointer
- uint8_t i2cHandle [WM8904_I2C_HANDLER_SIZE]
  i2c handle

Macro Definition Documentation

83.3.1 #define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 4, 3))

83.3.2 #define WM8904_I2C_ADDRESS (0x1A)

83.3.3 #define WM8904_I2C_BITRATE (400000U)

Enumeration Type Documentation

83.4.1 anonymous enum

Enumerator

kStatus_WM8904_Success Success.
kStatus_WM8904_Fail Failure.

83.4.2 anonymous enum

Enumerator

kWM8904_LRCPolarityNormal LRC polarity normal.
kWM8904_LRCPolarityInverted LRC polarity inverted.

83.4.3 enum wm8904_module_t

Enumerator

kWM8904_ModuleADC moduel ADC
kWM8904_ModuleDAC module DAC
kWM8904_ModulePGA module PGA
kWM8904_ModuleHeadphone module headphone
kWM8904_ModuleLineout module line out

83.4.4 anonymous enum

83.4.5 enum wm8904_timeslot_t

Enumerator

kWM8904_TimeSlot0 time slot0
kWM8904_TimeSlot1 time slot1

83.4.6 enum wm8904_protocol_t

Enumerator

kWM8904_ProtocolI2S I2S type.
kWM8904_ProtocolLeftJustified Left justified mode.
kWM8904_ProtocolRightJustified Right justified mode.
kWM8904_ProtocolPCMA PCM A mode.
kWM8904_ProtocolPCMB PCM B mode.

83.4.7 enum wm8904_fs_ratio_t

Enumerator

kWM8904_FsRatio64X SYSCLK is 64 ∗ sample rate ∗ frame width.
kWM8904_FsRatio128X SYSCLK is 128 ∗ sample rate ∗ frame width.
kWM8904_FsRatio192X SYSCLK is 192 ∗ sample rate ∗ frame width.
kWM8904_FsRatio256X SYSCLK is 256 ∗ sample rate ∗ frame width.
kWM8904_FsRatio384X SYSCLK is 384 ∗ sample rate ∗ frame width.
kWM8904_FsRatio512X SYSCLK is 512 ∗ sample rate ∗ frame width.
kWM8904_FsRatio768X SYSCLK is 768 ∗ sample rate ∗ frame width.
kWM8904_FsRatio1024X SYSCLK is 1024 ∗ sample rate ∗ frame width.
kWM8904_FsRatio1408X SYSCLK is 1408 ∗ sample rate ∗ frame width.
kWM8904_FsRatio1536X SYSCLK is 1536 ∗ sample rate ∗ frame width.

83.4.8 enum wm8904_sample_rate_t

Enumerator

kWM8904_SampleRate8kHz 8 kHz
kWM8904_SampleRate12kHz 11.025kHz, 12kHz
kWM8904_SampleRate16kHz 16kHz

kWM8904_SampleRate24kHz 22.05kHz, 24kHz
kWM8904_SampleRate32kHz 32kHz
kWM8904_SampleRate48kHz 44.1kHz, 48kHz

83.4.9 enum wm8904_bit_width_t

Enumerator

kWM8904_BitWidth16 16 bits
kWM8904_BitWidth20 20 bits
kWM8904_BitWidth24 24 bits
kWM8904_BitWidth32 32 bits

83.4.10 anonymous enum

Enumerator

kWM8904_RecordSourceDifferentialLine record source from differential line
kWM8904_RecordSourceLineInput record source from line input
kWM8904_RecordSourceDifferentialMic record source from differential mic
kWM8904_RecordSourceDigitalMic record source from digital microphone

83.4.11 anonymous enum

Enumerator

kWM8904_RecordChannelLeft1 left record channel 1
kWM8904_RecordChannelLeft2 left record channel 2
kWM8904_RecordChannelLeft3 left record channel 3
kWM8904_RecordChannelRight1 right record channel 1
kWM8904_RecordChannelRight2 right record channel 2
kWM8904_RecordChannelRight3 right record channel 3
kWM8904_RecordChannelDifferentialPositive1 differential positive record channel 1
kWM8904_RecordChannelDifferentialPositive2 differential positive record channel 2
kWM8904_RecordChannelDifferentialPositive3 differential positive record channel 3
kWM8904_RecordChannelDifferentialNegative1 differential negative record channel 1
kWM8904_RecordChannelDifferentialNegative2 differential negative record channel 2
kWM8904_RecordChannelDifferentialNegative3 differential negative record channel 3

83.4.12 anonymous enum

Enumerator

kWM8904_PlaySourcePGA play source PGA, bypass ADC
kWM8904_PlaySourceDAC play source Input3

83.4.13 enum wm8904_sys_clk_source_t

Enumerator

kWM8904_SysClkSourceMCLK wm8904 system clock soure from MCLK
kWM8904_SysClkSourceFLL wm8904 system clock soure from FLL

83.4.14 enum wm8904_fll_clk_source_t

Enumerator

kWM8904_FLLClkSourceMCLK wm8904 FLL clock source from MCLK

Function Documentation

83.5.1 status_t WM8904_WriteRegister (wm8904_handle_t ∗ *handle*, uint8_t *reg*, uint16_t *value*)

Parameters

handle	WM8904 handle structure.
reg	register address.
value	value to write.

Returns

kStatus_Success, else failed.

83.5.2 status_t WM8904_ReadRegister (wm8904_handle_t ∗ *handle*, uint8_t *reg*, uint16_t ∗ *value*)

Parameters

handle	WM8904 handle structure.
reg	register address.
value	value to read.

Returns

kStatus_Success, else failed.

83.5.3 status_t WM8904_ModifyRegister (wm8904_handle_t * *handle*, uint8_t *reg*, uint16_t *mask*, uint16_t *value*)

Parameters

handle	WM8904 handle structure.
reg	register address.
mask	register bits mask.
value	value to write.

Returns

kStatus_Success, else failed.

83.5.4 status_t WM8904_Init (wm8904_handle_t * *handle*, wm8904_config_t * *wm8904Config*)

Parameters

handle	WM8904 handle structure.
wm8904Config	WM8904 configuration structure.

83.5.5 status_t WM8904_Deinit (wm8904_handle_t * *handle*)

This function resets WM8904.

Parameters

handle	WM8904 handle structure.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.6 void WM8904_GetDefaultConfig (wm8904_config_t ∗ *config*)

The default values are:

master = false; protocol = kWM8904_ProtocolI2S; format.fsRatio = kWM8904_FsRatio64X; format.-sampleRate = kWM8904_SampleRate48kHz; format.bitWidth = kWM8904_BitWidth16;

Parameters

config	default configurations of wm8904.

83.5.7 status_t WM8904_SetMasterSlave (wm8904_handle_t ∗ *handle,* bool *master*)

Deprecated DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY WM8904_Se-MasterClock

Parameters

handle	WM8904 handle structure.
master	true for master, false for slave.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.8 status_t WM8904_SeMasterClock (wm8904_handle_t ∗ *handle,* uint32_t *sysclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	WM8904 handle structure.
sysclk	system clock rate.
sampleRate	sample rate
bitWidth	bit width

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.9 status_t WM8904_SetFLLConfig (wm8904_handle_t ∗ *handle,* wm8904_fll_config_t ∗ *config*)

Parameters

handle	wm8904 handler pointer.
config	FLL configuration pointer.

83.5.10 status_t WM8904_SetProtocol (wm8904_handle_t ∗ *handle,* wm8904_protocol_t *protocol*)

Parameters

handle	WM8904 handle structure.
protocol	Audio transfer protocol.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.11 status_t WM8904_SetAudioFormat (wm8904_handle_t ∗ *handle,* uint32_t *sysclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	WM8904 handle structure.
sysclk	System clock frequency for codec, user should pay attention to this parater, sysclk is caculate as SYSCLK = MCLK / MCLKDIV, MCLKDIV is bit0 of WM8904_CLK_-RATES_0.
sampleRate	Sample rate frequency in Hz.
bitWidth	Audio data bit width.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.12 status_t WM8904_CheckAudioFormat (wm8904_handle_t ∗ *handle,* wm8904_audio_format_t ∗ *format,* uint32_t *mclkFreq*)

This api is used check the fsRatio setting based on the mclk and sample rate, if fsRatio setting is not correct, it will correct it according to mclk and sample rate.

Parameters

handle	WM8904 handle structure.
format	audio data format
mclkFreq	mclk frequency

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.13 status_t WM8904_SetVolume (wm8904_handle_t ∗ *handle,* uint16_t *volumeLeft,* uint16_t *volumeRight*)

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57DB, 63 for 6DB.

Parameters

handle	WM8904 handle structure.
volumeLeft	left channel volume.
volumeRight	right channel volume.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.14 status_t WM8904_SetMute (wm8904_handle_t * *handle,* bool *muteLeft,* bool *muteRight*)

Parameters

handle	WM8904 handle structure.
muteLeft	true to mute left channel, false to unmute.
muteRight	true to mute right channel, false to unmute.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.15 status_t WM8904_SelectLRCPolarity (wm8904_handle_t * *handle,* uint32_t *polarity*)

Parameters

handle	WM8904 handle structure.
polarity	LRC clock polarity.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.16 status_t WM8904_EnableDACTDMMode (wm8904_handle_t * *handle,* wm8904_timeslot_t *timeSlot*)

Parameters

handle	WM8904 handle structure.
timeSlot	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.17 status_t WM8904_EnableADCTDMMode (wm8904_handle_t * *handle*, wm8904_timeslot_t *timeSlot*)

Parameters

handle	WM8904 handle structure.
timeSlot	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.18 status_t WM8904_SetModulePower (wm8904_handle_t * *handle*, wm8904_module_t *module*, bool *isEnabled*)

param handle WM8904 handle structure. param module wm8904 module. param isEnabled, true is power on, false is power down.

return kStatus_WM8904_Success if successful, different code otherwise..

83.5.19 status_t WM8904_SetChannelVolume (wm8904_handle_t * *handle*, uint32_t *channel*, uint32_t *volume*)

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57dB, 63 for 6DB.

Parameters

handle	codec handle structure.
channel	codec channel.
volume	volume value from 0 -63.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.20 status_t WM8904_SetRecord (wm8904_handle_t * *handle,* uint32_t *recordSource*)

Parameters

handle	WM8904 handle structure.
recordSource	record source , can be a value of kCODEC_ModuleRecordSourceDifferential-Line, kCODEC_ModuleRecordSourceDifferentialMic, kCODEC_ModuleRecord-SourceSingleEndMic, kCODEC_ModuleRecordSourceDigitalMic.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

83.5.21 status_t WM8904_SetRecordChannel (wm8904_handle_t * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	WM8904 handle structure.
leftRecord-Channel	channel number of left record channel when using differential source, channel number of single end left channel when using single end source, channel number of digital mic when using digital mic source.

rightRecord-Channel	channel number of right record channel when using differential source, channel number of single end right channel when using single end source.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

83.5.22 status_t WM8904_SetPlay (wm8904_handle_t ∗ *handle,* uint32_t *playSource*)

Parameters

handle	WM8904 handle structure.
playSource	play source , can be a value of kCODEC_ModuleHeadphoneSourcePGA, kCODEC-_ModuleHeadphoneSourceDAC, kCODEC_ModuleLineoutSourcePGA, kCODEC-_ModuleLineoutSourceDAC.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

83.5.23 status_t WM8904_SetChannelMute (wm8904_handle_t ∗ *handle,* uint32_t *channel,* bool *isMute*)

Parameters

handle	codec handle structure.
channel	codec module name.
isMute	true is mute, false unmute.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

Chapter 84
Wm8960

Overview

Data Structures

- struct wm8960_audio_format_t
  wm8960 audio format More...
- struct wm8960_config_t
  Initialize structure of WM8960. More...
- struct wm8960_handle_t
  wm8960 codec handler More...

Macros

- #define WM8960_I2C_HANDLER_SIZE CODEC_I2C_MASTER_HANDLER_SIZE
  wm8960 handle size
- #define WM8960_LINVOL 0x0U
  Define the register address of WM8960.
- #define WM8960_CACHEREGNUM 56U
  Cache register number.
- #define WM8960_IFACE1_FORMAT_MASK 0x03U
  WM8960_IFACE1 FORMAT bits.
- #define WM8960_IFACE1_WL_MASK 0x0CU
  WM8960_IFACE1 WL bits.
- #define WM8960_IFACE1_LRP_MASK 0x10U
  WM8960_IFACE1 LRP bit.
- #define WM8960_IFACE1_DLRSWAP_MASK 0x20U
  WM8960_IFACE1 DLRSWAP bit.
- #define WM8960_IFACE1_MS_MASK 0x40U
  WM8960_IFACE1 MS bit.
- #define WM8960_IFACE1_BCLKINV_MASK 0x80U
  WM8960_IFACE1 BCLKINV bit.
- #define WM8960_IFACE1_ALRSWAP_MASK 0x100U
  WM8960_IFACE1 ALRSWAP bit.
- #define WM8960_POWER1_VREF_MASK 0x40U
  WM8960_POWER1.
- #define WM8960_POWER2_DACL_MASK 0x100U
  WM8960_POWER2.
- #define WM8960_I2C_ADDR 0x1A
  WM8960 I2C address.
- #define WM8960_I2C_BAUDRATE (100000U)
  WM8960 I2C baudrate.
- #define WM8960_ADC_MAX_VOLUME_vALUE 0xFFU
  WM8960 maximum volume value.

Enumerations

- enum wm8960_module_t {
  kWM8960_ModuleADC = 0,
  kWM8960_ModuleDAC = 1,
  kWM8960_ModuleVREF = 2,
  kWM8960_ModuleHP = 3,
  kWM8960_ModuleMICB = 4,
  kWM8960_ModuleMIC = 5,
  kWM8960_ModuleLineIn = 6,
  kWM8960_ModuleLineOut = 7,
  kWM8960_ModuleSpeaker = 8,
  kWM8960_ModuleOMIX = 9 }
  Modules in WM8960 board.
- enum {
  kWM8960_HeadphoneLeft = 1,
  kWM8960_HeadphoneRight = 2,
  kWM8960_SpeakerLeft = 4,
  kWM8960_SpeakerRight = 8 }
  wm8960 play channel
- enum wm8960_play_source_t {
  kWM8960_PlaySourcePGA = 1,
  kWM8960_PlaySourceInput = 2,
  kWM8960_PlaySourceDAC = 4 }
  wm8960 play source
- enum wm8960_route_t {
  kWM8960_RouteBypass = 0,
  kWM8960_RoutePlayback = 1,
  kWM8960_RoutePlaybackandRecord = 2,
  kWM8960_RouteRecord = 5 }
  WM8960 data route.
- enum wm8960_protocol_t {
  kWM8960_BusI2S = 2,
  kWM8960_BusLeftJustified = 1,
  kWM8960_BusRightJustified = 0,
  kWM8960_BusPCMA = 3,
  kWM8960_BusPCMB = 3 | (1 << 4) }
  The audio data transfer protocol choice.
- enum wm8960_input_t {
  kWM8960_InputClosed = 0,
  kWM8960_InputSingleEndedMic = 1,
  kWM8960_InputDifferentialMicInput2 = 2,
  kWM8960_InputDifferentialMicInput3 = 3,
  kWM8960_InputLineINPUT2 = 4,
  kWM8960_InputLineINPUT3 = 5 }
  wm8960 input source
- enum {

kWM8960_AudioSampleRate8KHz = 8000U,
kWM8960_AudioSampleRate11025Hz = 11025U,
kWM8960_AudioSampleRate12KHz = 12000U,
kWM8960_AudioSampleRate16KHz = 16000U,
kWM8960_AudioSampleRate22050Hz = 22050U,
kWM8960_AudioSampleRate24KHz = 24000U,
kWM8960_AudioSampleRate32KHz = 32000U,
kWM8960_AudioSampleRate44100Hz = 44100U,
kWM8960_AudioSampleRate48KHz = 48000U,
kWM8960_AudioSampleRate96KHz = 96000U,
kWM8960_AudioSampleRate192KHz = 192000U,
kWM8960_AudioSampleRate384KHz = 384000U }
audio sample rate definition
- enum {
kWM8960_AudioBitWidth16bit = 16U,
kWM8960_AudioBitWidth20bit = 20U,
kWM8960_AudioBitWidth24bit = 24U,
kWM8960_AudioBitWidth32bit = 32U }
audio bit width

Functions

- status_t WM8960_Init (wm8960_handle_t ∗handle, const wm8960_config_t ∗config)
WM8960 initialize function.
- status_t WM8960_Deinit (wm8960_handle_t ∗handle)
Deinit the WM8960 codec.
- status_t WM8960_SetDataRoute (wm8960_handle_t ∗handle, wm8960_route_t route)
Set audio data route in WM8960.
- status_t WM8960_SetLeftInput (wm8960_handle_t ∗handle, wm8960_input_t input)
Set left audio input source in WM8960.
- status_t WM8960_SetRightInput (wm8960_handle_t ∗handle, wm8960_input_t input)
Set right audio input source in WM8960.
- status_t WM8960_SetProtocol (wm8960_handle_t ∗handle, wm8960_protocol_t protocol)
Set the audio transfer protocol.
- void WM8960_SetMasterSlave (wm8960_handle_t ∗handle, bool master)
Set WM8960 as master or slave.
- status_t WM8960_SetVolume (wm8960_handle_t ∗handle, wm8960_module_t module, uint32_t volume)
Set the volume of different modules in WM8960.
- uint32_t WM8960_GetVolume (wm8960_handle_t ∗handle, wm8960_module_t module)
Get the volume of different modules in WM8960.
- status_t WM8960_SetMute (wm8960_handle_t ∗handle, wm8960_module_t module, bool isEnabled)
Mute modules in WM8960.
- status_t WM8960_SetModule (wm8960_handle_t ∗handle, wm8960_module_t module, bool isEnabled)
Enable/disable expected devices.
- status_t WM8960_SetPlay (wm8960_handle_t ∗handle, uint32_t playSource)
SET the WM8960 play source.

- status_t WM8960_ConfigDataFormat (wm8960_handle_t ∗handle, uint32_t sysclk, uint32_t sample_rate, uint32_t bits)
  Configure the data format of audio data.
- status_t WM8960_SetJackDetect (wm8960_handle_t ∗handle, bool isEnabled)
  Enable/disable jack detect feature.
- status_t WM8960_WriteReg (wm8960_handle_t ∗handle, uint8_t reg, uint16_t val)
  Write register to WM8960 using I2C.
- status_t WM8960_ReadReg (uint8_t reg, uint16_t ∗val)
  Read register from WM8960 using I2C.
- status_t WM8960_ModifyReg (wm8960_handle_t ∗handle, uint8_t reg, uint16_t mask, uint16_t val)
  Modify some bits in the register using I2C.

Driver version

- #define FSL_WM8960_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))
  CLOCK driver version 2.1.3.

Data Structure Documentation

84.2.1 struct wm8960_audio_format_t

Data Fields

- uint32_t mclk_HZ
  master clock frequency
- uint32_t sampleRate
  sample rate
- uint32_t bitWidth
  bit width

84.2.2 struct wm8960_config_t

Data Fields

- wm8960_route_t route
  Audio data route.
- wm8960_protocol_t bus
  Audio transfer protocol.
- wm8960_audio_format_t format
  Audio format.
- bool master_slave
  Master or slave.
- bool enableSpeaker
  True means enable class D speaker as output, false means no.
- wm8960_input_t leftInputSource
  Left input source for WM8960.
- wm8960_input_t rightInputSource

Right input source for wm8960.
- wm8960_play_source_t playSource
  play source
- uint8_t slaveAddress
  wm8960 device address
- codec_i2c_config_t i2cConfig
  i2c configuration

84.2.2.0.0.66 Field Documentation

84.2.2.0.0.66.1 wm8960_route_t wm8960_config_t::route

84.2.2.0.0.66.2 bool wm8960_config_t::master_slave

84.2.3 struct wm8960_handle_t

Data Fields

- const wm8960_config_t ∗ config
  wm8904 config pointer
- uint8_t i2cHandle [WM8960_I2C_HANDLER_SIZE]
  i2c handle

Macro Definition Documentation

84.3.1 #define WM8960_LINVOL 0x0U

84.3.2 #define WM8960_I2C_ADDR 0x1A

Enumeration Type Documentation

84.4.1 enum wm8960_module_t

Enumerator

kWM8960_ModuleADC ADC module in WM8960.
kWM8960_ModuleDAC DAC module in WM8960.
kWM8960_ModuleVREF VREF module.
kWM8960_ModuleHP Headphone.
kWM8960_ModuleMICB Mic bias.
kWM8960_ModuleMIC Input Mic.
kWM8960_ModuleLineIn Analog in PGA.
kWM8960_ModuleLineOut Line out module.
kWM8960_ModuleSpeaker Speaker module.
kWM8960_ModuleOMIX Output mixer.

84.4.2 anonymous enum

Enumerator

kWM8960_HeadphoneLeft wm8960 headphone left channel
kWM8960_HeadphoneRight wm8960 headphone right channel
kWM8960_SpeakerLeft wm8960 speaker left channel
kWM8960_SpeakerRight wm8960 speaker right channel

84.4.3 enum wm8960_play_source_t

Enumerator

kWM8960_PlaySourcePGA wm8960 play source PGA
kWM8960_PlaySourceInput wm8960 play source Input
kWM8960_PlaySourceDAC wm8960 play source DAC

84.4.4 enum wm8960_route_t

Only provide some typical data route, not all route listed. Note: Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

kWM8960_RouteBypass LINEIN->Headphone.
kWM8960_RoutePlayback I2SIN->DAC->Headphone.
kWM8960_RoutePlaybackandRecord I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.
kWM8960_RouteRecord LINEIN->ADC->I2SOUT.

84.4.5 enum wm8960_protocol_t

WM8960 only supports I2S format and PCM format.

Enumerator

kWM8960_BusI2S I2S type.
kWM8960_BusLeftJustified Left justified mode.
kWM8960_BusRightJustified Right justified mode.
kWM8960_BusPCMA PCM A mode.
kWM8960_BusPCMB PCM B mode.

84.4.6 enum wm8960_input_t

Enumerator

kWM8960_InputClosed Input device is closed.
kWM8960_InputSingleEndedMic Input as single ended mic, only use L/RINPUT1.
kWM8960_InputDifferentialMicInput2 Input as differential mic, use L/RINPUT1 and L/RINPUT2.
kWM8960_InputDifferentialMicInput3 Input as differential mic, use L/RINPUT1 and L/RINPUT3.
kWM8960_InputLineINPUT2 Input as line input, only use L/RINPUT2.
kWM8960_InputLineINPUT3 Input as line input, only use L/RINPUT3.

84.4.7 anonymous enum

Enumerator

kWM8960_AudioSampleRate8KHz Sample rate 8000 Hz.
kWM8960_AudioSampleRate11025Hz Sample rate 11025 Hz.
kWM8960_AudioSampleRate12KHz Sample rate 12000 Hz.
kWM8960_AudioSampleRate16KHz Sample rate 16000 Hz.
kWM8960_AudioSampleRate22050Hz Sample rate 22050 Hz.
kWM8960_AudioSampleRate24KHz Sample rate 24000 Hz.
kWM8960_AudioSampleRate32KHz Sample rate 32000 Hz.
kWM8960_AudioSampleRate44100Hz Sample rate 44100 Hz.
kWM8960_AudioSampleRate48KHz Sample rate 48000 Hz.
kWM8960_AudioSampleRate96KHz Sample rate 96000 Hz.
kWM8960_AudioSampleRate192KHz Sample rate 192000 Hz.
kWM8960_AudioSampleRate384KHz Sample rate 384000 Hz.

84.4.8 anonymous enum

Enumerator

kWM8960_AudioBitWidth16bit audio bit width 16
kWM8960_AudioBitWidth20bit audio bit width 20
kWM8960_AudioBitWidth24bit audio bit width 24
kWM8960_AudioBitWidth32bit audio bit width 32

Function Documentation

84.5.1 status_t WM8960_Init (wm8960_handle_t * *handle*, const wm8960_config_t * *config*)

The second parameter is NULL to WM8960 in this version. If users want to change the settings, they have to use wm8960_write_reg() or wm8960_modify_reg() to set the register value of WM8960. Note-: If the codec_config is NULL, it would initialize WM8960 using default settings. The default setting: codec_config->route = kWM8960_RoutePlaybackandRecord codec_config->bus = kWM8960_BusI2S codec_config->master = slave

Parameters

handle	WM8960 handle structure.
config	WM8960 configuration structure.

84.5.2 status_t WM8960_Deinit (wm8960_handle_t * *handle*)

This function close all modules in WM8960 to save power.

Parameters

handle	WM8960 handle structure pointer.

84.5.3 status_t WM8960_SetDataRoute (wm8960_handle_t * *handle*, wm8960_route_t *route*)

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules. Note: If a new route is set, the previous route would not work.

Parameters

handle	WM8960 handle structure.
route	Audio data route in WM8960.

84.5.4 status_t WM8960_SetLeftInput (wm8960_handle_t * *handle*, wm8960_input_t *input*)

Parameters

handle	WM8960 handle structure.
input	Audio input source.

84.5.5 status_t WM8960_SetRightInput (wm8960_handle_t * *handle*, wm8960_input_t *input*)

Parameters

handle	WM8960 handle structure.
input	Audio input source.

84.5.6 status_t WM8960_SetProtocol (wm8960_handle_t * *handle*, wm8960_protocol_t *protocol*)

WM8960 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

handle	WM8960 handle structure.
protocol	Audio data transfer protocol.

84.5.7 void WM8960_SetMasterSlave (wm8960_handle_t * *handle*, bool *master*)

Parameters

handle	WM8960 handle structure.
master	1 represent master, 0 represent slave.

84.5.8 status_t WM8960_SetVolume (wm8960_handle_t * *handle*, wm8960_module_t *module*, uint32_t *volume*)

This function would set the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Module:kWM8960_ModuleADC, volume range value: 0 is mute, 1-255 is -97db to 30db Module:kWM8960_ModuleDAC, volume range value: 0 is mute, 1-255 is -127db to 0db Module:kWM8960_ModuleHP, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db Module:kWM8960_ModuleLineIn, volume range value: 0 - 0x3F is -17.25db to 30db Module:kWM8960_ModuleSpeaker, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db

Parameters

handle	WM8960 handle structure.
module	Module to set volume, it can be ADC, DAC, Headphone and so on.
volume	Volume value need to be set.

84.5.9 uint32_t WM8960_GetVolume (wm8960_handle_t ∗ *handle*, wm8960_module_t *module*)

This function gets the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	WM8960 handle structure.
module	Module to set volume, it can be ADC, DAC, Headphone and so on.

Returns

Volume value of the module.

84.5.10 status_t WM8960_SetMute (wm8960_handle_t ∗ *handle*, wm8960_module_t *module*, bool *isEnabled*)

Parameters

handle	WM8960 handle structure.
module	Modules need to be mute.

isEnabled	Mute or unmute, 1 represent mute.

84.5.11 status_t WM8960_SetModule (wm8960_handle_t ∗ *handle*, wm8960_module_t *module*, bool *isEnabled*)

Parameters

handle	WM8960 handle structure.
module	Module expected to enable.
isEnabled	Enable or disable moudles.

84.5.12 status_t WM8960_SetPlay (wm8960_handle_t ∗ *handle*, uint32_t *playSource*)

Parameters

handle	WM8960 handle structure.
playSource	play source , can be a value combine of kWM8960_ModuleHeadphoneSourcePG-A, kWM8960_ModuleHeadphoneSourceDAC, kWM8960_ModulePlaySourceInput, kWM8960_ModulePlayMonoRight, kWM8960_ModulePlayMonoLeft.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

84.5.13 status_t WM8960_ConfigDataFormat (wm8960_handle_t ∗ *handle*, uint32_t *sysclk*, uint32_t *sample_rate*, uint32_t *bits*)

This function would configure the registers about the sample rate, bit depths.

Parameters

handle	WM8960 handle structure pointer.
sysclk	system clock of the codec which can be generated by MCLK or PLL output.
sample_rate	Sample rate of audio file running in WM8960. WM8960 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
bits	Bit depth of audio file (WM8960 only supports 16bit, 20bit, 24bit and 32 bit in HW).

84.5.14 status_t WM8960_SetJackDetect (wm8960_handle_t ∗ *handle,* bool *isEnabled*)

Parameters

handle	WM8960 handle structure.
isEnabled	Enable or disable moudles.

84.5.15 status_t WM8960_WriteReg (wm8960_handle_t ∗ *handle,* uint8_t *reg,* uint16_t *val*)

Parameters

handle	WM8960 handle structure.
reg	The register address in WM8960.
val	Value needs to write into the register.

84.5.16 status_t WM8960_ReadReg (uint8_t *reg,* uint16_t ∗ *val*)

Parameters

reg	The register address in WM8960.
val	Value written to.

84.5.17 status_t WM8960_ModifyReg (wm8960_handle_t ∗ *handle,* uint8_t *reg,* uint16_t *mask,* uint16_t *val*)

Parameters

handle	WM8960 handle structure.
reg	The register address in WM8960.
mask	The mask code for the bits want to write. The bit you want to write should be 0.
val	Value needs to write into the register.

Chapter 85
Serial_Manager

Overview

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_PORT_TYPE_UART (0U)
  Enable or disable uart port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_USBCDC (0U)
  Enable or disable USB CDC port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_SWO (0U)
  Enable or disable SWO port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_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_MANAGER_TASK_HANDLE_TX (0U)
  Enable or disable SerialManager_Task() handle TX to prevent recursive calling.
- #define SERIAL_MANAGER_TIME_DELAY_DEFAULT_VALUE (1U)
  Set the default delay time in ms used by SerialManager_TimeDelay().
- #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_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)
  SERIAL_PORT_UART_HANDLE_SIZE/SERIAL_PORT_USB_CDC_HANDLE_SIZE + serial manager dedicated 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_USE_COMMON_TASK (0U)
  Macro to determine whether use common task.
- #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)
  callback function

Enumerations

- enum serial_port_type_t {
  kSerialPort_Uart = 1U,
  kSerialPort_UsbCdc,
  kSerialPort_Swo,
  kSerialPort_Virtual,
  kSerialPort_Rpmsg }
  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.

Data Structure Documentation

85.2.1 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.

85.2.1.0.0.67 Field Documentation

85.2.1.0.0.67.1 uint8_t* serial_manager_config_t::ringBuffer

Besides, the memory space cannot be free during the lifetime of the serial manager module.

85.2.2 struct serial_manager_callback_message_t

Data Fields

- uint8_t * buffer
  Transferred buffer.
- uint32_t length
  Transferred data length.

Macro Definition Documentation

85.3.1 #define SERIAL_MANAGER_TIME_DELAY_DEFAULT_VALUE (1U)

85.3.2 #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_-SIZE_TEMP + 12U)

Definition of serial manager handle size.

85.3.3 #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.

85.3.4 #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.

85.3.5 #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.

85.3.6 #define SERIAL_MANAGER_USE_COMMON_TASK (0U)

85.3.7 #define SERIAL_MANAGER_TASK_PRIORITY (2U)

85.3.8 #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)

Enumeration Type Documentation

85.4.1 enum serial_port_type_t

Enumerator

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.

85.4.2 enum serial_manager_type_t

Enumerator

kSerialManager_NonBlocking None blocking handle.
kSerialManager_Blocking Blocking handle.

85.4.3 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.

Function Documentation

85.5.1 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.

85.5.2 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.

85.5.3 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);
*
```

85.5.4 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.

85.5.5 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));
*
```

85.5.6 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.

85.5.7 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.

85.5.8 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.

85.5.9 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.

85.5.10 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.

Chapter 86
Serial_port_swo

Overview

Data Structures

- struct serial_port_swo_config_t
  serial port swo config struct More...

Macros

- #define SERIAL_PORT_SWO_HANDLE_SIZE (12U)
  serial port swo handle size

Enumerations

- enum serial_port_swo_protocol_t {
  kSerialManager_SwoProtocolManchester = 1U,
  kSerialManager_SwoProtocolNrz = 2U }
  serial port swo protocol

Data Structure Documentation

86.2.1 struct serial_port_swo_config_t

Data Fields

- uint32_t clockRate
  clock rate
- uint32_t baudRate
  baud rate
- uint32_t port
  Port used to transfer data.
- serial_port_swo_protocol_t protocol
  SWO protocol.

Enumeration Type Documentation

86.3.1 enum serial_port_swo_protocol_t

Enumerator

kSerialManager_SwoProtocolManchester SWO Manchester protocol.
kSerialManager_SwoProtocolNrz SWO UART/NRZ protocol.

Chapter 87
Serial_port_uart

Overview

Macros

- #define SERIAL_PORT_UART_HANDLE_SIZE (HAL_UART_HANDLE_SIZE)
  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 = 0x1U,
  kSerialManager_UartParityOdd = 0x2U }
  serial port uart parity mode
- enum serial_port_uart_stop_bit_count_t {
  kSerialManager_UartOneStopBit = 0U,
  kSerialManager_UartTwoStopBit = 1U }
  serial port uart stop bit count

Enumeration Type Documentation

87.2.1 enum serial_port_uart_parity_mode_t

Enumerator

kSerialManager_UartParityDisabled Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

87.2.2 enum serial_port_uart_stop_bit_count_t

Enumerator

kSerialManager_UartOneStopBit One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

Chapter 88
Serial_port_usb

Overview

Data Structures

- struct serial_port_usb_cdc_config_t
  serial port usb config struct More...

Macros

- #define SERIAL_PORT_USB_CDC_HANDLE_SIZE (72U)
  serial port usb handle size
- #define USB_DEVICE_INTERRUPT_PRIORITY (3U)
  USB interrupt priority.

Enumerations

- enum serial_port_usb_cdc_controller_index_t {
  kSerialManager_UsbControllerKhci0 = 0U,
  kSerialManager_UsbControllerKhci1 = 1U,
  kSerialManager_UsbControllerEhci0 = 2U,
  kSerialManager_UsbControllerEhci1 = 3U,
  kSerialManager_UsbControllerLpcIp3511Fs0 = 4U,
  kSerialManager_UsbControllerLpcIp3511Fs1 = 5U,
  kSerialManager_UsbControllerLpcIp3511Hs0 = 6U,
  kSerialManager_UsbControllerLpcIp3511Hs1 = 7U,
  kSerialManager_UsbControllerOhci0 = 8U,
  kSerialManager_UsbControllerOhci1 = 9U,
  kSerialManager_UsbControllerIp3516Hs0 = 10U,
  kSerialManager_UsbControllerIp3516Hs1 = 11U }
  USB controller ID.

Data Structure Documentation

88.2.1 struct serial_port_usb_cdc_config_t

Data Fields

- serial_port_usb_cdc_controller_index_t controllerIndex
  controller index

Enumeration Type Documentation

88.3.1 enum serial_port_usb_cdc_controller_index_t

Enumerator

kSerialManager_UsbControllerKhci0 KHCI 0U.

kSerialManager_UsbControllerKhci1 KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerEhci0 EHCI 0U.

kSerialManager_UsbControllerEhci1 EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerLpcIp3511Fs0 LPC USB IP3511 FS controller 0.

kSerialManager_UsbControllerLpcIp3511Fs1 LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerLpcIp3511Hs0 LPC USB IP3511 HS controller 0.

kSerialManager_UsbControllerLpcIp3511Hs1 LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerOhci0 OHCI 0U.

kSerialManager_UsbControllerOhci1 OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerIp3516Hs0 IP3516HS 0U.

kSerialManager_UsbControllerIp3516Hs1 IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.

Chapter 89
Data Structure Documentation

89.0.2 codec_i2c_config_t Struct Reference

CODEC I2C configurations structure.

`#include <fsl_codec_i2c.h>`

Data Fields

- uint32_t codecI2CInstance
  i2c bus instance
- uint32_t codecI2CSourceClock
  i2c bus source clock frequency

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support