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

Section Number	Title	Page Number

Contents

Contents

Contents

Section Number	Title	Page Number

Contents

Contents

Contents

Section Number	Title	Page Number

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™. 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

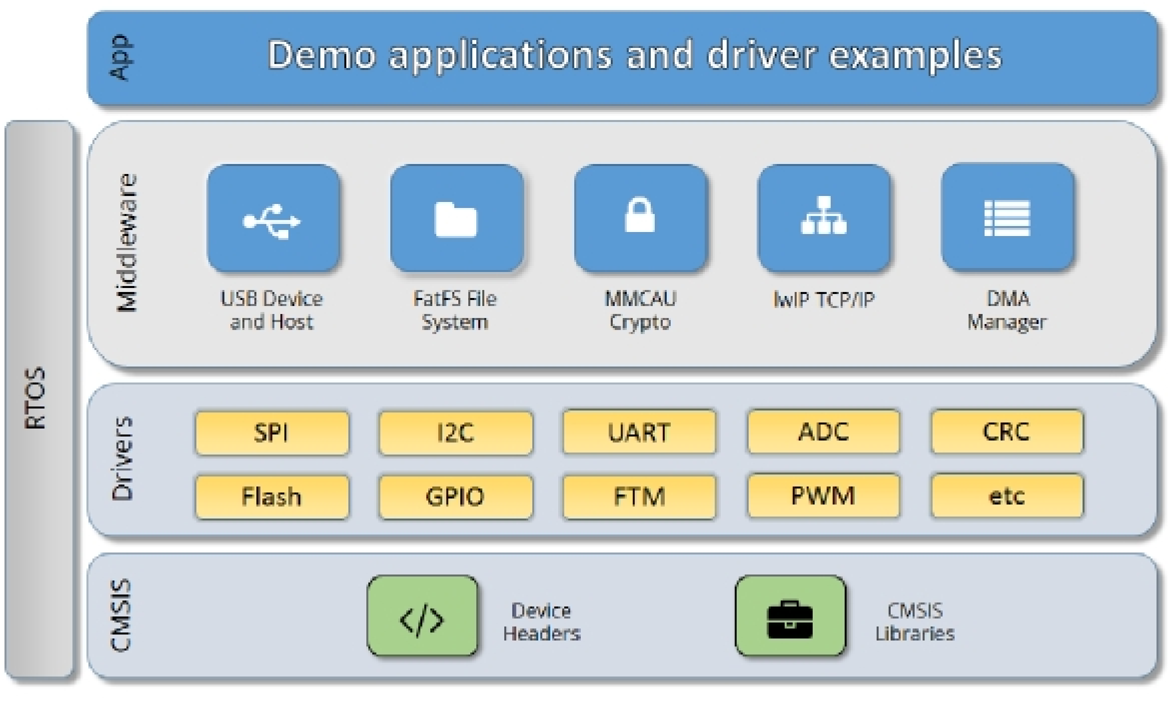

Figure 1: MCUXpresso SDK Block Diagram

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_ECSPI_Busy = 6400
- kStatus_ECSPI_Idle = 6401
- kStatus_ECSPI_Error = 6402
- kStatus_ECSPI_HardwareOverFlow = 6403
- 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_UART_TxBusy = 2800
- kStatus_UART_RxBusy = 2801
- kStatus_UART_TxIdle = 2802
- kStatus_UART_RxIdle = 2803
- kStatus_UART_TxWatermarkTooLarge = 2804
- kStatus_UART_RxWatermarkTooLarge = 2805
- kStatus_UART_FlagCannotClearManually = 2806
- kStatus_UART_Error = 2807
- kStatus_UART_RxRingBufferOverrun = 2808
- kStatus_UART_RxHardwareOverrun = 2809
- kStatus_UART_NoiseError = 2810
- kStatus_UART_FramingError = 2811
- kStatus_UART_ParityError = 2812
- kStatus_UART_BaudrateNotSupport = 2813
- kStatus_UART_BreakDetect = 2814
- kStatus_QSPI_Idle = 4500
- kStatus_QSPI_Busy = 4501
- kStatus_QSPI_Error = 4502
- 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

Chapter 5
Deprecated List

Global GPIO_ClearPinsOutput (GPIO_Type *base, uint32_t mask)

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 It has been superceded by GPIO_PortClear.

Global GPIO_DisableInterrupts (GPIO_Type *base, uint32_t mask)

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 It has been superceded by GPIO_PortDisableInterrupts.

Global GPIO_ReadPadStatus (GPIO_Type *base, uint32_t pin)

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 It has been superceded by GPIO_PinReadPadStatus.

Global GPIO_ReadPinInput (GPIO_Type *base, uint32_t pin)

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 It has been superceded by GPIO_PinRead.

Global GPIO_SetPinInterruptConfig (GPIO_Type *base, uint32_t pin, gpio_interrupt_mode_t pinInterruptMode)

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 It has been superceded by GPIO_PinSetInterruptConfig.

Global GPIO_SetPinsOutput (GPIO_Type *base, uint32_t mask)

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 It has been superceded by GPIO_PortSet.

Global GPIO_WritePinOutput (GPIO_Type *base, uint32_t pin, uint8_t output)

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 It has been superceded by GPIO_PinWrite.

Global SAI_RxGetDefaultConfig (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_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 mclkSourceClockHz, 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

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

Data Structures

- struct osc_config_t
  OSC configuration structure. More...
- struct ccm_analog_frac_pll_config_t
  Fractional-N PLL configuration. More...
- struct ccm_analog_sscg_pll_config_t
  SSCG PLL configuration. More...

Macros

- #define OSC25M_CLK_FREQ 25000000U
  XTAL 25M clock frequency.
- #define OSC27M_CLK_FREQ 27000000U
  XTAL 27M clock frequency.
- #define HDMI_PHY_27M_FREQ 27000000U
  HDMI PHY 27M clock frequency.
- #define CLKPN_FREQ 0U
  clock1PN frequency.
- #define ECSPI_CLOCKS
  Clock ip name array for ECSPI.
- #define GPIO_CLOCKS
  Clock ip name array for GPIO.
- #define GPT_CLOCKS
  Clock ip name array for GPT.
- #define I2C_CLOCKS
  Clock ip name array for I2C.
- #define IOMUX_CLOCKS
  Clock ip name array for IOMUX.
- #define IPMUX_CLOCKS
  Clock ip name array for IPMUX.
- #define PWM_CLOCKS
  Clock ip name array for PWM.
- #define RDC_CLOCKS
  Clock ip name array for RDC.

- #define SAI_CLOCKS
  Clock ip name array for SAI.
- #define RDC_SEMA42_CLOCKS
  Clock ip name array for RDC SEMA42.
- #define UART_CLOCKS
  Clock ip name array for UART.
- #define USDHC_CLOCKS
  Clock ip name array for USDHC.
- #define WDOG_CLOCKS
  Clock ip name array for WDOG.
- #define TMU_CLOCKS
  Clock ip name array for TEMPSENSOR.
- #define SDMA_CLOCKS
  Clock ip name array for SDMA.
- #define MU_CLOCKS
  Clock ip name array for MU.
- #define QSPI_CLOCKS
  Clock ip name array for QSPI.
- #define CCM_BIT_FIELD_EXTRACTION(val, mask, shift) (((val) & (mask)) >> (shift))
  CCM reg macros to extract corresponding registers bit field.
- #define CCM_REG_OFF(root, off) (∗((volatile uint32_t ∗)((uint32_t)(root) + (off))))
  CCM reg macros to map corresponding registers.
- #define AUDIO_PLL1_CFG0_OFFSET 0x00
  CCM Analog registers offset.
- #define CCM_ANALOG_TUPLE(reg, shift) ((((reg)&0xFFFFU) << 16U) | (shift))
  CCM ANALOG tuple macros to map corresponding registers and bit fields.
- #define CCM_TUPLE(ccgr, root) ((ccgr) << 16U | (root))
  CCM CCGR and root tuple.
- #define kCLOCK_CoreSysClk kCLOCK_CoreM4Clk
  For compatible with other platforms without CCM.
- #define CLOCK_GetCoreSysClkFreq CLOCK_GetCoreM4Freq
  For compatible with other platforms without CCM.

Enumerations

- enum clock_name_t {
  kCLOCK_CoreM4Clk,
  kCLOCK_AxiClk,
  kCLOCK_AhbClk,
  kCLOCK_IpgClk }
  Clock name used to get clock frequency.
- enum clock_ip_name_t { ,

kCLOCK_Debug = CCM_TUPLE(4U, 32U),
kCLOCK_Dram = CCM_TUPLE(5U, 64U),
kCLOCK_Ecspi1 = CCM_TUPLE(7U, 101U),
kCLOCK_Ecspi2 = CCM_TUPLE(8U, 102U),
kCLOCK_Ecspi3 = CCM_TUPLE(9U, 131U),
kCLOCK_Gpio1 = CCM_TUPLE(11U, 33U),
kCLOCK_Gpio2 = CCM_TUPLE(12U, 33U),
kCLOCK_Gpio3 = CCM_TUPLE(13U, 33U),
kCLOCK_Gpio4 = CCM_TUPLE(14U, 33U),
kCLOCK_Gpio5 = CCM_TUPLE(15U, 33U),
kCLOCK_Gpt1 = CCM_TUPLE(16U, 107U),
kCLOCK_Gpt2 = CCM_TUPLE(17U, 108U),
kCLOCK_Gpt3 = CCM_TUPLE(18U, 109U),
kCLOCK_Gpt4 = CCM_TUPLE(19U, 110U),
kCLOCK_Gpt5 = CCM_TUPLE(20U, 111U),
kCLOCK_Gpt6 = CCM_TUPLE(21U, 112U),
kCLOCK_I2c1 = CCM_TUPLE(23U, 90U),
kCLOCK_I2c2 = CCM_TUPLE(24U, 91U),
kCLOCK_I2c3 = CCM_TUPLE(25U, 92U),
kCLOCK_I2c4 = CCM_TUPLE(26U, 93U),
kCLOCK_Iomux = CCM_TUPLE(27U, 33U),
kCLOCK_Ipmux1 = CCM_TUPLE(28U, 33U),
kCLOCK_Ipmux2 = CCM_TUPLE(29U, 33U),
kCLOCK_Ipmux3 = CCM_TUPLE(30U, 33U),
kCLOCK_Ipmux4 = CCM_TUPLE(31U, 33U),
kCLOCK_M4 = CCM_TUPLE(32U, 1U),
kCLOCK_Mu = CCM_TUPLE(33U, 33U),
kCLOCK_Ocram = CCM_TUPLE(35U, 16U),
kCLOCK_OcramS = CCM_TUPLE(36U, 32U),
kCLOCK_Pwm1 = CCM_TUPLE(40U, 103U),
kCLOCK_Pwm2 = CCM_TUPLE(41U, 104U),
kCLOCK_Pwm3 = CCM_TUPLE(42U, 105U),
kCLOCK_Pwm4 = CCM_TUPLE(43U, 106U),
kCLOCK_Qspi = CCM_TUPLE(47U, 87U),
kCLOCK_Rdc = CCM_TUPLE(49U, 33U),
kCLOCK_Sai1 = CCM_TUPLE(51U, 75U),
kCLOCK_Sai2 = CCM_TUPLE(52U, 76U),
kCLOCK_Sai3 = CCM_TUPLE(53U, 77U),
kCLOCK_Sai4 = CCM_TUPLE(54U, 78U),
kCLOCK_Sai5 = CCM_TUPLE(55U, 79U),
kCLOCK_Sai6 = CCM_TUPLE(56U, 80U),
kCLOCK_Sdma1 = CCM_TUPLE(58U, 33U),
kCLOCK_Sdma2 = CCM_TUPLE(59U, 35U),
kCLOCK_Sec_Debug = CCM_TUPLE(60U, 33U),
kCLOCK_Sema42_1 = CCM_TUPLE(61U, 33U),
kCLOCK_Sema42_2 = CCM_TUPLE(62U, 33U),
kCLOCK_Sim_display = CCM_TUPLE(63U, 16U),
kCLOCK_Sim_m = CCM_TUPLE(65U, 32U),
kCLOCK_Sim_main = CCM_TUPLE(66U, 16U),

kCLOCK_TempSensor = CCM_TUPLE(98U, 0xFFFF) }
CCM CCGR gate control.
- enum clock_root_control_t {
kCLOCK_RootM4 = (uint32_t)(&(CCM)->ROOT[1].TARGET_ROOT),
kCLOCK_RootAxi = (uint32_t)(&(CCM)->ROOT[16].TARGET_ROOT),
kCLOCK_RootNoc = (uint32_t)(&(CCM)->ROOT[26].TARGET_ROOT),
kCLOCK_RootAhb = (uint32_t)(&(CCM)->ROOT[32].TARGET_ROOT),
kCLOCK_RootIpg = (uint32_t)(&(CCM)->ROOT[33].TARGET_ROOT),
kCLOCK_RootDramAlt = (uint32_t)(&(CCM)->ROOT[64].TARGET_ROOT),
kCLOCK_RootSai1 = (uint32_t)(&(CCM)->ROOT[75].TARGET_ROOT),
kCLOCK_RootSai2 = (uint32_t)(&(CCM)->ROOT[76].TARGET_ROOT),
kCLOCK_RootSai3 = (uint32_t)(&(CCM)->ROOT[77].TARGET_ROOT),
kCLOCK_RootSai4 = (uint32_t)(&(CCM)->ROOT[78].TARGET_ROOT),
kCLOCK_RootSai5 = (uint32_t)(&(CCM)->ROOT[79].TARGET_ROOT),
kCLOCK_RootSai6 = (uint32_t)(&(CCM)->ROOT[80].TARGET_ROOT),
kCLOCK_RootQspi = (uint32_t)(&(CCM)->ROOT[87].TARGET_ROOT),
kCLOCK_RootI2c1 = (uint32_t)(&(CCM)->ROOT[90].TARGET_ROOT),
kCLOCK_RootI2c2 = (uint32_t)(&(CCM)->ROOT[91].TARGET_ROOT),
kCLOCK_RootI2c3 = (uint32_t)(&(CCM)->ROOT[92].TARGET_ROOT),
kCLOCK_RootI2c4 = (uint32_t)(&(CCM)->ROOT[93].TARGET_ROOT),
kCLOCK_RootUart1 = (uint32_t)(&(CCM)->ROOT[94].TARGET_ROOT),
kCLOCK_RootUart2 = (uint32_t)(&(CCM)->ROOT[95].TARGET_ROOT),
kCLOCK_RootUart3 = (uint32_t)(&(CCM)->ROOT[96].TARGET_ROOT),
kCLOCK_RootUart4 = (uint32_t)(&(CCM)->ROOT[97].TARGET_ROOT),
kCLOCK_RootEcspi1 = (uint32_t)(&(CCM)->ROOT[101].TARGET_ROOT),
kCLOCK_RootEcspi2 = (uint32_t)(&(CCM)->ROOT[102].TARGET_ROOT),
kCLOCK_RootEcspi3 = (uint32_t)(&(CCM)->ROOT[131].TARGET_ROOT),
kCLOCK_RootPwm1 = (uint32_t)(&(CCM)->ROOT[103].TARGET_ROOT),
kCLOCK_RootPwm2 = (uint32_t)(&(CCM)->ROOT[104].TARGET_ROOT),
kCLOCK_RootPwm3 = (uint32_t)(&(CCM)->ROOT[105].TARGET_ROOT),
kCLOCK_RootPwm4 = (uint32_t)(&(CCM)->ROOT[106].TARGET_ROOT),
kCLOCK_RootGpt1 = (uint32_t)(&(CCM)->ROOT[107].TARGET_ROOT),
kCLOCK_RootGpt2 = (uint32_t)(&(CCM)->ROOT[108].TARGET_ROOT),
kCLOCK_RootGpt3 = (uint32_t)(&(CCM)->ROOT[109].TARGET_ROOT),
kCLOCK_RootGpt4 = (uint32_t)(&(CCM)->ROOT[110].TARGET_ROOT),
kCLOCK_RootGpt5 = (uint32_t)(&(CCM)->ROOT[111].TARGET_ROOT),
kCLOCK_RootGpt6 = (uint32_t)(&(CCM)->ROOT[112].TARGET_ROOT),
kCLOCK_RootWdog = (uint32_t)(&(CCM)->ROOT[114].TARGET_ROOT) }
ccm root name used to get clock frequency.
- enum clock_rootmux_m4_clk_sel_t {

kCLOCK_M4RootmuxOsc25m = 0U,
kCLOCK_M4RootmuxSysPll2Div5 = 1U,
kCLOCK_M4RootmuxSysPll2Div4 = 2U,
kCLOCK_M4RootmuxSysPll1Div3 = 3U,
kCLOCK_M4RootmuxSysPll1 = 4U,
kCLOCK_M4RootmuxAudioPll1 = 5U,
kCLOCK_M4RootmuxVideoPll1 = 6U,
kCLOCK_M4RootmuxSysPll3 = 7U }
Root clock select enumeration for ARM Cortex-M4 core.
- enum clock_rootmux_axi_clk_sel_t {
kCLOCK_AxiRootmuxOsc25m = 0U,
kCLOCK_AxiRootmuxSysPll2Div3 = 1U,
kCLOCK_AxiRootmuxSysPll1 = 2U,
kCLOCK_AxiRootmuxSysPll2Div4 = 3U,
kCLOCK_AxiRootmuxSysPll2 = 4U,
kCLOCK_AxiRootmuxAudioPll1 = 5U,
kCLOCK_AxiRootmuxVideoPll1 = 6U,
kCLOCK_AxiRootmuxSysPll1Div8 = 7U }
Root clock select enumeration for AXI bus.
- enum clock_rootmux_ahb_clk_sel_t {
kCLOCK_AhbRootmuxOsc25m = 0U,
kCLOCK_AhbRootmuxSysPll1Div6 = 1U,
kCLOCK_AhbRootmuxSysPll1 = 2U,
kCLOCK_AhbRootmuxSysPll1Div2 = 3U,
kCLOCK_AhbRootmuxSysPll2Div8 = 4U,
kCLOCK_AhbRootmuxSysPll3 = 5U,
kCLOCK_AhbRootmuxAudioPll1 = 6U,
kCLOCK_AhbRootmuxVideoPll1 = 7U }
Root clock select enumeration for AHB bus.
- enum clock_rootmux_qspi_clk_sel_t {
kCLOCK_QspiRootmuxOsc25m = 0U,
kCLOCK_QspiRootmuxSysPll1Div2 = 1U,
kCLOCK_QspiRootmuxSysPll1 = 2U,
kCLOCK_QspiRootmuxSysPll2Div2 = 3U,
kCLOCK_QspiRootmuxAudioPll2 = 4,
kCLOCK_QspiRootmuxSysPll1Div3 = 5U,
kCLOCK_QspiRootmuxSysPll3 = 6U,
kCLOCK_QspiRootmuxSysPll1Div8 = 7U }
Root clock select enumeration for QSPI peripheral.
- enum clock_rootmux_ecspi_clk_sel_t {

kCLOCK_EcspiRootmuxOsc25m = 0U,
kCLOCK_EcspiRootmuxSysPll2Div5 = 1U,
kCLOCK_EcspiRootmuxSysPll1Div20 = 2U,
kCLOCK_EcspiRootmuxSysPll1Div5 = 3U,
kCLOCK_EcspiRootmuxSysPll1 = 4U,
kCLOCK_EcspiRootmuxSysPll3 = 5U,
kCLOCK_EcspiRootmuxSysPll2Div4 = 6U,
kCLOCK_EcspiRootmuxAudioPll2 = 7U }
Root clock select enumeration for ECSPI peripheral.

- enum clock_rootmux_i2c_clk_sel_t {
kCLOCK_I2cRootmuxOsc25m = 0U,
kCLOCK_I2cRootmuxSysPll1Div5 = 1U,
kCLOCK_I2cRootmuxSysPll2Div20 = 2U,
kCLOCK_I2cRootmuxSysPll3 = 3U,
kCLOCK_I2cRootmuxAudioPll1 = 4U,
kCLOCK_I2cRootmuxVideoPll1 = 5U,
kCLOCK_I2cRootmuxAudioPll2 = 6U,
kCLOCK_I2cRootmuxSysPll1Div6 = 7U }
Root clock select enumeration for I2C peripheral.
- enum clock_rootmux_uart_clk_sel_t {
kCLOCK_UartRootmuxOsc25m = 0U,
kCLOCK_UartRootmuxSysPll1Div10 = 1U,
kCLOCK_UartRootmuxSysPll2Div5 = 2U,
kCLOCK_UartRootmuxSysPll2Div10 = 3U,
kCLOCK_UartRootmuxSysPll3 = 4U,
kCLOCK_UartRootmuxExtClk2 = 5U,
kCLOCK_UartRootmuxExtClk34 = 6U,
kCLOCK_UartRootmuxAudioPll2 = 7U }
Root clock select enumeration for UART peripheral.
- enum clock_rootmux_gpt_t {
kCLOCK_GptRootmuxOsc25m = 0U,
kCLOCK_GptRootmuxSystemPll2Div10 = 1U,
kCLOCK_GptRootmuxSysPll1Div2 = 2U,
kCLOCK_GptRootmuxSysPll1Div20 = 3U,
kCLOCK_GptRootmuxVideoPll1 = 4U,
kCLOCK_GptRootmuxSystemPll1Div10 = 5U,
kCLOCK_GptRootmuxAudioPll1 = 6U,
kCLOCK_GptRootmuxExtClk123 = 7U }
Root clock select enumeration for GPT peripheral.
- enum clock_rootmux_wdog_clk_sel_t {

kCLOCK_WdogRootmuxOsc25m = 0U,
kCLOCK_WdogRootmuxSysPll1Div6 = 1U,
kCLOCK_WdogRootmuxSysPll1Div5 = 2U,
kCLOCK_WdogRootmuxVpuPll = 3U,
kCLOCK_WdogRootmuxSystemPll2Div8 = 4U,
kCLOCK_WdogRootmuxSystemPll3 = 5U,
kCLOCK_WdogRootmuxSystemPll1Div10 = 6U,
kCLOCK_WdogRootmuxSystemPll2Div6 = 7U }

Root clock select enumeration for WDOG peripheral.

- enum clock_rootmux_Pwm_clk_sel_t {
kCLOCK_PwmRootmuxOsc25m = 0U,
kCLOCK_PwmRootmuxSysPll2Div10 = 1U,
kCLOCK_PwmRootmuxSysPll1Div5 = 2U,
kCLOCK_PwmRootmuxSysPll1Div20 = 3U,
kCLOCK_PwmRootmuxSystemPll3 = 4U,
kCLOCK_PwmRootmuxExtClk12 = 5U,
kCLOCK_PwmRootmuxSystemPll1Div10 = 6U,
kCLOCK_PwmRootmuxVideoPll1 = 7U }

  Root clock select enumeration for PWM peripheral.
- enum clock_rootmux_sai_clk_sel_t {
kCLOCK_SaiRootmuxOsc25m = 0U,
kCLOCK_SaiRootmuxAudioPll1 = 1U,
kCLOCK_SaiRootmuxAudioPll2 = 2U,
kCLOCK_SaiRootmuxVideoPll1 = 3U,
kCLOCK_SaiRootmuxSysPll1Div6 = 4U,
kCLOCK_SaiRootmuxOsc27m = 5U,
kCLOCK_SaiRootmuxExtClk123 = 6U,
kCLOCK_SaiRootmuxExtClk234 = 7U }

  Root clock select enumeration for SAI peripheral.
- enum clock_rootmux_noc_clk_sel_t {
kCLOCK_NocRootmuxOsc25m = 0U,
kCLOCK_NocRootmuxSysPll1 = 1U,
kCLOCK_NocRootmuxSysPll3 = 2U,
kCLOCK_NocRootmuxSysPll2 = 3U,
kCLOCK_NocRootmuxSysPll2Div2 = 4U,
kCLOCK_NocRootmuxAudioPll1 = 5U,
kCLOCK_NocRootmuxVideoPll1 = 6U,
kCLOCK_NocRootmuxAudioPll2 = 7U }

  Root clock select enumeration for NOC CLK.
- enum clock_pll_gate_t {

kCLOCK_ArmPllGate = (uint32_t)(&(CCM)->PLL_CTRL[12].PLL_CTRL),
kCLOCK_GpuPllGate = (uint32_t)(&(CCM)->PLL_CTRL[13].PLL_CTRL),
kCLOCK_VpuPllGate = (uint32_t)(&(CCM)->PLL_CTRL[14].PLL_CTRL),
kCLOCK_DramPllGate = (uint32_t)(&(CCM)->PLL_CTRL[15].PLL_CTRL),
kCLOCK_SysPll1Gate = (uint32_t)(&(CCM)->PLL_CTRL[16].PLL_CTRL),
kCLOCK_SysPll1Div2Gate = (uint32_t)(&(CCM)->PLL_CTRL[17].PLL_CTRL),
kCLOCK_SysPll1Div3Gate = (uint32_t)(&(CCM)->PLL_CTRL[18].PLL_CTRL),
kCLOCK_SysPll1Div4Gate = (uint32_t)(&(CCM)->PLL_CTRL[19].PLL_CTRL),
kCLOCK_SysPll1Div5Gate = (uint32_t)(&(CCM)->PLL_CTRL[20].PLL_CTRL),
kCLOCK_SysPll1Div6Gate = (uint32_t)(&(CCM)->PLL_CTRL[21].PLL_CTRL),
kCLOCK_SysPll1Div8Gate = (uint32_t)(&(CCM)->PLL_CTRL[22].PLL_CTRL),
kCLOCK_SysPll1Div10Gate = (uint32_t)(&(CCM)->PLL_CTRL[23].PLL_CTRL),
kCLOCK_SysPll1Div20Gate = (uint32_t)(&(CCM)->PLL_CTRL[24].PLL_CTRL),
kCLOCK_SysPll2Gate = (uint32_t)(&(CCM)->PLL_CTRL[25].PLL_CTRL),
kCLOCK_SysPll2Div2Gate = (uint32_t)(&(CCM)->PLL_CTRL[26].PLL_CTRL),
kCLOCK_SysPll2Div3Gate = (uint32_t)(&(CCM)->PLL_CTRL[27].PLL_CTRL),
kCLOCK_SysPll2Div4Gate = (uint32_t)(&(CCM)->PLL_CTRL[28].PLL_CTRL),
kCLOCK_SysPll2Div5Gate = (uint32_t)(&(CCM)->PLL_CTRL[29].PLL_CTRL),
kCLOCK_SysPll2Div6Gate = (uint32_t)(&(CCM)->PLL_CTRL[30].PLL_CTRL),
kCLOCK_SysPll2Div8Gate = (uint32_t)(&(CCM)->PLL_CTRL[31].PLL_CTRL),
kCLOCK_SysPll2Div10Gate = (uint32_t)(&(CCM)->PLL_CTRL[32].PLL_CTRL),
kCLOCK_SysPll2Div20Gate = (uint32_t)(&(CCM)->PLL_CTRL[33].PLL_CTRL),
kCLOCK_SysPll3Gate = (uint32_t)(&(CCM)->PLL_CTRL[34].PLL_CTRL),
kCLOCK_AudioPll1Gate = (uint32_t)(&(CCM)->PLL_CTRL[35].PLL_CTRL),
kCLOCK_AudioPll2Gate = (uint32_t)(&(CCM)->PLL_CTRL[36].PLL_CTRL),
kCLOCK_VideoPll1Gate = (uint32_t)(&(CCM)->PLL_CTRL[37].PLL_CTRL),
kCLOCK_VideoPll2Gate = (uint32_t)(&(CCM)->PLL_CTRL[38].PLL_CTRL) }

CCM PLL gate control.

- enum clock_gate_value_t {
kCLOCK_ClockNotNeeded = 0x0U,
kCLOCK_ClockNeededRun = 0x1111U,
kCLOCK_ClockNeededRunWait = 0x2222U,
kCLOCK_ClockNeededAll = 0x3333U }

CCM gate control value.

- enum clock_pll_bypass_ctrl_t {

kCLOCK_AudioPll1BypassCtrl,
kCLOCK_AudioPll2BypassCtrl,
kCLOCK_VideoPll1BypassCtrl,
kCLOCK_GpuPLLPwrBypassCtrl,
kCLOCK_VpuPllPwrBypassCtrl,
kCLOCK_ArmPllPwrBypassCtrl,
kCLOCK_SysPll1InternalPll1BypassCtrl,
kCLOCK_SysPll1InternalPll2BypassCtrl,
kCLOCK_SysPll2InternalPll1BypassCtrl,
kCLOCK_SysPll2InternalPll2BypassCtrl,
kCLOCK_SysPll3InternalPll1BypassCtrl,
kCLOCK_SysPll3InternalPll2BypassCtrl,
kCLOCK_VideoPll2InternalPll1BypassCtrl,
kCLOCK_VideoPll2InternalPll2BypassCtrl,
kCLOCK_DramPllInternalPll1BypassCtrl,
kCLOCK_DramPllInternalPll2BypassCtrl }
PLL control names for PLL bypass.
- enum clock_pll_clke_t {

kCLOCK_AudioPll1Clke,
kCLOCK_AudioPll2Clke,
kCLOCK_VideoPll1Clke,
kCLOCK_GpuPllClke,
kCLOCK_VpuPllClke,
kCLOCK_ArmPllClke,
kCLOCK_SystemPll1Clke,
kCLOCK_SystemPll1Div2Clke,
kCLOCK_SystemPll1Div3Clke,
kCLOCK_SystemPll1Div4Clke,
kCLOCK_SystemPll1Div5Clke,
kCLOCK_SystemPll1Div6Clke,
kCLOCK_SystemPll1Div8Clke,
kCLOCK_SystemPll1Div10Clke,
kCLOCK_SystemPll1Div20Clke,
kCLOCK_SystemPll2Clke,
kCLOCK_SystemPll2Div2Clke,
kCLOCK_SystemPll2Div3Clke,
kCLOCK_SystemPll2Div4Clke,
kCLOCK_SystemPll2Div5Clke,
kCLOCK_SystemPll2Div6Clke,
kCLOCK_SystemPll2Div8Clke,
kCLOCK_SystemPll2Div10Clke,
kCLOCK_SystemPll2Div20Clke,
kCLOCK_SystemPll3Clke,
kCLOCK_VideoPll2Clke,
kCLOCK_DramPllClke,
kCLOCK_OSC25MClke,
kCLOCK_OSC27MClke }

PLL clock names for clock enable/disable settings.

- enum clock_pll_ctrl_t

  ANALOG Power down override control.
- enum _osc_mode {
  kOSC_OscMode = 0U,
  kOSC_ExtMode = 1U }

  OSC work mode.
- enum osc32_src_t {
  kOSC32_Src25MDiv800 = 0U,
  kOSC32_SrcRTC }

  OSC 32K input select.
- enum _ccm_analog_pll_ref_clk {
  kANALOG_PllRefOsc25M = 0U,
  kANALOG_PllRefOsc27M = 1U,
  kANALOG_PllRefOscHdmiPhy27M = 2U,
  kANALOG_PllRefClkPN = 3U }

  PLL reference clock select.

Driver version

- #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 3, 3))
  CLOCK driver version 2.3.3.

CCM Root Clock Setting

- static void CLOCK_SetRootMux (clock_root_control_t rootClk, uint32_t mux)
  Set clock root mux.
- static uint32_t CLOCK_GetRootMux (clock_root_control_t rootClk)
  Get clock root mux.
- static void CLOCK_EnableRoot (clock_root_control_t rootClk)
  Enable clock root.
- static void CLOCK_DisableRoot (clock_root_control_t rootClk)
  Disable clock root.
- static bool CLOCK_IsRootEnabled (clock_root_control_t rootClk)
  Check whether clock root is enabled.
- void CLOCK_UpdateRoot (clock_root_control_t ccmRootClk, uint32_t mux, uint32_t pre, uint32_t post)
  Update clock root in one step, for dynamical clock switching Note: The PRE and POST dividers in this function are the actually divider, software will map it to register value.
- void CLOCK_SetRootDivider (clock_root_control_t ccmRootClk, uint32_t pre, uint32_t post)
  Set root clock divider Note: The PRE and POST dividers in this function are the actually divider, software will map it to register value.
- static uint32_t CLOCK_GetRootPreDivider (clock_root_control_t rootClk)
  Get clock root PRE_PODF.
- static uint32_t CLOCK_GetRootPostDivider (clock_root_control_t rootClk)
  Get clock root POST_PODF.

OSC setting

- void CLOCK_InitOSC25M (const osc_config_t *config)
  OSC25M init.
- void CLOCK_DeinitOSC25M (void)
  OSC25M deinit.
- void CLOCK_InitOSC27M (const osc_config_t *config)
  OSC27M init.
- void CLOCK_DeinitOSC27M (void)
  OSC27M deinit.
- static void CLOCK_SwitchOSC32Src (osc32_src_t sel)
  switch 32KHZ OSC input

CCM Gate Control

- static void CLOCK_ControlGate (uint32_t ccmGate, clock_gate_value_t control)
  Set PLL or CCGR gate control.
- void CLOCK_EnableClock (clock_ip_name_t ccmGate)
  Enable CCGR clock gate and root clock gate for each module User should set specific gate for each module according to the description of the table of system clocks, gating and override in CCM chapter of reference manual.
- void CLOCK_DisableClock (clock_ip_name_t ccmGate)

Disable CCGR clock gate for the each module User should set specific gate for each module according to the description of the table of system clocks, gating and override in CCM chapter of reference manual.

CCM Analog PLL Operatoin Functions

- static void CLOCK_PowerUpPll (CCM_ANALOG_Type ∗base, clock_pll_ctrl_t pllControl)
  Power up PLL.
- static void CLOCK_PowerDownPll (CCM_ANALOG_Type ∗base, clock_pll_ctrl_t pllControl)
  Power down PLL.
- static void CLOCK_SetPllBypass (CCM_ANALOG_Type ∗base, clock_pll_bypass_ctrl_t pllControl, bool bypass)
  PLL bypass setting.
- static bool CLOCK_IsPllBypassed (CCM_ANALOG_Type ∗base, clock_pll_bypass_ctrl_t pllControl)
  Check if PLL is bypassed.
- static bool CLOCK_IsPllLocked (CCM_ANALOG_Type ∗base, clock_pll_ctrl_t pllControl)
  Check if PLL clock is locked.
- static void CLOCK_EnableAnalogClock (CCM_ANALOG_Type ∗base, clock_pll_clke_t pllClock)
  Enable PLL clock.
- static void CLOCK_DisableAnalogClock (CCM_ANALOG_Type ∗base, clock_pll_clke_t pllClock)
  Disable PLL clock.
- static void CLOCK_OverrideAnalogClke (CCM_ANALOG_Type ∗base, clock_pll_clke_t ovClock, bool override)
  Override PLL clock output enable.
- static void CLOCK_OverridePllPd (CCM_ANALOG_Type ∗base, clock_pll_ctrl_t pdClock, bool override)
  Override PLL power down.
- void CLOCK_InitArmPll (const ccm_analog_frac_pll_config_t ∗config)
  Initializes the ANALOG ARM PLL.
- void CLOCK_DeinitArmPll (void)
  De-initialize the ARM PLL.
- void CLOCK_InitSysPll1 (const ccm_analog_sscg_pll_config_t ∗config)
  Initializes the ANALOG SYS PLL1.
- void CLOCK_DeinitSysPll1 (void)
  De-initialize the System PLL1.
- void CLOCK_InitSysPll2 (const ccm_analog_sscg_pll_config_t ∗config)
  Initializes the ANALOG SYS PLL2.
- void CLOCK_DeinitSysPll2 (void)
  De-initialize the System PLL2.
- void CLOCK_InitSysPll3 (const ccm_analog_sscg_pll_config_t ∗config)
  Initializes the ANALOG SYS PLL3.
- void CLOCK_DeinitSysPll3 (void)
  De-initialize the System PLL3.
- void CLOCK_InitDramPll (const ccm_analog_sscg_pll_config_t ∗config)
  Initializes the ANALOG DDR PLL.
- void CLOCK_DeinitDramPll (void)
  De-initialize the Dram PLL.
- void CLOCK_InitAudioPll1 (const ccm_analog_frac_pll_config_t ∗config)

Initializes the ANALOG AUDIO PLL1.
- void CLOCK_DeinitAudioPll1 (void)
  De-initialize the Audio PLL1.
- void CLOCK_InitAudioPll2 (const ccm_analog_frac_pll_config_t ∗config)
  Initializes the ANALOG AUDIO PLL2.
- void CLOCK_DeinitAudioPll2 (void)
  De-initialize the Audio PLL2.
- void CLOCK_InitVideoPll1 (const ccm_analog_frac_pll_config_t ∗config)
  Initializes the ANALOG VIDEO PLL1.
- void CLOCK_DeinitVideoPll1 (void)
  De-initialize the Video PLL1.
- void CLOCK_InitVideoPll2 (const ccm_analog_sscg_pll_config_t ∗config)
  Initializes the ANALOG VIDEO PLL2.
- void CLOCK_DeinitVideoPll2 (void)
  De-initialize the Video PLL2.
- void CLOCK_InitSSCGPll (CCM_ANALOG_Type ∗base, const ccm_analog_sscg_pll_config_t ∗config, clock_pll_ctrl_t type)
  Initializes the ANALOG SSCG PLL.
- uint32_t CLOCK_GetSSCGPllFreq (CCM_ANALOG_Type ∗base, clock_pll_ctrl_t type, uint32_t refClkFreq, bool pll1Bypass)
  Get the ANALOG SSCG PLL clock frequency.
- void CLOCK_InitFracPll (CCM_ANALOG_Type ∗base, const ccm_analog_frac_pll_config_t ∗config, clock_pll_ctrl_t type)
  Initializes the ANALOG Fractional PLL.
- uint32_t CLOCK_GetFracPllFreq (CCM_ANALOG_Type ∗base, clock_pll_ctrl_t type, uint32_t refClkFreq)
  Gets the ANALOG Fractional PLL clock frequency.
- uint32_t CLOCK_GetPllFreq (clock_pll_ctrl_t pll)
  Gets PLL clock frequency.
- uint32_t CLOCK_GetPllRefClkFreq (clock_pll_ctrl_t ctrl)
  Gets PLL reference clock frequency.

CCM Get frequency

- uint32_t CLOCK_GetFreq (clock_name_t clockName)
  Gets the clock frequency for a specific clock name.
- uint32_t CLOCK_GetCoreM4Freq (void)
  Get the CCM Cortex M4 core frequency.
- uint32_t CLOCK_GetAxiFreq (void)
  Get the CCM Axi bus frequency.
- uint32_t CLOCK_GetAhbFreq (void)
  Get the CCM Ahb bus frequency.

Data Structure Documentation

6.2.1 struct osc_config_t

Data Fields

- uint8_t oscMode

ext or osc mode
- uint8_t oscDiv
  osc divider

6.2.2 struct ccm_analog_frac_pll_config_t

Note: all the dividers in this configuration structure are the actually divider, software will map it to register value

Data Fields

- uint8_t refSel
  pll reference clock sel
- uint8_t refDiv
  A 6bit divider to make sure the REF must be within the range 10MHZ~300MHZ.
- uint32_t fractionDiv
  Inlcude fraction divider(divider:1:2^24) output clock range is 2000MHZ-4000MHZ.
- uint8_t outDiv
  output clock divide, output clock range is 30MHZ to 2000MHZ, must be a even value

6.2.3 struct ccm_analog_sscg_pll_config_t

Note: all the dividers in this configuration structure are the actually divider, software will map it to register value

Data Fields

- uint8_t refSel
  pll reference clock sel
- uint8_t refDiv1
  A 3bit divider to make sure the REF must be within the range 25MHZ~235MHZ ,post_divide REF must be within the range 25MHZ~54MHZ.
- uint8_t refDiv2
  A 6bit divider to make sure the post_divide REF must be within the range 54MHZ~75MHZ.
- uint32_t loopDivider1
  A 6bit internal PLL1 feedback clock divider, output clock range must be within the range 1600MHZ-2400-MHZ.
- uint32_t loopDivider2
  A 6bit internal PLL2 feedback clock divider, output clock range must be within the range 1200MHZ-2400-MHZ.
- uint8_t outDiv
  A 6bit output clock divide, output clock range is 20MHZ to 1200MHZ.

Macro Definition Documentation

6.3.1 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 3, 3))

6.3.2 #define ECSPI_CLOCKS

Value:

```
{                                                                  \
        kCLOCK_IpInvalid, kCLOCK_Ecspi1, kCLOCK_Ecspi2,
      kCLOCK_Ecspi3, \
    }
```

6.3.3 #define GPIO_CLOCKS

Value:

```
{                                                                                          \
        kCLOCK_IpInvalid, kCLOCK_Gpio1, kCLOCK_Gpio2,
      kCLOCK_Gpio3, kCLOCK_Gpio4, kCLOCK_Gpio5, \
    }
```

6.3.4 #define GPT_CLOCKS

Value:

```
{                                                                                                  \
        kCLOCK_IpInvalid, kCLOCK_Gpt1, kCLOCK_Gpt2,
      kCLOCK_Gpt3, kCLOCK_Gpt4, kCLOCK_Gpt5,
      kCLOCK_Gpt6, \
    }
```

6.3.5 #define I2C_CLOCKS

Value:

```
{                                                                          \
        kCLOCK_IpInvalid, kCLOCK_I2c1, kCLOCK_I2c2,
      kCLOCK_I2c3, kCLOCK_I2c4, \
    }
```

6.3.6 #define IOMUX_CLOCKS

Value:

```
{                     \
        kCLOCK_Iomux, \
    }
```

6.3.7 #define IPMUX_CLOCKS

Value:

```
{                                                                \
        kCLOCK_Ipmux1, kCLOCK_Ipmux2,
      kCLOCK_Ipmux3, kCLOCK_Ipmux4, \
    }
```

6.3.8 #define PWM_CLOCKS

Value:

```
{                                                                          \
        kCLOCK_IpInvalid, kCLOCK_Pwm1, kCLOCK_Pwm2,
      kCLOCK_Pwm3, kCLOCK_Pwm4, \
    }
```

6.3.9 #define RDC_CLOCKS

Value:

```
{                   \
        kCLOCK_Rdc, \
    }
```

6.3.10 #define SAI_CLOCKS

Value:

```
{                                                                                                      \
        kCLOCK_IpInvalid, kCLOCK_Sai1, kCLOCK_Sai2,
      kCLOCK_Sai3, kCLOCK_Sai4, kCLOCK_Sai5,
      kCLOCK_Sai6, \
    }
```

6.3.11 #define RDC_SEMA42_CLOCKS

Value:

```
{                                                    \
        kCLOCK_IpInvalid, kCLOCK_Sema42_1, kCLOCK_Sema42_2 \
    }
```

6.3.12 #define UART_CLOCKS

Value:

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

6.3.13 #define USDHC_CLOCKS

Value:

```
{                                                \
        kCLOCK_IpInvalid, kCLOCK_Usdhc1, kCLOCK_Usdhc2 \
    }
```

6.3.14 #define WDOG_CLOCKS

Value:

```
{                                                              \
        kCLOCK_IpInvalid, kCLOCK_Wdog1, kCLOCK_Wdog2,
      kCLOCK_Wdog3 \
    }
```

6.3.15 #define TMU_CLOCKS

Value:

```
{                      \
        kCLOCK_TempSensor, \
    }
```

6.3.16 #define SDMA_CLOCKS

Value:

```
{                               \
        kCLOCK_Sdma1, kCLOCK_Sdma2 \
    }
```

6.3.17 #define MU_CLOCKS

Value:

```
{             \
        kCLOCK_Mu \
    }
```

6.3.18 #define QSPI_CLOCKS

Value:

```
{               \
        kCLOCK_Qspi \
    }
```

6.3.19 #define kCLOCK_CoreSysClk kCLOCK_CoreM4Clk

6.3.20 #define CLOCK_GetCoreSysClkFreq CLOCK_GetCoreM4Freq

Enumeration Type Documentation

6.4.1 enum clock_name_t

Enumerator

kCLOCK_CoreM4Clk ARM M4 Core clock.
kCLOCK_AxiClk Main AXI bus clock.
kCLOCK_AhbClk AHB bus clock.
kCLOCK_IpgClk IPG bus clock.

6.4.2 enum clock_ip_name_t

Enumerator

kCLOCK_Debug DEBUG Clock Gate.
kCLOCK_Dram DRAM Clock Gate.
kCLOCK_Ecspi1 ECSPI1 Clock Gate.
kCLOCK_Ecspi2 ECSPI2 Clock Gate.
kCLOCK_Ecspi3 ECSPI3 Clock Gate.
kCLOCK_Gpio1 GPIO1 Clock Gate.
kCLOCK_Gpio2 GPIO2 Clock Gate.
kCLOCK_Gpio3 GPIO3 Clock Gate.
kCLOCK_Gpio4 GPIO4 Clock Gate.
kCLOCK_Gpio5 GPIO5 Clock Gate.
kCLOCK_Gpt1 GPT1 Clock Gate.
kCLOCK_Gpt2 GPT2 Clock Gate.
kCLOCK_Gpt3 GPT3 Clock Gate.
kCLOCK_Gpt4 GPT4 Clock Gate.
kCLOCK_Gpt5 GPT5 Clock Gate.
kCLOCK_Gpt6 GPT6 Clock Gate.
kCLOCK_I2c1 I2C1 Clock Gate.
kCLOCK_I2c2 I2C2 Clock Gate.
kCLOCK_I2c3 I2C3 Clock Gate.
kCLOCK_I2c4 I2C4 Clock Gate.
kCLOCK_Iomux IOMUX Clock Gate.
kCLOCK_Ipmux1 IPMUX1 Clock Gate.
kCLOCK_Ipmux2 IPMUX2 Clock Gate.
kCLOCK_Ipmux3 IPMUX3 Clock Gate.
kCLOCK_Ipmux4 IPMUX4 Clock Gate.
kCLOCK_M4 M4 Clock Gate.
kCLOCK_Mu MU Clock Gate.
kCLOCK_Ocram OCRAM Clock Gate.
kCLOCK_OcramS OCRAM S Clock Gate.
kCLOCK_Pwm1 PWM1 Clock Gate.
kCLOCK_Pwm2 PWM2 Clock Gate.
kCLOCK_Pwm3 PWM3 Clock Gate.
kCLOCK_Pwm4 PWM4 Clock Gate.
kCLOCK_Qspi QSPI Clock Gate.
kCLOCK_Rdc RDC Clock Gate.
kCLOCK_Sai1 SAI1 Clock Gate.
kCLOCK_Sai2 SAI2 Clock Gate.
kCLOCK_Sai3 SAI3 Clock Gate.
kCLOCK_Sai4 SAI4 Clock Gate.
kCLOCK_Sai5 SAI5 Clock Gate.
kCLOCK_Sai6 SAI6 Clock Gate.

kCLOCK_Sdma1 SDMA1 Clock Gate.
kCLOCK_Sdma2 SDMA2 Clock Gate.
kCLOCK_Sec_Debug SEC_DEBUG Clock Gate.
kCLOCK_Sema42_1 RDC SEMA42 Clock Gate.
kCLOCK_Sema42_2 RDC SEMA42 Clock Gate.
kCLOCK_Sim_display SIM_Display Clock Gate.
kCLOCK_Sim_m SIM_M Clock Gate.
kCLOCK_Sim_main SIM_MAIN Clock Gate.
kCLOCK_Sim_s SIM_S Clock Gate.
kCLOCK_Sim_wakeup SIM_WAKEUP Clock Gate.
kCLOCK_Uart1 UART1 Clock Gate.
kCLOCK_Uart2 UART2 Clock Gate.
kCLOCK_Uart3 UART3 Clock Gate.
kCLOCK_Uart4 UART4 Clock Gate.
kCLOCK_Wdog1 WDOG1 Clock Gate.
kCLOCK_Wdog2 WDOG2 Clock Gate.
kCLOCK_Wdog3 WDOG3 Clock Gate.
kCLOCK_TempSensor TempSensor Clock Gate.

6.4.3 enum clock_root_control_t

Enumerator

kCLOCK_RootM4 ARM Cortex-M4 Clock control name.
kCLOCK_RootAxi AXI Clock control name.
kCLOCK_RootNoc NOC Clock control name.
kCLOCK_RootAhb AHB Clock control name.
kCLOCK_RootIpg IPG Clock control name.
kCLOCK_RootDramAlt DRAM ALT Clock control name.
kCLOCK_RootSai1 SAI1 Clock control name.
kCLOCK_RootSai2 SAI2 Clock control name.
kCLOCK_RootSai3 SAI3 Clock control name.
kCLOCK_RootSai4 SAI4 Clock control name.
kCLOCK_RootSai5 SAI5 Clock control name.
kCLOCK_RootSai6 SAI6 Clock control name.
kCLOCK_RootQspi QSPI Clock control name.
kCLOCK_RootI2c1 I2C1 Clock control name.
kCLOCK_RootI2c2 I2C2 Clock control name.
kCLOCK_RootI2c3 I2C3 Clock control name.
kCLOCK_RootI2c4 I2C4 Clock control name.
kCLOCK_RootUart1 UART1 Clock control name.
kCLOCK_RootUart2 UART2 Clock control name.
kCLOCK_RootUart3 UART3 Clock control name.
kCLOCK_RootUart4 UART4 Clock control name.

kCLOCK_RootEcspi1 ECSPI1 Clock control name.
kCLOCK_RootEcspi2 ECSPI2 Clock control name.
kCLOCK_RootEcspi3 ECSPI3 Clock control name.
kCLOCK_RootPwm1 PWM1 Clock control name.
kCLOCK_RootPwm2 PWM2 Clock control name.
kCLOCK_RootPwm3 PWM3 Clock control name.
kCLOCK_RootPwm4 PWM4 Clock control name.
kCLOCK_RootGpt1 GPT1 Clock control name.
kCLOCK_RootGpt2 GPT2 Clock control name.
kCLOCK_RootGpt3 GPT3 Clock control name.
kCLOCK_RootGpt4 GPT4 Clock control name.
kCLOCK_RootGpt5 GPT5 Clock control name.
kCLOCK_RootGpt6 GPT6 Clock control name.
kCLOCK_RootWdog WDOG Clock control name.

6.4.4 enum clock_rootmux_m4_clk_sel_t

Enumerator

kCLOCK_M4RootmuxOsc25m ARM Cortex-M4 Clock from OSC 25M.
kCLOCK_M4RootmuxSysPll2Div5 ARM Cortex-M4 Clock from SYSTEM PLL2 divided by 5.
kCLOCK_M4RootmuxSysPll2Div4 ARM Cortex-M4 Clock from SYSTEM PLL2 divided by 4.
kCLOCK_M4RootmuxSysPll1Div3 ARM Cortex-M4 Clock from SYSTEM PLL1 divided by 3.
kCLOCK_M4RootmuxSysPll1 ARM Cortex-M4 Clock from SYSTEM PLL1.
kCLOCK_M4RootmuxAudioPll1 ARM Cortex-M4 Clock from AUDIO PLL1.
kCLOCK_M4RootmuxVideoPll1 ARM Cortex-M4 Clock from VIDEO PLL1.
kCLOCK_M4RootmuxSysPll3 ARM Cortex-M4 Clock from SYSTEM PLL3.

6.4.5 enum clock_rootmux_axi_clk_sel_t

Enumerator

kCLOCK_AxiRootmuxOsc25m ARM AXI Clock from OSC 25M.
kCLOCK_AxiRootmuxSysPll2Div3 ARM AXI Clock from SYSTEM PLL2 divided by 3.
kCLOCK_AxiRootmuxSysPll1 ARM AXI Clock from SYSTEM PLL1.
kCLOCK_AxiRootmuxSysPll2Div4 ARM AXI Clock from SYSTEM PLL2 divided by 4.
kCLOCK_AxiRootmuxSysPll2 ARM AXI Clock from SYSTEM PLL2.
kCLOCK_AxiRootmuxAudioPll1 ARM AXI Clock from AUDIO PLL1.
kCLOCK_AxiRootmuxVideoPll1 ARM AXI Clock from VIDEO PLL1.
kCLOCK_AxiRootmuxSysPll1Div8 ARM AXI Clock from SYSTEM PLL1 divided by 8.

6.4.6 enum clock_rootmux_ahb_clk_sel_t

Enumerator

kCLOCK_AhbRootmuxOsc25m ARM AHB Clock from OSC 25M.
kCLOCK_AhbRootmuxSysPll1Div6 ARM AHB Clock from SYSTEM PLL1 divided by 6.
kCLOCK_AhbRootmuxSysPll1 ARM AHB Clock from SYSTEM PLL1.
kCLOCK_AhbRootmuxSysPll1Div2 ARM AHB Clock from SYSTEM PLL1 divided by 2.
kCLOCK_AhbRootmuxSysPll2Div8 ARM AHB Clock from SYSTEM PLL2 divided by 8.
kCLOCK_AhbRootmuxSysPll3 ARM AHB Clock from SYSTEM PLL3.
kCLOCK_AhbRootmuxAudioPll1 ARM AHB Clock from AUDIO PLL1.
kCLOCK_AhbRootmuxVideoPll1 ARM AHB Clock from VIDEO PLL1.

6.4.7 enum clock_rootmux_qspi_clk_sel_t

Enumerator

kCLOCK_QspiRootmuxOsc25m ARM QSPI Clock from OSC 25M.
kCLOCK_QspiRootmuxSysPll1Div2 ARM QSPI Clock from SYSTEM PLL1 divided by 2.
kCLOCK_QspiRootmuxSysPll1 ARM QSPI Clock from SYSTEM PLL1.
kCLOCK_QspiRootmuxSysPll2Div2 ARM QSPI Clock from SYSTEM PLL2 divided by 2.
kCLOCK_QspiRootmuxAudioPll2 ARM QSPI Clock from AUDIO PLL2.
kCLOCK_QspiRootmuxSysPll1Div3 ARM QSPI Clock from SYSTEM PLL1 divided by 3.
kCLOCK_QspiRootmuxSysPll3 ARM QSPI Clock from SYSTEM PLL3.
kCLOCK_QspiRootmuxSysPll1Div8 ARM QSPI Clock from SYSTEM PLL1 divided by 8.

6.4.8 enum clock_rootmux_ecspi_clk_sel_t

Enumerator

kCLOCK_EcspiRootmuxOsc25m ECSPI Clock from OSC 25M.
kCLOCK_EcspiRootmuxSysPll2Div5 ECSPI Clock from SYSTEM PLL2 divided by 5.
kCLOCK_EcspiRootmuxSysPll1Div20 ECSPI Clock from SYSTEM PLL1 divided by 20.
kCLOCK_EcspiRootmuxSysPll1Div5 ECSPI Clock from SYSTEM PLL1 divided by 5.
kCLOCK_EcspiRootmuxSysPll1 ECSPI Clock from SYSTEM PLL1.
kCLOCK_EcspiRootmuxSysPll3 ECSPI Clock from SYSTEM PLL3.
kCLOCK_EcspiRootmuxSysPll2Div4 ECSPI Clock from SYSTEM PLL2 divided by 4.
kCLOCK_EcspiRootmuxAudioPll2 ECSPI Clock from AUDIO PLL2.

6.4.9 enum clock_rootmux_i2c_clk_sel_t

Enumerator

kCLOCK_I2cRootmuxOsc25m I2C Clock from OSC 25M.
kCLOCK_I2cRootmuxSysPll1Div5 I2C Clock from SYSTEM PLL1 divided by 5.
kCLOCK_I2cRootmuxSysPll2Div20 I2C Clock from SYSTEM PLL2 divided by 20.
kCLOCK_I2cRootmuxSysPll3 I2C Clock from SYSTEM PLL3 .
kCLOCK_I2cRootmuxAudioPll1 I2C Clock from AUDIO PLL1.
kCLOCK_I2cRootmuxVideoPll1 I2C Clock from VIDEO PLL1.
kCLOCK_I2cRootmuxAudioPll2 I2C Clock from AUDIO PLL2.
kCLOCK_I2cRootmuxSysPll1Div6 I2C Clock from SYSTEM PLL1 divided by 6.

6.4.10 enum clock_rootmux_uart_clk_sel_t

Enumerator

kCLOCK_UartRootmuxOsc25m UART Clock from OSC 25M.
kCLOCK_UartRootmuxSysPll1Div10 UART Clock from SYSTEM PLL1 divided by 10.
kCLOCK_UartRootmuxSysPll2Div5 UART Clock from SYSTEM PLL2 divided by 5.
kCLOCK_UartRootmuxSysPll2Div10 UART Clock from SYSTEM PLL2 divided by 10.
kCLOCK_UartRootmuxSysPll3 UART Clock from SYSTEM PLL3.
kCLOCK_UartRootmuxExtClk2 UART Clock from External Clock 2.
kCLOCK_UartRootmuxExtClk34 UART Clock from External Clock 3, External Clock 4.
kCLOCK_UartRootmuxAudioPll2 UART Clock from Audio PLL2.

6.4.11 enum clock_rootmux_gpt_t

Enumerator

kCLOCK_GptRootmuxOsc25m GPT Clock from OSC 25M.
kCLOCK_GptRootmuxSystemPll2Div10 GPT Clock from SYSTEM PLL2 divided by 10.
kCLOCK_GptRootmuxSysPll1Div2 GPT Clock from SYSTEM PLL1 divided by 2.
kCLOCK_GptRootmuxSysPll1Div20 GPT Clock from SYSTEM PLL1 divided by 20.
kCLOCK_GptRootmuxVideoPll1 GPT Clock from VIDEO PLL1.
kCLOCK_GptRootmuxSystemPll1Div10 GPT Clock from SYSTEM PLL1 divided by 10.
kCLOCK_GptRootmuxAudioPll1 GPT Clock from AUDIO PLL1.
kCLOCK_GptRootmuxExtClk123 GPT Clock from External Clock1, External Clock2, External Clock3.

6.4.12 enum clock_rootmux_wdog_clk_sel_t

Enumerator

kCLOCK_WdogRootmuxOsc25m WDOG Clock from OSC 25M.
kCLOCK_WdogRootmuxSysPll1Div6 WDOG Clock from SYSTEM PLL1 divided by 6.
kCLOCK_WdogRootmuxSysPll1Div5 WDOG Clock from SYSTEM PLL1 divided by 5.
kCLOCK_WdogRootmuxVpuPll WDOG Clock from VPU DLL.
kCLOCK_WdogRootmuxSystemPll2Div8 WDOG Clock from SYSTEM PLL2 divided by 8.
kCLOCK_WdogRootmuxSystemPll3 WDOG Clock from SYSTEM PLL3.
kCLOCK_WdogRootmuxSystemPll1Div10 WDOG Clock from SYSTEM PLL1 divided by 10.
kCLOCK_WdogRootmuxSystemPll2Div6 WDOG Clock from SYSTEM PLL2 divided by 6.

6.4.13 enum clock_rootmux_Pwm_clk_sel_t

Enumerator

kCLOCK_PwmRootmuxOsc25m PWM Clock from OSC 25M.
kCLOCK_PwmRootmuxSysPll2Div10 PWM Clock from SYSTEM PLL2 divided by 10.
kCLOCK_PwmRootmuxSysPll1Div5 PWM Clock from SYSTEM PLL1 divided by 5.
kCLOCK_PwmRootmuxSysPll1Div20 PWM Clock from SYSTEM PLL1 divided by 20.
kCLOCK_PwmRootmuxSystemPll3 PWM Clock from SYSTEM PLL3.
kCLOCK_PwmRootmuxExtClk12 PWM Clock from External Clock1, External Clock2.
kCLOCK_PwmRootmuxSystemPll1Div10 PWM Clock from SYSTEM PLL1 divided by 10.
kCLOCK_PwmRootmuxVideoPll1 PWM Clock from VIDEO PLL1.

6.4.14 enum clock_rootmux_sai_clk_sel_t

Enumerator

kCLOCK_SaiRootmuxOsc25m SAI Clock from OSC 25M.
kCLOCK_SaiRootmuxAudioPll1 SAI Clock from AUDIO PLL1.
kCLOCK_SaiRootmuxAudioPll2 SAI Clock from AUDIO PLL2.
kCLOCK_SaiRootmuxVideoPll1 SAI Clock from VIDEO PLL1.
kCLOCK_SaiRootmuxSysPll1Div6 SAI Clock from SYSTEM PLL1 divided by 6.
kCLOCK_SaiRootmuxOsc27m SAI Clock from OSC 27M.
kCLOCK_SaiRootmuxExtClk123 SAI Clock from External Clock1, External Clock2, External Clock3.
kCLOCK_SaiRootmuxExtClk234 SAI Clock from External Clock2, External Clock3, External Clock4.

6.4.15 enum clock_rootmux_noc_clk_sel_t

Enumerator

kCLOCK_NocRootmuxOsc25m NOC Clock from OSC 25M.
kCLOCK_NocRootmuxSysPll1 NOC Clock from SYSTEM PLL1.
kCLOCK_NocRootmuxSysPll3 NOC Clock from SYSTEM PLL3.
kCLOCK_NocRootmuxSysPll2 NOC Clock from SYSTEM PLL2.
kCLOCK_NocRootmuxSysPll2Div2 NOC Clock from SYSTEM PLL2 divided by 2.
kCLOCK_NocRootmuxAudioPll1 NOC Clock from AUDIO PLL1.
kCLOCK_NocRootmuxVideoPll1 NOC Clock from VIDEO PLL1.
kCLOCK_NocRootmuxAudioPll2 NOC Clock from AUDIO PLL2.

6.4.16 enum clock_pll_gate_t

Enumerator

kCLOCK_ArmPllGate ARM PLL Gate.
kCLOCK_GpuPllGate GPU PLL Gate.
kCLOCK_VpuPllGate VPU PLL Gate.
kCLOCK_DramPllGate DRAM PLL1 Gate.
kCLOCK_SysPll1Gate SYSTEM PLL1 Gate.
kCLOCK_SysPll1Div2Gate SYSTEM PLL1 Div2 Gate.
kCLOCK_SysPll1Div3Gate SYSTEM PLL1 Div3 Gate.
kCLOCK_SysPll1Div4Gate SYSTEM PLL1 Div4 Gate.
kCLOCK_SysPll1Div5Gate SYSTEM PLL1 Div5 Gate.
kCLOCK_SysPll1Div6Gate SYSTEM PLL1 Div6 Gate.
kCLOCK_SysPll1Div8Gate SYSTEM PLL1 Div8 Gate.
kCLOCK_SysPll1Div10Gate SYSTEM PLL1 Div10 Gate.
kCLOCK_SysPll1Div20Gate SYSTEM PLL1 Div20 Gate.
kCLOCK_SysPll2Gate SYSTEM PLL2 Gate.
kCLOCK_SysPll2Div2Gate SYSTEM PLL2 Div2 Gate.
kCLOCK_SysPll2Div3Gate SYSTEM PLL2 Div3 Gate.
kCLOCK_SysPll2Div4Gate SYSTEM PLL2 Div4 Gate.
kCLOCK_SysPll2Div5Gate SYSTEM PLL2 Div5 Gate.
kCLOCK_SysPll2Div6Gate SYSTEM PLL2 Div6 Gate.
kCLOCK_SysPll2Div8Gate SYSTEM PLL2 Div8 Gate.
kCLOCK_SysPll2Div10Gate SYSTEM PLL2 Div10 Gate.
kCLOCK_SysPll2Div20Gate SYSTEM PLL2 Div20 Gate.
kCLOCK_SysPll3Gate SYSTEM PLL3 Gate.
kCLOCK_AudioPll1Gate AUDIO PLL1 Gate.
kCLOCK_AudioPll2Gate AUDIO PLL2 Gate.
kCLOCK_VideoPll1Gate VIDEO PLL1 Gate.
kCLOCK_VideoPll2Gate VIDEO PLL2 Gate.

6.4.17 enum clock_gate_value_t

Enumerator

kCLOCK_ClockNotNeeded Clock always disabled.
kCLOCK_ClockNeededRun Clock enabled when CPU is running.
kCLOCK_ClockNeededRunWait Clock enabled when CPU is running or in WAIT mode.
kCLOCK_ClockNeededAll Clock always enabled.

6.4.18 enum clock_pll_bypass_ctrl_t

These constants define the PLL control names for PLL bypass.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: bypass bit shift.

Enumerator

kCLOCK_AudioPll1BypassCtrl CCM Audio PLL1 bypass Control.
kCLOCK_AudioPll2BypassCtrl CCM Audio PLL2 bypass Control.
kCLOCK_VideoPll1BypassCtrl CCM Video Pll1 bypass Control.
kCLOCK_GpuPLLPwrBypassCtrl CCM Gpu PLL bypass Control.
kCLOCK_VpuPllPwrBypassCtrl CCM Vpu PLL bypass Control.
kCLOCK_ArmPllPwrBypassCtrl CCM Arm PLL bypass Control.
kCLOCK_SysPll1InternalPll1BypassCtrl CCM System PLL1 internal pll1 bypass Control.
kCLOCK_SysPll1InternalPll2BypassCtrl CCM System PLL1 internal pll2 bypass Control.
kCLOCK_SysPll2InternalPll1BypassCtrl CCM Analog System PLL1 internal pll1 bypass Control.

kCLOCK_SysPll2InternalPll2BypassCtrl CCM Analog VIDEO System PLL1 internal pll1 bypass Control.
kCLOCK_SysPll3InternalPll1BypassCtrl CCM Analog VIDEO PLL bypass Control.
kCLOCK_SysPll3InternalPll2BypassCtrl CCM Analog VIDEO PLL bypass Control.
kCLOCK_VideoPll2InternalPll1BypassCtrl CCM Analog 480M PLL bypass Control.
kCLOCK_VideoPll2InternalPll2BypassCtrl CCM Analog 480M PLL bypass Control.
kCLOCK_DramPllInternalPll1BypassCtrl CCM Analog 480M PLL bypass Control.
kCLOCK_DramPllInternalPll2BypassCtrl CCM Analog 480M PLL bypass Control.

6.4.19 enum clock_pll_clke_t

These constants define the PLL clock names for PLL clock enable/disable operations.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: Clock enable bit shift.

Enumerator

kCLOCK_AudioPll1Clke Audio pll1 clke.
kCLOCK_AudioPll2Clke Audio pll2 clke.
kCLOCK_VideoPll1Clke Video pll1 clke.
kCLOCK_GpuPllClke Gpu pll clke.
kCLOCK_VpuPllClke Vpu pll clke.
kCLOCK_ArmPllClke Arm pll clke.
kCLOCK_SystemPll1Clke System pll1 clke.
kCLOCK_SystemPll1Div2Clke System pll1 Div2 clke.
kCLOCK_SystemPll1Div3Clke System pll1 Div3 clke.
kCLOCK_SystemPll1Div4Clke System pll1 Div4 clke.
kCLOCK_SystemPll1Div5Clke System pll1 Div5 clke.
kCLOCK_SystemPll1Div6Clke System pll1 Div6 clke.
kCLOCK_SystemPll1Div8Clke System pll1 Div8 clke.
kCLOCK_SystemPll1Div10Clke System pll1 Div10 clke.
kCLOCK_SystemPll1Div20Clke System pll1 Div20 clke.
kCLOCK_SystemPll2Clke System pll2 clke.
kCLOCK_SystemPll2Div2Clke System pll2 Div2 clke.
kCLOCK_SystemPll2Div3Clke System pll2 Div3 clke.
kCLOCK_SystemPll2Div4Clke System pll2 Div4 clke.
kCLOCK_SystemPll2Div5Clke System pll2 Div5 clke.
kCLOCK_SystemPll2Div6Clke System pll2 Div6 clke.
kCLOCK_SystemPll2Div8Clke System pll2 Div8 clke.
kCLOCK_SystemPll2Div10Clke System pll2 Div10 clke.
kCLOCK_SystemPll2Div20Clke System pll2 Div20 clke.
kCLOCK_SystemPll3Clke System pll3 clke.
kCLOCK_VideoPll2Clke Video pll2 clke.
kCLOCK_DramPllClke Dram pll clke.
kCLOCK_OSC25MClke OSC25M clke.
kCLOCK_OSC27MClke OSC27M clke.

6.4.20 enum _osc_mode

Enumerator

kOSC_OscMode OSC oscillator mode.
kOSC_ExtMode OSC external mode.

6.4.21 enum osc32_src_t

Enumerator

kOSC32_Src25MDiv800 source from 25M divide 800

kOSC32_SrcRTC source from RTC

6.4.22 enum _ccm_analog_pll_ref_clk

Enumerator

kANALOG_PllRefOsc25M reference OSC 25M
kANALOG_PllRefOsc27M reference OSC 27M
kANALOG_PllRefOscHdmiPhy27M reference HDMI PHY 27M
kANALOG_PllRefClkPN reference CLK_P_N

Function Documentation

6.5.1 static void CLOCK_SetRootMux (clock_root_control_t *rootClk*, uint32_t *mux*) [inline], [static]

User maybe need to set more than one mux ROOT according to the clock tree description in the reference manual.

Parameters

rootClk	Root clock control (see clock_root_control_t enumeration).
mux	Root mux value (see _ccm_rootmux_xxx enumeration).

6.5.2 static uint32_t CLOCK_GetRootMux (clock_root_control_t *rootClk*) [inline], [static]

In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

Parameters

rootClk	Root clock control (see clock_root_control_t enumeration).

Returns

Root mux value (see _ccm_rootmux_xxx enumeration).

6.5.3 static void CLOCK_EnableRoot (clock_root_control_t *rootClk*) [inline], [static]

Parameters

rootClk	Root clock control (see clock_root_control_t enumeration)

6.5.4 static void CLOCK_DisableRoot (clock_root_control_t *rootClk*) [inline], [static]

Parameters

rootClk	Root control (see clock_root_control_t enumeration)

6.5.5 static bool CLOCK_IsRootEnabled (clock_root_control_t *rootClk*) [inline], [static]

Parameters

rootClk	Root control (see clock_root_control_t enumeration)

Returns

CCM root enabled or not.
- true: Clock root is enabled.
- false: Clock root is disabled.

6.5.6 void CLOCK_UpdateRoot (clock_root_control_t *ccmRootClk*, uint32_t *mux*, uint32_t *pre*, uint32_t *post*)

Parameters

ccmRootClk	Root control (see clock_root_control_t enumeration)
mux	root mux value (see _ccm_rootmux_xxx enumeration)
pre	Pre divider value (0-7, divider=n+1)

post	Post divider value (0-63, divider=n+1)

6.5.7 void CLOCK_SetRootDivider (clock_root_control_t *ccmRootClk*, uint32_t *pre*, uint32_t *post*)

Parameters

ccmRootClk	Root control (see clock_root_control_t enumeration)
pre	Pre divider value (1-8)
post	Post divider value (1-64)

6.5.8 static uint32_t CLOCK_GetRootPreDivider (clock_root_control_t *rootClk*) [inline], [static]

In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

Parameters

rootClk	Root clock name (see clock_root_control_t enumeration).

Returns

Root Pre divider value.

6.5.9 static uint32_t CLOCK_GetRootPostDivider (clock_root_control_t *rootClk*) [inline], [static]

In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

Parameters

rootClk	Root clock name (see clock_root_control_t enumeration).

Returns

Root Post divider value.

6.5.10 void CLOCK_InitOSC25M (const osc_config_t * *config*)

Parameters

config	osc configuration.

6.5.11 void CLOCK_DeinitOSC25M (void)

6.5.12 void CLOCK_InitOSC27M (const osc_config_t * *config*)

Parameters

config	osc configuration.

6.5.13 void CLOCK_DeinitOSC27M (void)

6.5.14 static void CLOCK_SwitchOSC32Src (osc32_src_t *sel*) [inline], [static]

Parameters

sel	OSC32 input clock select

6.5.15 static void CLOCK_ControlGate (uint32_t *ccmGate*, clock_gate_value_t *control*) [inline], [static]

Parameters

ccmGate	Gate control (see clock_pll_gate_t and clock_ip_name_t enumeration)
control	Gate control value (see clock_gate_value_t)

6.5.16 void CLOCK_EnableClock (clock_ip_name_t *ccmGate*)

Take care of that one module may need to set more than one clock gate.

Parameters

ccmGate	Gate control for each module (see clock_ip_name_t enumeration).

6.5.17 void CLOCK_DisableClock (clock_ip_name_t *ccmGate*)

Take care of that one module may need to set more than one clock gate.

Parameters

ccmGate	Gate control for each module (see clock_ip_name_t enumeration).

6.5.18 static void CLOCK_PowerUpPll (CCM_ANALOG_Type * *base*, clock_pll_ctrl_t *pllControl*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllControl	PLL control name (see clock_pll_ctrl_t enumeration)

6.5.19 static void CLOCK_PowerDownPll (CCM_ANALOG_Type * *base*, clock_pll_ctrl_t *pllControl*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllControl	PLL control name (see clock_pll_ctrl_t enumeration)

6.5.20 static void CLOCK_SetPllBypass (CCM_ANALOG_Type * *base*, clock_pll_bypass_ctrl_t *pllControl*, bool *bypass*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllControl	PLL control name (see ccm_analog_pll_control_t enumeration)
bypass	Bypass the PLL. • true: Bypass the PLL. • false: Do not bypass the PLL.

6.5.21 static bool CLOCK_IsPllBypassed (CCM_ANALOG_Type * *base*, clock_pll_bypass_ctrl_t *pllControl*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllControl	PLL control name (see ccm_analog_pll_control_t enumeration)

Returns

PLL bypass status.

- true: The PLL is bypassed.
- false: The PLL is not bypassed.

6.5.22 static bool CLOCK_IsPllLocked (CCM_ANALOG_Type * *base*, clock_pll_ctrl_t *pllControl*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllControl	PLL control name (see clock_pll_ctrl_t enumeration)

Returns

PLL lock status.

- true: The PLL clock is locked.
- false: The PLL clock is not locked.

6.5.23 static void CLOCK_EnableAnalogClock (CCM_ANALOG_Type ∗ *base*, clock_pll_clke_t *pllClock*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllClock	PLL clock name (see ccm_analog_pll_clock_t enumeration)

6.5.24 static void CLOCK_DisableAnalogClock (CCM_ANALOG_Type ∗ *base*, clock_pll_clke_t *pllClock*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllClock	PLL clock name (see ccm_analog_pll_clock_t enumeration)

6.5.25 static void CLOCK_OverrideAnalogClke (CCM_ANALOG_Type ∗ *base*, clock_pll_clke_t *ovClock*, bool *override*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
ovClock	PLL clock name (see clock_pll_clke_t enumeration)
override	Override the PLL. • true: Override the PLL clke, CCM will handle it. • false: Do not override the PLL clke.

6.5.26 static void CLOCK_OverridePllPd (CCM_ANALOG_Type ∗ *base,* clock_pll_ctrl_t *pdClock,* bool *override*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pdClock	PLL clock name (see clock_pll_ctrl_t enumeration)
override	Override the PLL. • true: Override the PLL clke, CCM will handle it. • false: Do not override the PLL clke.

6.5.27 void CLOCK_InitArmPll (const ccm_analog_frac_pll_config_t ∗ *config*)

Parameters

config	Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).

Note

This function can't detect whether the Arm PLL has been enabled and used by some IPs.

6.5.28 void CLOCK_InitSysPll1 (const ccm_analog_sscg_pll_config_t ∗ *config*)

Parameters

config	Pointer to the configuration structure(see ccm_analog_sscg_pll_config_t enumeration).

Note

This function can't detect whether the SYS PLL has been enabled and used by some IPs.

6.5.29 void CLOCK_InitSysPll2 (const ccm_analog_sscg_pll_config_t * *config*)

Parameters

config	Pointer to the configuration structure(see ccm_analog_sscg_pll_config_t enumeration).

Note

This function can't detect whether the SYS PLL has been enabled and used by some IPs.

6.5.30 void CLOCK_InitSysPll3 (const ccm_analog_sscg_pll_config_t * *config*)

Parameters

config	Pointer to the configuration structure(see ccm_analog_sscg_pll_config_t enumeration).

Note

This function can't detect whether the SYS PLL has been enabled and used by some IPs.

6.5.31 void CLOCK_InitDramPll (const ccm_analog_sscg_pll_config_t * *config*)

Parameters

config	Pointer to the configuration structure(see ccm_analog_sscg_pll_config_t enumeration).

Note

This function can't detect whether the DDR PLL has been enabled and used by some IPs.

6.5.32 void CLOCK_InitAudioPll1 (const ccm_analog_frac_pll_config_t ∗ *config*)

Parameters

config	Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).

Note

This function can't detect whether the AUDIO PLL has been enabled and used by some IPs.

6.5.33 void CLOCK_InitAudioPll2 (const ccm_analog_frac_pll_config_t ∗ *config*)

Parameters

config	Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).

Note

This function can't detect whether the AUDIO PLL has been enabled and used by some IPs.

6.5.34 void CLOCK_InitVideoPll1 (const ccm_analog_frac_pll_config_t ∗ *config*)

Parameters

config	Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).

6.5.35 void CLOCK_InitVideoPll2 (const ccm_analog_sscg_pll_config_t * *config*)

Parameters

config	Pointer to the configuration structure(see ccm_analog_sscg_pll_config_t enumeration).

Note

This function can't detect whether the VIDEO PLL has been enabled and used by some IPs.

6.5.36 void CLOCK_InitSSCGPll (CCM_ANALOG_Type * *base,* const ccm_analog_sscg_pll_config_t * *config,* clock_pll_ctrl_t *type*)

Parameters

base	CCM ANALOG base address
config	Pointer to the configuration structure(see ccm_analog_sscg_pll_config_t enumeration).
type	sscg pll type

6.5.37 uint32_t CLOCK_GetSSCGPllFreq (CCM_ANALOG_Type * *base,* clock_pll_ctrl_t *type,* uint32_t *refClkFreq,* bool *pll1Bypass*)

Parameters

base	CCM ANALOG base address.

type	sscg pll type
refClkFreq	reference clock frequency
pll1Bypass	pll1 bypass flag

Returns

Clock frequency

6.5.38 void CLOCK_InitFracPll (CCM_ANALOG_Type * *base*, const ccm_analog_frac_pll_config_t * *config*, clock_pll_ctrl_t *type*)

Parameters

base	CCM ANALOG base address.
config	Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).
type	fractional pll type.

6.5.39 uint32_t CLOCK_GetFracPllFreq (CCM_ANALOG_Type * *base*, clock_pll_ctrl_t *type*, uint32_t *refClkFreq*)

Parameters

base	CCM_ANALOG base pointer.
type	fractional pll type.
refClkFreq	reference clock frequency

Returns

Clock frequency

6.5.40 uint32_t CLOCK_GetPllFreq (clock_pll_ctrl_t *pll*)

Parameters

pll	fractional pll type.

Returns

Clock frequency

6.5.41 uint32_t CLOCK_GetPllRefClkFreq (clock_pll_ctrl_t *ctrl*)

Parameters

ctrl	fractional pll type.

Returns

Clock frequency

6.5.42 uint32_t CLOCK_GetFreq (clock_name_t *clockName*)

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock_name_t.

Parameters

clockName	Clock names defined in clock_name_t

Returns

Clock frequency value in hertz

6.5.43 uint32_t CLOCK_GetCoreM4Freq (void)

Returns

Clock frequency; If the clock is invalid, returns 0.

6.5.44 uint32_t CLOCK_GetAxiFreq (void)

Returns

Clock frequency; If the clock is invalid, returns 0.

6.5.45 uint32_t CLOCK_GetAhbFreq (void)

Returns

Clock frequency; If the clock is invalid, returns 0.

Chapter 7
IOMUXC: IOMUX Controller

Overview

IOMUXC driver provides APIs for pin configuration. It also supports the miscellaneous functions integrated in IOMUXC.

Files

- file fsl_iomuxc.h

Driver version

- #define FSL_IOMUXC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  IOMUXC driver version 2.0.1.

Pin function ID

The pin function ID is a tuple of <muxRegister muxMode inputRegister inputDaisy configRegister>

- #define **IOMUXC_PMIC_STBY_REQ** 0x30330014, 0x0, 0x00000000, 0x0, 0x3033027C
- #define **IOMUXC_PMIC_ON_REQ** 0x30330018, 0x0, 0x00000000, 0x0, 0x30330280
- #define **IOMUXC_ONOFF** 0x3033001C, 0x0, 0x00000000, 0x0, 0x30330284
- #define **IOMUXC_POR_B** 0x30330020, 0x0, 0x00000000, 0x0, 0x30330288
- #define **IOMUXC_RTC_RESET_B** 0x30330024, 0x0, 0x00000000, 0x0, 0x3033028C
- #define **IOMUXC_GPIO1_IO00_GPIO1_IO00** 0x30330028, 0x0, 0x00000000, 0x0, 0x30330290
- #define **IOMUXC_GPIO1_IO00_CCM_ENET_PHY_REF_CLK_ROOT** 0x30330028, 0x1, 0x00000000, 0x0, 0x30330290
- #define **IOMUXC_GPIO1_IO00_XTALOSC_REF_CLK_32K** 0x30330028, 0x5, 0x00000000, 0x0, 0x30330290
- #define **IOMUXC_GPIO1_IO00_CCM_EXT_CLK1** 0x30330028, 0x6, 0x00000000, 0x0, 0x30330290
- #define **IOMUXC_GPIO1_IO01_GPIO1_IO01** 0x3033002C, 0x0, 0x00000000, 0x0, 0x30330294
- #define **IOMUXC_GPIO1_IO01_PWM1_OUT** 0x3033002C, 0x1, 0x00000000, 0x0, 0x30330294
- #define **IOMUXC_GPIO1_IO01_XTALOSC_REF_CLK_24M** 0x3033002C, 0x5, 0x00000000, 0x0, 0x30330294
- #define **IOMUXC_GPIO1_IO01_CCM_EXT_CLK2** 0x3033002C, 0x6, 0x00000000, 0x0, 0x30330294
- #define **IOMUXC_GPIO1_IO02_GPIO1_IO02** 0x30330030, 0x0, 0x00000000, 0x0, 0x30330298
- #define **IOMUXC_GPIO1_IO02_WDOG1_WDOG_B** 0x30330030, 0x1, 0x00000000, 0x0, 0x30330298
- #define **IOMUXC_GPIO1_IO02_WDOG1_WDOG_ANY** 0x30330030, 0x5, 0x00000000, 0x0, 0x30330298
- #define **IOMUXC_GPIO1_IO03_GPIO1_IO03** 0x30330034, 0x0, 0x00000000, 0x0, 0x3033029-C
- #define **IOMUXC_GPIO1_IO03_USDHC1_VSELECT** 0x30330034, 0x1, 0x00000000, 0x0, 0x3033029C

- #define **IOMUXC_GPIO1_IO03_SDMA1_EXT_EVENT0** 0x30330034, 0x5, 0x00000000, 0x0, 0x3033029C
- #define **IOMUXC_GPIO1_IO04_GPIO1_IO04** 0x30330038, 0x0, 0x00000000, 0x0, 0x303302-A0
- #define **IOMUXC_GPIO1_IO04_USDHC2_VSELECT** 0x30330038, 0x1, 0x00000000, 0x0, 0x303302A0
- #define **IOMUXC_GPIO1_IO04_SDMA1_EXT_EVENT1** 0x30330038, 0x5, 0x00000000, 0x0, 0x303302A0
- #define **IOMUXC_GPIO1_IO05_GPIO1_IO05** 0x3033003C, 0x0, 0x00000000, 0x0, 0x303302-A4
- #define **IOMUXC_GPIO1_IO05_M4_NMI** 0x3033003C, 0x1, 0x00000000, 0x0, 0x303302A4
- #define **IOMUXC_GPIO1_IO05_CCM_PMIC_READY** 0x3033003C, 0x5, 0x303304BC, 0x0, 0x303302A4
- #define **IOMUXC_GPIO1_IO06_GPIO1_IO06** 0x30330040, 0x0, 0x00000000, 0x0, 0x303302-A8
- #define **IOMUXC_GPIO1_IO06_ENET1_MDC** 0x30330040, 0x1, 0x00000000, 0x0, 0x303302-A8
- #define **IOMUXC_GPIO1_IO06_USDHC1_CD_B** 0x30330040, 0x5, 0x00000000, 0x0, 0x303302A8
- #define **IOMUXC_GPIO1_IO06_CCM_EXT_CLK3** 0x30330040, 0x6, 0x00000000, 0x0, 0x303302A8
- #define **IOMUXC_GPIO1_IO07_GPIO1_IO07** 0x30330044, 0x0, 0x00000000, 0x0, 0x303302-AC
- #define **IOMUXC_GPIO1_IO07_ENET1_MDIO** 0x30330044, 0x1, 0x303304C0, 0x0, 0x303302AC
- #define **IOMUXC_GPIO1_IO07_USDHC1_WP** 0x30330044, 0x5, 0x00000000, 0x0, 0x303302-AC
- #define **IOMUXC_GPIO1_IO07_CCM_EXT_CLK4** 0x30330044, 0x6, 0x00000000, 0x0, 0x303302AC
- #define **IOMUXC_GPIO1_IO08_GPIO1_IO08** 0x30330048, 0x0, 0x00000000, 0x0, 0x303302-B0
- #define **IOMUXC_GPIO1_IO08_ENET1_1588_EVENT0_IN** 0x30330048, 0x1, 0x00000000, 0x0, 0x303302B0
- #define **IOMUXC_GPIO1_IO08_USDHC2_RESET_B** 0x30330048, 0x5, 0x00000000, 0x0, 0x303302B0
- #define **IOMUXC_GPIO1_IO09_GPIO1_IO09** 0x3033004C, 0x0, 0x00000000, 0x0, 0x303302-B4
- #define **IOMUXC_GPIO1_IO09_ENET1_1588_EVENT0_OUT** 0x3033004C, 0x1, 0x00000000, 0x0, 0x303302B4
- #define **IOMUXC_GPIO1_IO09_SDMA2_EXT_EVENT0** 0x3033004C, 0x5, 0x00000000, 0x0, 0x303302B4
- #define **IOMUXC_GPIO1_IO10_GPIO1_IO10** 0x30330050, 0x0, 0x00000000, 0x0, 0x303302-B8
- #define **IOMUXC_GPIO1_IO10_USB1_OTG_ID** 0x30330050, 0x1, 0x00000000, 0x0, 0x303302B8
- #define **IOMUXC_GPIO1_IO11_GPIO1_IO11** 0x30330054, 0x0, 0x00000000, 0x0, 0x303302-BC
- #define **IOMUXC_GPIO1_IO11_USB2_OTG_ID** 0x30330054, 0x1, 0x00000000, 0x0, 0x303302BC
- #define **IOMUXC_GPIO1_IO11_CCM_PMIC_READY** 0x30330054, 0x5, 0x303304BC, 0x1,

0x303302BC
- #define **IOMUXC_GPIO1_IO12_GPIO1_IO12** 0x30330058, 0x0, 0x00000000, 0x0, 0x303302C0
- #define **IOMUXC_GPIO1_IO12_USB1_OTG_PWR** 0x30330058, 0x1, 0x00000000, 0x0, 0x303302C0
- #define **IOMUXC_GPIO1_IO12_SDMA2_EXT_EVENT1** 0x30330058, 0x5, 0x00000000, 0x0, 0x303302C0
- #define **IOMUXC_GPIO1_IO13_GPIO1_IO13** 0x3033005C, 0x0, 0x00000000, 0x0, 0x303302C4
- #define **IOMUXC_GPIO1_IO13_USB1_OTG_OC** 0x3033005C, 0x1, 0x00000000, 0x0, 0x303302C4
- #define **IOMUXC_GPIO1_IO13_PWM2_OUT** 0x3033005C, 0x5, 0x00000000, 0x0, 0x303302C4
- #define **IOMUXC_GPIO1_IO14_GPIO1_IO14** 0x30330060, 0x0, 0x00000000, 0x0, 0x303302C8
- #define **IOMUXC_GPIO1_IO14_USB2_OTG_PWR** 0x30330060, 0x1, 0x00000000, 0x0, 0x303302C8
- #define **IOMUXC_GPIO1_IO14_PWM3_OUT** 0x30330060, 0x5, 0x00000000, 0x0, 0x303302C8
- #define **IOMUXC_GPIO1_IO14_CCM_CLKO1** 0x30330060, 0x6, 0x00000000, 0x0, 0x303302C8
- #define **IOMUXC_GPIO1_IO15_GPIO1_IO15** 0x30330064, 0x0, 0x00000000, 0x0, 0x303302CC
- #define **IOMUXC_GPIO1_IO15_USB2_OTG_OC** 0x30330064, 0x1, 0x00000000, 0x0, 0x303302CC
- #define **IOMUXC_GPIO1_IO15_PWM4_OUT** 0x30330064, 0x5, 0x00000000, 0x0, 0x303302CC
- #define **IOMUXC_GPIO1_IO15_CCM_CLKO2** 0x30330064, 0x6, 0x00000000, 0x0, 0x303302CC
- #define **IOMUXC_ENET_MDC_ENET1_MDC** 0x30330068, 0x0, 0x00000000, 0x0, 0x303302D0
- #define **IOMUXC_ENET_MDC_GPIO1_IO16** 0x30330068, 0x5, 0x00000000, 0x0, 0x303302D0
- #define **IOMUXC_ENET_MDIO_ENET1_MDIO** 0x3033006C, 0x0, 0x303304C0, 0x1, 0x303302D4
- #define **IOMUXC_ENET_MDIO_GPIO1_IO17** 0x3033006C, 0x5, 0x00000000, 0x0, 0x303302D4
- #define **IOMUXC_ENET_TD3_ENET1_RGMII_TD3** 0x30330070, 0x0, 0x00000000, 0x0, 0x303302D8
- #define **IOMUXC_ENET_TD3_GPIO1_IO18** 0x30330070, 0x5, 0x00000000, 0x0, 0x303302D8
- #define **IOMUXC_ENET_TD2_ENET1_RGMII_TD2** 0x30330074, 0x0, 0x00000000, 0x0, 0x303302DC
- #define **IOMUXC_ENET_TD2_ENET1_TX_CLK** 0x30330074, 0x1, 0x00000000, 0x0, 0x303302DC
- #define **IOMUXC_ENET_TD2_GPIO1_IO19** 0x30330074, 0x5, 0x00000000, 0x0, 0x303302DC
- #define **IOMUXC_ENET_TD1_ENET1_RGMII_TD1** 0x30330078, 0x0, 0x00000000, 0x0, 0x303302E0
- #define **IOMUXC_ENET_TD1_GPIO1_IO20** 0x30330078, 0x5, 0x00000000, 0x0, 0x303302E0
- #define **IOMUXC_ENET_TD0_ENET1_RGMII_TD0** 0x3033007C, 0x0, 0x00000000, 0x0, 0x303302E4

- #define **IOMUXC_ENET_TD0_GPIO1_IO21** 0x3033007C, 0x5, 0x00000000, 0x0, 0x303302E4
- #define **IOMUXC_ENET_TX_CTL_ENET1_RGMII_TX_CTL** 0x30330080, 0x0, 0x00000000, 0x0, 0x303302E8
- #define **IOMUXC_ENET_TX_CTL_GPIO1_IO22** 0x30330080, 0x5, 0x00000000, 0x0, 0x303302E8
- #define **IOMUXC_ENET_TXC_ENET1_RGMII_TXC** 0x30330084, 0x0, 0x00000000, 0x0, 0x303302EC
- #define **IOMUXC_ENET_TXC_ENET1_TX_ER** 0x30330084, 0x1, 0x00000000, 0x0, 0x303302EC
- #define **IOMUXC_ENET_TXC_GPIO1_IO23** 0x30330084, 0x5, 0x00000000, 0x0, 0x303302E-C
- #define **IOMUXC_ENET_RX_CTL_ENET1_RGMII_RX_CTL** 0x30330088, 0x0, 0x00000000, 0x0, 0x303302F0
- #define **IOMUXC_ENET_RX_CTL_GPIO1_IO24** 0x30330088, 0x5, 0x00000000, 0x0, 0x303302F0
- #define **IOMUXC_ENET_RXC_ENET1_RGMII_RXC** 0x3033008C, 0x0, 0x00000000, 0x0, 0x303302F4
- #define **IOMUXC_ENET_RXC_ENET1_RX_ER** 0x3033008C, 0x1, 0x00000000, 0x0, 0x303302F4
- #define **IOMUXC_ENET_RXC_GPIO1_IO25** 0x3033008C, 0x5, 0x00000000, 0x0, 0x303302-F4
- #define **IOMUXC_ENET_RD0_ENET1_RGMII_RD0** 0x30330090, 0x0, 0x00000000, 0x0, 0x303302F8
- #define **IOMUXC_ENET_RD0_GPIO1_IO26** 0x30330090, 0x5, 0x00000000, 0x0, 0x303302F8
- #define **IOMUXC_ENET_RD1_ENET1_RGMII_RD1** 0x30330094, 0x0, 0x00000000, 0x0, 0x303302FC
- #define **IOMUXC_ENET_RD1_GPIO1_IO27** 0x30330094, 0x5, 0x00000000, 0x0, 0x303302FC
- #define **IOMUXC_ENET_RD2_ENET1_RGMII_RD2** 0x30330098, 0x0, 0x00000000, 0x0, 0x30330300
- #define **IOMUXC_ENET_RD2_GPIO1_IO28** 0x30330098, 0x5, 0x00000000, 0x0, 0x30330300
- #define **IOMUXC_ENET_RD3_ENET1_RGMII_RD3** 0x3033009C, 0x0, 0x00000000, 0x0, 0x30330304
- #define **IOMUXC_ENET_RD3_GPIO1_IO29** 0x3033009C, 0x5, 0x00000000, 0x0, 0x30330304
- #define **IOMUXC_SD1_CLK_USDHC1_CLK** 0x303300A0, 0x0, 0x00000000, 0x0, 0x30330308
- #define **IOMUXC_SD1_CLK_GPIO2_IO00** 0x303300A0, 0x5, 0x00000000, 0x0, 0x30330308
- #define **IOMUXC_SD1_CMD_USDHC1_CMD** 0x303300A4, 0x0, 0x00000000, 0x0, 0x3033030-C
- #define **IOMUXC_SD1_CMD_GPIO2_IO01** 0x303300A4, 0x5, 0x00000000, 0x0, 0x3033030C
- #define **IOMUXC_SD1_DATA0_USDHC1_DATA0** 0x303300A8, 0x0, 0x00000000, 0x0, 0x30330310
- #define **IOMUXC_SD1_DATA0_GPIO2_IO02** 0x303300A8, 0x5, 0x00000000, 0x0, 0x30330310
- #define **IOMUXC_SD1_DATA1_USDHC1_DATA1** 0x303300AC, 0x0, 0x00000000, 0x0, 0x30330314
- #define **IOMUXC_SD1_DATA1_GPIO2_IO03** 0x303300AC, 0x5, 0x00000000, 0x0, 0x30330314
- #define **IOMUXC_SD1_DATA2_USDHC1_DATA2** 0x303300B0, 0x0, 0x00000000, 0x0, 0x30330318
- #define **IOMUXC_SD1_DATA2_GPIO2_IO04** 0x303300B0, 0x5, 0x00000000, 0x0, 0x30330318
- #define **IOMUXC_SD1_DATA3_USDHC1_DATA3** 0x303300B4, 0x0, 0x00000000, 0x0, 0x3033031C
- #define **IOMUXC_SD1_DATA3_GPIO2_IO05** 0x303300B4, 0x5, 0x00000000, 0x0, 0x3033031-C

- #define **IOMUXC_SD1_DATA4_USDHC1_DATA4** 0x303300B8, 0x0, 0x00000000, 0x0, 0x30330320
- #define **IOMUXC_SD1_DATA4_GPIO2_IO06** 0x303300B8, 0x5, 0x00000000, 0x0, 0x30330320
- #define **IOMUXC_SD1_DATA5_USDHC1_DATA5** 0x303300BC, 0x0, 0x00000000, 0x0, 0x30330324
- #define **IOMUXC_SD1_DATA5_GPIO2_IO07** 0x303300BC, 0x5, 0x00000000, 0x0, 0x30330324
- #define **IOMUXC_SD1_DATA6_USDHC1_DATA6** 0x303300C0, 0x0, 0x00000000, 0x0, 0x30330328
- #define **IOMUXC_SD1_DATA6_GPIO2_IO08** 0x303300C0, 0x5, 0x00000000, 0x0, 0x30330328
- #define **IOMUXC_SD1_DATA7_USDHC1_DATA7** 0x303300C4, 0x0, 0x00000000, 0x0, 0x3033032C
- #define **IOMUXC_SD1_DATA7_GPIO2_IO09** 0x303300C4, 0x5, 0x00000000, 0x0, 0x3033032-C
- #define **IOMUXC_SD1_RESET_B_USDHC1_RESET_B** 0x303300C8, 0x0, 0x00000000, 0x0, 0x30330330
- #define **IOMUXC_SD1_RESET_B_GPIO2_IO10** 0x303300C8, 0x5, 0x00000000, 0x0, 0x30330330
- #define **IOMUXC_SD1_STROBE_USDHC1_STROBE** 0x303300CC, 0x0, 0x00000000, 0x0, 0x30330334
- #define **IOMUXC_SD1_STROBE_GPIO2_IO11** 0x303300CC, 0x5, 0x00000000, 0x0, 0x30330334
- #define **IOMUXC_SD2_CD_B_USDHC2_CD_B** 0x303300D0, 0x0, 0x00000000, 0x0, 0x30330338
- #define **IOMUXC_SD2_CD_B_GPIO2_IO12** 0x303300D0, 0x5, 0x00000000, 0x0, 0x30330338
- #define **IOMUXC_SD2_CLK_USDHC2_CLK** 0x303300D4, 0x0, 0x00000000, 0x0, 0x3033033-C
- #define **IOMUXC_SD2_CLK_GPIO2_IO13** 0x303300D4, 0x5, 0x00000000, 0x0, 0x3033033C
- #define **IOMUXC_SD2_CMD_USDHC2_CMD** 0x303300D8, 0x0, 0x00000000, 0x0, 0x30330340
- #define **IOMUXC_SD2_CMD_GPIO2_IO14** 0x303300D8, 0x5, 0x00000000, 0x0, 0x30330340
- #define **IOMUXC_SD2_DATA0_USDHC2_DATA0** 0x303300DC, 0x0, 0x00000000, 0x0, 0x30330344
- #define **IOMUXC_SD2_DATA0_GPIO2_IO15** 0x303300DC, 0x5, 0x00000000, 0x0, 0x30330344
- #define **IOMUXC_SD2_DATA1_USDHC2_DATA1** 0x303300E0, 0x0, 0x00000000, 0x0, 0x30330348
- #define **IOMUXC_SD2_DATA1_GPIO2_IO16** 0x303300E0, 0x5, 0x00000000, 0x0, 0x30330348
- #define **IOMUXC_SD2_DATA2_USDHC2_DATA2** 0x303300E4, 0x0, 0x00000000, 0x0, 0x3033034C
- #define **IOMUXC_SD2_DATA2_GPIO2_IO17** 0x303300E4, 0x5, 0x00000000, 0x0, 0x3033034-C
- #define **IOMUXC_SD2_DATA3_USDHC2_DATA3** 0x303300E8, 0x0, 0x00000000, 0x0, 0x30330350
- #define **IOMUXC_SD2_DATA3_GPIO2_IO18** 0x303300E8, 0x5, 0x00000000, 0x0, 0x30330350
- #define **IOMUXC_SD2_RESET_B_USDHC2_RESET_B** 0x303300EC, 0x0, 0x00000000, 0x0, 0x30330354
- #define **IOMUXC_SD2_RESET_B_GPIO2_IO19** 0x303300EC, 0x5, 0x00000000, 0x0, 0x30330354
- #define **IOMUXC_SD2_WP_USDHC2_WP** 0x303300F0, 0x0, 0x00000000, 0x0, 0x30330358
- #define **IOMUXC_SD2_WP_GPIO2_IO20** 0x303300F0, 0x5, 0x00000000, 0x0, 0x30330358
- #define **IOMUXC_NAND_ALE_RAWNAND_ALE** 0x303300F4, 0x0, 0x00000000, 0x0, 0x3033035C
- #define **IOMUXC_NAND_ALE_QSPI_A_SCLK** 0x303300F4, 0x1, 0x00000000, 0x0,

0x3033035C
- #define **IOMUXC_NAND_ALE_GPIO3_IO00** 0x303300F4, 0x5, 0x00000000, 0x0, 0x3033035C
- #define **IOMUXC_NAND_CE0_B_RAWNAND_CE0_B** 0x303300F8, 0x0, 0x00000000, 0x0, 0x30330360
- #define **IOMUXC_NAND_CE0_B_QSPI_A_SS0_B** 0x303300F8, 0x1, 0x00000000, 0x0, 0x30330360
- #define **IOMUXC_NAND_CE0_B_GPIO3_IO01** 0x303300F8, 0x5, 0x00000000, 0x0, 0x30330360
- #define **IOMUXC_NAND_CE1_B_RAWNAND_CE1_B** 0x303300FC, 0x0, 0x00000000, 0x0, 0x30330364
- #define **IOMUXC_NAND_CE1_B_QSPI_A_SS1_B** 0x303300FC, 0x1, 0x00000000, 0x0, 0x30330364
- #define **IOMUXC_NAND_CE1_B_GPIO3_IO02** 0x303300FC, 0x5, 0x00000000, 0x0, 0x30330364
- #define **IOMUXC_NAND_CE2_B_RAWNAND_CE2_B** 0x30330100, 0x0, 0x00000000, 0x0, 0x30330368
- #define **IOMUXC_NAND_CE2_B_QSPI_B_SS0_B** 0x30330100, 0x1, 0x00000000, 0x0, 0x30330368
- #define **IOMUXC_NAND_CE2_B_GPIO3_IO03** 0x30330100, 0x5, 0x00000000, 0x0, 0x30330368
- #define **IOMUXC_NAND_CE3_B_RAWNAND_CE3_B** 0x30330104, 0x0, 0x00000000, 0x0, 0x3033036C
- #define **IOMUXC_NAND_CE3_B_QSPI_B_SS1_B** 0x30330104, 0x1, 0x00000000, 0x0, 0x3033036C
- #define **IOMUXC_NAND_CE3_B_GPIO3_IO04** 0x30330104, 0x5, 0x00000000, 0x0, 0x3033036C
- #define **IOMUXC_NAND_CLE_RAWNAND_CLE** 0x30330108, 0x0, 0x00000000, 0x0, 0x30330370
- #define **IOMUXC_NAND_CLE_QSPI_B_SCLK** 0x30330108, 0x1, 0x00000000, 0x0, 0x30330370
- #define **IOMUXC_NAND_CLE_GPIO3_IO05** 0x30330108, 0x5, 0x00000000, 0x0, 0x30330370
- #define **IOMUXC_NAND_DATA00_RAWNAND_DATA00** 0x3033010C, 0x0, 0x00000000, 0x0, 0x30330374
- #define **IOMUXC_NAND_DATA00_QSPI_A_DATA0** 0x3033010C, 0x1, 0x00000000, 0x0, 0x30330374
- #define **IOMUXC_NAND_DATA00_GPIO3_IO06** 0x3033010C, 0x5, 0x00000000, 0x0, 0x30330374
- #define **IOMUXC_NAND_DATA01_RAWNAND_DATA01** 0x30330110, 0x0, 0x00000000, 0x0, 0x30330378
- #define **IOMUXC_NAND_DATA01_QSPI_A_DATA1** 0x30330110, 0x1, 0x00000000, 0x0, 0x30330378
- #define **IOMUXC_NAND_DATA01_GPIO3_IO07** 0x30330110, 0x5, 0x00000000, 0x0, 0x30330378
- #define **IOMUXC_NAND_DATA02_RAWNAND_DATA02** 0x30330114, 0x0, 0x00000000, 0x0, 0x3033037C
- #define **IOMUXC_NAND_DATA02_QSPI_A_DATA2** 0x30330114, 0x1, 0x00000000, 0x0, 0x3033037C
- #define **IOMUXC_NAND_DATA02_GPIO3_IO08** 0x30330114, 0x5, 0x00000000, 0x0, 0x3033037C

- #define **IOMUXC_NAND_DATA03_RAWNAND_DATA03** 0x30330118, 0x0, 0x00000000, 0x0, 0x30330380
- #define **IOMUXC_NAND_DATA03_QSPI_A_DATA3** 0x30330118, 0x1, 0x00000000, 0x0, 0x30330380
- #define **IOMUXC_NAND_DATA03_GPIO3_IO09** 0x30330118, 0x5, 0x00000000, 0x0, 0x30330380
- #define **IOMUXC_NAND_DATA04_RAWNAND_DATA04** 0x3033011C, 0x0, 0x00000000, 0x0, 0x30330384
- #define **IOMUXC_NAND_DATA04_QSPI_B_DATA0** 0x3033011C, 0x1, 0x00000000, 0x0, 0x30330384
- #define **IOMUXC_NAND_DATA04_GPIO3_IO10** 0x3033011C, 0x5, 0x00000000, 0x0, 0x30330384
- #define **IOMUXC_NAND_DATA05_RAWNAND_DATA05** 0x30330120, 0x0, 0x00000000, 0x0, 0x30330388
- #define **IOMUXC_NAND_DATA05_QSPI_B_DATA1** 0x30330120, 0x1, 0x00000000, 0x0, 0x30330388
- #define **IOMUXC_NAND_DATA05_GPIO3_IO11** 0x30330120, 0x5, 0x00000000, 0x0, 0x30330388
- #define **IOMUXC_NAND_DATA06_RAWNAND_DATA06** 0x30330124, 0x0, 0x00000000, 0x0, 0x3033038C
- #define **IOMUXC_NAND_DATA06_QSPI_B_DATA2** 0x30330124, 0x1, 0x00000000, 0x0, 0x3033038C
- #define **IOMUXC_NAND_DATA06_GPIO3_IO12** 0x30330124, 0x5, 0x00000000, 0x0, 0x3033038C
- #define **IOMUXC_NAND_DATA07_RAWNAND_DATA07** 0x30330128, 0x0, 0x00000000, 0x0, 0x30330390
- #define **IOMUXC_NAND_DATA07_QSPI_B_DATA3** 0x30330128, 0x1, 0x00000000, 0x0, 0x30330390
- #define **IOMUXC_NAND_DATA07_GPIO3_IO13** 0x30330128, 0x5, 0x00000000, 0x0, 0x30330390
- #define **IOMUXC_NAND_DQS_RAWNAND_DQS** 0x3033012C, 0x0, 0x00000000, 0x0, 0x30330394
- #define **IOMUXC_NAND_DQS_QSPI_A_DQS** 0x3033012C, 0x1, 0x00000000, 0x0, 0x30330394
- #define **IOMUXC_NAND_DQS_GPIO3_IO14** 0x3033012C, 0x5, 0x00000000, 0x0, 0x30330394
- #define **IOMUXC_NAND_RE_B_RAWNAND_RE_B** 0x30330130, 0x0, 0x00000000, 0x0, 0x30330398
- #define **IOMUXC_NAND_RE_B_QSPI_B_DQS** 0x30330130, 0x1, 0x00000000, 0x0, 0x30330398
- #define **IOMUXC_NAND_RE_B_GPIO3_IO15** 0x30330130, 0x5, 0x00000000, 0x0, 0x30330398
- #define **IOMUXC_NAND_READY_B_RAWNAND_READY_B** 0x30330134, 0x0, 0x00000000, 0x0, 0x3033039C
- #define **IOMUXC_NAND_READY_B_GPIO3_IO16** 0x30330134, 0x5, 0x00000000, 0x0, 0x3033039C
- #define **IOMUXC_NAND_WE_B_RAWNAND_WE_B** 0x30330138, 0x0, 0x00000000, 0x0, 0x303303A0
- #define **IOMUXC_NAND_WE_B_GPIO3_IO17** 0x30330138, 0x5, 0x00000000, 0x0, 0x303303-A0
- #define **IOMUXC_NAND_WP_B_RAWNAND_WP_B** 0x3033013C, 0x0, 0x00000000, 0x0, 0x303303A4
- #define **IOMUXC_NAND_WP_B_GPIO3_IO18** 0x3033013C, 0x5, 0x00000000, 0x0, 0x303303-A4

- #define **IOMUXC_SAI5_RXFS_SAI5_RX_SYNC** 0x30330140, 0x0, 0x303304E4, 0x0, 0x303303A8
- #define **IOMUXC_SAI5_RXFS_SAI1_TX_DATA0** 0x30330140, 0x1, 0x00000000, 0x0, 0x303303A8
- #define **IOMUXC_SAI5_RXFS_GPIO3_IO19** 0x30330140, 0x5, 0x00000000, 0x0, 0x303303A8
- #define **IOMUXC_SAI5_RXC_SAI5_RX_BCLK** 0x30330144, 0x0, 0x303304D0, 0x0, 0x303303AC
- #define **IOMUXC_SAI5_RXC_SAI1_TX_DATA1** 0x30330144, 0x1, 0x00000000, 0x0, 0x303303AC
- #define **IOMUXC_SAI5_RXC_GPIO3_IO20** 0x30330144, 0x5, 0x00000000, 0x0, 0x303303AC
- #define **IOMUXC_SAI5_RXD0_SAI5_RX_DATA0** 0x30330148, 0x0, 0x303304D4, 0x0, 0x303303B0
- #define **IOMUXC_SAI5_RXD0_SAI1_TX_DATA2** 0x30330148, 0x1, 0x00000000, 0x0, 0x303303B0
- #define **IOMUXC_SAI5_RXD0_GPIO3_IO21** 0x30330148, 0x5, 0x00000000, 0x0, 0x303303B0
- #define **IOMUXC_SAI5_RXD1_SAI5_RX_DATA1** 0x3033014C, 0x0, 0x303304D8, 0x0, 0x303303B4
- #define **IOMUXC_SAI5_RXD1_SAI1_TX_DATA3** 0x3033014C, 0x1, 0x00000000, 0x0, 0x303303B4
- #define **IOMUXC_SAI5_RXD1_SAI1_TX_SYNC** 0x3033014C, 0x2, 0x303304CC, 0x0, 0x303303B4
- #define **IOMUXC_SAI5_RXD1_SAI5_TX_SYNC** 0x3033014C, 0x3, 0x303304EC, 0x0, 0x303303B4
- #define **IOMUXC_SAI5_RXD1_GPIO3_IO22** 0x3033014C, 0x5, 0x00000000, 0x0, 0x303303-B4
- #define **IOMUXC_SAI5_RXD2_SAI5_RX_DATA2** 0x30330150, 0x0, 0x303304DC, 0x0, 0x303303B8
- #define **IOMUXC_SAI5_RXD2_SAI1_TX_DATA4** 0x30330150, 0x1, 0x00000000, 0x0, 0x303303B8
- #define **IOMUXC_SAI5_RXD2_SAI1_TX_SYNC** 0x30330150, 0x2, 0x303304CC, 0x1, 0x303303B8
- #define **IOMUXC_SAI5_RXD2_SAI5_TX_BCLK** 0x30330150, 0x3, 0x303304E8, 0x0, 0x303303B8
- #define **IOMUXC_SAI5_RXD2_GPIO3_IO23** 0x30330150, 0x5, 0x00000000, 0x0, 0x303303B8
- #define **IOMUXC_SAI5_RXD3_SAI5_RX_DATA3** 0x30330154, 0x0, 0x303304E0, 0x0, 0x303303BC
- #define **IOMUXC_SAI5_RXD3_SAI1_TX_DATA5** 0x30330154, 0x1, 0x00000000, 0x0, 0x303303BC
- #define **IOMUXC_SAI5_RXD3_SAI1_TX_SYNC** 0x30330154, 0x2, 0x303304CC, 0x2, 0x303303BC
- #define **IOMUXC_SAI5_RXD3_SAI5_TX_DATA0** 0x30330154, 0x3, 0x00000000, 0x0, 0x303303BC
- #define **IOMUXC_SAI5_RXD3_GPIO3_IO24** 0x30330154, 0x5, 0x00000000, 0x0, 0x303303B-C
- #define **IOMUXC_SAI5_MCLK_SAI5_MCLK** 0x30330158, 0x0, 0x3033052C, 0x0, 0x303303-C0
- #define **IOMUXC_SAI5_MCLK_SAI1_TX_BCLK** 0x30330158, 0x1, 0x303304C8, 0x0, 0x303303C0
- #define **IOMUXC_SAI5_MCLK_SAI4_MCLK** 0x30330158, 0x2, 0x00000000, 0x0, 0x303303-C0

- #define **IOMUXC_SAI5_MCLK_GPIO3_IO25** 0x30330158, 0x5, 0x00000000, 0x0, 0x303303C0
- #define **IOMUXC_SAI1_RXFS_SAI1_RX_SYNC** 0x3033015C, 0x0, 0x303304C4, 0x0, 0x303303C4
- #define **IOMUXC_SAI1_RXFS_SAI5_RX_SYNC** 0x3033015C, 0x1, 0x303304E4, 0x1, 0x303303C4
- #define **IOMUXC_SAI1_RXFS_CORESIGHT_TRACE_CLK** 0x3033015C, 0x4, 0x00000000, 0x0, 0x303303C4
- #define **IOMUXC_SAI1_RXFS_GPIO4_IO00** 0x3033015C, 0x5, 0x00000000, 0x0, 0x303303C4
- #define **IOMUXC_SAI1_RXC_SAI1_RX_BCLK** 0x30330160, 0x0, 0x00000000, 0x0, 0x303303C8
- #define **IOMUXC_SAI1_RXC_SAI5_RX_BCLK** 0x30330160, 0x1, 0x303304D0, 0x1, 0x303303C8
- #define **IOMUXC_SAI1_RXC_CORESIGHT_TRACE_CTL** 0x30330160, 0x4, 0x00000000, 0x0, 0x303303C8
- #define **IOMUXC_SAI1_RXC_GPIO4_IO01** 0x30330160, 0x5, 0x00000000, 0x0, 0x303303C8
- #define **IOMUXC_SAI1_RXD0_SAI1_RX_DATA0** 0x30330164, 0x0, 0x00000000, 0x0, 0x303303CC
- #define **IOMUXC_SAI1_RXD0_SAI5_RX_DATA0** 0x30330164, 0x1, 0x303304D4, 0x1, 0x303303CC
- #define **IOMUXC_SAI1_RXD0_CORESIGHT_TRACE0** 0x30330164, 0x4, 0x00000000, 0x0, 0x303303CC
- #define **IOMUXC_SAI1_RXD0_GPIO4_IO02** 0x30330164, 0x5, 0x00000000, 0x0, 0x303303CC
- #define **IOMUXC_SAI1_RXD0_SRC_BOOT_CFG0** 0x30330164, 0x6, 0x00000000, 0x0, 0x303303CC
- #define **IOMUXC_SAI1_RXD1_SAI1_RX_DATA1** 0x30330168, 0x0, 0x00000000, 0x0, 0x303303D0
- #define **IOMUXC_SAI1_RXD1_SAI5_RX_DATA1** 0x30330168, 0x1, 0x303304D8, 0x1, 0x303303D0
- #define **IOMUXC_SAI1_RXD1_CORESIGHT_TRACE1** 0x30330168, 0x4, 0x00000000, 0x0, 0x303303D0
- #define **IOMUXC_SAI1_RXD1_GPIO4_IO03** 0x30330168, 0x5, 0x00000000, 0x0, 0x303303D0
- #define **IOMUXC_SAI1_RXD1_SRC_BOOT_CFG1** 0x30330168, 0x6, 0x00000000, 0x0, 0x303303D0
- #define **IOMUXC_SAI1_RXD2_SAI1_RX_DATA2** 0x3033016C, 0x0, 0x00000000, 0x0, 0x303303D4
- #define **IOMUXC_SAI1_RXD2_SAI5_RX_DATA2** 0x3033016C, 0x1, 0x303304DC, 0x1, 0x303303D4
- #define **IOMUXC_SAI1_RXD2_CORESIGHT_TRACE2** 0x3033016C, 0x4, 0x00000000, 0x0, 0x303303D4
- #define **IOMUXC_SAI1_RXD2_GPIO4_IO04** 0x3033016C, 0x5, 0x00000000, 0x0, 0x303303D4
- #define **IOMUXC_SAI1_RXD2_SRC_BOOT_CFG2** 0x3033016C, 0x6, 0x00000000, 0x0, 0x303303D4
- #define **IOMUXC_SAI1_RXD3_SAI1_RX_DATA3** 0x30330170, 0x0, 0x00000000, 0x0, 0x303303D8
- #define **IOMUXC_SAI1_RXD3_SAI5_RX_DATA3** 0x30330170, 0x1, 0x303304E0, 0x1, 0x303303D8

- #define **IOMUXC_SAI1_RXD3_CORESIGHT_TRACE3** 0x30330170, 0x4, 0x00000000, 0x0, 0x303303D8
- #define **IOMUXC_SAI1_RXD3_GPIO4_IO05** 0x30330170, 0x5, 0x00000000, 0x0, 0x303303D8
- #define **IOMUXC_SAI1_RXD3_SRC_BOOT_CFG3** 0x30330170, 0x6, 0x00000000, 0x0, 0x303303D8
- #define **IOMUXC_SAI1_RXD4_SAI1_RX_DATA4** 0x30330174, 0x0, 0x00000000, 0x0, 0x303303DC
- #define **IOMUXC_SAI1_RXD4_SAI6_TX_BCLK** 0x30330174, 0x1, 0x3033051C, 0x0, 0x303303DC
- #define **IOMUXC_SAI1_RXD4_SAI6_RX_BCLK** 0x30330174, 0x2, 0x30330510, 0x0, 0x303303DC
- #define **IOMUXC_SAI1_RXD4_CORESIGHT_TRACE4** 0x30330174, 0x4, 0x00000000, 0x0, 0x303303DC
- #define **IOMUXC_SAI1_RXD4_GPIO4_IO06** 0x30330174, 0x5, 0x00000000, 0x0, 0x303303D-C
- #define **IOMUXC_SAI1_RXD4_SRC_BOOT_CFG4** 0x30330174, 0x6, 0x00000000, 0x0, 0x303303DC
- #define **IOMUXC_SAI1_RXD5_SAI1_RX_DATA5** 0x30330178, 0x0, 0x00000000, 0x0, 0x303303E0
- #define **IOMUXC_SAI1_RXD5_SAI6_TX_DATA0** 0x30330178, 0x1, 0x00000000, 0x0, 0x303303E0
- #define **IOMUXC_SAI1_RXD5_SAI6_RX_DATA0** 0x30330178, 0x2, 0x30330514, 0x0, 0x303303E0
- #define **IOMUXC_SAI1_RXD5_SAI1_RX_SYNC** 0x30330178, 0x3, 0x303304C4, 0x1, 0x303303E0
- #define **IOMUXC_SAI1_RXD5_CORESIGHT_TRACE5** 0x30330178, 0x4, 0x00000000, 0x0, 0x303303E0
- #define **IOMUXC_SAI1_RXD5_GPIO4_IO07** 0x30330178, 0x5, 0x00000000, 0x0, 0x303303E0
- #define **IOMUXC_SAI1_RXD5_SRC_BOOT_CFG5** 0x30330178, 0x6, 0x00000000, 0x0, 0x303303E0
- #define **IOMUXC_SAI1_RXD6_SAI1_RX_DATA6** 0x3033017C, 0x0, 0x00000000, 0x0, 0x303303E4
- #define **IOMUXC_SAI1_RXD6_SAI6_TX_SYNC** 0x3033017C, 0x1, 0x30330520, 0x0, 0x303303E4
- #define **IOMUXC_SAI1_RXD6_SAI6_RX_SYNC** 0x3033017C, 0x2, 0x30330518, 0x0, 0x303303E4
- #define **IOMUXC_SAI1_RXD6_CORESIGHT_TRACE6** 0x3033017C, 0x4, 0x00000000, 0x0, 0x303303E4
- #define **IOMUXC_SAI1_RXD6_GPIO4_IO08** 0x3033017C, 0x5, 0x00000000, 0x0, 0x303303-E4
- #define **IOMUXC_SAI1_RXD6_SRC_BOOT_CFG6** 0x3033017C, 0x6, 0x00000000, 0x0, 0x303303E4
- #define **IOMUXC_SAI1_RXD7_SAI1_RX_DATA7** 0x30330180, 0x0, 0x00000000, 0x0, 0x303303E8
- #define **IOMUXC_SAI1_RXD7_SAI6_MCLK** 0x30330180, 0x1, 0x30330530, 0x0, 0x303303E8
- #define **IOMUXC_SAI1_RXD7_SAI1_TX_SYNC** 0x30330180, 0x2, 0x303304CC, 0x4, 0x303303E8
- #define **IOMUXC_SAI1_RXD7_SAI1_TX_DATA4** 0x30330180, 0x3, 0x00000000, 0x0, 0x303303E8
- #define **IOMUXC_SAI1_RXD7_CORESIGHT_TRACE7** 0x30330180, 0x4, 0x00000000, 0x0,

0x303303E8
- #define **IOMUXC_SAI1_RXD7_GPIO4_IO09** 0x30330180, 0x5, 0x00000000, 0x0, 0x303303E8
- #define **IOMUXC_SAI1_RXD7_SRC_BOOT_CFG7** 0x30330180, 0x6, 0x00000000, 0x0, 0x303303E8
- #define **IOMUXC_SAI1_TXFS_SAI1_TX_SYNC** 0x30330184, 0x0, 0x303304CC, 0x3, 0x303303EC
- #define **IOMUXC_SAI1_TXFS_SAI5_TX_SYNC** 0x30330184, 0x1, 0x303304EC, 0x1, 0x303303EC
- #define **IOMUXC_SAI1_TXFS_CORESIGHT_EVENTO** 0x30330184, 0x4, 0x00000000, 0x0, 0x303303EC
- #define **IOMUXC_SAI1_TXFS_GPIO4_IO10** 0x30330184, 0x5, 0x00000000, 0x0, 0x303303EC
- #define **IOMUXC_SAI1_TXC_SAI1_TX_BCLK** 0x30330188, 0x0, 0x303304C8, 0x1, 0x303303F0
- #define **IOMUXC_SAI1_TXC_SAI5_TX_BCLK** 0x30330188, 0x1, 0x303304E8, 0x1, 0x303303F0
- #define **IOMUXC_SAI1_TXC_CORESIGHT_EVENTI** 0x30330188, 0x4, 0x00000000, 0x0, 0x303303F0
- #define **IOMUXC_SAI1_TXC_GPIO4_IO11** 0x30330188, 0x5, 0x00000000, 0x0, 0x303303F0
- #define **IOMUXC_SAI1_TXD0_SAI1_TX_DATA0** 0x3033018C, 0x0, 0x00000000, 0x0, 0x303303F4
- #define **IOMUXC_SAI1_TXD0_SAI5_TX_DATA0** 0x3033018C, 0x1, 0x00000000, 0x0, 0x303303F4
- #define **IOMUXC_SAI1_TXD0_CORESIGHT_TRACE8** 0x3033018C, 0x4, 0x00000000, 0x0, 0x303303F4
- #define **IOMUXC_SAI1_TXD0_GPIO4_IO12** 0x3033018C, 0x5, 0x00000000, 0x0, 0x303303F4
- #define **IOMUXC_SAI1_TXD0_SRC_BOOT_CFG8** 0x3033018C, 0x6, 0x00000000, 0x0, 0x303303F4
- #define **IOMUXC_SAI1_TXD1_SAI1_TX_DATA1** 0x30330190, 0x0, 0x00000000, 0x0, 0x303303F8
- #define **IOMUXC_SAI1_TXD1_SAI5_TX_DATA1** 0x30330190, 0x1, 0x00000000, 0x0, 0x303303F8
- #define **IOMUXC_SAI1_TXD1_CORESIGHT_TRACE9** 0x30330190, 0x4, 0x00000000, 0x0, 0x303303F8
- #define **IOMUXC_SAI1_TXD1_GPIO4_IO13** 0x30330190, 0x5, 0x00000000, 0x0, 0x303303F8
- #define **IOMUXC_SAI1_TXD1_SRC_BOOT_CFG9** 0x30330190, 0x6, 0x00000000, 0x0, 0x303303F8
- #define **IOMUXC_SAI1_TXD2_SAI1_TX_DATA2** 0x30330194, 0x0, 0x00000000, 0x0, 0x303303FC
- #define **IOMUXC_SAI1_TXD2_SAI5_TX_DATA2** 0x30330194, 0x1, 0x00000000, 0x0, 0x303303FC
- #define **IOMUXC_SAI1_TXD2_CORESIGHT_TRACE10** 0x30330194, 0x4, 0x00000000, 0x0, 0x303303FC
- #define **IOMUXC_SAI1_TXD2_GPIO4_IO14** 0x30330194, 0x5, 0x00000000, 0x0, 0x303303FC
- #define **IOMUXC_SAI1_TXD2_SRC_BOOT_CFG10** 0x30330194, 0x6, 0x00000000, 0x0, 0x303303FC
- #define **IOMUXC_SAI1_TXD3_SAI1_TX_DATA3** 0x30330198, 0x0, 0x00000000, 0x0, 0x30330400
- #define **IOMUXC_SAI1_TXD3_SAI5_TX_DATA3** 0x30330198, 0x1, 0x00000000, 0x0, 0x30330400
- #define **IOMUXC_SAI1_TXD3_CORESIGHT_TRACE11** 0x30330198, 0x4, 0x00000000, 0x0,

0x30330400
- #define **IOMUXC_SAI1_TXD3_GPIO4_IO15** 0x30330198, 0x5, 0x00000000, 0x0, 0x30330400
- #define **IOMUXC_SAI1_TXD3_SRC_BOOT_CFG11** 0x30330198, 0x6, 0x00000000, 0x0, 0x30330400
- #define **IOMUXC_SAI1_TXD4_SAI1_TX_DATA4** 0x3033019C, 0x0, 0x00000000, 0x0, 0x30330404
- #define **IOMUXC_SAI1_TXD4_SAI6_RX_BCLK** 0x3033019C, 0x1, 0x30330510, 0x1, 0x30330404
- #define **IOMUXC_SAI1_TXD4_SAI6_TX_BCLK** 0x3033019C, 0x2, 0x3033051C, 0x1, 0x30330404
- #define **IOMUXC_SAI1_TXD4_CORESIGHT_TRACE12** 0x3033019C, 0x4, 0x00000000, 0x0, 0x30330404
- #define **IOMUXC_SAI1_TXD4_GPIO4_IO16** 0x3033019C, 0x5, 0x00000000, 0x0, 0x30330404
- #define **IOMUXC_SAI1_TXD4_SRC_BOOT_CFG12** 0x3033019C, 0x6, 0x00000000, 0x0, 0x30330404
- #define **IOMUXC_SAI1_TXD5_SAI1_TX_DATA5** 0x303301A0, 0x0, 0x00000000, 0x0, 0x30330408
- #define **IOMUXC_SAI1_TXD5_SAI6_RX_DATA0** 0x303301A0, 0x1, 0x30330514, 0x1, 0x30330408
- #define **IOMUXC_SAI1_TXD5_SAI6_TX_DATA0** 0x303301A0, 0x2, 0x00000000, 0x0, 0x30330408
- #define **IOMUXC_SAI1_TXD5_CORESIGHT_TRACE13** 0x303301A0, 0x4, 0x00000000, 0x0, 0x30330408
- #define **IOMUXC_SAI1_TXD5_GPIO4_IO17** 0x303301A0, 0x5, 0x00000000, 0x0, 0x30330408
- #define **IOMUXC_SAI1_TXD5_SRC_BOOT_CFG13** 0x303301A0, 0x6, 0x00000000, 0x0, 0x30330408
- #define **IOMUXC_SAI1_TXD6_SAI1_TX_DATA6** 0x303301A4, 0x0, 0x00000000, 0x0, 0x3033040C
- #define **IOMUXC_SAI1_TXD6_SAI6_RX_SYNC** 0x303301A4, 0x1, 0x30330518, 0x1, 0x3033040C
- #define **IOMUXC_SAI1_TXD6_SAI6_TX_SYNC** 0x303301A4, 0x2, 0x30330520, 0x1, 0x3033040C
- #define **IOMUXC_SAI1_TXD6_CORESIGHT_TRACE14** 0x303301A4, 0x4, 0x00000000, 0x0, 0x3033040C
- #define **IOMUXC_SAI1_TXD6_GPIO4_IO18** 0x303301A4, 0x5, 0x00000000, 0x0, 0x3033040-C
- #define **IOMUXC_SAI1_TXD6_SRC_BOOT_CFG14** 0x303301A4, 0x6, 0x00000000, 0x0, 0x3033040C
- #define **IOMUXC_SAI1_TXD7_SAI1_TX_DATA7** 0x303301A8, 0x0, 0x00000000, 0x0, 0x30330410
- #define **IOMUXC_SAI1_TXD7_SAI6_MCLK** 0x303301A8, 0x1, 0x30330530, 0x1, 0x30330410
- #define **IOMUXC_SAI1_TXD7_CORESIGHT_TRACE15** 0x303301A8, 0x4, 0x00000000, 0x0, 0x30330410
- #define **IOMUXC_SAI1_TXD7_GPIO4_IO19** 0x303301A8, 0x5, 0x00000000, 0x0, 0x30330410
- #define **IOMUXC_SAI1_TXD7_SRC_BOOT_CFG15** 0x303301A8, 0x6, 0x00000000, 0x0, 0x30330410
- #define **IOMUXC_SAI1_MCLK_SAI1_MCLK** 0x303301AC, 0x0, 0x00000000, 0x0, 0x30330414
- #define **IOMUXC_SAI1_MCLK_SAI5_MCLK** 0x303301AC, 0x1, 0x3033052C, 0x1, 0x30330414
- #define **IOMUXC_SAI1_MCLK_SAI1_TX_BCLK** 0x303301AC, 0x2, 0x303304C8, 0x2, 0x30330414
- #define **IOMUXC_SAI1_MCLK_GPIO4_IO20** 0x303301AC, 0x5, 0x00000000, 0x0, 0x30330414

- #define **IOMUXC_SAI2_RXFS_SAI2_RX_SYNC** 0x303301B0, 0x0, 0x00000000, 0x0, 0x30330418
- #define **IOMUXC_SAI2_RXFS_SAI5_TX_SYNC** 0x303301B0, 0x1, 0x303304EC, 0x2, 0x30330418
- #define **IOMUXC_SAI2_RXFS_GPIO4_IO21** 0x303301B0, 0x5, 0x00000000, 0x0, 0x30330418
- #define **IOMUXC_SAI2_RXC_SAI2_RX_BCLK** 0x303301B4, 0x0, 0x00000000, 0x0, 0x3033041C
- #define **IOMUXC_SAI2_RXC_SAI5_TX_BCLK** 0x303301B4, 0x1, 0x303304E8, 0x2, 0x3033041C
- #define **IOMUXC_SAI2_RXC_GPIO4_IO22** 0x303301B4, 0x5, 0x00000000, 0x0, 0x3033041C
- #define **IOMUXC_SAI2_RXD0_SAI2_RX_DATA0** 0x303301B8, 0x0, 0x00000000, 0x0, 0x30330420
- #define **IOMUXC_SAI2_RXD0_SAI5_TX_DATA0** 0x303301B8, 0x1, 0x00000000, 0x0, 0x30330420
- #define **IOMUXC_SAI2_RXD0_GPIO4_IO23** 0x303301B8, 0x5, 0x00000000, 0x0, 0x30330420
- #define **IOMUXC_SAI2_TXFS_SAI2_TX_SYNC** 0x303301BC, 0x0, 0x00000000, 0x0, 0x30330424
- #define **IOMUXC_SAI2_TXFS_SAI5_TX_DATA1** 0x303301BC, 0x1, 0x00000000, 0x0, 0x30330424
- #define **IOMUXC_SAI2_TXFS_GPIO4_IO24** 0x303301BC, 0x5, 0x00000000, 0x0, 0x30330424
- #define **IOMUXC_SAI2_TXC_SAI2_TX_BCLK** 0x303301C0, 0x0, 0x00000000, 0x0, 0x30330428
- #define **IOMUXC_SAI2_TXC_SAI5_TX_DATA2** 0x303301C0, 0x1, 0x00000000, 0x0, 0x30330428
- #define **IOMUXC_SAI2_TXC_GPIO4_IO25** 0x303301C0, 0x5, 0x00000000, 0x0, 0x30330428
- #define **IOMUXC_SAI2_TXD0_SAI2_TX_DATA0** 0x303301C4, 0x0, 0x00000000, 0x0, 0x3033042C
- #define **IOMUXC_SAI2_TXD0_SAI5_TX_DATA3** 0x303301C4, 0x1, 0x00000000, 0x0, 0x3033042C
- #define **IOMUXC_SAI2_TXD0_GPIO4_IO26** 0x303301C4, 0x5, 0x00000000, 0x0, 0x3033042-C
- #define **IOMUXC_SAI2_MCLK_SAI2_MCLK** 0x303301C8, 0x0, 0x00000000, 0x0, 0x30330430
- #define **IOMUXC_SAI2_MCLK_SAI5_MCLK** 0x303301C8, 0x1, 0x3033052C, 0x2, 0x30330430
- #define **IOMUXC_SAI2_MCLK_GPIO4_IO27** 0x303301C8, 0x5, 0x00000000, 0x0, 0x30330430
- #define **IOMUXC_SAI3_RXFS_SAI3_RX_SYNC** 0x303301CC, 0x0, 0x00000000, 0x0, 0x30330434
- #define **IOMUXC_SAI3_RXFS_GPT1_CAPTURE1** 0x303301CC, 0x1, 0x00000000, 0x0, 0x30330434
- #define **IOMUXC_SAI3_RXFS_SAI5_RX_SYNC** 0x303301CC, 0x2, 0x303304E4, 0x2, 0x30330434
- #define **IOMUXC_SAI3_RXFS_GPIO4_IO28** 0x303301CC, 0x5, 0x00000000, 0x0, 0x30330434
- #define **IOMUXC_SAI3_RXC_SAI3_RX_BCLK** 0x303301D0, 0x0, 0x00000000, 0x0, 0x30330438
- #define **IOMUXC_SAI3_RXC_GPT1_CAPTURE2** 0x303301D0, 0x1, 0x00000000, 0x0, 0x30330438
- #define **IOMUXC_SAI3_RXC_SAI5_RX_BCLK** 0x303301D0, 0x2, 0x303304D0, 0x2, 0x30330438
- #define **IOMUXC_SAI3_RXC_GPIO4_IO29** 0x303301D0, 0x5, 0x00000000, 0x0, 0x30330438
- #define **IOMUXC_SAI3_RXD_SAI3_RX_DATA0** 0x303301D4, 0x0, 0x00000000, 0x0, 0x3033043C
- #define **IOMUXC_SAI3_RXD_GPT1_COMPARE1** 0x303301D4, 0x1, 0x00000000, 0x0,

0x3033043C
- #define **IOMUXC_SAI3_RXD_SAI5_RX_DATA0** 0x303301D4, 0x2, 0x303304D4, 0x2, 0x3033043C
- #define **IOMUXC_SAI3_RXD_GPIO4_IO30** 0x303301D4, 0x5, 0x00000000, 0x0, 0x3033043C
- #define **IOMUXC_SAI3_TXFS_SAI3_TX_SYNC** 0x303301D8, 0x0, 0x00000000, 0x0, 0x30330440
- #define **IOMUXC_SAI3_TXFS_GPT1_CLK** 0x303301D8, 0x1, 0x00000000, 0x0, 0x30330440
- #define **IOMUXC_SAI3_TXFS_SAI5_RX_DATA1** 0x303301D8, 0x2, 0x303304D8, 0x2, 0x30330440
- #define **IOMUXC_SAI3_TXFS_GPIO4_IO31** 0x303301D8, 0x5, 0x00000000, 0x0, 0x30330440
- #define **IOMUXC_SAI3_TXC_SAI3_TX_BCLK** 0x303301DC, 0x0, 0x00000000, 0x0, 0x30330444
- #define **IOMUXC_SAI3_TXC_GPT1_COMPARE2** 0x303301DC, 0x1, 0x00000000, 0x0, 0x30330444
- #define **IOMUXC_SAI3_TXC_SAI5_RX_DATA2** 0x303301DC, 0x2, 0x303304DC, 0x2, 0x30330444
- #define **IOMUXC_SAI3_TXC_GPIO5_IO00** 0x303301DC, 0x5, 0x00000000, 0x0, 0x30330444
- #define **IOMUXC_SAI3_TXD_SAI3_TX_DATA0** 0x303301E0, 0x0, 0x00000000, 0x0, 0x30330448
- #define **IOMUXC_SAI3_TXD_GPT1_COMPARE3** 0x303301E0, 0x1, 0x00000000, 0x0, 0x30330448
- #define **IOMUXC_SAI3_TXD_SAI5_RX_DATA3** 0x303301E0, 0x2, 0x303304E0, 0x2, 0x30330448
- #define **IOMUXC_SAI3_TXD_GPIO5_IO01** 0x303301E0, 0x5, 0x00000000, 0x0, 0x30330448
- #define **IOMUXC_SAI3_MCLK_SAI3_MCLK** 0x303301E4, 0x0, 0x00000000, 0x0, 0x3033044C
- #define **IOMUXC_SAI3_MCLK_PWM4_OUT** 0x303301E4, 0x1, 0x00000000, 0x0, 0x3033044C
- #define **IOMUXC_SAI3_MCLK_SAI5_MCLK** 0x303301E4, 0x2, 0x3033052C, 0x3, 0x3033044C
- #define **IOMUXC_SAI3_MCLK_GPIO5_IO02** 0x303301E4, 0x5, 0x00000000, 0x0, 0x3033044C
- #define **IOMUXC_SPDIF_TX_SPDIF1_OUT** 0x303301E8, 0x0, 0x00000000, 0x0, 0x30330450
- #define **IOMUXC_SPDIF_TX_PWM3_OUT** 0x303301E8, 0x1, 0x00000000, 0x0, 0x30330450
- #define **IOMUXC_SPDIF_TX_GPIO5_IO03** 0x303301E8, 0x5, 0x00000000, 0x0, 0x30330450
- #define **IOMUXC_SPDIF_RX_SPDIF1_IN** 0x303301EC, 0x0, 0x00000000, 0x0, 0x30330454
- #define **IOMUXC_SPDIF_RX_PWM2_OUT** 0x303301EC, 0x1, 0x00000000, 0x0, 0x30330454
- #define **IOMUXC_SPDIF_RX_GPIO5_IO04** 0x303301EC, 0x5, 0x00000000, 0x0, 0x30330454
- #define **IOMUXC_SPDIF_EXT_CLK_SPDIF1_EXT_CLK** 0x303301F0, 0x0, 0x00000000, 0x0, 0x30330458
- #define **IOMUXC_SPDIF_EXT_CLK_PWM1_OUT** 0x303301F0, 0x1, 0x00000000, 0x0, 0x30330458
- #define **IOMUXC_SPDIF_EXT_CLK_GPIO5_IO05** 0x303301F0, 0x5, 0x00000000, 0x0, 0x30330458
- #define **IOMUXC_ECSPI1_SCLK_ECSPI1_SCLK** 0x303301F4, 0x0, 0x00000000, 0x0, 0x3033045C
- #define **IOMUXC_ECSPI1_SCLK_UART3_RX** 0x303301F4, 0x1, 0x30330504, 0x0, 0x3033045C
- #define **IOMUXC_ECSPI1_SCLK_UART3_TX** 0x303301F4, 0x1, 0x00000000, 0X0, 0x3033045C
- #define **IOMUXC_ECSPI1_SCLK_GPIO5_IO06** 0x303301F4, 0x5, 0x00000000, 0x0,

0x3033045C
- #define **IOMUXC_ECSPI1_MOSI_ECSPI1_MOSI** 0x303301F8, 0x0, 0x00000000, 0x0, 0x30330460
- #define **IOMUXC_ECSPI1_MOSI_UART3_TX** 0x303301F8, 0x1, 0x00000000, 0X0, 0x30330460
- #define **IOMUXC_ECSPI1_MOSI_UART3_RX** 0x303301F8, 0x1, 0x30330504, 0x1, 0x30330460
- #define **IOMUXC_ECSPI1_MOSI_GPIO5_IO07** 0x303301F8, 0x5, 0x00000000, 0x0, 0x30330460
- #define **IOMUXC_ECSPI1_MISO_ECSPI1_MISO** 0x303301FC, 0x0, 0x00000000, 0x0, 0x30330464
- #define **IOMUXC_ECSPI1_MISO_UART3_CTS_B** 0x303301FC, 0x1, 0x00000000, 0X0, 0x30330464
- #define **IOMUXC_ECSPI1_MISO_UART3_RTS_B** 0x303301FC, 0x1, 0x30330500, 0x0, 0x30330464
- #define **IOMUXC_ECSPI1_MISO_GPIO5_IO08** 0x303301FC, 0x5, 0x00000000, 0x0, 0x30330464
- #define **IOMUXC_ECSPI1_SS0_ECSPI1_SS0** 0x30330200, 0x0, 0x00000000, 0x0, 0x30330468
- #define **IOMUXC_ECSPI1_SS0_UART3_RTS_B** 0x30330200, 0x1, 0x30330500, 0x1, 0x30330468
- #define **IOMUXC_ECSPI1_SS0_UART3_CTS_B** 0x30330200, 0x1, 0x00000000, 0X0, 0x30330468
- #define **IOMUXC_ECSPI1_SS0_GPIO5_IO09** 0x30330200, 0x5, 0x00000000, 0x0, 0x30330468
- #define **IOMUXC_ECSPI2_SCLK_ECSPI2_SCLK** 0x30330204, 0x0, 0x00000000, 0x0, 0x3033046C
- #define **IOMUXC_ECSPI2_SCLK_UART4_RX** 0x30330204, 0x1, 0x3033050C, 0x0, 0x3033046-C
- #define **IOMUXC_ECSPI2_SCLK_UART4_TX** 0x30330204, 0x1, 0x00000000, 0X0, 0x3033046C
- #define **IOMUXC_ECSPI2_SCLK_GPIO5_IO10** 0x30330204, 0x5, 0x00000000, 0x0, 0x3033046C
- #define **IOMUXC_ECSPI2_MOSI_ECSPI2_MOSI** 0x30330208, 0x0, 0x00000000, 0x0, 0x30330470
- #define **IOMUXC_ECSPI2_MOSI_UART4_TX** 0x30330208, 0x1, 0x00000000, 0X0, 0x30330470
- #define **IOMUXC_ECSPI2_MOSI_UART4_RX** 0x30330208, 0x1, 0x3033050C, 0x1, 0x30330470
- #define **IOMUXC_ECSPI2_MOSI_GPIO5_IO11** 0x30330208, 0x5, 0x00000000, 0x0, 0x30330470
- #define **IOMUXC_ECSPI2_MISO_ECSPI2_MISO** 0x3033020C, 0x0, 0x00000000, 0x0, 0x30330474
- #define **IOMUXC_ECSPI2_MISO_UART4_CTS_B** 0x3033020C, 0x1, 0x00000000, 0X0, 0x30330474
- #define **IOMUXC_ECSPI2_MISO_UART4_RTS_B** 0x3033020C, 0x1, 0x30330508, 0x0, 0x30330474
- #define **IOMUXC_ECSPI2_MISO_GPIO5_IO12** 0x3033020C, 0x5, 0x00000000, 0x0, 0x30330474
- #define **IOMUXC_ECSPI2_SS0_ECSPI2_SS0** 0x30330210, 0x0, 0x00000000, 0x0, 0x30330478
- #define **IOMUXC_ECSPI2_SS0_UART4_RTS_B** 0x30330210, 0x1, 0x30330508, 0x1, 0x30330478
- #define **IOMUXC_ECSPI2_SS0_UART4_CTS_B** 0x30330210, 0x1, 0x00000000, 0X0, 0x30330478
- #define **IOMUXC_ECSPI2_SS0_GPIO5_IO13** 0x30330210, 0x5, 0x00000000, 0x0, 0x30330478
- #define **IOMUXC_I2C1_SCL_I2C1_SCL** 0x30330214, 0x0, 0x00000000, 0x0, 0x3033047C

- #define **IOMUXC_I2C1_SCL_ENET1_MDC** 0x30330214, 0x1, 0x00000000, 0x0, 0x3033047C
- #define **IOMUXC_I2C1_SCL_GPIO5_IO14** 0x30330214, 0x5, 0x00000000, 0x0, 0x3033047C
- #define **IOMUXC_I2C1_SDA_I2C1_SDA** 0x30330218, 0x0, 0x00000000, 0x0, 0x30330480
- #define **IOMUXC_I2C1_SDA_ENET1_MDIO** 0x30330218, 0x1, 0x303304C0, 0x2, 0x30330480
- #define **IOMUXC_I2C1_SDA_GPIO5_IO15** 0x30330218, 0x5, 0x00000000, 0x0, 0x30330480
- #define **IOMUXC_I2C2_SCL_I2C2_SCL** 0x3033021C, 0x0, 0x00000000, 0x0, 0x30330484
- #define **IOMUXC_I2C2_SCL_ENET1_1588_EVENT1_IN** 0x3033021C, 0x1, 0x00000000, 0x0, 0x30330484
- #define **IOMUXC_I2C2_SCL_GPIO5_IO16** 0x3033021C, 0x5, 0x00000000, 0x0, 0x30330484
- #define **IOMUXC_I2C2_SDA_I2C2_SDA** 0x30330220, 0x0, 0x00000000, 0x0, 0x30330488
- #define **IOMUXC_I2C2_SDA_ENET1_1588_EVENT1_OUT** 0x30330220, 0x1, 0x00000000, 0x0, 0x30330488
- #define **IOMUXC_I2C2_SDA_GPIO5_IO17** 0x30330220, 0x5, 0x00000000, 0x0, 0x30330488
- #define **IOMUXC_I2C3_SCL_I2C3_SCL** 0x30330224, 0x0, 0x00000000, 0x0, 0x3033048C
- #define **IOMUXC_I2C3_SCL_PWM4_OUT** 0x30330224, 0x1, 0x00000000, 0x0, 0x3033048C
- #define **IOMUXC_I2C3_SCL_GPT2_CLK** 0x30330224, 0x2, 0x00000000, 0x0, 0x3033048C
- #define **IOMUXC_I2C3_SCL_GPIO5_IO18** 0x30330224, 0x5, 0x00000000, 0x0, 0x3033048C
- #define **IOMUXC_I2C3_SDA_I2C3_SDA** 0x30330228, 0x0, 0x00000000, 0x0, 0x30330490
- #define **IOMUXC_I2C3_SDA_PWM3_OUT** 0x30330228, 0x1, 0x00000000, 0x0, 0x30330490
- #define **IOMUXC_I2C3_SDA_GPT3_CLK** 0x30330228, 0x2, 0x00000000, 0x0, 0x30330490
- #define **IOMUXC_I2C3_SDA_GPIO5_IO19** 0x30330228, 0x5, 0x00000000, 0x0, 0x30330490
- #define **IOMUXC_I2C4_SCL_I2C4_SCL** 0x3033022C, 0x0, 0x00000000, 0x0, 0x30330494
- #define **IOMUXC_I2C4_SCL_PWM2_OUT** 0x3033022C, 0x1, 0x00000000, 0x0, 0x30330494
- #define **IOMUXC_I2C4_SCL_PCIE1_CLKREQ_B** 0x3033022C, 0x2, 0x30330524, 0x0, 0x30330494
- #define **IOMUXC_I2C4_SCL_GPIO5_IO20** 0x3033022C, 0x5, 0x00000000, 0x0, 0x30330494
- #define **IOMUXC_I2C4_SDA_I2C4_SDA** 0x30330230, 0x0, 0x00000000, 0x0, 0x30330498
- #define **IOMUXC_I2C4_SDA_PWM1_OUT** 0x30330230, 0x1, 0x00000000, 0x0, 0x30330498
- #define **IOMUXC_I2C4_SDA_PCIE2_CLKREQ_B** 0x30330230, 0x2, 0x30330528, 0x0, 0x30330498
- #define **IOMUXC_I2C4_SDA_GPIO5_IO21** 0x30330230, 0x5, 0x00000000, 0x0, 0x30330498
- #define **IOMUXC_UART1_RXD_UART1_RX** 0x30330234, 0x0, 0x303304F4, 0x0, 0x3033049-C
- #define **IOMUXC_UART1_RXD_UART1_TX** 0x30330234, 0x0, 0x00000000, 0X0, 0x3033049-C
- #define **IOMUXC_UART1_RXD_ECSPI3_SCLK** 0x30330234, 0x1, 0x00000000, 0x0, 0x3033049C
- #define **IOMUXC_UART1_RXD_GPIO5_IO22** 0x30330234, 0x5, 0x00000000, 0x0, 0x3033049-C
- #define **IOMUXC_UART1_TXD_UART1_TX** 0x30330238, 0x0, 0x00000000, 0X0, 0x303304-A0
- #define **IOMUXC_UART1_TXD_UART1_RX** 0x30330238, 0x0, 0x303304F4, 0x1, 0x303304-A0
- #define **IOMUXC_UART1_TXD_ECSPI3_MOSI** 0x30330238, 0x1, 0x00000000, 0x0, 0x303304A0
- #define **IOMUXC_UART1_TXD_GPIO5_IO23** 0x30330238, 0x5, 0x00000000, 0x0, 0x303304-A0
- #define **IOMUXC_UART2_RXD_UART2_RX** 0x3033023C, 0x0, 0x303304FC, 0x0, 0x303304-A4
- #define **IOMUXC_UART2_RXD_UART2_TX** 0x3033023C, 0x0, 0x00000000, 0X0, 0x303304-A4
- #define **IOMUXC_UART2_RXD_ECSPI3_MISO** 0x3033023C, 0x1, 0x00000000, 0x0,

0x303304A4
- #define **IOMUXC_UART2_RXD_GPIO5_IO24** 0x3033023C, 0x5, 0x00000000, 0x0, 0x303304-A4
- #define **IOMUXC_UART2_TXD_UART2_TX** 0x30330240, 0x0, 0x00000000, 0X0, 0x303304-A8
- #define **IOMUXC_UART2_TXD_UART2_RX** 0x30330240, 0x0, 0x303304FC, 0x1, 0x303304-A8
- #define **IOMUXC_UART2_TXD_ECSPI3_SS0** 0x30330240, 0x1, 0x00000000, 0x0, 0x303304-A8
- #define **IOMUXC_UART2_TXD_GPIO5_IO25** 0x30330240, 0x5, 0x00000000, 0x0, 0x303304-A8
- #define **IOMUXC_UART3_RXD_UART3_RX** 0x30330244, 0x0, 0x30330504, 0x2, 0x303304-AC
- #define **IOMUXC_UART3_RXD_UART3_TX** 0x30330244, 0x0, 0x00000000, 0X0, 0x303304-AC
- #define **IOMUXC_UART3_RXD_UART1_CTS_B** 0x30330244, 0x1, 0x00000000, 0X0, 0x303304AC
- #define **IOMUXC_UART3_RXD_UART1_RTS_B** 0x30330244, 0x1, 0x303304F0, 0x0, 0x303304AC
- #define **IOMUXC_UART3_RXD_GPIO5_IO26** 0x30330244, 0x5, 0x00000000, 0x0, 0x303304-AC
- #define **IOMUXC_UART3_TXD_UART3_TX** 0x30330248, 0x0, 0x00000000, 0X0, 0x303304-B0
- #define **IOMUXC_UART3_TXD_UART3_RX** 0x30330248, 0x0, 0x30330504, 0x3, 0x303304-B0
- #define **IOMUXC_UART3_TXD_UART1_RTS_B** 0x30330248, 0x1, 0x303304F0, 0x1, 0x303304B0
- #define **IOMUXC_UART3_TXD_UART1_CTS_B** 0x30330248, 0x1, 0x00000000, 0X0, 0x303304B0
- #define **IOMUXC_UART3_TXD_GPIO5_IO27** 0x30330248, 0x5, 0x00000000, 0x0, 0x303304-B0
- #define **IOMUXC_UART4_RXD_UART4_RX** 0x3033024C, 0x0, 0x3033050C, 0x2, 0x303304-B4
- #define **IOMUXC_UART4_RXD_UART4_TX** 0x3033024C, 0x0, 0x00000000, 0X0, 0x303304-B4
- #define **IOMUXC_UART4_RXD_UART2_CTS_B** 0x3033024C, 0x1, 0x00000000, 0X0, 0x303304B4
- #define **IOMUXC_UART4_RXD_UART2_RTS_B** 0x3033024C, 0x1, 0x303304F8, 0x0, 0x303304B4
- #define **IOMUXC_UART4_RXD_PCIE1_CLKREQ_B** 0x3033024C, 0x2, 0x30330524, 0x1, 0x303304B4
- #define **IOMUXC_UART4_RXD_GPIO5_IO28** 0x3033024C, 0x5, 0x00000000, 0x0, 0x303304-B4
- #define **IOMUXC_UART4_TXD_UART4_TX** 0x30330250, 0x0, 0x00000000, 0X0, 0x303304-B8
- #define **IOMUXC_UART4_TXD_UART4_RX** 0x30330250, 0x0, 0x3033050C, 0x3, 0x303304-B8
- #define **IOMUXC_UART4_TXD_UART2_RTS_B** 0x30330250, 0x1, 0x303304F8, 0x1, 0x303304B8
- #define **IOMUXC_UART4_TXD_UART2_CTS_B** 0x30330250, 0x1, 0x00000000, 0X0,

0x303304B8
- #define **IOMUXC_UART4_TXD_PCIE2_CLKREQ_B** 0x30330250, 0x2, 0x30330528, 0x1, 0x303304B8
- #define **IOMUXC_UART4_TXD_GPIO5_IO29** 0x30330250, 0x5, 0x00000000, 0x0, 0x303304-B8
- #define **IOMUXC_TEST_MODE** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330254
- #define **IOMUXC_BOOT_MODE0** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330258
- #define **IOMUXC_BOOT_MODE1** 0x00000000, 0x0, 0x00000000, 0x0, 0x3033025C
- #define **IOMUXC_JTAG_MOD** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330260
- #define **IOMUXC_JTAG_TRST_B** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330264
- #define **IOMUXC_JTAG_TDI** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330268
- #define **IOMUXC_JTAG_TMS** 0x00000000, 0x0, 0x00000000, 0x0, 0x3033026C
- #define **IOMUXC_JTAG_TCK** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330270
- #define **IOMUXC_JTAG_TDO** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330274
- #define **IOMUXC_RTC** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330278

Configuration

- static void IOMUXC_SetPinMux (uint32_t muxRegister, uint32_t muxMode, uint32_t inputRegister, uint32_t inputDaisy, uint32_t configRegister, uint32_t inputOnfield)
  Sets the IOMUXC pin mux mode.
- static void IOMUXC_SetPinConfig (uint32_t muxRegister, uint32_t muxMode, uint32_t inputRegister, uint32_t inputDaisy, uint32_t configRegister, uint32_t configValue)
  Sets the IOMUXC pin configuration.

Macro Definition Documentation

7.2.1 #define FSL_IOMUXC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Function Documentation

7.3.1 static void IOMUXC_SetPinMux (uint32_t *muxRegister,* uint32_t *muxMode,* uint32_t *inputRegister,* uint32_t *inputDaisy,* uint32_t *configRegister,* uint32_t *inputOnfield*) [inline], [static]

Note

The first five parameters can be filled with the pin function ID macros.

This is an example to set the I2C4_SDA as the pwm1_OUT:

```
* IOMUXC_SetPinMux(IOMUXC_I2C4_SDA_PWM1_OUT, 0);
*
```

Parameters

muxRegister	The pin mux register_
muxMode	The pin mux mode_
inputRegister	The select input register_
inputDaisy	The input daisy_
configRegister	The config register_
inputOnfield	The pad->module input inversion_

7.3.2 static void IOMUXC_SetPinConfig (uint32_t *muxRegister*, uint32_t *muxMode*, uint32_t *inputRegister*, uint32_t *inputDaisy*, uint32_t *configRegister*, uint32_t *configValue*) [inline], [static]

Note

The previous five parameters can be filled with the pin function ID macros.

This is an example to set pin configuration for IOMUXC_I2C4_SDA_PWM1_OUT:

```
* IOMUXC_SetPinConfig(IOMUXC_I2C4_SDA_PWM1_OUT, IOMUXC_SW_PAD_CTL_PAD_ODE_MASK |
      IOMUXC0_SW_PAD_CTL_PAD_DSE(2U))
*
```

Parameters

muxRegister	The pin mux register_
muxMode	The pin mux mode_
inputRegister	The select input register_
inputDaisy	The input daisy_
configRegister	The config register_
configValue	The pin config value_

Chapter 8
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

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

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

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

8.2.4 #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U

8.2.5 #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U

8.2.6 #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U

8.2.7 #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U

8.2.8 #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U

8.2.9 #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U

8.2.10 #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U

8.2.11 #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U

8.2.12 #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U

8.2.13 #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U

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

Typedef Documentation

8.3.1 typedef int32_t status_t

Enumeration Type Documentation

8.4.1 enum _status_groups

Enumerator

kStatusGroup_Generic Group number for generic status codes.

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

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

8.4.2 anonymous enum

Enumerator

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

Function Documentation

8.5.1 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

8.5.2 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

8.5.3 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.

8.5.4 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().

8.5.5 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.

8.5.6 void SDK_Free (void * *ptr*)

Parameters

ptr	The memory to be release.

8.5.7 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 9
ECSPI: Enhanced Configurable Serial Peripheral Interface Driver

Overview

Modules

- ECSPI Driver

ECSPI Driver

9.2.1 Overview

ECSPI driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for ECSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. ECSPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the spi_handle_t as the first parameter. Initialize the handle by calling the SPI_MasterTransferCreateHandle() or SPI_SlaveTransferCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions SPI_MasterTransferNonBlocking() and SPI_SlaveTransferNonBlocking() set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_SPI_Idle status.

9.2.2 Typical use case

9.2.2.1 SPI master transfer using polling method

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

9.2.2.2 SPI master transfer using an interrupt method

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

Data Structures

- struct ecspi_channel_config_t
  ECSPI user channel configure structure. More...
- struct ecspi_master_config_t
  ECSPI master configure structure. More...
- struct ecspi_slave_config_t
  ECSPI slave configure structure. More...
- struct ecspi_transfer_t
  ECSPI transfer structure. More...
- struct ecspi_master_handle_t
  ECSPI master handle structure. More...

Macros

- #define ECSPI_DUMMYDATA (0xFFFFFFFFU)
  ECSPI dummy transfer data, the data is sent while txBuff is NULL.
- #define SPI_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Retry times for waiting flag.

Typedefs

- typedef ecspi_master_handle_t ecspi_slave_handle_t
  Slave handle is the same with master handle.
- typedef void(∗ ecspi_master_callback_t)(ECSPI_Type ∗base, ecspi_master_handle_t ∗handle, status_t status, void ∗userData)
  ECSPI master callback for finished transmit.
- typedef void(∗ ecspi_slave_callback_t)(ECSPI_Type ∗base, ecspi_slave_handle_t ∗handle, status_t status, void ∗userData)
  ECSPI slave callback for finished transmit.

Enumerations

- enum {
  kStatus_ECSPI_Busy = MAKE_STATUS(kStatusGroup_ECSPI, 0),
  kStatus_ECSPI_Idle = MAKE_STATUS(kStatusGroup_ECSPI, 1),
  kStatus_ECSPI_Error = MAKE_STATUS(kStatusGroup_ECSPI, 2),
  kStatus_ECSPI_HardwareOverFlow = MAKE_STATUS(kStatusGroup_ECSPI, 3),
  kStatus_ECSPI_Timeout = MAKE_STATUS(kStatusGroup_ECSPI, 4) }
  Return status for the ECSPI driver.
- enum ecspi_clock_polarity_t {
  kECSPI_PolarityActiveHigh = 0x0U,
  kECSPI_PolarityActiveLow }
  ECSPI clock polarity configuration.
- enum ecspi_clock_phase_t {
  kECSPI_ClockPhaseFirstEdge,
  kECSPI_ClockPhaseSecondEdge }
  ECSPI clock phase configuration.
- enum {
  kECSPI_TxfifoEmptyInterruptEnable = ECSPI_INTREG_TEEN_MASK,
  kECSPI_TxFifoDataRequstInterruptEnable = ECSPI_INTREG_TDREN_MASK,
  kECSPI_TxFifoFullInterruptEnable = ECSPI_INTREG_TFEN_MASK,
  kECSPI_RxFifoReadyInterruptEnable = ECSPI_INTREG_RREN_MASK,
  kECSPI_RxFifoDataRequstInterruptEnable = ECSPI_INTREG_RDREN_MASK,
  kECSPI_RxFifoFullInterruptEnable = ECSPI_INTREG_RFEN_MASK,
  kECSPI_RxFifoOverFlowInterruptEnable = ECSPI_INTREG_ROEN_MASK,
  kECSPI_TransferCompleteInterruptEnable = ECSPI_INTREG_TCEN_MASK,

kECSPI_AllInterruptEnable }
ECSPI interrupt sources.
- enum {
kECSPI_TxfifoEmptyFlag = ECSPI_STATREG_TE_MASK,
kECSPI_TxFifoDataRequstFlag = ECSPI_STATREG_TDR_MASK,
kECSPI_TxFifoFullFlag = ECSPI_STATREG_TF_MASK,
kECSPI_RxFifoReadyFlag = ECSPI_STATREG_RR_MASK,
kECSPI_RxFifoDataRequstFlag = ECSPI_STATREG_RDR_MASK,
kECSPI_RxFifoFullFlag = ECSPI_STATREG_RF_MASK,
kECSPI_RxFifoOverFlowFlag = ECSPI_STATREG_RO_MASK,
kECSPI_TransferCompleteFlag = ECSPI_STATREG_TC_MASK }
ECSPI status flags.
- enum {
kECSPI_TxDmaEnable = ECSPI_DMAREG_TEDEN_MASK,
kECSPI_RxDmaEnable = ECSPI_DMAREG_RXDEN_MASK,
kECSPI_DmaAllEnable = (ECSPI_DMAREG_TEDEN_MASK | ECSPI_DMAREG_RXDEN_MASK) }
ECSPI DMA enable.
- enum ecspi_Data_ready_t {
kECSPI_DataReadyIgnore = 0x0U,
kECSPI_DataReadyFallingEdge,
kECSPI_DataReadyLowLevel }
ECSPI SPI_RDY signal configuration.
- enum ecspi_channel_source_t {
kECSPI_Channel0 = 0x0U,
kECSPI_Channel1,
kECSPI_Channel2,
kECSPI_Channel3 }
ECSPI channel select source.
- enum ecspi_master_slave_mode_t {
kECSPI_Slave = 0U,
kECSPI_Master }
ECSPI master or slave mode configuration.
- enum ecspi_data_line_inactive_state_t {
kECSPI_DataLineInactiveStateHigh = 0x0U,
kECSPI_DataLineInactiveStateLow }
ECSPI data line inactive state configuration.
- enum ecspi_clock_inactive_state_t {
kECSPI_ClockInactiveStateLow = 0x0U,
kECSPI_ClockInactiveStateHigh }
ECSPI clock inactive state configuration.
- enum ecspi_chip_select_active_state_t {
kECSPI_ChipSelectActiveStateLow = 0x0U,
kECSPI_ChipSelectActiveStateHigh }
ECSPI active state configuration.
- enum ecspi_sample_period_clock_source_t {
kECSPI_spiClock = 0x0U,

kECSPI_lowFreqClock }
ECSPI sample period clock configuration.

Functions

- uint32_t ECSPI_GetInstance (ECSPI_Type *base)
  Get the instance for ECSPI module.

Driver version

- #define FSL_ECSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  ECSPI driver version.

Initialization and deinitialization

- void ECSPI_MasterGetDefaultConfig (ecspi_master_config_t *config)
  Sets the ECSPI configuration structure to default values.
- void ECSPI_MasterInit (ECSPI_Type *base, const ecspi_master_config_t *config, uint32_t srcClock_Hz)
  Initializes the ECSPI with configuration.
- void ECSPI_SlaveGetDefaultConfig (ecspi_slave_config_t *config)
  Sets the ECSPI configuration structure to default values.
- void ECSPI_SlaveInit (ECSPI_Type *base, const ecspi_slave_config_t *config)
  Initializes the ECSPI with configuration.
- void ECSPI_Deinit (ECSPI_Type *base)
  De-initializes the ECSPI.
- static void ECSPI_Enable (ECSPI_Type *base, bool enable)
  Enables or disables the ECSPI.

Status

- static uint32_t ECSPI_GetStatusFlags (ECSPI_Type *base)
  Gets the status flag.
- static void ECSPI_ClearStatusFlags (ECSPI_Type *base, uint32_t mask)
  Clear the status flag.

Interrupts

- static void ECSPI_EnableInterrupts (ECSPI_Type *base, uint32_t mask)
  Enables the interrupt for the ECSPI.
- static void ECSPI_DisableInterrupts (ECSPI_Type *base, uint32_t mask)
  Disables the interrupt for the ECSPI.

Software Reset

- static void ECSPI_SoftwareReset (ECSPI_Type *base)
  Software reset.

Channel mode check

- static bool ECSPI_IsMaster (ECSPI_Type *base, ecspi_channel_source_t channel)
  Mode check.

DMA Control

- static void ECSPI_EnableDMA (ECSPI_Type *base, uint32_t mask, bool enable)
  Enables the DMA source for ECSPI.

FIFO Operation

- static uint8_t ECSPI_GetTxFifoCount (ECSPI_Type *base)
  Get the Tx FIFO data count.
- static uint8_t ECSPI_GetRxFifoCount (ECSPI_Type *base)
  Get the Rx FIFO data count.

Bus Operations

- static void ECSPI_SetChannelSelect (ECSPI_Type *base, ecspi_channel_source_t channel)
  Set channel select for transfer.
- void ECSPI_SetChannelConfig (ECSPI_Type *base, ecspi_channel_source_t channel, const ecspi_channel_config_t *config)
  Set channel select configuration for transfer.
- void ECSPI_SetBaudRate (ECSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the baud rate for ECSPI transfer.
- status_t ECSPI_WriteBlocking (ECSPI_Type *base, uint32_t *buffer, size_t size)
  Sends a buffer of data bytes using a blocking method.
- static void ECSPI_WriteData (ECSPI_Type *base, uint32_t data)
  Writes a data into the ECSPI data register.
- static uint32_t ECSPI_ReadData (ECSPI_Type *base)
  Gets a data from the ECSPI data register.

Transactional

- void ECSPI_MasterTransferCreateHandle (ECSPI_Type *base, ecspi_master_handle_t *handle, ecspi_master_callback_t callback, void *userData)
  Initializes the ECSPI master handle.
- status_t ECSPI_MasterTransferBlocking (ECSPI_Type *base, ecspi_transfer_t *xfer)

Transfers a block of data using a polling method.
- status_t ECSPI_MasterTransferNonBlocking (ECSPI_Type ∗base, ecspi_master_handle_t ∗handle, ecspi_transfer_t ∗xfer)
  Performs a non-blocking ECSPI interrupt transfer.
- status_t ECSPI_MasterTransferGetCount (ECSPI_Type ∗base, ecspi_master_handle_t ∗handle, size_t ∗count)
  Gets the bytes of the ECSPI interrupt transferred.
- void ECSPI_MasterTransferAbort (ECSPI_Type ∗base, ecspi_master_handle_t ∗handle)
  Aborts an ECSPI transfer using interrupt.
- void ECSPI_MasterTransferHandleIRQ (ECSPI_Type ∗base, ecspi_master_handle_t ∗handle)
  Interrupts the handler for the ECSPI.
- void ECSPI_SlaveTransferCreateHandle (ECSPI_Type ∗base, ecspi_slave_handle_t ∗handle, ecspi_slave_callback_t callback, void ∗userData)
  Initializes the ECSPI slave handle.
- static status_t ECSPI_SlaveTransferNonBlocking (ECSPI_Type ∗base, ecspi_slave_handle_t ∗handle, ecspi_transfer_t ∗xfer)
  Performs a non-blocking ECSPI slave interrupt transfer.
- static status_t ECSPI_SlaveTransferGetCount (ECSPI_Type ∗base, ecspi_slave_handle_t ∗handle, size_t ∗count)
  Gets the bytes of the ECSPI interrupt transferred.
- static void ECSPI_SlaveTransferAbort (ECSPI_Type ∗base, ecspi_slave_handle_t ∗handle)
  Aborts an ECSPI slave transfer using interrupt.
- void ECSPI_SlaveTransferHandleIRQ (ECSPI_Type ∗base, ecspi_slave_handle_t ∗handle)
  Interrupts a handler for the ECSPI slave.

9.2.3 Data Structure Documentation

9.2.3.1 struct ecspi_channel_config_t

Data Fields

- ecspi_master_slave_mode_t channelMode
  Channel mode.
- ecspi_clock_inactive_state_t clockInactiveState
  Clock line (SCLK) inactive state.
- ecspi_data_line_inactive_state_t dataLineInactiveState
  Data line (MOSI&MISO) inactive state.
- ecspi_chip_select_active_state_t chipSlectActiveState
  Chip select(SS) line active state.
- ecspi_clock_polarity_t polarity
  Clock polarity.
- ecspi_clock_phase_t phase
  Clock phase.

9.2.3.2 struct ecspi_master_config_t

Data Fields

- ecspi_channel_source_t channel
  Channel number.
- ecspi_channel_config_t channelConfig
  Channel configuration.
- ecspi_sample_period_clock_source_t samplePeriodClock
  Sample period clock source.
- uint8_t burstLength
  Burst length.
- uint8_t chipSelectDelay
  SS delay time.
- uint16_t samplePeriod
  Sample period.
- uint8_t txFifoThreshold
  TX Threshold.
- uint8_t rxFifoThreshold
  RX Threshold.
- uint32_t baudRate_Bps
  ECSPI baud rate for master mode.
- bool enableLoopback
  Enable the ECSPI loopback test.

9.2.3.2.0.1 Field Documentation

9.2.3.2.0.1.1 bool ecspi_master_config_t::enableLoopback

9.2.3.3 struct ecspi_slave_config_t

Data Fields

- uint8_t burstLength
  Burst length.
- uint8_t txFifoThreshold
  TX Threshold.
- uint8_t rxFifoThreshold
  RX Threshold.
- ecspi_channel_config_t channelConfig
  Channel configuration.

9.2.3.4 struct ecspi_transfer_t

Data Fields

- uint32_t ∗ txData
  Send buffer.
- uint32_t ∗ rxData
  Receive buffer.

- size_t dataSize
  Transfer bytes.
- ecspi_channel_source_t channel
  ECSPI channel select.

9.2.3.5 struct _ecspi_master_handle

Data Fields

- ecspi_channel_source_t channel
  Channel number.
- uint32_t ∗volatile txData
  Transfer buffer.
- uint32_t ∗volatile rxData
  Receive buffer.
- volatile size_t txRemainingBytes
  Send data remaining in bytes.
- volatile size_t rxRemainingBytes
  Receive data remaining in bytes.
- volatile uint32_t state
  ECSPI internal state.
- size_t transferSize
  Bytes to be transferred.
- ecspi_master_callback_t callback
  ECSPI callback.
- void ∗ userData
  Callback parameter.

9.2.4 Macro Definition Documentation

9.2.4.1 #define FSL_ECSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

9.2.4.2 #define ECSPI_DUMMYDATA (0xFFFFFFFFU)

9.2.4.3 #define SPI_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/

9.2.5 Enumeration Type Documentation

9.2.5.1 anonymous enum

Enumerator

kStatus_ECSPI_Busy ECSPI bus is busy.
kStatus_ECSPI_Idle ECSPI is idle.
kStatus_ECSPI_Error ECSPI error.

kStatus_ECSPI_HardwareOverFlow ECSPI hardware overflow.
kStatus_ECSPI_Timeout ECSPI timeout polling status flags.

9.2.5.2 enum ecspi_clock_polarity_t

Enumerator

kECSPI_PolarityActiveHigh Active-high ECSPI polarity high (idles low).
kECSPI_PolarityActiveLow Active-low ECSPI polarity low (idles high).

9.2.5.3 enum ecspi_clock_phase_t

Enumerator

kECSPI_ClockPhaseFirstEdge First edge on SPSCK occurs at the middle of the first cycle of a data transfer.
kECSPI_ClockPhaseSecondEdge First edge on SPSCK occurs at the start of the first cycle of a data transfer.

9.2.5.4 anonymous enum

Enumerator

kECSPI_TxfifoEmptyInterruptEnable Transmit FIFO buffer empty interrupt.
kECSPI_TxFifoDataRequstInterruptEnable Transmit FIFO data requst interrupt.
kECSPI_TxFifoFullInterruptEnable Transmit FIFO full interrupt.
kECSPI_RxFifoReadyInterruptEnable Receiver FIFO ready interrupt.
kECSPI_RxFifoDataRequstInterruptEnable Receiver FIFO data requst interrupt.
kECSPI_RxFifoFullInterruptEnable Receiver FIFO full interrupt.
kECSPI_RxFifoOverFlowInterruptEnable Receiver FIFO buffer overflow interrupt.
kECSPI_TransferCompleteInterruptEnable Transfer complete interrupt.
kECSPI_AllInterruptEnable All interrupt.

9.2.5.5 anonymous enum

Enumerator

kECSPI_TxfifoEmptyFlag Transmit FIFO buffer empty flag.
kECSPI_TxFifoDataRequstFlag Transmit FIFO data requst flag.
kECSPI_TxFifoFullFlag Transmit FIFO full flag.
kECSPI_RxFifoReadyFlag Receiver FIFO ready flag.
kECSPI_RxFifoDataRequstFlag Receiver FIFO data requst flag.
kECSPI_RxFifoFullFlag Receiver FIFO full flag.
kECSPI_RxFifoOverFlowFlag Receiver FIFO buffer overflow flag.
kECSPI_TransferCompleteFlag Transfer complete flag.

9.2.5.6 anonymous enum

Enumerator

kECSPI_TxDmaEnable Tx DMA request source.
kECSPI_RxDmaEnable Rx DMA request source.
kECSPI_DmaAllEnable All DMA request source.

9.2.5.7 enum ecspi_Data_ready_t

Enumerator

kECSPI_DataReadyIgnore SPI_RDY signal is ignored.
kECSPI_DataReadyFallingEdge SPI_RDY signal will be triggerd by the falling edge.
kECSPI_DataReadyLowLevel SPI_RDY signal will be triggerd by a low level.

9.2.5.8 enum ecspi_channel_source_t

Enumerator

kECSPI_Channel0 Channel 0 is selectd.
kECSPI_Channel1 Channel 1 is selectd.
kECSPI_Channel2 Channel 2 is selectd.
kECSPI_Channel3 Channel 3 is selectd.

9.2.5.9 enum ecspi_master_slave_mode_t

Enumerator

kECSPI_Slave ECSPI peripheral operates in slave mode.
kECSPI_Master ECSPI peripheral operates in master mode.

9.2.5.10 enum ecspi_data_line_inactive_state_t

Enumerator

kECSPI_DataLineInactiveStateHigh The data line inactive state stays high.
kECSPI_DataLineInactiveStateLow The data line inactive state stays low.

9.2.5.11 enum ecspi_clock_inactive_state_t

Enumerator

kECSPI_ClockInactiveStateLow The SCLK inactive state stays low.
kECSPI_ClockInactiveStateHigh The SCLK inactive state stays high.

9.2.5.12 enum ecspi_chip_select_active_state_t

Enumerator

kECSPI_ChipSelectActiveStateLow The SS signal line active stays low.
kECSPI_ChipSelectActiveStateHigh The SS signal line active stays high.

9.2.5.13 enum ecspi_sample_period_clock_source_t

Enumerator

kECSPI_spiClock The sample period clock source is SCLK.
kECSPI_lowFreqClock The sample seriod clock source is low_frequency reference clock(32.768 kHz).

9.2.6 Function Documentation

9.2.6.1 uint32_t ECSPI_GetInstance (ECSPI_Type * *base*)

Parameters

base	ECSPI base address

9.2.6.2 void ECSPI_MasterGetDefaultConfig (ecspi_master_config_t * *config*)

The purpose of this API is to get the configuration structure initialized for use in ECSPI_MasterInit(). User may use the initialized structure unchanged in ECSPI_MasterInit, or modify some fields of the structure before calling ECSPI_MasterInit. After calling this API, the master is ready to transfer. Example:

```
ecspi_master_config_t config;
ECSPI_MasterGetDefaultConfig(&config);
```

Parameters

config	pointer to config structure

9.2.6.3 void ECSPI_MasterInit (ECSPI_Type ∗ *base*, const ecspi_master_config_t ∗ *config*, uint32_t *srcClock_Hz*)

The configuration structure can be filled by user from scratch, or be set with default values by ECSPI_MasterGetDefaultConfig(). After calling this API, the slave is ready to transfer. Example

```
ecspi_master_config_t config = {
.baudRate_Bps = 400000,
...
};
ECSPI_MasterInit(ECSPI0, &config);
```

Parameters

base	ECSPI base pointer
config	pointer to master configuration structure
srcClock_Hz	Source clock frequency.

9.2.6.4 void ECSPI_SlaveGetDefaultConfig (ecspi_slave_config_t ∗ *config*)

The purpose of this API is to get the configuration structure initialized for use in ECSPI_SlaveInit(). User may use the initialized structure unchanged in ECSPI_SlaveInit(), or modify some fields of the structure before calling ECSPI_SlaveInit(). After calling this API, the master is ready to transfer. Example:

```
ecspi_Slaveconfig_t config;
ECSPI_SlaveGetDefaultConfig(&config);
```

Parameters

config	pointer to config structure

9.2.6.5 void ECSPI_SlaveInit (ECSPI_Type ∗ *base*, const ecspi_slave_config_t ∗ *config*)

The configuration structure can be filled by user from scratch, or be set with default values by ECSPI_SlaveGetDefaultConfig(). After calling this API, the slave is ready to transfer. Example

```
ecspi_Salveconfig_t config = {
.baudRate_Bps = 400000,
...
};
ECSPI_SlaveInit(ECSPI1, &config);
```

Parameters

base	ECSPI base pointer
config	pointer to master configuration structure

9.2.6.6 void ECSPI_Deinit (ECSPI_Type ∗ *base*)

Calling this API resets the ECSPI module, gates the ECSPI clock. The ECSPI module can't work unless calling the ECSPI_MasterInit/ECSPI_SlaveInit to initialize module.

Parameters

base	ECSPI base pointer

9.2.6.7 static void ECSPI_Enable (ECSPI_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

base	ECSPI base pointer
enable	pass true to enable module, false to disable module

9.2.6.8 static uint32_t ECSPI_GetStatusFlags (ECSPI_Type ∗ *base*) [inline], [static]

Parameters

base	ECSPI base pointer

Returns

ECSPI Status, use status flag to AND _ecspi_flags could get the related status.

9.2.6.9 static void ECSPI_ClearStatusFlags (ECSPI_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	ECSPI base pointer
mask	ECSPI Status, use status flag to AND _ecspi_flags could get the related status.

9.2.6.10 static void ECSPI_EnableInterrupts (ECSPI_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	ECSPI base pointer
mask	ECSPI interrupt source. The parameter can be any combination of the following values: • kECSPI_TxfifoEmptyInterruptEnable • kECSPI_TxFifoDataRequstInterruptEnable • kECSPI_TxFifoFullInterruptEnable • kECSPI_RxFifoReadyInterruptEnable • kECSPI_RxFifoDataRequstInterruptEnable • kECSPI_RxFifoFullInterruptEnable • kECSPI_RxFifoOverFlowInterruptEnable • kECSPI_TransferCompleteInterruptEnable • kECSPI_AllInterruptEnable

9.2.6.11 static void ECSPI_DisableInterrupts (ECSPI_Type * base, uint32_t mask) [inline], [static]

Parameters

base	ECSPI base pointer
mask	ECSPI interrupt source. The parameter can be any combination of the following values: • kECSPI_TxfifoEmptyInterruptEnable • kECSPI_TxFifoDataRequstInterruptEnable • kECSPI_TxFifoFullInterruptEnable • kECSPI_RxFifoReadyInterruptEnable • kECSPI_RxFifoDataRequstInterruptEnable • kECSPI_RxFifoFullInterruptEnable • kECSPI_RxFifoOverFlowInterruptEnable • kECSPI_TransferCompleteInterruptEnable • kECSPI_AllInterruptEnable

9.2.6.12 static void ECSPI_SoftwareReset (ECSPI_Type * base) [inline], [static]

Parameters

base	ECSPI base pointer

9.2.6.13 static bool ECSPI_IsMaster (ECSPI_Type * base, ecspi_channel_source_t channel) [inline], [static]

Parameters

base	ECSPI base pointer
channel	ECSPI channel source

Returns

mode of channel

9.2.6.14 static void ECSPI_EnableDMA (ECSPI_Type * base, uint32_t mask, bool enable) [inline], [static]

Parameters

base	ECSPI base pointer
mask	ECSPI DMA source. The parameter can be any of the following values: • kECSPI_TxDmaEnable • kECSPI_RxDmaEnable • kECSPI_DmaAllEnable
enable	True means enable DMA, false means disable DMA

9.2.6.15 static uint8_t ECSPI_GetTxFifoCount (ECSPI_Type * *base*) [inline], [static]

Parameters

base	ECSPI base pointer.

Returns

the number of words in Tx FIFO buffer.

9.2.6.16 static uint8_t ECSPI_GetRxFifoCount (ECSPI_Type * *base*) [inline], [static]

Parameters

base	ECSPI base pointer.

Returns

the number of words in Rx FIFO buffer.

9.2.6.17 static void ECSPI_SetChannelSelect (ECSPI_Type * *base*, ecspi_channel_source_t *channel*) [inline], [static]

Parameters

base	ECSPI base pointer
channel	Channel source.

9.2.6.18 void ECSPI_SetChannelConfig (ECSPI_Type ∗ *base*, ecspi_channel_source_t *channel*, const ecspi_channel_config_t ∗ *config*)

The purpose of this API is to set the channel will be use to transfer. User may use this API after instance has been initialized or before transfer start. The configuration structure *ecspi_channel_config* can be filled by user from scratch. After calling this API, user can select this channel as transfer channel.

Parameters

base	ECSPI base pointer
channel	Channel source.
config	Configuration struct of channel

9.2.6.19 void ECSPI_SetBaudRate (ECSPI_Type ∗ *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

This is only used in master.

Parameters

base	ECSPI base pointer
baudRate_Bps	baud rate needed in Hz.
srcClock_Hz	ECSPI source clock frequency in Hz.

9.2.6.20 status_t ECSPI_WriteBlocking (ECSPI_Type ∗ *base*, uint32_t ∗ *buffer*, size_t *size*)

Note

This function blocks via polling until all bytes have been sent.

Parameters

base	ECSPI base pointer
buffer	The data bytes to send
size	The number of data bytes to send

Return values

kStatus_Success	Successfully start a transfer.
kStatus_ECSPI_Timeout	The transfer timed out and was aborted.

9.2.6.21 static void ECSPI_WriteData (ECSPI_Type ∗ *base*, uint32_t *data*) [inline], [static]

Parameters

base	ECSPI base pointer
data	Data needs to be write.

9.2.6.22 static uint32_t ECSPI_ReadData (ECSPI_Type ∗ *base*) [inline], [static]

Parameters

base	ECSPI base pointer

Returns

Data in the register.

9.2.6.23 void ECSPI_MasterTransferCreateHandle (ECSPI_Type ∗ *base*, ecspi_master_handle_t ∗ *handle*, ecspi_master_callback_t *callback*, void ∗ *userData*)

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

Parameters

base	ECSPI peripheral base address.
handle	ECSPI handle pointer.
callback	Callback function.
userData	User data.

9.2.6.24 status_t ECSPI_MasterTransferBlocking (ECSPI_Type ∗ *base*, ecspi_transfer_t ∗ *xfer*)

Parameters

base	SPI base pointer
xfer	pointer to spi_xfer_config_t structure

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_ECSPI_Timeout	The transfer timed out and was aborted.

9.2.6.25 status_t ECSPI_MasterTransferNonBlocking (ECSPI_Type ∗ *base*, ecspi_master_handle_t ∗ *handle*, ecspi_transfer_t ∗ *xfer*)

Note

> The API immediately returns after transfer initialization is finished.
> If ECSPI transfer data frame size is 16 bits, the transfer size cannot be an odd number.

Parameters

base	ECSPI peripheral base address.
handle	pointer to ecspi_master_handle_t structure which stores the transfer state
xfer	pointer to ecspi_transfer_t structure

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_ECSPI_Busy	ECSPI is not idle, is running another transfer.

9.2.6.26 status_t ECSPI_MasterTransferGetCount (ECSPI_Type ∗ *base*, ecspi_master_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

base	ECSPI peripheral base address.
handle	Pointer to ECSPI transfer handle, this should be a static variable.
count	Transferred bytes of ECSPI master.

Return values

kStatus_ECSPI_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

9.2.6.27 void ECSPI_MasterTransferAbort (ECSPI_Type ∗ *base*, ecspi_master_handle_t ∗ *handle*)

Parameters

base	ECSPI peripheral base address.
handle	Pointer to ECSPI transfer handle, this should be a static variable.

9.2.6.28 void ECSPI_MasterTransferHandleIRQ (ECSPI_Type ∗ *base*, ecspi_master_handle_t ∗ *handle*)

Parameters

base	ECSPI peripheral base address.
handle	pointer to ecspi_master_handle_t structure which stores the transfer state.

9.2.6.29 void ECSPI_SlaveTransferCreateHandle (ECSPI_Type * *base*, ecspi_slave_handle_t * *handle*, ecspi_slave_callback_t *callback*, void * *userData*)

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

Parameters

base	ECSPI peripheral base address.
handle	ECSPI handle pointer.
callback	Callback function.
userData	User data.

9.2.6.30 static status_t ECSPI_SlaveTransferNonBlocking (ECSPI_Type * *base*, ecspi_slave_handle_t * *handle*, ecspi_transfer_t * *xfer*) [inline], [static]

Note

The API returns immediately after the transfer initialization is finished.

Parameters

base	ECSPI peripheral base address.
handle	pointer to ecspi_master_handle_t structure which stores the transfer state
xfer	pointer to ecspi_transfer_t structure

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_ECSPI_Busy	ECSPI is not idle, is running another transfer.

9.2.6.31 static status_t ECSPI_SlaveTransferGetCount (ECSPI_Type * *base*, ecspi_slave_handle_t * *handle*, size_t * *count*) [inline], [static]

Parameters

base	ECSPI peripheral base address.
handle	Pointer to ECSPI transfer handle, this should be a static variable.
count	Transferred bytes of ECSPI slave.

Return values

kStatus_ECSPI_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

9.2.6.32 static void ECSPI_SlaveTransferAbort (ECSPI_Type ∗ *base*, ecspi_slave_handle_t ∗ *handle*) [inline], [static]

Parameters

base	ECSPI peripheral base address.
handle	Pointer to ECSPI transfer handle, this should be a static variable.

9.2.6.33 void ECSPI_SlaveTransferHandleIRQ (ECSPI_Type ∗ *base*, ecspi_slave_handle_t ∗ *handle*)

Parameters

base	ECSPI peripheral base address.
handle	pointer to ecspi_slave_handle_t structure which stores the transfer state

Chapter 10
GPT: General Purpose Timer

Overview

The MCUXpresso SDK provides a driver for the General Purpose Timer (GPT) of MCUXpresso SDK devices.

Function groups

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

10.2.1 Initialization and deinitialization

The function GPT_Init() initializes the gpt with specified configurations. The function GPT_GetDefaultConfig() gets the default configurations. The initialization function configures the restart/free-run mode and input selection when running.

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

Typical use case

10.3.1 GPT interrupt example

Set up a channel to trigger a periodic interrupt after every 1 second. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/gpt

Data Structures

- struct gpt_config_t
  Structure to configure the running mode. More...

Enumerations

- enum gpt_clock_source_t {
  kGPT_ClockSource_Off = 0U,
  kGPT_ClockSource_Periph = 1U,
  kGPT_ClockSource_HighFreq = 2U,
  kGPT_ClockSource_Ext = 3U,
  kGPT_ClockSource_LowFreq = 4U,
  kGPT_ClockSource_Osc = 5U }
  List of clock sources.

- enum gpt_input_capture_channel_t {
  kGPT_InputCapture_Channel1 = 0U,
  kGPT_InputCapture_Channel2 = 1U }
  List of input capture channel number.
- enum gpt_input_operation_mode_t {
  kGPT_InputOperation_Disabled = 0U,
  kGPT_InputOperation_RiseEdge = 1U,
  kGPT_InputOperation_FallEdge = 2U,
  kGPT_InputOperation_BothEdge = 3U }
  List of input capture operation mode.
- enum gpt_output_compare_channel_t {
  kGPT_OutputCompare_Channel1 = 0U,
  kGPT_OutputCompare_Channel2 = 1U,
  kGPT_OutputCompare_Channel3 = 2U }
  List of output compare channel number.
- enum gpt_output_operation_mode_t {
  kGPT_OutputOperation_Disconnected = 0U,
  kGPT_OutputOperation_Toggle = 1U,
  kGPT_OutputOperation_Clear = 2U,
  kGPT_OutputOperation_Set = 3U,
  kGPT_OutputOperation_Activelow = 4U }
  List of output compare operation mode.
- enum gpt_interrupt_enable_t {
  kGPT_OutputCompare1InterruptEnable = GPT_IR_OF1IE_MASK,
  kGPT_OutputCompare2InterruptEnable = GPT_IR_OF2IE_MASK,
  kGPT_OutputCompare3InterruptEnable = GPT_IR_OF3IE_MASK,
  kGPT_InputCapture1InterruptEnable = GPT_IR_IF1IE_MASK,
  kGPT_InputCapture2InterruptEnable = GPT_IR_IF2IE_MASK,
  kGPT_RollOverFlagInterruptEnable = GPT_IR_ROVIE_MASK }
  List of GPT interrupts.
- enum gpt_status_flag_t {
  kGPT_OutputCompare1Flag = GPT_SR_OF1_MASK,
  kGPT_OutputCompare2Flag = GPT_SR_OF2_MASK,
  kGPT_OutputCompare3Flag = GPT_SR_OF3_MASK,
  kGPT_InputCapture1Flag = GPT_SR_IF1_MASK,
  kGPT_InputCapture2Flag = GPT_SR_IF2_MASK,
  kGPT_RollOverFlag = GPT_SR_ROV_MASK }
  Status flag.

Driver version

- #define **FSL_GPT_DRIVER_VERSION** (MAKE_VERSION(2, 0, 2))

Initialization and deinitialization

- void GPT_Init (GPT_Type ∗base, const gpt_config_t ∗initConfig)
  Initialize GPT to reset state and initialize running mode.

- void GPT_Deinit (GPT_Type ∗base)
  Disables the module and gates the GPT clock.
- void GPT_GetDefaultConfig (gpt_config_t ∗config)
  Fills in the GPT configuration structure with default settings.

Software Reset

- static void GPT_SoftwareReset (GPT_Type ∗base)
  Software reset of GPT module.

Clock source and frequency control

- static void GPT_SetClockSource (GPT_Type ∗base, gpt_clock_source_t source)
  Set clock source of GPT.
- static gpt_clock_source_t GPT_GetClockSource (GPT_Type ∗base)
  Get clock source of GPT.
- static void GPT_SetClockDivider (GPT_Type ∗base, uint32_t divider)
  Set pre scaler of GPT.
- static uint32_t GPT_GetClockDivider (GPT_Type ∗base)
  Get clock divider in GPT module.
- static void GPT_SetOscClockDivider (GPT_Type ∗base, uint32_t divider)
  OSC 24M pre-scaler before selected by clock source.
- static uint32_t GPT_GetOscClockDivider (GPT_Type ∗base)
  Get OSC 24M clock divider in GPT module.

Timer Start and Stop

- static void GPT_StartTimer (GPT_Type ∗base)
  Start GPT timer.
- static void GPT_StopTimer (GPT_Type ∗base)
  Stop GPT timer.

Read the timer period

- static uint32_t GPT_GetCurrentTimerCount (GPT_Type ∗base)
  Reads the current GPT counting value.

GPT Input/Output Signal Control

- static void GPT_SetInputOperationMode (GPT_Type ∗base, gpt_input_capture_channel_t channel, gpt_input_operation_mode_t mode)
  Set GPT operation mode of input capture channel.
- static gpt_input_operation_mode_t GPT_GetInputOperationMode (GPT_Type ∗base, gpt_input_capture_channel_t channel)
  Get GPT operation mode of input capture channel.
- static uint32_t GPT_GetInputCaptureValue (GPT_Type ∗base, gpt_input_capture_channel_t channel)
  Get GPT input capture value of certain channel.
- static void GPT_SetOutputOperationMode (GPT_Type ∗base, gpt_output_compare_channel_t channel, gpt_output_operation_mode_t mode)

Set GPT operation mode of output compare channel.
- static gpt_output_operation_mode_t GPT_GetOutputOperationMode (GPT_Type ∗base, gpt_output_compare_channel_t channel)
  Get GPT operation mode of output compare channel.
- static void GPT_SetOutputCompareValue (GPT_Type ∗base, gpt_output_compare_channel_t channel, uint32_t value)
  Set GPT output compare value of output compare channel.
- static uint32_t GPT_GetOutputCompareValue (GPT_Type ∗base, gpt_output_compare_channel_t channel)
  Get GPT output compare value of output compare channel.
- static void GPT_ForceOutput (GPT_Type ∗base, gpt_output_compare_channel_t channel)
  Force GPT output action on output compare channel, ignoring comparator.

GPT Interrupt and Status Interface

- static void GPT_EnableInterrupts (GPT_Type ∗base, uint32_t mask)
  Enables the selected GPT interrupts.
- static void GPT_DisableInterrupts (GPT_Type ∗base, uint32_t mask)
  Disables the selected GPT interrupts.
- static uint32_t GPT_GetEnabledInterrupts (GPT_Type ∗base)
  Gets the enabled GPT interrupts.

Status Interface

- static uint32_t GPT_GetStatusFlags (GPT_Type ∗base, gpt_status_flag_t flags)
  Get GPT status flags.
- static void GPT_ClearStatusFlags (GPT_Type ∗base, gpt_status_flag_t flags)
  Clears the GPT status flags.

Data Structure Documentation

10.4.1 struct gpt_config_t

Data Fields

- gpt_clock_source_t clockSource
  clock source for GPT module.
- uint32_t divider
  clock divider (prescaler+1) from clock source to counter.
- bool enableFreeRun
  true: FreeRun mode, false: Restart mode.
- bool enableRunInWait
  GPT enabled in wait mode.
- bool enableRunInStop
  GPT enabled in stop mode.
- bool enableRunInDoze
  GPT enabled in doze mode.
- bool enableRunInDbg
  GPT enabled in debug mode.

- bool enableMode
```
true:  counter reset to 0 when enabled;
```
  false: counter retain its value when enabled.

10.4.1.0.0.2 Field Documentation

10.4.1.0.0.2.1 gpt_clock_source_t gpt_config_t::clockSource

10.4.1.0.0.2.2 uint32_t gpt_config_t::divider

10.4.1.0.0.2.3 bool gpt_config_t::enableFreeRun

10.4.1.0.0.2.4 bool gpt_config_t::enableRunInWait

10.4.1.0.0.2.5 bool gpt_config_t::enableRunInStop

10.4.1.0.0.2.6 bool gpt_config_t::enableRunInDoze

10.4.1.0.0.2.7 bool gpt_config_t::enableRunInDbg

10.4.1.0.0.2.8 bool gpt_config_t::enableMode

Enumeration Type Documentation

10.5.1 enum gpt_clock_source_t

Note

Actual number of clock sources is SoC dependent

Enumerator

kGPT_ClockSource_Off GPT Clock Source Off.
kGPT_ClockSource_Periph GPT Clock Source from Peripheral Clock.
kGPT_ClockSource_HighFreq GPT Clock Source from High Frequency Reference Clock.
kGPT_ClockSource_Ext GPT Clock Source from external pin.
kGPT_ClockSource_LowFreq GPT Clock Source from Low Frequency Reference Clock.
kGPT_ClockSource_Osc GPT Clock Source from Crystal oscillator.

10.5.2 enum gpt_input_capture_channel_t

Enumerator

kGPT_InputCapture_Channel1 GPT Input Capture Channel1.
kGPT_InputCapture_Channel2 GPT Input Capture Channel2.

10.5.3 enum gpt_input_operation_mode_t

Enumerator

kGPT_InputOperation_Disabled Don't capture.
kGPT_InputOperation_RiseEdge Capture on rising edge of input pin.
kGPT_InputOperation_FallEdge Capture on falling edge of input pin.
kGPT_InputOperation_BothEdge Capture on both edges of input pin.

10.5.4 enum gpt_output_compare_channel_t

Enumerator

kGPT_OutputCompare_Channel1 Output Compare Channel1.
kGPT_OutputCompare_Channel2 Output Compare Channel2.
kGPT_OutputCompare_Channel3 Output Compare Channel3.

10.5.5 enum gpt_output_operation_mode_t

Enumerator

kGPT_OutputOperation_Disconnected Don't change output pin.
kGPT_OutputOperation_Toggle Toggle output pin.
kGPT_OutputOperation_Clear Set output pin low.
kGPT_OutputOperation_Set Set output pin high.
kGPT_OutputOperation_Activelow Generate a active low pulse on output pin.

10.5.6 enum gpt_interrupt_enable_t

Enumerator

kGPT_OutputCompare1InterruptEnable Output Compare Channel1 interrupt enable.
kGPT_OutputCompare2InterruptEnable Output Compare Channel2 interrupt enable.
kGPT_OutputCompare3InterruptEnable Output Compare Channel3 interrupt enable.
kGPT_InputCapture1InterruptEnable Input Capture Channel1 interrupt enable.
kGPT_InputCapture2InterruptEnable Input Capture Channel1 interrupt enable.
kGPT_RollOverFlagInterruptEnable Counter rolled over interrupt enable.

10.5.7 enum gpt_status_flag_t

Enumerator

kGPT_OutputCompare1Flag Output compare channel 1 event.
kGPT_OutputCompare2Flag Output compare channel 2 event.
kGPT_OutputCompare3Flag Output compare channel 3 event.
kGPT_InputCapture1Flag Input Capture channel 1 event.
kGPT_InputCapture2Flag Input Capture channel 2 event.
kGPT_RollOverFlag Counter reaches maximum value and rolled over to 0 event.

Function Documentation

10.6.1 void GPT_Init (GPT_Type ∗ *base*, const gpt_config_t ∗ *initConfig*)

Parameters

base	GPT peripheral base address.
initConfig	GPT mode setting configuration.

10.6.2 void GPT_Deinit (GPT_Type ∗ *base*)

Parameters

base	GPT peripheral base address.

10.6.3 void GPT_GetDefaultConfig (gpt_config_t ∗ *config*)

The default values are:

```
*    config->clockSource = kGPT_ClockSource_Periph;
*    config->divider = 1U;
*    config->enableRunInStop = true;
*    config->enableRunInWait = true;
*    config->enableRunInDoze = false;
*    config->enableRunInDbg = false;
*    config->enableFreeRun = false;
*    config->enableMode  = true;
*
```

Parameters

config	Pointer to the user configuration structure.

10.6.4 static void GPT_SoftwareReset (GPT_Type * *base*) [inline], [static]

Parameters

base	GPT peripheral base address.

10.6.5 static void GPT_SetClockSource (GPT_Type * *base*, gpt_clock_source_t *source*) [inline], [static]

Parameters

base	GPT peripheral base address.
source	Clock source (see gpt_clock_source_t typedef enumeration).

10.6.6 static gpt_clock_source_t GPT_GetClockSource (GPT_Type * *base*) [inline], [static]

Parameters

base	GPT peripheral base address.

Returns

clock source (see gpt_clock_source_t typedef enumeration).

10.6.7 static void GPT_SetClockDivider (GPT_Type * *base*, uint32_t *divider*) [inline], [static]

Parameters

base	GPT peripheral base address.
divider	Divider of GPT (1-4096).

10.6.8 static uint32_t GPT_GetClockDivider (GPT_Type ∗ *base*) [inline], [static]

Parameters

base	GPT peripheral base address.

Returns

clock divider in GPT module (1-4096).

10.6.9 static void GPT_SetOscClockDivider (GPT_Type ∗ *base*, uint32_t *divider*) [inline], [static]

Parameters

base	GPT peripheral base address.
divider	OSC Divider(1-16).

10.6.10 static uint32_t GPT_GetOscClockDivider (GPT_Type ∗ *base*) [inline], [static]

Parameters

base	GPT peripheral base address.

Returns

OSC clock divider in GPT module (1-16).

10.6.11 static void GPT_StartTimer (GPT_Type ∗ *base*) [inline], [static]

Parameters

base	GPT peripheral base address.

10.6.12 static void GPT_StopTimer (GPT_Type ∗ *base*) [inline], [static]

Parameters

base	GPT peripheral base address.

10.6.13 static uint32_t GPT_GetCurrentTimerCount (GPT_Type ∗ *base*) [inline], [static]

Parameters

base	GPT peripheral base address.

Returns

Current GPT counter value.

10.6.14 static void GPT_SetInputOperationMode (GPT_Type ∗ *base*, gpt_input_capture_channel_t *channel*, gpt_input_operation_mode_t *mode*) [inline], [static]

Parameters

base	GPT peripheral base address.
channel	GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).
mode	GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

10.6.15 static gpt_input_operation_mode_t GPT_GetInputOperationMode (GPT_Type ∗ *base*, gpt_input_capture_channel_t *channel*) [inline], [static]

Parameters

base	GPT peripheral base address.
channel	GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

10.6.16 static uint32_t GPT_GetInputCaptureValue (GPT_Type * *base*, gpt_input_capture_channel_t *channel*) [inline], [static]

Parameters

base	GPT peripheral base address.
channel	GPT capture channel (see gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture value.

10.6.17 static void GPT_SetOutputOperationMode (GPT_Type * *base*, gpt_output_compare_channel_t *channel*, gpt_output_operation_mode_t *mode*) [inline], [static]

Parameters

base	GPT peripheral base address.
channel	GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).
mode	GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

10.6.18 static gpt_output_operation_mode_t GPT_GetOutputOperationMode (GPT_Type * *base*, gpt_output_compare_channel_t *channel*) [inline], [static]

Parameters

base	GPT peripheral base address.
channel	GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

10.6.19 static void GPT_SetOutputCompareValue (GPT_Type * *base*, gpt_output_compare_channel_t *channel*, uint32_t *value*) [inline], [static]

Parameters

base	GPT peripheral base address.
channel	GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).
value	GPT output compare value.

10.6.20 static uint32_t GPT_GetOutputCompareValue (GPT_Type * *base*, gpt_output_compare_channel_t *channel*) [inline], [static]

Parameters

base	GPT peripheral base address.
channel	GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output compare value.

10.6.21 static void GPT_ForceOutput (GPT_Type * *base*, gpt_output_compare_channel_t *channel*) [inline], [static]

Parameters

base	GPT peripheral base address.
channel	GPT output compare channel (see gpt_output_compare_channel_t typedef enumeration).

10.6.22 static void GPT_EnableInterrupts (GPT_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

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

10.6.23 static void GPT_DisableInterrupts (GPT_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	GPT peripheral base address
mask	The interrupts to disable. This is a logical OR of members of the enumeration gpt_interrupt_enable_t

10.6.24 static uint32_t GPT_GetEnabledInterrupts (GPT_Type * *base*) [inline], [static]

Parameters

base	GPT peripheral base address

Returns

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

10.6.25 static uint32_t GPT_GetStatusFlags (GPT_Type * *base*, gpt_status_flag_t *flags*) [inline], [static]

Parameters

base	GPT peripheral base address.
flags	GPT status flag mask (see gpt_status_flag_t for bit definition).

Returns

GPT status, each bit represents one status flag.

10.6.26 static void GPT_ClearStatusFlags (GPT_Type * *base*, gpt_status_flag_t *flags*) [inline], [static]

Parameters

base	GPT peripheral base address.
flags	GPT status flag mask (see gpt_status_flag_t for bit definition).

Chapter 11
GPIO: General-Purpose Input/Output Driver

Overview

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

Typical use case

11.2.1 Input Operation

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

Data Structures

- struct gpio_pin_config_t
  GPIO Init structure definition. More...

Enumerations

- enum gpio_pin_direction_t {
  kGPIO_DigitalInput = 0U,
  kGPIO_DigitalOutput = 1U }
  GPIO direction definition.
- enum gpio_interrupt_mode_t {
  kGPIO_NoIntmode = 0U,
  kGPIO_IntLowLevel = 1U,
  kGPIO_IntHighLevel = 2U,
  kGPIO_IntRisingEdge = 3U,
  kGPIO_IntFallingEdge = 4U,
  kGPIO_IntRisingOrFallingEdge = 5U }
  GPIO interrupt mode definition.

Driver version

- #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))
  GPIO driver version.

GPIO Initialization and Configuration functions

- void GPIO_PinInit (GPIO_Type ∗base, uint32_t pin, const gpio_pin_config_t ∗Config)
  Initializes the GPIO peripheral according to the specified parameters in the initConfig.

GPIO Reads and Write Functions

- void GPIO_PinWrite (GPIO_Type ∗base, uint32_t pin, uint8_t output)
  Sets the output level of the individual GPIO pin to logic 1 or 0.
- static void GPIO_WritePinOutput (GPIO_Type ∗base, uint32_t pin, uint8_t output)
  Sets the output level of the individual GPIO pin to logic 1 or 0.
- static void GPIO_PortSet (GPIO_Type ∗base, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 1.
- static void GPIO_SetPinsOutput (GPIO_Type ∗base, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 1.
- static void GPIO_PortClear (GPIO_Type ∗base, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 0.
- static void GPIO_ClearPinsOutput (GPIO_Type ∗base, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 0.
- static void GPIO_PortToggle (GPIO_Type ∗base, uint32_t mask)
  Reverses the current output logic of the multiple GPIO pins.
- static uint32_t GPIO_PinRead (GPIO_Type ∗base, uint32_t pin)
  Reads the current input value of the GPIO port.
- static uint32_t GPIO_ReadPinInput (GPIO_Type ∗base, uint32_t pin)
  Reads the current input value of the GPIO port.

GPIO Reads Pad Status Functions

- static uint8_t GPIO_PinReadPadStatus (GPIO_Type ∗base, uint32_t pin)
  Reads the current GPIO pin pad status.
- static uint8_t GPIO_ReadPadStatus (GPIO_Type ∗base, uint32_t pin)
  Reads the current GPIO pin pad status.

Interrupts and flags management functions

- void GPIO_PinSetInterruptConfig (GPIO_Type ∗base, uint32_t pin, gpio_interrupt_mode_t pinInterruptMode)
  Sets the current pin interrupt mode.
- static void GPIO_SetPinInterruptConfig (GPIO_Type ∗base, uint32_t pin, gpio_interrupt_mode_t pinInterruptMode)
  Sets the current pin interrupt mode.
- static void GPIO_PortEnableInterrupts (GPIO_Type ∗base, uint32_t mask)
  Enables the specific pin interrupt.
- static void GPIO_EnableInterrupts (GPIO_Type ∗base, uint32_t mask)
  Enables the specific pin interrupt.
- static void GPIO_PortDisableInterrupts (GPIO_Type ∗base, uint32_t mask)
  Disables the specific pin interrupt.
- static void GPIO_DisableInterrupts (GPIO_Type ∗base, uint32_t mask)
  Disables the specific pin interrupt.
- static uint32_t GPIO_PortGetInterruptFlags (GPIO_Type ∗base)
  Reads individual pin interrupt status.
- static uint32_t GPIO_GetPinsInterruptFlags (GPIO_Type ∗base)
  Reads individual pin interrupt status.
- static void GPIO_PortClearInterruptFlags (GPIO_Type ∗base, uint32_t mask)
  Clears pin interrupt flag.
- static void GPIO_ClearPinsInterruptFlags (GPIO_Type ∗base, uint32_t mask)

Clears pin interrupt flag.

Data Structure Documentation

11.3.1 struct gpio_pin_config_t

Data Fields

- gpio_pin_direction_t direction
  Specifies the pin direction.
- uint8_t outputLogic
  Set a default output logic, which has no use in input.
- gpio_interrupt_mode_t interruptMode
  Specifies the pin interrupt mode, a value of gpio_interrupt_mode_t.

11.3.1.0.0.3 Field Documentation

11.3.1.0.0.3.1 gpio_pin_direction_t gpio_pin_config_t::direction

11.3.1.0.0.3.2 gpio_interrupt_mode_t gpio_pin_config_t::interruptMode

Macro Definition Documentation

11.4.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

Enumeration Type Documentation

11.5.1 enum gpio_pin_direction_t

Enumerator

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

11.5.2 enum gpio_interrupt_mode_t

Enumerator

kGPIO_NoIntmode Set current pin general IO functionality.
kGPIO_IntLowLevel Set current pin interrupt is low-level sensitive.
kGPIO_IntHighLevel Set current pin interrupt is high-level sensitive.
kGPIO_IntRisingEdge Set current pin interrupt is rising-edge sensitive.
kGPIO_IntFallingEdge Set current pin interrupt is falling-edge sensitive.
kGPIO_IntRisingOrFallingEdge Enable the edge select bit to override the ICR register's configuration.

Function Documentation

11.6.1 void GPIO_PinInit (GPIO_Type * *base*, uint32_t *pin*, const gpio_pin_config_t * *Config*)

Parameters

base	GPIO base pointer.
pin	Specifies the pin number
Config	pointer to a gpio_pin_config_t structure that contains the configuration information.

11.6.2 void GPIO_PinWrite (GPIO_Type * *base*, uint32_t *pin*, uint8_t *output*)

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.
output	GPIOpin output logic level. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

11.6.3 static void GPIO_WritePinOutput (GPIO_Type * *base*, uint32_t *pin*, uint8_t *output*) [inline], [static]

Deprecated Do not use this function. It has been superceded by GPIO_PinWrite.

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

Parameters

base	GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.)
mask	GPIO pin number macro

11.6.5 static void GPIO_SetPinsOutput (GPIO_Type * *base*, uint32_t *mask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by GPIO_PortSet.

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

Parameters

base	GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.)
mask	GPIO pin number macro

11.6.7 static void GPIO_ClearPinsOutput (GPIO_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by GPIO_PortClear.

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

Parameters

base	GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.)
mask	GPIO pin number macro

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

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.

Return values

GPIO	port input value.

11.6.10 static uint32_t GPIO_ReadPinInput (GPIO_Type ∗ *base*, uint32_t *pin*) [inline], [static]

Deprecated Do not use this function. It has been superceded by GPIO_PinRead.

11.6.11 static uint8_t GPIO_PinReadPadStatus (GPIO_Type * *base*, uint32_t *pin*) [inline], [static]

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.

Return values

GPIO	pin pad status value.

11.6.12 static uint8_t GPIO_ReadPadStatus (GPIO_Type * *base*, uint32_t *pin*) [inline], [static]

Deprecated Do not use this function. It has been superceded by GPIO_PinReadPadStatus.

11.6.13 void GPIO_PinSetInterruptConfig (GPIO_Type * *base*, uint32_t *pin*, gpio_interrupt_mode_t *pinInterruptMode*)

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.
pinInterrupt-Mode	pointer to a gpio_interrupt_mode_t structure that contains the interrupt mode information.

11.6.14 static void GPIO_SetPinInterruptConfig (GPIO_Type * *base*, uint32_t *pin*, gpio_interrupt_mode_t *pinInterruptMode*) [inline], [static]

Deprecated Do not use this function. It has been superceded by GPIO_PinSetInterruptConfig.

11.6.15 static void GPIO_PortEnableInterrupts (GPIO_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	GPIO base pointer.
mask	GPIO pin number macro.

11.6.16 static void GPIO_EnableInterrupts (GPIO_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO base pointer.
mask	GPIO pin number macro.

11.6.17 static void GPIO_PortDisableInterrupts (GPIO_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO base pointer.
mask	GPIO pin number macro.

11.6.18 static void GPIO_DisableInterrupts (GPIO_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by GPIO_PortDisableInterrupts.

11.6.19 static uint32_t GPIO_PortGetInterruptFlags (GPIO_Type ∗ *base*) [inline], [static]

Parameters

base	GPIO base pointer.

Return values

current	pin interrupt status flag.

11.6.20 static uint32_t GPIO_GetPinsInterruptFlags (GPIO_Type * *base*) [inline], [static]

Parameters

base	GPIO base pointer.

Return values

current	pin interrupt status flag.

11.6.21 static void GPIO_PortClearInterruptFlags (GPIO_Type * *base*, uint32_t *mask*) [inline], [static]

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

base	GPIO base pointer.
mask	GPIO pin number macro.

11.6.22 static void GPIO_ClearPinsInterruptFlags (GPIO_Type * *base*, uint32_t *mask*) [inline], [static]

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

base	GPIO base pointer.
mask	GPIO pin number macro.

Chapter 12
I2C: Inter-Integrated Circuit Driver

Overview

Modules

- I2C Driver

I2C Driver

12.2.1 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.

12.2.2 Typical use case

12.2.2.1 Master Operation in functional method

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

12.2.2.2 Master Operation in interrupt transactional method

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

12.2.2.3 Slave Operation in functional method

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

12.2.2.4 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_master_transfer_t
  I2C master transfer structure. More...
- struct i2c_master_handle_t
  I2C master handle structure. More...
- struct i2c_slave_config_t
  I2C slave user configuration. 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.

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_I2SR_RXAK_MASK,
  kI2C_IntPendingFlag = I2C_I2SR_IIF_MASK,
  kI2C_TransferDirectionFlag = I2C_I2SR_SRW_MASK,
  kI2C_ArbitrationLostFlag = I2C_I2SR_IAL_MASK,
  kI2C_BusBusyFlag = I2C_I2SR_IBB_MASK,
  kI2C_AddressMatchFlag = I2C_I2SR_IAAS_MASK,
  kI2C_TransferCompleteFlag = I2C_I2SR_ICF_MASK }

I2C peripheral flags.
- enum _i2c_interrupt_enable { kI2C_GlobalInterruptEnable = I2C_I2CR_IIEN_MASK }
  I2C feature interrupt source.
- enum i2c_direction_t {
  kI2C_Write = 0x0U,
  kI2C_Read = 0x1U }
  The direction of master and slave transfers.
- 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_SlaveCompletionEvent = 0x20U,
  kI2C_SlaveAllEvents }
  Set of events sent to the callback for nonblocking slave transfers.

Driver version

- #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 7))
  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_MasterDeinit (I2C_Type ∗base)
  De-initializes the I2C master peripheral.
- void I2C_MasterGetDefaultConfig (i2c_master_config_t ∗masterConfig)
  Sets the I2C master configuration structure to default values.
- void I2C_SlaveInit (I2C_Type ∗base, const i2c_slave_config_t ∗slaveConfig)
  Initializes the I2C peripheral.
- void I2C_SlaveDeinit (I2C_Type ∗base)
  De-initializes the I2C slave peripheral.
- 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

- static uint32_t I2C_MasterGetStatusFlags (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 uint32_t I2C_SlaveGetStatusFlags (I2C_Type ∗base)
  Gets the I2C status flags.
- 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.

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.

12.2.3 Data Structure Documentation

12.2.3.1 struct i2c_master_config_t

Data Fields

- bool enableMaster
  Enables the I2C peripheral at initialization time.
- uint32_t baudRate_Bps
  Baud rate configuration of I2C peripheral.

12.2.3.1.0.4 Field Documentation

12.2.3.1.0.4.1 bool i2c_master_config_t::enableMaster

12.2.3.1.0.4.2 uint32_t i2c_master_config_t::baudRate_Bps

12.2.3.2 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.

12.2.3.2.0.5 Field Documentation

12.2.3.2.0.5.1 uint32_t i2c_master_transfer_t::flags

12.2.3.2.0.5.2 uint8_t i2c_master_transfer_t::slaveAddress

12.2.3.2.0.5.3 i2c_direction_t i2c_master_transfer_t::direction

12.2.3.2.0.5.4 uint32_t i2c_master_transfer_t::subaddress

Transferred MSB first.

12.2.3.2.0.5.5 uint8_t i2c_master_transfer_t::subaddressSize

12.2.3.2.0.5.6 uint8_t* volatile i2c_master_transfer_t::data

12.2.3.2.0.5.7 volatile size_t i2c_master_transfer_t::dataSize

12.2.3.3 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.

12.2.3.3.0.6 Field Documentation

12.2.3.3.0.6.1 i2c_master_transfer_t i2c_master_handle_t::transfer

12.2.3.3.0.6.2 size_t i2c_master_handle_t::transferSize

12.2.3.3.0.6.3 uint8_t i2c_master_handle_t::state

12.2.3.3.0.6.4 i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback

12.2.3.3.0.6.5 void∗ i2c_master_handle_t::userData

12.2.3.4 struct i2c_slave_config_t

Data Fields

- bool enableSlave
  Enables the I2C peripheral at initialization time.
- uint16_t slaveAddress
  A slave address configuration.

12.2.3.4.0.7 Field Documentation

12.2.3.4.0.7.1 bool i2c_slave_config_t::enableSlave

12.2.3.4.0.7.2 uint16_t i2c_slave_config_t::slaveAddress

12.2.3.5 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.

12.2.3.5.0.8 Field Documentation

12.2.3.5.0.8.1 i2c_slave_transfer_event_t i2c_slave_transfer_t::event

12.2.3.5.0.8.2 uint8_t* volatile i2c_slave_transfer_t::data

12.2.3.5.0.8.3 volatile size_t i2c_slave_transfer_t::dataSize

12.2.3.5.0.8.4 status_t i2c_slave_transfer_t::completionStatus

Only applies for kI2C_SlaveCompletionEvent.

12.2.3.5.0.8.5 size_t i2c_slave_transfer_t::transferredCount

12.2.3.6 struct _i2c_slave_handle

I2C slave handle typedef.

Data Fields

- volatile uint8_t state
  A transfer state maintained during transfer.
- 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.

12.2.3.6.0.9 Field Documentation

12.2.3.6.0.9.1 volatile uint8_t i2c_slave_handle_t::state

12.2.3.6.0.9.2 i2c_slave_transfer_t i2c_slave_handle_t::transfer

12.2.3.6.0.9.3 uint32_t i2c_slave_handle_t::eventMask

12.2.3.6.0.9.4 i2c_slave_transfer_callback_t i2c_slave_handle_t::callback

12.2.3.6.0.9.5 void* i2c_slave_handle_t::userData

12.2.4 Macro Definition Documentation

12.2.4.1 #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 7))

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

12.2.5 Typedef Documentation

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

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

12.2.6 Enumeration Type Documentation

12.2.6.1 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.

12.2.6.2 enum _i2c_flags

The following status register flags can be cleared:

- kI2C_ArbitrationLostFlag

- kI2C_IntPendingFlag

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.
kI2C_TransferDirectionFlag I2C transfer direction flag.
kI2C_ArbitrationLostFlag I2C arbitration lost flag.
kI2C_BusBusyFlag I2C bus busy flag.
kI2C_AddressMatchFlag I2C address match flag.
kI2C_TransferCompleteFlag I2C transfer complete flag.

12.2.6.3 enum _i2c_interrupt_enable

Enumerator

kI2C_GlobalInterruptEnable I2C global interrupt.

12.2.6.4 enum i2c_direction_t

Enumerator

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

12.2.6.5 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.

12.2.6.6 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_SlaveCompletionEvent A stop was detected or finished transfer, completing the transfer.
kI2C_SlaveAllEvents A bit mask of all available events.

12.2.7 Function Documentation

12.2.7.1 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,
* .baudRate_Bps = 100000
* };
* 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

12.2.7.2 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

12.2.7.3 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_MasterInit(). Use the initialized structure unchanged in the I2C_MasterInit() or modify the structure before calling the I2C_MasterInit(). This is an example.

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

Parameters

masterConfig	A pointer to the master configuration structure.

12.2.7.4 void I2C_SlaveInit (I2C_Type ∗ *base*, const i2c_slave_config_t ∗ *slaveConfig*)

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,
* .slaveAddress = 0x1DU,
* };
* I2C_SlaveInit(I2C0, &config);
*
```

Parameters

base	I2C base pointer
slaveConfig	A pointer to the slave configuration structure

12.2.7.5 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

12.2.7.6 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_SlaveInit(). Modify fields of the structure before calling the I2C_SlaveInit(). This is an example.

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

Parameters

slaveConfig	A pointer to the slave configuration structure.

12.2.7.7 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.

12.2.7.8 static uint32_t I2C_MasterGetStatusFlags (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.

12.2.7.9 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_ArbitrationLostFlag • kI2C_IntPendingFlag

12.2.7.10 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.

12.2.7.11 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_IntPendingFlagFlag

12.2.7.12 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

12.2.7.13 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

12.2.7.14 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

12.2.7.15 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.

12.2.7.16 status_t I2C_MasterStop (I2C_Type * *base*)

Return values

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

12.2.7.17 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.

12.2.7.18 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.

12.2.7.19 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.

12.2.7.20 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.

12.2.7.21 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.

12.2.7.22 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_ArbitrationLost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.

12.2.7.23 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.

12.2.7.24 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_MasterGetTransferCount 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.

12.2.7.25 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.

12.2.7.26 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.

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

Parameters

base	I2C base pointer.
i2cHandle	pointer to i2c_master_handle_t structure.

12.2.7.28 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.

12.2.7.29 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.

12.2.7.30 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.

12.2.7.31 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.

12.2.7.32 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 13
PWM: Pulse Width Modulation Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the Pulse Width Modulation (PWM) module of MCUXpresso SDK devices.

PWM Driver

13.2.1 Initialization and deinitialization

The function PWM_Init() initializes the PWM with a specified configurations. The function PWM_GetDefaultConfig() gets the default configurations.The initialization function configures the PWM for the requested register update mode for registers with buffers.

The function PWM_Deinit() disables the PWM counter and turns off the module clock.

Typical use case

13.3.1 PWM output

Output PWM signal on PWM3 module with different dutycycles. Periodically update the PWM signal duty cycle. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/pwm

Enumerations

- enum pwm_clock_source_t {
  kPWM_PeripheralClock = 1U,
  kPWM_HighFrequencyClock,
  kPWM_LowFrequencyClock }

  PWM clock source select.
- enum pwm_fifo_water_mark_t {
  kPWM_FIFOWaterMark_1 = 0U,
  kPWM_FIFOWaterMark_2,
  kPWM_FIFOWaterMark_3,
  kPWM_FIFOWaterMark_4 }

  PWM FIFO water mark select.
- enum pwm_byte_data_swap_t {
  kPWM_ByteNoSwap = 0U,
  kPWM_ByteSwap }

  PWM byte data swap select.

- enum pwm_half_word_data_swap_t {
  kPWM_HalfWordNoSwap = 0U,
  kPWM_HalfWordSwap }
  PWM half-word data swap select.
- enum pwm_output_configuration_t {
  kPWM_SetAtRolloverAndClearAtcomparison = 0U,
  kPWM_ClearAtRolloverAndSetAtcomparison,
  kPWM_NoConfigure }
  PWM Output Configuration.
- enum pwm_sample_repeat_t {
  kPWM_EachSampleOnce = 0u,
  kPWM_EachSampletwice,
  kPWM_EachSampleFourTimes,
  kPWM_EachSampleEightTimes }
  PWM FIFO sample repeat It determines the number of times each sample from the FIFO is to be used.
- enum pwm_interrupt_enable_t {
  kPWM_FIFOEmptyInterruptEnable = (1U << 0),
  kPWM_RolloverInterruptEnable = (1U << 1),
  kPWM_CompareInterruptEnable = (1U << 2) }
  List of PWM interrupt options.
- enum pwm_status_flags_t {
  kPWM_FIFOEmptyFlag = (1U << 3),
  kPWM_RolloverFlag = (1U << 4),
  kPWM_CompareFlag = (1U << 5),
  kPWM_FIFOWriteErrorFlag }
  List of PWM status flags.
- enum pwm_fifo_available_t {
  kPWM_NoDataInFIFOFlag = 0U,
  kPWM_OneWordInFIFOFlag,
  kPWM_TwoWordsInFIFOFlag,
  kPWM_ThreeWordsInFIFOFlag,
  kPWM_FourWordsInFIFOFlag }
  List of PWM FIFO available.

Functions

- static void PWM_SoftwareReset (PWM_Type ∗base)
  Sofrware reset.
- static void PWM_SetPeriodValue (PWM_Type ∗base, uint32_t value)
  Sets the PWM period value.
- static uint32_t PWM_GetPeriodValue (PWM_Type ∗base)
  Gets the PWM period value.
- static uint32_t PWM_GetCounterValue (PWM_Type ∗base)
  Gets the PWM counter value.

Driver version

- #define FSL_PWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

Version 2.0.0.

Initialization and deinitialization

- status_t PWM_Init (PWM_Type ∗base, const pwm_config_t ∗config)
  Ungates the PWM clock and configures the peripheral for basic operation.
- void PWM_Deinit (PWM_Type ∗base)
  Gate the PWM submodule clock.
- void PWM_GetDefaultConfig (pwm_config_t ∗config)
  Fill in the PWM config struct with the default settings.

PWM start and stop.

- static void PWM_StartTimer (PWM_Type ∗base)
  Starts the PWM counter when the PWM is enabled.
- static void PWM_StopTimer (PWM_Type ∗base)
  Stops the PWM counter when the pwm is disabled.

Interrupt Interface

- static void PWM_EnableInterrupts (PWM_Type ∗base, uint32_t mask)
  Enables the selected PWM interrupts.
- static void PWM_DisableInterrupts (PWM_Type ∗base, uint32_t mask)
  Disables the selected PWM interrupts.
- static uint32_t PWM_GetEnabledInterrupts (PWM_Type ∗base)
  Gets the enabled PWM interrupts.

Status Interface

- static uint32_t PWM_GetStatusFlags (PWM_Type ∗base)
  Gets the PWM status flags.
- static void PWM_clearStatusFlags (PWM_Type ∗base, uint32_t mask)
  Clears the PWM status flags.
- static uint32_t PWM_GetFIFOAvailable (PWM_Type ∗base)
  Gets the PWM FIFO available.

Sample Interface

- static void PWM_SetSampleValue (PWM_Type ∗base, uint32_t value)
  Sets the PWM sample value.
- static uint32_t PWM_GetSampleValue (PWM_Type ∗base)
  Gets the PWM sample value.

Enumeration Type Documentation

13.4.1 enum pwm_clock_source_t

Enumerator

kPWM_PeripheralClock The Peripheral clock is used as the clock.

kPWM_HighFrequencyClock High-frequency reference clock is used as the clock.
kPWM_LowFrequencyClock Low-frequency reference clock(32KHz) is used as the clock.

13.4.2 enum pwm_fifo_water_mark_t

Sets the data level at which the FIFO empty flag will be set

Enumerator

kPWM_FIFOWaterMark_1 FIFO empty flag is set when there are more than or equal to 1 empty slots.
kPWM_FIFOWaterMark_2 FIFO empty flag is set when there are more than or equal to 2 empty slots.
kPWM_FIFOWaterMark_3 FIFO empty flag is set when there are more than or equal to 3 empty slots.
kPWM_FIFOWaterMark_4 FIFO empty flag is set when there are more than or equal to 4 empty slots.

13.4.3 enum pwm_byte_data_swap_t

It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

Enumerator

kPWM_ByteNoSwap byte ordering remains the same
kPWM_ByteSwap byte ordering is reversed

13.4.4 enum pwm_half_word_data_swap_t

Enumerator

kPWM_HalfWordNoSwap Half word swapping does not take place.
kPWM_HalfWordSwap Half word from write data bus are swapped.

13.4.5 enum pwm_output_configuration_t

Enumerator

kPWM_SetAtRolloverAndClearAtcomparison Output pin is set at rollover and cleared at comparison.

kPWM_ClearAtRolloverAndSetAtcomparison Output pin is cleared at rollover and set at comparison.
kPWM_NoConfigure PWM output is disconnected.

13.4.6 enum pwm_sample_repeat_t

Enumerator

kPWM_EachSampleOnce Use each sample once.
kPWM_EachSampletwice Use each sample twice.
kPWM_EachSampleFourTimes Use each sample four times.
kPWM_EachSampleEightTimes Use each sample eight times.

13.4.7 enum pwm_interrupt_enable_t

Enumerator

kPWM_FIFOEmptyInterruptEnable This bit controls the generation of the FIFO Empty interrupt.

kPWM_RolloverInterruptEnable This bit controls the generation of the Rollover interrupt.
kPWM_CompareInterruptEnable This bit controls the generation of the Compare interrupt.

13.4.8 enum pwm_status_flags_t

Enumerator

kPWM_FIFOEmptyFlag This bit indicates the FIFO data level in comparison to the water level set by FWM field in the control register.
kPWM_RolloverFlag This bit shows that a roll-over event has occurred.
kPWM_CompareFlag This bit shows that a compare event has occurred.
kPWM_FIFOWriteErrorFlag This bit shows that an attempt has been made to write FIFO when it is full.

13.4.9 enum pwm_fifo_available_t

Enumerator

kPWM_NoDataInFIFOFlag No data available.
kPWM_OneWordInFIFOFlag 1 word of data in FIFO
kPWM_TwoWordsInFIFOFlag 2 word of data in FIFO
kPWM_ThreeWordsInFIFOFlag 3 word of data in FIFO
kPWM_FourWordsInFIFOFlag 4 word of data in FIFO

Function Documentation

13.5.1 status_t PWM_Init (PWM_Type * *base,* const pwm_config_t * *config*)

Note

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

Parameters

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

Returns

kStatus_Success means success; else failed.

13.5.2 void PWM_Deinit (PWM_Type * *base*)

Parameters

base	PWM peripheral base address

13.5.3 void PWM_GetDefaultConfig (pwm_config_t * *config*)

The default values are:

```
*   config->enableStopMode = false;
*   config->enableDozeMode = false;
*   config->enableWaitMode = false;
*   config->enableDozeMode = false;
*   config->clockSource = kPWM_LowFrequencyClock;
*   config->prescale = 0U;
*   config->outputConfig = kPWM_SetAtRolloverAndClearAtcomparison;
*   config->fifoWater = kPWM_FIFOWaterMark_2;
*   config->sampleRepeat = kPWM_EachSampleOnce;
*   config->byteSwap = kPWM_ByteNoSwap;
*   config->halfWordSwap = kPWM_HalfWordNoSwap;
*
```

Parameters

config	Pointer to user's PWM config structure.

13.5.4 static void PWM_StartTimer (PWM_Type * *base*) [inline], [static]

When the PWM is enabled, it begins a new period, the output pin is set to start a new period while the prescaler and counter are released and counting begins.

Parameters

base	PWM peripheral base address

13.5.5 static void PWM_StopTimer (PWM_Type * *base*) [inline], [static]

Parameters

base	PWM peripheral base address

13.5.6 static void PWM_SoftwareReset (PWM_Type * *base*) [inline], [static]

PWM is reset when this bit is set to 1. It is a self clearing bit. Setting this bit resets all the registers to their reset values except for the STOPEN, DOZEN, WAITEN, and DBGEN bits in this control register.

Parameters

base	PWM peripheral base address

13.5.7 static void PWM_EnableInterrupts (PWM_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

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

13.5.8 static void PWM_DisableInterrupts (PWM_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	PWM peripheral base address
mask	The interrupts to disable. This is a logical OR of members of the enumeration pwm_interrupt_enable_t

13.5.9 static uint32_t PWM_GetEnabledInterrupts (PWM_Type * *base*) [inline], [static]

Parameters

base	PWM peripheral base address

Returns

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

13.5.10 static uint32_t PWM_GetStatusFlags (PWM_Type * *base*) [inline], [static]

Parameters

base	PWM peripheral base address

Returns

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

13.5.11 static void PWM_clearStatusFlags (PWM_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

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

13.5.12 static uint32_t PWM_GetFIFOAvailable (PWM_Type * *base*) [inline], [static]

Parameters

base	PWM peripheral base address

Returns

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

13.5.13 static void PWM_SetSampleValue (PWM_Type * *base*, uint32_t *value*) [inline], [static]

Parameters

base	PWM peripheral base address
value	The sample value. This is the input to the 4x16 FIFO. The value in this register denotes the value of the sample being currently used.

13.5.14 static uint32_t PWM_GetSampleValue (PWM_Type * *base*) [inline], [static]

Parameters

base	PWM peripheral base address

Returns

The sample value. It can be read only when the PWM is enable.

13.5.15 static void PWM_SetPeriodValue (PWM_Type * *base*, uint32_t *value*) [inline], [static]

Parameters

base	PWM peripheral base address
value	The period value. The PWM period register (PWM_PWMPR) determines the period of the PWM output signal. Writing 0xFFFF to this register will achieve the same result as writing 0xFFFE. PWMO (Hz) = PCLK(Hz) / (period +2)

13.5.16 static uint32_t PWM_GetPeriodValue (PWM_Type * *base*) [inline], [static]

Parameters

base	PWM peripheral base address

Returns

The period value. The PWM period register (PWM_PWMPR) determines the period of the PWM output signal.

13.5.17 static uint32_t PWM_GetCounterValue (PWM_Type * *base*) [inline], [static]

Parameters

base	PWM peripheral base address

Returns

The counter value. The current count value.

Chapter 14
UART: Universal Asynchronous Receiver/Transmitter Driver

Overview

Modules

- UART Driver
- UART FreeRTOS Driver

UART Driver

14.2.1 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 the purpose of optimization/customization. Using the functional API requires the knowledge of the UART peripheral and how to organize functional APIs to meet the application requirements. All functional APIs use the peripheral base address as the first parameter. UART functional operation groups provide the functional API set.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the uart_handle_t as the second parameter. Initialize the handle by calling the UART_TransferCreateHandle() API.

Transactional APIs support asynchronous transfer, which means that the functions UART_TransferSendNonBlocking() and UART_TransferReceiveNonBlocking() set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_UART_TxIdle and kStatus_UART_RxIdle.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size while calling the UART_TransferCreateHandle(). If passing NULL, the ring buffer feature is disabled. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The UART_TransferReceiveNonBlocking() function first gets data from the ring buffer. If the ring buffer does not have enough data, the function first returns the data in the ring buffer and then saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the kStatus_UART_RxIdle.

If the receive ring buffer is full, the upper layer is informed through a callback with the kStatus_UART_RxRingBufferOverrun. In the callback function, the upper layer reads data out from the ring buffer. If not, existing data is overwritten by the new data.

The ring buffer size is specified when creating the handle. Note that one byte is reserved for the ring buffer maintenance. When creating handle using the following code.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/uart In this example, the buffer size is 32, but only 31 bytes are used for saving data.

14.2.2 Typical use case

14.2.2.1 UART Send/receive using a polling method

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

14.2.2.2 UART Send/receive using an interrupt method

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

14.2.2.3 UART Receive using the ringbuffer feature

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

14.2.2.4 UART automatic baud rate detect feature

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_IUART, 0),
  kStatus_UART_RxBusy = MAKE_STATUS(kStatusGroup_IUART, 1),
  kStatus_UART_TxIdle = MAKE_STATUS(kStatusGroup_IUART, 2),
  kStatus_UART_RxIdle = MAKE_STATUS(kStatusGroup_IUART, 3),
  kStatus_UART_TxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_IUART, 4),
  kStatus_UART_RxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_IUART, 5),
  kStatus_UART_FlagCannotClearManually,
  kStatus_UART_Error = MAKE_STATUS(kStatusGroup_IUART, 7),
  kStatus_UART_RxRingBufferOverrun = MAKE_STATUS(kStatusGroup_IUART, 8),
  kStatus_UART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_IUART, 9),
  kStatus_UART_NoiseError = MAKE_STATUS(kStatusGroup_IUART, 10),
  kStatus_UART_FramingError = MAKE_STATUS(kStatusGroup_IUART, 11),
  kStatus_UART_ParityError = MAKE_STATUS(kStatusGroup_IUART, 12),
  kStatus_UART_BaudrateNotSupport,
  kStatus_UART_BreakDetect = MAKE_STATUS(kStatusGroup_IUART, 14),
  kStatus_UART_Timeout = MAKE_STATUS(kStatusGroup_IUART, 15) }

  Error codes for the UART driver.
- enum uart_data_bits_t {
  kUART_SevenDataBits = 0x0U,
  kUART_EightDataBits = 0x1U }

  UART data bits count.
- 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 = 0x0U,
  kUART_TwoStopBit = 0x1U }

  UART stop bit count.
- enum uart_idle_condition_t {
  kUART_IdleFor4Frames = 0x0U,
  kUART_IdleFor8Frames = 0x1U,
  kUART_IdleFor16Frames = 0x2U,
  kUART_IdleFor32Frames = 0x3U }

  UART idle condition detect.
- enum _uart_interrupt_enable

  This structure contains the settings for all of the UART interrupt configurations.
- enum {

kUART_RxCharReadyFlag = 0x0000000FU,
kUART_RxErrorFlag = 0x0000000EU,
kUART_RxOverrunErrorFlag = 0x0000000DU,
kUART_RxFrameErrorFlag = 0x0000000CU,
kUART_RxBreakDetectFlag = 0x0000000BU,
kUART_RxParityErrorFlag = 0x0000000AU,
kUART_ParityErrorFlag = 0x0094000FU,
kUART_RtsStatusFlag = 0x0094000EU,
kUART_TxReadyFlag = 0x0094000DU,
kUART_RtsDeltaFlag = 0x0094000CU,
kUART_EscapeFlag = 0x0094000BU,
kUART_FrameErrorFlag = 0x0094000AU,
kUART_RxReadyFlag = 0x00940009U,
kUART_AgingTimerFlag = 0x00940008U,
kUART_DtrDeltaFlag = 0x00940007U,
kUART_RxDsFlag = 0x00940006U,
kUART_tAirWakeFlag = 0x00940005U,
kUART_AwakeFlag = 0x00940004U,
kUART_Rs485SlaveAddrMatchFlag = 0x00940003U,
kUART_AutoBaudFlag = 0x0098000FU,
kUART_TxEmptyFlag = 0x0098000EU,
kUART_DtrFlag = 0x0098000DU,
kUART_IdleFlag = 0x0098000CU,
kUART_AutoBaudCntStopFlag = 0x0098000BU,
kUART_RiDeltaFlag = 0x0098000AU,
kUART_RiFlag = 0x00980009U,
kUART_IrFlag = 0x00980008U,
kUART_WakeFlag = 0x00980007U,
kUART_DcdDeltaFlag = 0x00980006U,
kUART_DcdFlag = 0x00980005U,
kUART_RtsFlag = 0x00980004U,
kUART_TxCompleteFlag = 0x00980003U,
kUART_BreakDetectFlag = 0x00980002U,
kUART_RxOverrunFlag = 0x00980001U,
kUART_RxDataReadyFlag = 0x00980000U }
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, 2, 0))
  UART driver version.

Software Reset

- static void UART_SoftwareReset (UART_Type *base)
  Resets the UART using software.

Initialization and deinitialization

- status_t UART_Init (UART_Type *base, const uart_config_t *config, uint32_t srcClock_Hz)
  Initializes an UART instance with the user configuration structure and the peripheral clock.
- void UART_Deinit (UART_Type *base)
  Deinitializes a UART instance.
- void UART_GetDefaultConfig (uart_config_t *config)
  l
- status_t UART_SetBaudRate (UART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the UART instance baud rate.
- static void UART_Enable (UART_Type *base)
  This function is used to Enable the UART Module.
- static void UART_SetIdleCondition (UART_Type *base, uart_idle_condition_t condition)
  This function is used to configure the IDLE line condition.
- static void UART_Disable (UART_Type *base)
  This function is used to Disable the UART Module.

Status

- bool UART_GetStatusFlag (UART_Type *base, uint32_t flag)
  This function is used to get the current status of specific UART status flag(including interrupt flag).
- void UART_ClearStatusFlag (UART_Type *base, uint32_t flag)
  This function is used to clear the current status of specific UART status flag.

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 enabled UART interrupts.

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 transmitter register.
- static uint8_t UART_ReadByte (UART_Type ∗base)
  Reads the receiver register.
- 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 written to the UART TX register.
- 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.
- void UART_TransferHandleIRQ (UART_Type ∗base, uart_handle_t ∗handle)
  UART IRQ handle function.

DMA control functions.

- 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 request.

FIFO control functions.

- static void UART_SetTxFifoWatermark (UART_Type ∗base, uint8_t watermark)
  This function is used to set the watermark of UART Tx FIFO.
- static void UART_SetRxRTSWatermark (UART_Type ∗base, uint8_t watermark)
  This function is used to set the watermark of UART RTS deassertion.
- static void UART_SetRxFifoWatermark (UART_Type ∗base, uint8_t watermark)
  This function is used to set the watermark of UART Rx FIFO.

Auto baud rate detection.

- static void UART_EnableAutoBaudRate (UART_Type ∗base, bool enable)
  This function is used to set the enable condition of Automatic Baud Rate Detection feature.
- static bool UART_IsAutoBaudRateComplete (UART_Type ∗base)
  This function is used to read if the automatic baud rate detection has finished.

14.2.3 Data Structure Documentation

14.2.3.1 struct uart_config_t

Data Fields

- uint32_t baudRate_Bps
  UART baud rate.
- uart_parity_mode_t parityMode
  Parity error check mode of this module.
- uart_data_bits_t dataBitsCount
  Data bits count, eight (default), seven.
- uart_stop_bit_count_t stopBitCount
  Number of stop bits in one frame.
- uint8_t txFifoWatermark
  TX FIFO watermark.
- uint8_t rxFifoWatermark
  RX FIFO watermark.
- uint8_t rxRTSWatermark
  RX RTS watermark, RX FIFO data count being larger than this triggers RTS deassertion.
- bool enableAutoBaudRate
  Enable automatic baud rate detection.
- bool enableTx

Enable TX.
- bool enableRx
  Enable RX.
- bool enableRxRTS
  RX RTS enable.
- bool enableTxCTS
  TX CTS enable.

14.2.3.1.0.10 Field Documentation

14.2.3.1.0.10.1 uint32_t uart_config_t::baudRate_Bps

14.2.3.1.0.10.2 uart_parity_mode_t uart_config_t::parityMode

14.2.3.1.0.10.3 uart_stop_bit_count_t uart_config_t::stopBitCount

14.2.3.2 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.

14.2.3.2.0.11 Field Documentation

14.2.3.2.0.11.1 uint8_t∗ uart_transfer_t::data

14.2.3.2.0.11.2 size_t uart_transfer_t::dataSize

14.2.3.3 struct _uart_handle

Forward declaration of the handle typedef.

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.

14.2.3.3.0.12 Field Documentation

14.2.3.3.0.12.1 uint8_t* volatile uart_handle_t::txData

14.2.3.3.0.12.2 volatile size_t uart_handle_t::txDataSize

14.2.3.3.0.12.3 size_t uart_handle_t::txDataSizeAll

14.2.3.3.0.12.4 uint8_t* volatile uart_handle_t::rxData

14.2.3.3.0.12.5 volatile size_t uart_handle_t::rxDataSize

14.2.3.3.0.12.6 size_t uart_handle_t::rxDataSizeAll

14.2.3.3.0.12.7 uint8_t* uart_handle_t::rxRingBuffer

14.2.3.3.0.12.8 size_t uart_handle_t::rxRingBufferSize

14.2.3.3.0.12.9 volatile uint16_t uart_handle_t::rxRingBufferHead

14.2.3.3.0.12.10 volatile uint16_t uart_handle_t::rxRingBufferTail

14.2.3.3.0.12.11 uart_transfer_callback_t uart_handle_t::callback

14.2.3.3.0.12.12 void* uart_handle_t::userData

14.2.3.3.0.12.13 volatile uint8_t uart_handle_t::txState

14.2.4 Macro Definition Documentation

14.2.4.1 #define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

14.2.4.2 #define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */

14.2.5 Typedef Documentation

14.2.5.1 typedef void(* uart_transfer_callback_t)(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)

14.2.6 Enumeration Type Documentation

14.2.6.1 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_BreakDetect Receiver detect BREAK signal.
kStatus_UART_Timeout UART times out.

14.2.6.2 enum uart_data_bits_t

Enumerator

kUART_SevenDataBits Seven data bit.
kUART_EightDataBits Eight data bit.

14.2.6.3 enum uart_parity_mode_t

Enumerator

kUART_ParityDisabled Parity disabled.
kUART_ParityEven Even error check is selected.
kUART_ParityOdd Odd error check is selected.

14.2.6.4 enum uart_stop_bit_count_t

Enumerator

kUART_OneStopBit One stop bit.
kUART_TwoStopBit Two stop bits.

14.2.6.5 enum uart_idle_condition_t

Enumerator

kUART_IdleFor4Frames Idle for more than 4 frames.

kUART_IdleFor8Frames Idle for more than 8 frames.
kUART_IdleFor16Frames Idle for more than 16 frames.
kUART_IdleFor32Frames Idle for more than 32 frames.

14.2.6.6 enum _uart_interrupt_enable

14.2.6.7 anonymous enum

This provides constants for the UART status flags for use in the UART functions.

Enumerator

kUART_RxCharReadyFlag Rx Character Ready Flag.
kUART_RxErrorFlag Rx Error Detect Flag.
kUART_RxOverrunErrorFlag Rx Overrun Flag.
kUART_RxFrameErrorFlag Rx Frame Error Flag.
kUART_RxBreakDetectFlag Rx Break Detect Flag.
kUART_RxParityErrorFlag Rx Parity Error Flag.
kUART_ParityErrorFlag Parity Error Interrupt Flag.
kUART_RtsStatusFlag RTS_B Pin Status Flag.
kUART_TxReadyFlag Transmitter Ready Interrupt/DMA Flag.
kUART_RtsDeltaFlag RTS Delta Flag.
kUART_EscapeFlag Escape Sequence Interrupt Flag.
kUART_FrameErrorFlag Frame Error Interrupt Flag.
kUART_RxReadyFlag Receiver Ready Interrupt/DMA Flag.
kUART_AgingTimerFlag Aging Timer Interrupt Flag.
kUART_DtrDeltaFlag DTR Delta Flag.
kUART_RxDsFlag Receiver IDLE Interrupt Flag.
kUART_tAirWakeFlag Asynchronous IR WAKE Interrupt Flag.
kUART_AwakeFlag Asynchronous WAKE Interrupt Flag.
kUART_Rs485SlaveAddrMatchFlag RS-485 Slave Address Detected Interrupt Flag.
kUART_AutoBaudFlag Automatic Baud Rate Detect Complete Flag.
kUART_TxEmptyFlag Transmit Buffer FIFO Empty.
kUART_DtrFlag DTR edge triggered interrupt flag.
kUART_IdleFlag Idle Condition Flag.
kUART_AutoBaudCntStopFlag Auto-baud Counter Stopped Flag.
kUART_RiDeltaFlag Ring Indicator Delta Flag.
kUART_RiFlag Ring Indicator Input Flag.
kUART_IrFlag Serial Infrared Interrupt Flag.
kUART_WakeFlag Wake Flag.
kUART_DcdDeltaFlag Data Carrier Detect Delta Flag.
kUART_DcdFlag Data Carrier Detect Input Flag.
kUART_RtsFlag RTS Edge Triggered Interrupt Flag.
kUART_TxCompleteFlag Transmitter Complete Flag.

kUART_BreakDetectFlag BREAK Condition Detected Flag.
kUART_RxOverrunFlag Overrun Error Flag.
kUART_RxDataReadyFlag Receive Data Ready Flag.

14.2.7 Function Documentation

14.2.7.1 uint32_t UART_GetInstance (UART_Type ∗ *base*)

Parameters

base	UART peripheral base address.

Returns

UART instance.

14.2.7.2 static void UART_SoftwareReset (UART_Type ∗ *base*) [inline], [static]

This function resets the transmit and receive state machines, all FIFOs and register USR1, USR2, UBIR, UBMR, UBRC , URXD, UTXD and UTS[6-3]

Parameters

base	UART peripheral base address.

14.2.7.3 status_t UART_Init (UART_Type ∗ *base*, const uart_config_t ∗ *config*, uint32_t *srcClock_Hz*)

This function configures the UART module with user-defined settings. Call the UART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the UART.

```
*  uart_config_t uartConfig;
*  uartConfig.baudRate_Bps = 115200U;
*  uartConfig.parityMode = kUART_ParityDisabled;
*  uartConfig.dataBitsCount = kUART_EightDataBits;
*  uartConfig.stopBitCount = kUART_OneStopBit;
*  uartConfig.txFifoWatermark = 2;
*  uartConfig.rxFifoWatermark = 1;
*  uartConfig.enableAutoBaudrate = false;
*  uartConfig.enableTx = true;
*  uartConfig.enableRx = true;
*  UART_Init(UART1, &uartConfig, 24000000U);
*
```

Parameters

base	UART peripheral base address.
config	Pointer to a user-defined configuration structure.
srcClock_Hz	UART clock source frequency in HZ.

Return values

kStatus_Success	UART initialize succeed

14.2.7.4 void UART_Deinit (UART_Type ∗ *base*)

This function waits for transmit to complete, disables TX and RX, and disables the UART clock.

Parameters

base	UART peripheral base address.

14.2.7.5 void UART_GetDefaultConfig (uart_config_t ∗ *config*)

Gets the default configuration structure.

This function initializes the UART configuration structure to a default value. The default values are-: uartConfig->baudRate_Bps = 115200U; uartConfig->parityMode = kUART_ParityDisabled; uartConfig->dataBitsCount = kUART_EightDataBits; uartConfig->stopBitCount = kUART_OneStopBit; uartConfig->txFifoWatermark = 2; uartConfig->rxFifoWatermark = 1; uartConfig->enableAutoBaudrate = flase; uartConfig->enableTx = false; uartConfig->enableRx = false;

Parameters

config	Pointer to a configuration structure.

14.2.7.6 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.

14.2.7.7 static void UART_Enable (UART_Type ∗ *base*) [inline], [static]

Parameters

base	UART base pointer.

14.2.7.8 static void UART_SetIdleCondition (UART_Type ∗ *base*, uart_idle_condition_t *condition*) [inline], [static]

Parameters

base	UART base pointer.
condition	IDLE line detect condition of the enumerators in uart_idle_condition_t.

14.2.7.9 static void UART_Disable (UART_Type ∗ *base*) [inline], [static]

Parameters

base	UART base pointer.

14.2.7.10 bool UART_GetStatusFlag (UART_Type ∗ *base*, uint32_t *flag*)

The available status flag can be select from uart_status_flag_t enumeration.

Parameters

base	UART base pointer.
flag	Status flag to check.

Return values

current	state of corresponding status flag.

14.2.7.11 void UART_ClearStatusFlag (UART_Type * *base*, uint32_t *flag*)

The available status flag can be select from uart_status_flag_t enumeration.

Parameters

base	UART base pointer.
flag	Status flag to clear.

14.2.7.12 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 data ready interrupt, do the following.

```
*     UART_EnableInterrupts(UART1,kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
*
```

Parameters

base	UART peripheral base address.
mask	The interrupts to enable. Logical OR of _uart_interrupt_enable.

14.2.7.13 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 data ready interrupt do the following.

```
*     UART_EnableInterrupts(UART1,kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
*
```

Parameters

base	UART peripheral base address.
mask	The interrupts to disable. Logical OR of _uart_interrupt_enable.

14.2.7.14 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 interrupt enable status, compare the return value with enumerators in _uart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
*     uint32_t enabledInterrupts = UART_GetEnabledInterrupts(UART1);
*
*     if (kUART_TxEmptyEnable & enabledInterrupts)
*     {
*         ...
*     }
*
```

Parameters

base	UART peripheral base address.

Returns

UART interrupt flags which are logical OR of the enumerators in _uart_interrupt_enable.

14.2.7.15 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.

14.2.7.16 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.

14.2.7.17 static void UART_WriteByte (UART_Type * *base*, uint8_t *data*) [inline], [static]

This function is used to write data to transmitter register. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

base	UART peripheral base address.
data	Data write to the TX register.

14.2.7.18 static uint8_t UART_ReadByte (UART_Type * *base*) [inline], [static]

This function is used to read data from receiver register. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

base	UART peripheral base address.

Returns

Data read from data register.

14.2.7.19 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.

14.2.7.20 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.

14.2.7.21 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.

14.2.7.22 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.

14.2.7.23 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.

14.2.7.24 size_t UART_TransferGetRxRingBufferLength (uart_handle_t * *handle*)

Parameters

handle	UART handle pointer.

Returns

Length of received data in RX ring buffer.

14.2.7.25 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.

14.2.7.26 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.

14.2.7.27 status_t UART_TransferGetSendCount (UART_Type ∗ *base*, uart_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of bytes written to the UART TX register 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`;

14.2.7.28 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.

14.2.7.29 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.

14.2.7.30 status_t UART_TransferGetReceiveCount (UART_Type ∗ *base*, uart_handle_t ∗ *handle*, uint32_t ∗ *count*)

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

Parameters

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

Return values

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

14.2.7.31 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.

14.2.7.32 static void UART_EnableTxDMA (UART_Type ∗ *base*, bool *enable*) [inline], [static]

This function enables or disables the transmit request when the transmitter has one or more slots available in the TxFIFO. The fill level in the TxFIFO that generates the DMA request is controlled by the TXTL bits.

Parameters

base	UART peripheral base address.
enable	True to enable, false to disable.

14.2.7.33 static void UART_EnableRxDMA (UART_Type ∗ *base*, bool *enable*) [inline], [static]

This function enables or disables the receive request when the receiver has data in the RxFIFO. The fill level in the RxFIFO at which a DMA request is generated is controlled by the RXTL bits .

Parameters

base	UART peripheral base address.
enable	True to enable, false to disable.

14.2.7.34 static void UART_SetTxFifoWatermark (UART_Type ∗ *base*, uint8_t *watermark*) [inline], [static]

```
A maskable interrupt is generated whenever the data level in
the TxFIFO falls below the Tx FIFO watermark.
```

Parameters

base	UART base pointer.
watermark	The Tx FIFO watermark.

14.2.7.35 static void UART_SetRxRTSWatermark (UART_Type ∗ *base*, uint8_t *watermark*) [inline], [static]

The RTS signal deasserts whenever the data count in RxFIFO reaches the Rx RTS watermark.

Parameters

base	UART base pointer.
watermark	The Rx RTS watermark.

14.2.7.36 static void UART_SetRxFifoWatermark (UART_Type ∗ *base*, uint8_t *watermark*) [inline], [static]

```
A maskable interrupt is generated whenever the data level in
the RxFIFO reaches the Rx FIFO watermark.
```

Parameters

base	UART base pointer.

watermark	The Rx FIFO watermark.

14.2.7.37 static void UART_EnableAutoBaudRate (UART_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	UART base pointer.
enable	Enable/Disable Automatic Baud Rate Detection feature. • true: Enable Automatic Baud Rate Detection feature. • false: Disable Automatic Baud Rate Detection feature.

14.2.7.38 static bool UART_IsAutoBaudRateComplete (UART_Type * *base*) [inline], [static]

Parameters

base	UART base pointer.

Returns

- true: Automatic baud rate detection has finished.
  - false: Automatic baud rate detection has not finished.

UART FreeRTOS Driver

14.3.1 Overview

Data Structures

- struct uart_rtos_config_t
  UART configuration structure. More...

Driver version

- #define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  UART FreeRTOS driver version 2.1.1.

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.

14.3.2 Data Structure Documentation

14.3.2.1 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.

14.3.3 Macro Definition Documentation

14.3.3.1 #define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

14.3.4 Function Documentation

14.3.4.1 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.

14.3.4.2 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.

14.3.4.3 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.

14.3.4.4 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 15
MU: Messaging Unit

Overview

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

Function description

The MU driver provides these functions:

- Functions to initialize the MU module.
- Functions to send and receive messages.
- Functions for MU flags for both MU sides.
- Functions for status flags and interrupts.
- Other miscellaneous functions.

15.2.1 MU initialization

The function MU_Init() initializes the MU module and enables the MU clock. It should be called before any other MU functions.

The function MU_Deinit() deinitializes the MU module and disables the MU clock. No MU functions can be called after this function.

15.2.2 MU message

The MU message must be sent when the transmit register is empty. The MU driver provides blocking API and non-blocking API to send message.

The MU_SendMsgNonBlocking() function writes a message to the MU transmit register without checking the transmit register status. The upper layer should check that the transmit register is empty before calling this function. This function can be used in the ISR for better performance.

The MU_SendMsg() function is a blocking function. It waits until the transmit register is empty and sends the message.

Correspondingly, there are blocking and non-blocking APIs for receiving a message. The MU_ReadMsg-NonBlocking() function is a non-blocking API. The MU_ReadMsg() function is the blocking API.

15.2.3 MU flags

The MU driver provides 3-bit general purpose flags. When the flags are set on one side, they are reflected on the other side.

The MU flags must be set when the previous flags have been updated to the other side. The MU driver provides a non-blocking function and a blocking function. The blocking function MU_SetFlags() waits until previous flags have been updated to the other side and then sets flags. The non-blocking function sets the flags directly. Ensure that the kMU_FlagsUpdatingFlag is not pending before calling this function.

The function MU_GetFlags() gets the MU flags on the current side.

15.2.4 Status and interrupt

The function MU_GetStatusFlags() returns all MU status flags. Use the _mu_status_flags to check for specific flags, for example, to check RX0 and RX1 register full, use the following code:

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/mu The receive full flags are cleared automatically after messages are read out. The transmit empty flags are cleared automatically after new messages are written to the transmit register. The general purpose interrupt flags must be cleared manually using the function MU_ClearStatusFlags().

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/mu To enable or disable a specific interrupt, use MU_EnableInterrupts() and MU_DisableInterrupts() functions. The interrupts to enable or disable should be passed in as a bit mask of the _mu_interrupt_enable.

The MU_TriggerInterrupts() function triggers general purpose interrupts and NMI to the other core. The interrupts to trigger are passed in as a bit mask of the _mu_interrupt_trigger. If previously triggered interrupts have not been processed by the other side, this function returns an error.

15.2.5 MU misc functions

The MU_BootCoreB() and MU_HoldCoreBReset() functions should only be used from A side. They are used to boot the core B or to hold core B in reset.

The MU_ResetBothSides() function resets MU at both A and B sides. However, only the A side can call this function.

If a core enters stop mode, the platform clock of this core is disabled by default. The function MU_SetClockOnOtherCoreEnable() forces the other core's platform clock to remain enabled even after that core has entered a stop mode. In this case, the other core's platform clock keeps running until the current core enters stop mode too.

Function MU_GetOtherCorePowerMode() gets the power mode of the other core.

Enumerations

- enum _mu_status_flags {
  kMU_Tx0EmptyFlag = (1U << (MU_SR_TEn_SHIFT + 3U)),
  kMU_Tx1EmptyFlag = (1U << (MU_SR_TEn_SHIFT + 2U)),
  kMU_Tx2EmptyFlag = (1U << (MU_SR_TEn_SHIFT + 1U)),
  kMU_Tx3EmptyFlag = (1U << (MU_SR_TEn_SHIFT + 0U)),
  kMU_Rx0FullFlag = (1U << (MU_SR_RFn_SHIFT + 3U)),
  kMU_Rx1FullFlag = (1U << (MU_SR_RFn_SHIFT + 2U)),
  kMU_Rx2FullFlag = (1U << (MU_SR_RFn_SHIFT + 1U)),
  kMU_Rx3FullFlag = (1U << (MU_SR_RFn_SHIFT + 0U)),
  kMU_GenInt0Flag = (1U << (MU_SR_GIPn_SHIFT + 3U)),
  kMU_GenInt1Flag = (1U << (MU_SR_GIPn_SHIFT + 2U)),
  kMU_GenInt2Flag = (1U << (MU_SR_GIPn_SHIFT + 1U)),
  kMU_GenInt3Flag = (1U << (MU_SR_GIPn_SHIFT + 0U)),
  kMU_EventPendingFlag = MU_SR_EP_MASK,
  kMU_FlagsUpdatingFlag = MU_SR_FUP_MASK,
  kMU_OtherSideInResetFlag = MU_SR_RS_MASK }
  MU status flags.
- enum _mu_interrupt_enable {
  kMU_Tx0EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 3U)),
  kMU_Tx1EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 2U)),
  kMU_Tx2EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 1U)),
  kMU_Tx3EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 0U)),
  kMU_Rx0FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 3U)),
  kMU_Rx1FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 2U)),
  kMU_Rx2FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 1U)),
  kMU_Rx3FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 0U)),
  kMU_GenInt0InterruptEnable = (int)(1U << (MU_CR_GIEn_SHIFT + 3U)),
  kMU_GenInt1InterruptEnable = (1U << (MU_CR_GIEn_SHIFT + 2U)),
  kMU_GenInt2InterruptEnable = (1U << (MU_CR_GIEn_SHIFT + 1U)),
  kMU_GenInt3InterruptEnable = (1U << (MU_CR_GIEn_SHIFT + 0U)) }
  MU interrupt source to enable.
- enum _mu_interrupt_trigger {
  kMU_GenInt0InterruptTrigger = (1U << (MU_CR_GIRn_SHIFT + 3U)),
  kMU_GenInt1InterruptTrigger = (1U << (MU_CR_GIRn_SHIFT + 2U)),
  kMU_GenInt2InterruptTrigger = (1U << (MU_CR_GIRn_SHIFT + 1U)),
  kMU_GenInt3InterruptTrigger = (1U << (MU_CR_GIRn_SHIFT + 0U)) }
  MU interrupt that could be triggered to the other core.

Driver version

- #define FSL_MU_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))
  MU driver version.

MU initialization.

- void MU_Init (MU_Type *base)
  Initializes the MU module.
- void MU_Deinit (MU_Type *base)
  De-initializes the MU module.

MU Message

- static void MU_SendMsgNonBlocking (MU_Type *base, uint32_t regIndex, uint32_t msg)
  Writes a message to the TX register.
- void MU_SendMsg (MU_Type *base, uint32_t regIndex, uint32_t msg)
  Blocks to send a message.
- static uint32_t MU_ReceiveMsgNonBlocking (MU_Type *base, uint32_t regIndex)
  Reads a message from the RX register.
- uint32_t MU_ReceiveMsg (MU_Type *base, uint32_t regIndex)
  Blocks to receive a message.

MU Flags

- static void MU_SetFlagsNonBlocking (MU_Type *base, uint32_t flags)
  Sets the 3-bit MU flags reflect on the other MU side.
- void MU_SetFlags (MU_Type *base, uint32_t flags)
  Blocks setting the 3-bit MU flags reflect on the other MU side.
- static uint32_t MU_GetFlags (MU_Type *base)
  Gets the current value of the 3-bit MU flags set by the other side.

Status and Interrupt.

- static uint32_t MU_GetStatusFlags (MU_Type *base)
  Gets the MU status flags.
- static uint32_t MU_GetInterruptsPending (MU_Type *base)
  Gets the MU IRQ pending status.
- static void MU_ClearStatusFlags (MU_Type *base, uint32_t mask)
  Clears the specific MU status flags.
- static void MU_EnableInterrupts (MU_Type *base, uint32_t mask)
  Enables the specific MU interrupts.
- static void MU_DisableInterrupts (MU_Type *base, uint32_t mask)
  Disables the specific MU interrupts.
- status_t MU_TriggerInterrupts (MU_Type *base, uint32_t mask)
  Triggers interrupts to the other core.

MU misc functions

- static void MU_MaskHardwareReset (MU_Type *base, bool mask)
  Mask hardware reset by the other core.

Macro Definition Documentation

15.3.1 #define FSL_MU_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))

Enumeration Type Documentation

15.4.1 enum _mu_status_flags

Enumerator

kMU_Tx0EmptyFlag TX0 empty.
kMU_Tx1EmptyFlag TX1 empty.
kMU_Tx2EmptyFlag TX2 empty.
kMU_Tx3EmptyFlag TX3 empty.
kMU_Rx0FullFlag RX0 full.
kMU_Rx1FullFlag RX1 full.
kMU_Rx2FullFlag RX2 full.
kMU_Rx3FullFlag RX3 full.
kMU_GenInt0Flag General purpose interrupt 0 pending.
kMU_GenInt1Flag General purpose interrupt 0 pending.
kMU_GenInt2Flag General purpose interrupt 0 pending.
kMU_GenInt3Flag General purpose interrupt 0 pending.
kMU_EventPendingFlag MU event pending.
kMU_FlagsUpdatingFlag MU flags update is on-going.
kMU_OtherSideInResetFlag The other side is in reset.

15.4.2 enum _mu_interrupt_enable

Enumerator

kMU_Tx0EmptyInterruptEnable TX0 empty.
kMU_Tx1EmptyInterruptEnable TX1 empty.
kMU_Tx2EmptyInterruptEnable TX2 empty.
kMU_Tx3EmptyInterruptEnable TX3 empty.
kMU_Rx0FullInterruptEnable RX0 full.
kMU_Rx1FullInterruptEnable RX1 full.
kMU_Rx2FullInterruptEnable RX2 full.
kMU_Rx3FullInterruptEnable RX3 full.
kMU_GenInt0InterruptEnable General purpose interrupt 0.
kMU_GenInt1InterruptEnable General purpose interrupt 1.
kMU_GenInt2InterruptEnable General purpose interrupt 2.
kMU_GenInt3InterruptEnable General purpose interrupt 3.

15.4.3 enum _mu_interrupt_trigger

Enumerator

kMU_GenInt0InterruptTrigger General purpose interrupt 0.

kMU_GenInt1InterruptTrigger	General purpose interrupt 1.
kMU_GenInt2InterruptTrigger	General purpose interrupt 2.
kMU_GenInt3InterruptTrigger	General purpose interrupt 3.

Function Documentation

15.5.1 void MU_Init (MU_Type * *base*)

This function enables the MU clock only.

Parameters

base	MU peripheral base address.

15.5.2 void MU_Deinit (MU_Type * *base*)

This function disables the MU clock only.

Parameters

base	MU peripheral base address.

15.5.3 static void MU_SendMsgNonBlocking (MU_Type * *base,* uint32_t *regIndex,* uint32_t *msg*) [inline], [static]

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
* while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { }  Wait for TX0
      register empty.
* MU_SendMsgNonBlocking(base, 0U, MSG_VAL);  Write message to the TX0 register.
*
```

Parameters

base	MU peripheral base address.

regIndex	TX register index.
msg	Message to send.

15.5.4 void MU_SendMsg (MU_Type * *base*, uint32_t *regIndex*, uint32_t *msg*)

This function waits until the TX register is empty and sends the message.

Parameters

base	MU peripheral base address.
regIndex	TX register index.
msg	Message to send.

15.5.5 static uint32_t MU_ReceiveMsgNonBlocking (MU_Type * *base*, uint32_t *regIndex*) [inline], [static]

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
* uint32_t msg;
* while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
* {
* }  Wait for the RX0 register full.
*
* msg = MU_ReceiveMsgNonBlocking(base, 0U);  Read message from RX0 register.
*
```

Parameters

base	MU peripheral base address.
regIndex	TX register index.

Returns

The received message.

15.5.6 uint32_t MU_ReceiveMsg (MU_Type * *base*, uint32_t *regIndex*)

This function waits until the RX register is full and receives the message.

Parameters

base	MU peripheral base address.
regIndex	RX register index.

Returns

The received message.

15.5.7 static void MU_SetFlagsNonBlocking (MU_Type * *base,* uint32_t *flags*) [inline], [static]

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag `kMU_FlagsUpdatingFlag` asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag `kMU_FlagsUpdatingFlag` is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag `kMU_FlagsUpdatingFlag` is cleared before calling this function.

```
* while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
* {
* }  Wait for previous MU flags updating.
*
* MU_SetFlagsNonBlocking(base, 0U);  Set the mU flags.
*
```

Parameters

base	MU peripheral base address.
flags	The 3-bit MU flags to set.

15.5.8 void MU_SetFlags (MU_Type * *base,* uint32_t *flags*)

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag `kMU_FlagsUpdatingFlag` asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag `kMU_FlagsUpdatingFlag` is cleared by hardware. During the flags updating period, the flags cannot be changed. This function waits for the MU status flag `kMU_FlagsUpdatingFlag` cleared and sets the 3-bit MU flags.

Parameters

base	MU peripheral base address.
flags	The 3-bit MU flags to set.

15.5.9 static uint32_t MU_GetFlags (MU_Type ∗ *base*) [inline], [static]

This function gets the current 3-bit MU flags on the current side.

Parameters

base	MU peripheral base address.

Returns

flags Current value of the 3-bit flags.

15.5.10 static uint32_t MU_GetStatusFlags (MU_Type ∗ *base*) [inline], [static]

This function returns the bit mask of the MU status flags. See _mu_status_flags.

```
* uint32_t flags;
* flags = MU_GetStatusFlags(base);  Get all status flags.
* if (kMU_Tx0EmptyFlag & flags)
* {
*     The TX0 register is empty. Message can be sent.
*     MU_SendMsgNonBlocking(base, 0U, MSG0_VAL);
* }
* if (kMU_Tx1EmptyFlag & flags)
* {
*     The TX1 register is empty. Message can be sent.
*     MU_SendMsgNonBlocking(base, 1U, MSG1_VAL);
* }
*
```

Parameters

base	MU peripheral base address.

Returns

Bit mask of the MU status flags, see _mu_status_flags.

15.5.11 static uint32_t MU_GetInterruptsPending (MU_Type * *base*) [inline], [static]

This function returns the bit mask of the pending MU IRQs.

Parameters

base	MU peripheral base address.

Returns

Bit mask of the MU IRQs pending.

15.5.12 static void MU_ClearStatusFlags (MU_Type ∗ *base,* uint32_t *mask*) [inline], [static]

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See _mu_status_flags.

```
* Clear general interrupt 0 and general interrupt 1 pending flags.
* MU_ClearStatusFlags(base, kMU_GenInt0Flag |
      kMU_GenInt1Flag);
*
```

Parameters

base	MU peripheral base address.
mask	Bit mask of the MU status flags. See _mu_status_flags. The following flags are cleared by hardware, this function could not clear them. • kMU_Tx0EmptyFlag • kMU_Tx1EmptyFlag • kMU_Tx2EmptyFlag • kMU_Tx3EmptyFlag • kMU_Rx0FullFlag • kMU_Rx1FullFlag • kMU_Rx2FullFlag • kMU_Rx3FullFlag • kMU_EventPendingFlag • kMU_FlagsUpdatingFlag • kMU_OtherSideInResetFlag

15.5.13 static void MU_EnableInterrupts (MU_Type ∗ *base,* uint32_t *mask*) [inline], [static]

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See _mu_interrupt_enable.

```
*    Enable general interrupt 0 and TX0 empty interrupt.
* MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable |
      kMU_Tx0EmptyInterruptEnable);
*
```

Parameters

base	MU peripheral base address.
mask	Bit mask of the MU interrupts. See _mu_interrupt_enable.

15.5.14 static void MU_DisableInterrupts (MU_Type ∗ *base,* uint32_t *mask*) [inline], [static]

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See _mu_interrupt_enable.

```
*    Disable general interrupt 0 and TX0 empty interrupt.
* MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable |
      kMU_Tx0EmptyInterruptEnable);
*
```

Parameters

base	MU peripheral base address.
mask	Bit mask of the MU interrupts. See _mu_interrupt_enable.

15.5.15 status_t MU_TriggerInterrupts (MU_Type ∗ *base,* uint32_t *mask*)

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See _mu_interrupt_trigger. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
* if (kStatus_Success != MU_TriggerInterrupts(base,
      kMU_GenInt0InterruptTrigger |
      kMU_GenInt2InterruptTrigger))
* {
*       Previous general purpose interrupt 0 or general purpose interrupt 2
*       has not been processed by the other core.
* }
*
```

Parameters

base	MU peripheral base address.
mask	Bit mask of the interrupts to trigger. See _mu_interrupt_trigger.

Return values

kStatus_Success	Interrupts have been triggered successfully.
kStatus_Fail	Previous interrupts have not been accepted.

15.5.16 static void MU_MaskHardwareReset (MU_Type * *base*, bool *mask*) [inline], [static]

The other core could call MU_HardwareResetOtherCore() to reset current core. To mask the reset, call this function and pass in true.

Parameters

base	MU peripheral base address.
mask	Pass true to mask the hardware reset, pass false to unmask it.

Chapter 16
QSPI: Quad Serial Peripheral Interface

Overview

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

QSPI driver includes functional APIs and EDMA transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for QSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the QSPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. QSPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the qspi_handle_t as the first parameter. Initialize the handle by calling the QSPI_TransferTxCreateHandleEDMA() or QSPI_TransferRxCreateHandleEDMA() API.

Modules

- Quad Serial Peripheral Interface Driver

Quad Serial Peripheral Interface Driver

16.2.1 Overview

Data Structures

- struct qspi_dqs_config_t
  DQS configure features. More...
- struct qspi_flash_timing_t
  Flash timing configuration. More...
- struct qspi_config_t
  QSPI configuration structure. More...
- struct qspi_flash_config_t
  External flash configuration items. More...
- struct qspi_transfer_t
  Transfer structure for QSPI. More...
- struct ip_command_config_t
  16-bit access reg for IPCR register More...

Macros

- #define QSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
  Macro functions for LUT table.
- #define QSPI_CMD (0x1U)
  Macro for QSPI LUT command.
- #define QSPI_PAD_1 (0x0U)
  Macro for QSPI PAD.

Enumerations

- enum {
  kStatus_QSPI_Idle = MAKE_STATUS(kStatusGroup_QSPI, 0),
  kStatus_QSPI_Busy = MAKE_STATUS(kStatusGroup_QSPI, 1),
  kStatus_QSPI_Error = MAKE_STATUS(kStatusGroup_QSPI, 2) }
  Status structure of QSPI.
- enum qspi_read_area_t {
  kQSPI_ReadAHB = 0x0U,
  kQSPI_ReadIP }
  QSPI read data area, from IP FIFO or AHB buffer.
- enum qspi_command_seq_t {
  kQSPI_IPSeq = QuadSPI_SPTRCLR_IPPTRC_MASK,
  kQSPI_BufferSeq = QuadSPI_SPTRCLR_BFPTRC_MASK }
  QSPI command sequence type.
- enum qspi_fifo_t {
  kQSPI_TxFifo = QuadSPI_MCR_CLR_TXF_MASK,
  kQSPI_RxFifo = QuadSPI_MCR_CLR_RXF_MASK,
  kQSPI_AllFifo = QuadSPI_MCR_CLR_TXF_MASK | QuadSPI_MCR_CLR_RXF_MASK }

QSPI buffer type.

- enum qspi_endianness_t {
  kQSPI_64BigEndian = 0x0U,
  kQSPI_32LittleEndian,
  kQSPI_32BigEndian,
  kQSPI_64LittleEndian }

  QSPI transfer endianess.
- enum _qspi_error_flags {
  kQSPI_DataLearningFail = (int)QuadSPI_FR_DLPFF_MASK,
  kQSPI_TxBufferFill = QuadSPI_FR_TBFF_MASK,
  kQSPI_TxBufferUnderrun = QuadSPI_FR_TBUF_MASK,
  kQSPI_IllegalInstruction = QuadSPI_FR_ILLINE_MASK,
  kQSPI_RxBufferOverflow = QuadSPI_FR_RBOF_MASK,
  kQSPI_RxBufferDrain = QuadSPI_FR_RBDF_MASK,
  kQSPI_AHBSequenceError = QuadSPI_FR_ABSEF_MASK,
  kQSPI_AHBBufferOverflow = QuadSPI_FR_ABOF_MASK,
  kQSPI_IPCommandUsageError = QuadSPI_FR_IUEF_MASK,
  kQSPI_IPCommandTriggerDuringAHBAccess = QuadSPI_FR_IPAEF_MASK,
  kQSPI_IPCommandTriggerDuringIPAccess = QuadSPI_FR_IPIEF_MASK,
  kQSPI_IPCommandTriggerDuringAHBGrant = QuadSPI_FR_IPGEF_MASK,
  kQSPI_IPCommandTransactionFinished = QuadSPI_FR_TFF_MASK,
  kQSPI_FlagAll = (int)0x8C83F8D1U }

  QSPI error flags.
- enum _qspi_flags {
  kQSPI_DataLearningSamplePoint = (int)QuadSPI_SR_DLPSMP_MASK,
  kQSPI_TxBufferFull = QuadSPI_SR_TXFULL_MASK,
  kQSPI_TxBufferEnoughData = QuadSPI_SR_TXEDA_MASK,
  kQSPI_RxDMA = QuadSPI_SR_RXDMA_MASK,
  kQSPI_RxBufferFull = QuadSPI_SR_RXFULL_MASK,
  kQSPI_RxWatermark = QuadSPI_SR_RXWE_MASK,
  kQSPI_AHB3BufferFull = QuadSPI_SR_AHB3FUL_MASK,
  kQSPI_AHB2BufferFull = QuadSPI_SR_AHB2FUL_MASK,
  kQSPI_AHB1BufferFull = QuadSPI_SR_AHB1FUL_MASK,
  kQSPI_AHB0BufferFull = QuadSPI_SR_AHB0FUL_MASK,
  kQSPI_AHB3BufferNotEmpty = QuadSPI_SR_AHB3NE_MASK,
  kQSPI_AHB2BufferNotEmpty = QuadSPI_SR_AHB2NE_MASK,
  kQSPI_AHB1BufferNotEmpty = QuadSPI_SR_AHB1NE_MASK,
  kQSPI_AHB0BufferNotEmpty = QuadSPI_SR_AHB0NE_MASK,
  kQSPI_AHBTransactionPending = QuadSPI_SR_AHBTRN_MASK,
  kQSPI_AHBCommandPriorityGranted = QuadSPI_SR_AHBGNT_MASK,
  kQSPI_AHBAccess = QuadSPI_SR_AHB_ACC_MASK,
  kQSPI_IPAccess = QuadSPI_SR_IP_ACC_MASK,
  kQSPI_Busy = QuadSPI_SR_BUSY_MASK,
  kQSPI_StateAll = (int)0xEF897FE7U }

  QSPI state bit.
- enum _qspi_interrupt_enable {

kQSPI_DataLearningFailInterruptEnable,
kQSPI_TxBufferFillInterruptEnable = QuadSPI_RSER_TBFIE_MASK,
kQSPI_TxBufferUnderrunInterruptEnable = QuadSPI_RSER_TBUIE_MASK,
kQSPI_IllegalInstructionInterruptEnable,
kQSPI_RxBufferOverflowInterruptEnable = QuadSPI_RSER_RBOIE_MASK,
kQSPI_RxBufferDrainInterruptEnable = QuadSPI_RSER_RBDIE_MASK,
kQSPI_AHBSequenceErrorInterruptEnable = QuadSPI_RSER_ABSEIE_MASK,
kQSPI_AHBBufferOverflowInterruptEnable = QuadSPI_RSER_ABOIE_MASK,
kQSPI_IPCommandUsageErrorInterruptEnable = QuadSPI_RSER_IUEIE_MASK,
kQSPI_IPCommandTriggerDuringAHBAccessInterruptEnable,
kQSPI_IPCommandTriggerDuringIPAccessInterruptEnable,
kQSPI_IPCommandTriggerDuringAHBGrantInterruptEnable,
kQSPI_IPCommandTransactionFinishedInterruptEnable,
kQSPI_AllInterruptEnable = (int)0x8C83F8D1U }
  QSPI interrupt enable.
- enum _qspi_dma_enable { kQSPI_RxBufferDrainDMAEnable = QuadSPI_RSER_RBDDE_MAS-K }
  QSPI DMA request flag.
- enum qspi_dqs_phrase_shift_t {
  kQSPI_DQSNoPhraseShift = 0x0U,
  kQSPI_DQSPhraseShift45Degree,
  kQSPI_DQSPhraseShift90Degree,
  kQSPI_DQSPhraseShift135Degree }
  Phrase shift number for DQS mode.
- enum qspi_dqs_read_sample_clock_t {
  kQSPI_ReadSampleClkInternalLoopback = 0x0U,
  kQSPI_ReadSampleClkLoopbackFromDqsPad = 0x1U,
  kQSPI_ReadSampleClkExternalInputFromDqsPad = 0x2U }
  Qspi read sampling option.

Driver version

- #define FSL_QSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))
  QSPI driver version 2.2.2.

Initialization and deinitialization

- uint32_t QSPI_GetInstance (QuadSPI_Type *base)
  Get the instance number for QSPI.
- void QSPI_Init (QuadSPI_Type *base, qspi_config_t *config, uint32_t srcClock_Hz)
  Initializes the QSPI module and internal state.
- void QSPI_GetDefaultQspiConfig (qspi_config_t *config)
  Gets default settings for QSPI.
- void QSPI_Deinit (QuadSPI_Type *base)
  Deinitializes the QSPI module.

- void QSPI_SetFlashConfig (QuadSPI_Type *base, qspi_flash_config_t *config)
  Configures the serial flash parameter.
- void QSPI_SoftwareReset (QuadSPI_Type *base)
  Software reset for the QSPI logic.
- static void QSPI_Enable (QuadSPI_Type *base, bool enable)
  Enables or disables the QSPI module.

Status

- static uint32_t QSPI_GetStatusFlags (QuadSPI_Type *base)
  Gets the state value of QSPI.
- static uint32_t QSPI_GetErrorStatusFlags (QuadSPI_Type *base)
  Gets QSPI error status flags.
- static void QSPI_ClearErrorFlag (QuadSPI_Type *base, uint32_t mask)
  Clears the QSPI error flags.

Interrupts

- static void QSPI_EnableInterrupts (QuadSPI_Type *base, uint32_t mask)
  Enables the QSPI interrupts.
- static void QSPI_DisableInterrupts (QuadSPI_Type *base, uint32_t mask)
  Disables the QSPI interrupts.

DMA Control

- static void QSPI_EnableDMA (QuadSPI_Type *base, uint32_t mask, bool enable)
  Enables the QSPI DMA source.
- static uint32_t QSPI_GetTxDataRegisterAddress (QuadSPI_Type *base)
  Gets the Tx data register address.
- uint32_t QSPI_GetRxDataRegisterAddress (QuadSPI_Type *base)
  Gets the Rx data register address used for DMA operation.

Bus Operations

- static void QSPI_SetIPCommandAddress (QuadSPI_Type *base, uint32_t addr)
  Sets the IP command address.
- static void QSPI_SetIPCommandSize (QuadSPI_Type *base, uint32_t size)
  Sets the IP command size.
- void QSPI_ExecuteIPCommand (QuadSPI_Type *base, uint32_t index)
  Executes IP commands located in LUT table.
- void QSPI_ExecuteAHBCommand (QuadSPI_Type *base, uint32_t index)
  Executes AHB commands located in LUT table.
- static void QSPI_EnableIPParallelMode (QuadSPI_Type *base, bool enable)
  Enables/disables the QSPI IP command parallel mode.
- static void QSPI_EnableAHBParallelMode (QuadSPI_Type *base, bool enable)
  Enables/disables the QSPI AHB command parallel mode.

- void QSPI_UpdateLUT (QuadSPI_Type ∗base, uint32_t index, uint32_t ∗cmd)
  Updates the LUT table.
- static void QSPI_ClearFifo (QuadSPI_Type ∗base, uint32_t mask)
  Clears the QSPI FIFO logic.
- static void QSPI_ClearCommandSequence (QuadSPI_Type ∗base, qspi_command_seq_t seq)
  @ brief Clears the command sequence for the IP/buffer command.
- static void QSPI_EnableDDRMode (QuadSPI_Type ∗base, bool enable)
  Enable or disable DDR mode.
- void QSPI_SetReadDataArea (QuadSPI_Type ∗base, qspi_read_area_t area)
  @ brief Set the RX buffer readout area.
- void QSPI_WriteBlocking (QuadSPI_Type ∗base, uint32_t ∗buffer, size_t size)
  Sends a buffer of data bytes using a blocking method.
- static void QSPI_WriteData (QuadSPI_Type ∗base, uint32_t data)
  Writes data into FIFO.
- void QSPI_ReadBlocking (QuadSPI_Type ∗base, uint32_t ∗buffer, size_t size)
  Receives a buffer of data bytes using a blocking method.
- uint32_t QSPI_ReadData (QuadSPI_Type ∗base)
  Receives data from data FIFO.

Transactional

- static void QSPI_TransferSendBlocking (QuadSPI_Type ∗base, qspi_transfer_t ∗xfer)
  Writes data to the QSPI transmit buffer.
- static void QSPI_TransferReceiveBlocking (QuadSPI_Type ∗base, qspi_transfer_t ∗xfer)
  Reads data from the QSPI receive buffer in polling way.

16.2.2 Data Structure Documentation

16.2.2.1 struct qspi_dqs_config_t

Data Fields

- uint32_t portADelayTapNum
  Delay chain tap number selection for QSPI port A DQS.
- qspi_dqs_phrase_shift_t shift
  Phase shift for internal DQS generation.
- qspi_dqs_read_sample_clock_t rxSampleClock
  Read sample clock for Dqs.
- bool enableDQSClkInverse
  Enable inverse clock for internal DQS generation.

16.2.2.1.0.13 Field Documentation

16.2.2.1.0.13.1 qspi_dqs_read_sample_clock_t qspi_dqs_config_t::rxSampleClock

16.2.2.2 struct qspi_flash_timing_t

Data Fields

- uint32_t dataHoldTime
  Serial flash data in hold time.
- uint32_t CSHoldTime
  Serial flash CS hold time in terms of serial flash clock cycles.
- uint32_t CSSetupTime
  Serial flash CS setup time in terms of serial flash clock cycles.

16.2.2.3 struct qspi_config_t

Data Fields

- uint32_t clockSource
  Clock source for QSPI module.
- uint32_t baudRate
  Serial flash clock baud rate.
- uint8_t txWatermark
  QSPI transmit watermark value.
- uint8_t rxWatermark
  QSPI receive watermark value.
- uint32_t AHBbufferSize [FSL_FEATURE_QSPI_AHB_BUFFER_COUNT]
  AHB buffer size.
- uint8_t AHBbufferMaster [FSL_FEATURE_QSPI_AHB_BUFFER_COUNT]
  AHB buffer master.
- bool enableAHBbuffer3AllMaster
  Is AHB buffer3 for all master.
- qspi_read_area_t area
  Which area Rx data readout.
- bool enableQspi
  Enable QSPI after initialization.

16.2.2.3.0.14 Field Documentation

16.2.2.3.0.14.1 uint8_t qspi_config_t::rxWatermark

16.2.2.3.0.14.2 uint32_t qspi_config_t::AHBbufferSize[FSL_FEATURE_QSPI_AHB_BUFFER_COUNT]

16.2.2.3.0.14.3 uint8_t qspi_config_t::AHBbufferMaster[FSL_FEATURE_QSPI_AHB_BUFFER_COUNT]

16.2.2.3.0.14.4 bool qspi_config_t::enableAHBbuffer3AllMaster

16.2.2.4 struct qspi_flash_config_t

Data Fields

- uint32_t flashA1Size
  Flash A1 size.
- uint32_t flashA2Size
  Flash A2 size.
- uint32_t flashB1Size
  Flash B1 size.
- uint32_t flashB2Size
  Flash B2 size.
- uint32_t lookuptable [FSL_FEATURE_QSPI_LUT_DEPTH]
  Flash command in LUT.
- uint32_t dataHoldTime
  Data line hold time.
- uint32_t CSHoldTime
  CS line hold time.
- uint32_t CSSetupTime
  CS line setup time.
- uint32_t cloumnspace
  Column space size.
- uint32_t dataLearnValue
  Data Learn value if enable data learn.
- qspi_endianness_t endian
  Flash data endianess.
- bool enableWordAddress
  If enable word address.

16.2.2.4.0.15 Field Documentation

16.2.2.4.0.15.1 uint32_t qspi_flash_config_t::dataHoldTime

16.2.2.4.0.15.2 qspi_endianness_t qspi_flash_config_t::endian

16.2.2.4.0.15.3 bool qspi_flash_config_t::enableWordAddress

16.2.2.5 struct qspi_transfer_t

Data Fields

- uint32_t * data
  Pointer to data to transmit.
- size_t dataSize
  Bytes to be transmit.

16.2.2.6 struct ip_command_config_t

16.2.3 Macro Definition Documentation

16.2.3.1 #define FSL_QSPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))

16.2.4 Enumeration Type Documentation

16.2.4.1 anonymous enum

Enumerator

kStatus_QSPI_Idle QSPI is in idle state.
kStatus_QSPI_Busy QSPI is busy.
kStatus_QSPI_Error Error occurred during QSPI transfer.

16.2.4.2 enum qspi_read_area_t

Enumerator

kQSPI_ReadAHB QSPI read from AHB buffer.
kQSPI_ReadIP QSPI read from IP FIFO.

16.2.4.3 enum qspi_command_seq_t

Enumerator

kQSPI_IPSeq IP command sequence.

kQSPI_BufferSeq Buffer command sequence.

16.2.4.4 enum qspi_fifo_t

Enumerator

kQSPI_TxFifo QSPI Tx FIFO.
kQSPI_RxFifo QSPI Rx FIFO.
kQSPI_AllFifo QSPI all FIFO, including Tx and Rx.

16.2.4.5 enum qspi_endianness_t

Enumerator

kQSPI_64BigEndian 64 bits big endian
kQSPI_32LittleEndian 32 bit little endian
kQSPI_32BigEndian 32 bit big endian
kQSPI_64LittleEndian 64 bit little endian

16.2.4.6 enum _qspi_error_flags

Enumerator

kQSPI_DataLearningFail Data learning pattern failure flag.
kQSPI_TxBufferFill Tx buffer fill flag.
kQSPI_TxBufferUnderrun Tx buffer underrun flag.
kQSPI_IllegalInstruction Illegal instruction error flag.
kQSPI_RxBufferOverflow Rx buffer overflow flag.
kQSPI_RxBufferDrain Rx buffer drain flag.
kQSPI_AHBSequenceError AHB sequence error flag.
kQSPI_AHBBufferOverflow AHB buffer overflow flag.
kQSPI_IPCommandUsageError IP command usage error flag.
kQSPI_IPCommandTriggerDuringAHBAccess IP command trigger during AHB access error.
kQSPI_IPCommandTriggerDuringIPAccess IP command trigger cannot be executed.
kQSPI_IPCommandTriggerDuringAHBGrant IP command trigger during AHB grant error.
kQSPI_IPCommandTransactionFinished IP command transaction finished flag.
kQSPI_FlagAll All error flag.

16.2.4.7 enum _qspi_flags

Enumerator

kQSPI_DataLearningSamplePoint Data learning sample point.

kQSPI_TxBufferFull Tx buffer full flag.
kQSPI_TxBufferEnoughData Tx buffer enough data available.
kQSPI_RxDMA Rx DMA is requesting or running.
kQSPI_RxBufferFull Rx buffer full.
kQSPI_RxWatermark Rx buffer watermark exceeded.
kQSPI_AHB3BufferFull AHB buffer 3 full.
kQSPI_AHB2BufferFull AHB buffer 2 full.
kQSPI_AHB1BufferFull AHB buffer 1 full.
kQSPI_AHB0BufferFull AHB buffer 0 full.
kQSPI_AHB3BufferNotEmpty AHB buffer 3 not empty.
kQSPI_AHB2BufferNotEmpty AHB buffer 2 not empty.
kQSPI_AHB1BufferNotEmpty AHB buffer 1 not empty.
kQSPI_AHB0BufferNotEmpty AHB buffer 0 not empty.
kQSPI_AHBTransactionPending AHB access transaction pending.
kQSPI_AHBCommandPriorityGranted AHB command priority granted.
kQSPI_AHBAccess AHB access.
kQSPI_IPAccess IP access.
kQSPI_Busy Module busy.
kQSPI_StateAll All flags.

16.2.4.8 enum _qspi_interrupt_enable

Enumerator

kQSPI_DataLearningFailInterruptEnable Data learning pattern failure interrupt enable.
kQSPI_TxBufferFillInterruptEnable Tx buffer fill interrupt enable.
kQSPI_TxBufferUnderrunInterruptEnable Tx buffer underrun interrupt enable.
kQSPI_IllegalInstructionInterruptEnable Illegal instruction error interrupt enable.
kQSPI_RxBufferOverflowInterruptEnable Rx buffer overflow interrupt enable.
kQSPI_RxBufferDrainInterruptEnable Rx buffer drain interrupt enable.
kQSPI_AHBSequenceErrorInterruptEnable AHB sequence error interrupt enable.
kQSPI_AHBBufferOverflowInterruptEnable AHB buffer overflow interrupt enable.
kQSPI_IPCommandUsageErrorInterruptEnable IP command usage error interrupt enable.
kQSPI_IPCommandTriggerDuringAHBAccessInterruptEnable IP command trigger during AHB access error.
kQSPI_IPCommandTriggerDuringIPAccessInterruptEnable IP command trigger cannot be executed.
kQSPI_IPCommandTriggerDuringAHBGrantInterruptEnable IP command trigger during AHB grant error.
kQSPI_IPCommandTransactionFinishedInterruptEnable IP command transaction finished interrupt enable.
kQSPI_AllInterruptEnable All error interrupt enable.

16.2.4.9 enum _qspi_dma_enable

Enumerator

kQSPI_RxBufferDrainDMAEnable Rx buffer drain DMA.

16.2.4.10 enum qspi_dqs_phrase_shift_t

Enumerator

kQSPI_DQSNoPhraseShift No phase shift.
kQSPI_DQSPhraseShift45Degree Select 45 degree phase shift.
kQSPI_DQSPhraseShift90Degree Select 90 degree phase shift.
kQSPI_DQSPhraseShift135Degree Select 135 degree phase shift.

16.2.4.11 enum qspi_dqs_read_sample_clock_t

Enumerator

kQSPI_ReadSampleClkInternalLoopback Read sample clock adopts internal loopback mode.
kQSPI_ReadSampleClkLoopbackFromDqsPad Dummy Read strobe generated by QSPI Controller and loopback from DQS pad.
kQSPI_ReadSampleClkExternalInputFromDqsPad Flash provided Read strobe and input from D-QS pad.

16.2.5 Function Documentation

16.2.5.1 uint32_t QSPI_GetInstance (QuadSPI_Type ∗ *base*)

Parameters

base	QSPI base pointer.

16.2.5.2 void QSPI_Init (QuadSPI_Type ∗ *base,* qspi_config_t ∗ *config,* uint32_t *srcClock_Hz*)

This function enables the clock for QSPI and also configures the QSPI with the input configure parameters. Users should call this function before any QSPI operations.

Parameters

base	Pointer to QuadSPI Type.
config	QSPI configure structure.
srcClock_Hz	QSPI source clock frequency in Hz.

16.2.5.3 void QSPI_GetDefaultQspiConfig (qspi_config_t * *config*)

Parameters

config	QSPI configuration structure.

16.2.5.4 void QSPI_Deinit (QuadSPI_Type * *base*)

Clears the QSPI state and QSPI module registers.

Parameters

base	Pointer to QuadSPI Type.

16.2.5.5 void QSPI_SetFlashConfig (QuadSPI_Type * *base,* qspi_flash_config_t * *config*)

This function configures the serial flash relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the QSPI features.

Parameters

base	Pointer to QuadSPI Type.
config	Flash configuration parameters.

16.2.5.6 void QSPI_SoftwareReset (QuadSPI_Type * *base*)

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

base	Pointer to QuadSPI Type.

16.2.5.7 static void QSPI_Enable (QuadSPI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.
enable	True means enable QSPI, false means disable.

16.2.5.8 static uint32_t QSPI_GetStatusFlags (QuadSPI_Type * *base*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.

Returns

status flag, use status flag to AND _qspi_flags could get the related status.

16.2.5.9 static uint32_t QSPI_GetErrorStatusFlags (QuadSPI_Type * *base*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.

Returns

status flag, use status flag to AND _qspi_error_flags could get the related status.

16.2.5.10 static void QSPI_ClearErrorFlag (QuadSPI_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.
mask	Which kind of QSPI flags to be cleared, a combination of _qspi_error_flags.

16.2.5.11 static void QSPI_EnableInterrupts (QuadSPI_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.
mask	QSPI interrupt source.

16.2.5.12 static void QSPI_DisableInterrupts (QuadSPI_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.
mask	QSPI interrupt source.

16.2.5.13 static void QSPI_EnableDMA (QuadSPI_Type ∗ *base,* uint32_t *mask,* bool *enable*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.
mask	QSPI DMA source.
enable	True means enable DMA, false means disable.

16.2.5.14 static uint32_t QSPI_GetTxDataRegisterAddress (QuadSPI_Type ∗ *base*) [inline], [static]

It is used for DMA operation.

Parameters

base	Pointer to QuadSPI Type.

Returns

QSPI Tx data register address.

16.2.5.15 uint32_t QSPI_GetRxDataRegisterAddress (QuadSPI_Type ∗ *base*)

This function returns the Rx data register address or Rx buffer address according to the Rx read area settings.

Parameters

base	Pointer to QuadSPI Type.

Returns

QSPI Rx data register address.

16.2.5.16 static void QSPI_SetIPCommandAddress (QuadSPI_Type ∗ *base*, uint32_t *addr*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.
addr	IP command address.

16.2.5.17 static void QSPI_SetIPCommandSize (QuadSPI_Type ∗ *base*, uint32_t *size*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.

size	IP command size.

16.2.5.18 void QSPI_ExecuteIPCommand (QuadSPI_Type * *base*, uint32_t *index*)

Parameters

base	Pointer to QuadSPI Type.
index	IP command located in which LUT table index.

16.2.5.19 void QSPI_ExecuteAHBCommand (QuadSPI_Type * *base*, uint32_t *index*)

Parameters

base	Pointer to QuadSPI Type.
index	AHB command located in which LUT table index.

16.2.5.20 static void QSPI_EnableIPParallelMode (QuadSPI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.
enable	True means enable parallel mode, false means disable parallel mode.

16.2.5.21 static void QSPI_EnableAHBParallelMode (QuadSPI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.
enable	True means enable parallel mode, false means disable parallel mode.

16.2.5.22 void QSPI_UpdateLUT (QuadSPI_Type * *base*, uint32_t *index*, uint32_t * *cmd*)

Parameters

base	Pointer to QuadSPI Type.
index	Which LUT index needs to be located. It should be an integer divided by 4.
cmd	Command sequence array.

16.2.5.23 static void QSPI_ClearFifo (QuadSPI_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	Pointer to QuadSPI Type.
mask	Which kind of QSPI FIFO to be cleared.

16.2.5.24 static void QSPI_ClearCommandSequence (QuadSPI_Type ∗ *base,* qspi_command_seq_t *seq*) [inline], [static]

This function can reset the command sequence.

Parameters

base	QSPI base address.
seq	Which command sequence need to reset, IP command, buffer command or both.

16.2.5.25 static void QSPI_EnableDDRMode (QuadSPI_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	QSPI base pointer
enable	True means enable DDR mode, false means disable DDR mode.

16.2.5.26 void QSPI_SetReadDataArea (QuadSPI_Type ∗ *base,* qspi_read_area_t *area*)

This function can set the RX buffer readout, from AHB bus or IP Bus.

Parameters

base	QSPI base address.
area	QSPI Rx buffer readout area. AHB bus buffer or IP bus buffer.

16.2.5.27 void QSPI_WriteBlocking (QuadSPI_Type * *base,* uint32_t * *buffer,* size_t *size*)

Note

This function blocks via polling until all bytes have been sent.

Parameters

base	QSPI base pointer
buffer	The data bytes to send
size	The number of data bytes to send

16.2.5.28 static void QSPI_WriteData (QuadSPI_Type * *base,* uint32_t *data*) [inline], [static]

Parameters

base	QSPI base pointer
data	The data bytes to send

16.2.5.29 void QSPI_ReadBlocking (QuadSPI_Type * *base,* uint32_t * *buffer,* size_t *size*)

Note

This function blocks via polling until all bytes have been sent. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

Parameters

base	QSPI base pointer
buffer	The data bytes to send
size	The number of data bytes to receive

16.2.5.30 uint32_t QSPI_ReadData (QuadSPI_Type * *base*)

Parameters

base	QSPI base pointer

Returns

The data in the FIFO.

16.2.5.31 static void QSPI_TransferSendBlocking (QuadSPI_Type * *base*, qspi_transfer_t * *xfer*) [inline], [static]

This function writes a continuous data to the QSPI transmit FIFO. This function is a block function and can return only when finished. This function uses polling methods.

Parameters

base	Pointer to QuadSPI Type.
xfer	QSPI transfer structure.

16.2.5.32 static void QSPI_TransferReceiveBlocking (QuadSPI_Type * *base*, qspi_transfer_t * *xfer*) [inline], [static]

This function reads continuous data from the QSPI receive buffer/FIFO. This function is a blocking function and can return only when finished. This function uses polling methods. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

Parameters

base	Pointer to QuadSPI Type.
xfer	QSPI transfer structure.

Chapter 17
RDC: Resource Domain Controller

Overview

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

The Resource Domain Controller (RDC) provides robust support for the isolation of destination memory mapped locations such as peripherals and memory to a single core, a bus master, or set of cores and bus masters.

The RDC driver should be used together with the RDC_SEMA42 driver.

Data Structures

- struct rdc_hardware_config_t
  RDC hardware configuration. More...
- struct rdc_domain_assignment_t
  Master domain assignment. More...
- struct rdc_periph_access_config_t
  Peripheral domain access permission configuration. More...
- struct rdc_mem_access_config_t
  Memory region domain access control configuration. More...
- struct rdc_mem_status_t
  Memory region access violation status. More...

Enumerations

- enum _rdc_interrupts { kRDC_RestoreCompleteInterrupt = RDC_INTCTRL_RCI_EN_MASK }
  RDC interrupts.
- enum _rdc_flags { kRDC_PowerDownDomainOn = RDC_STAT_PDS_MASK }
  RDC status.
- enum _rdc_access_policy {
  kRDC_NoAccess = 0,
  kRDC_WriteOnly = 1,
  kRDC_ReadOnly = 2,
  kRDC_ReadWrite = 3 }
  Access permission policy.

Functions

- void RDC_Init (RDC_Type ∗base)
  Initializes the RDC module.
- void RDC_Deinit (RDC_Type ∗base)
  De-initializes the RDC module.
- void RDC_GetHardwareConfig (RDC_Type ∗base, rdc_hardware_config_t ∗config)
  Gets the RDC hardware configuration.

- static void RDC_EnableInterrupts (RDC_Type ∗base, uint32_t mask)
  Enable interrupts.
- static void RDC_DisableInterrupts (RDC_Type ∗base, uint32_t mask)
  Disable interrupts.
- static uint32_t RDC_GetInterruptStatus (RDC_Type ∗base)
  Get the interrupt pending status.
- static void RDC_ClearInterruptStatus (RDC_Type ∗base, uint32_t mask)
  Clear interrupt pending status.
- static uint32_t RDC_GetStatus (RDC_Type ∗base)
  Get RDC status.
- static void RDC_ClearStatus (RDC_Type ∗base, uint32_t mask)
  Clear RDC status.
- void RDC_SetMasterDomainAssignment (RDC_Type ∗base, rdc_master_t master, const rdc_domain_assignment_t ∗domainAssignment)
  Set master domain assignment.
- void RDC_GetDefaultMasterDomainAssignment (rdc_domain_assignment_t ∗domainAssignment)
  Get default master domain assignment.
- static void RDC_LockMasterDomainAssignment (RDC_Type ∗base, rdc_master_t master)
  Lock master domain assignment.
- void RDC_SetPeriphAccessConfig (RDC_Type ∗base, const rdc_periph_access_config_t ∗config)
  Set peripheral access policy.
- void RDC_GetDefaultPeriphAccessConfig (rdc_periph_access_config_t ∗config)
  Get default peripheral access policy.
- static void RDC_LockPeriphAccessConfig (RDC_Type ∗base, rdc_periph_t periph)
  Lock peripheral access policy configuration.
- void RDC_SetMemAccessConfig (RDC_Type ∗base, const rdc_mem_access_config_t ∗config)
  Set memory region access policy.
- void RDC_GetDefaultMemAccessConfig (rdc_mem_access_config_t ∗config)
  Get default memory region access policy.
- static void RDC_LockMemAccessConfig (RDC_Type ∗base, rdc_mem_t mem)
  Lock memory access policy configuration.
- static void RDC_SetMemAccessValid (RDC_Type ∗base, rdc_mem_t mem, bool valid)
  Enable or disable memory access policy configuration.
- void RDC_GetMemViolationStatus (RDC_Type ∗base, rdc_mem_t mem, rdc_mem_status_t ∗status)
  Get the memory region violation status.
- static void RDC_ClearMemViolationFlag (RDC_Type ∗base, rdc_mem_t mem)
  Clear the memory region violation flag.
- static uint8_t RDC_GetCurrentMasterDomainId (RDC_Type ∗base)
  Gets the domain ID of the current bus master.

Data Structure Documentation

17.2.1 struct rdc_hardware_config_t

Data Fields

- uint32_t domainNumber: 4
  Number of domains.
- uint32_t masterNumber: 8

Number of bus masters.
- uint32_t periphNumber: 8
  Number of peripherals.
- uint32_t memNumber: 8
  Number of memory regions.

17.2.1.0.0.16 Field Documentation

17.2.1.0.0.16.1 uint32_t rdc_hardware_config_t::domainNumber

17.2.1.0.0.16.2 uint32_t rdc_hardware_config_t::masterNumber

17.2.1.0.0.16.3 uint32_t rdc_hardware_config_t::periphNumber

17.2.1.0.0.16.4 uint32_t rdc_hardware_config_t::memNumber

17.2.2 struct rdc_domain_assignment_t

Data Fields

- uint32_t domainId: 2U
  Domain ID.
- uint32_t __pad0__: 29U
  Reserved.
- uint32_t lock: 1U
  Lock the domain assignment.

17.2.2.0.0.17 Field Documentation

17.2.2.0.0.17.1 uint32_t rdc_domain_assignment_t::domainId

17.2.2.0.0.17.2 uint32_t rdc_domain_assignment_t::__pad0__

17.2.2.0.0.17.3 uint32_t rdc_domain_assignment_t::lock

17.2.3 struct rdc_periph_access_config_t

Data Fields

- rdc_periph_t periph
  Peripheral name.
- bool lock
  Lock the permission until reset.
- bool enableSema
  Enable semaphore or not, when enabled, master should call RDC_SEMA42_Lock to lock the semaphore gate accordingly before access the peripheral.
- uint16_t policy
  Access policy.

17.2.3.0.0.18 Field Documentation

17.2.3.0.0.18.1 rdc_periph_t rdc_periph_access_config_t::periph

17.2.3.0.0.18.2 bool rdc_periph_access_config_t::lock

17.2.3.0.0.18.3 bool rdc_periph_access_config_t::enableSema

17.2.3.0.0.18.4 uint16_t rdc_periph_access_config_t::policy

17.2.4 struct rdc_mem_access_config_t

Note that when setting the baseAddress and endAddress, should be aligned to the region resolution, see rdc_mem_t definitions.

Data Fields

- rdc_mem_t mem
  Memory region descriptor name.
- bool lock
  Lock the configuration.
- uint64_t baseAddress
  Start address of the memory region.
- uint64_t endAddress
  End address of the memory region.
- uint16_t policy
  Access policy.

17.2.4.0.0.19 Field Documentation

17.2.4.0.0.19.1 rdc_mem_t rdc_mem_access_config_t::mem

17.2.4.0.0.19.2 bool rdc_mem_access_config_t::lock

17.2.4.0.0.19.3 uint64_t rdc_mem_access_config_t::baseAddress

17.2.4.0.0.19.4 uint64_t rdc_mem_access_config_t::endAddress

17.2.4.0.0.19.5 uint16_t rdc_mem_access_config_t::policy

17.2.5 struct rdc_mem_status_t

Data Fields

- bool hasViolation
  Violating happens or not.
- uint8_t domainID
  Violating Domain ID.

- uint64_t address
  Violating Address.

17.2.5.0.0.20 Field Documentation

17.2.5.0.0.20.1 bool rdc_mem_status_t::hasViolation

17.2.5.0.0.20.2 uint8_t rdc_mem_status_t::domainID

17.2.5.0.0.20.3 uint64_t rdc_mem_status_t::address

Enumeration Type Documentation

17.3.1 enum _rdc_interrupts

Enumerator

kRDC_RestoreCompleteInterrupt Interrupt generated when the RDC has completed restoring state to a recently re-powered memory regions.

17.3.2 enum _rdc_flags

Enumerator

kRDC_PowerDownDomainOn Power down domain is ON.

17.3.3 enum _rdc_access_policy

Enumerator

kRDC_NoAccess Could not read or write.
kRDC_WriteOnly Write only.
kRDC_ReadOnly Read only.
kRDC_ReadWrite Read and write.

Function Documentation

17.4.1 void RDC_Init (RDC_Type * *base*)

This function enables the RDC clock.

Parameters

base	RDC peripheral base address.

17.4.2 void RDC_Deinit (RDC_Type * *base*)

This function disables the RDC clock.

Parameters

base	RDC peripheral base address.

17.4.3 void RDC_GetHardwareConfig (RDC_Type * *base*, rdc_hardware_config_t * *config*)

This function gets the RDC hardware configurations, including number of bus masters, number of domains, number of memory regions and number of peripherals.

Parameters

base	RDC peripheral base address.
config	Pointer to the structure to get the configuration.

17.4.4 static void RDC_EnableInterrupts (RDC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	RDC peripheral base address.
mask	Interrupts to enable, it is OR'ed value of enum _rdc_interrupts.

17.4.5 static void RDC_DisableInterrupts (RDC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	RDC peripheral base address.
mask	Interrupts to disable, it is OR'ed value of enum _rdc_interrupts.

17.4.6 static uint32_t RDC_GetInterruptStatus (RDC_Type * *base*) [inline], [static]

Parameters

base	RDC peripheral base address.

Returns

Interrupts pending status, it is OR'ed value of enum _rdc_interrupts.

17.4.7 static void RDC_ClearInterruptStatus (RDC_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	RDC peripheral base address.
mask	Status to clear, it is OR'ed value of enum _rdc_interrupts.

17.4.8 static uint32_t RDC_GetStatus (RDC_Type * *base*) [inline], [static]

Parameters

base	RDC peripheral base address.

Returns

mask RDC status, it is OR'ed value of enum _rdc_flags.

17.4.9 static void RDC_ClearStatus (RDC_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	RDC peripheral base address.
mask	RDC status to clear, it is OR'ed value of enum _rdc_flags.

17.4.10 void RDC_SetMasterDomainAssignment (RDC_Type * *base*, rdc_master_t *master*, const rdc_domain_assignment_t * *domainAssignment*)

Parameters

base	RDC peripheral base address.
master	Which master to set.
domain-Assignment	Pointer to the assignment.

17.4.11 void RDC_GetDefaultMasterDomainAssignment (rdc_domain_assignment-_t * *domainAssignment*)

The default configuration is:

```
assignment->domainId = 0U;
assignment->lock = 0U;
```

Parameters

domain-Assignment	Pointer to the assignment.

17.4.12 static void RDC_LockMasterDomainAssignment (RDC_Type * *base*, rdc_master_t *master*) [inline], [static]

Once locked, it could not be unlocked until next reset.

Parameters

base	RDC peripheral base address.
master	Which master to lock.

17.4.13 void RDC_SetPeriphAccessConfig (RDC_Type ∗ *base*, const rdc_periph_access_config_t ∗ *config*)

Parameters

base	RDC peripheral base address.
config	Pointer to the policy configuration.

17.4.14 void RDC_GetDefaultPeriphAccessConfig (rdc_periph_access_config_t ∗ *config*)

The default configuration is:

```
config->lock = false;
config->enableSema = false;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

Parameters

config	Pointer to the policy configuration.

17.4.15 static void RDC_LockPeriphAccessConfig (RDC_Type ∗ *base*, rdc_periph_t *periph*) [inline], [static]

Once locked, it could not be unlocked until reset.

Parameters

base	RDC peripheral base address.

periph	Which peripheral to lock.

17.4.16 void RDC_SetMemAccessConfig (RDC_Type ∗ *base*, const rdc_mem_access_config_t ∗ *config*)

Note that when setting the baseAddress and endAddress in `config`, should be aligned to the region resolution, see rdc_mem_t definitions.

Parameters

base	RDC peripheral base address.
config	Pointer to the policy configuration.

17.4.17 void RDC_GetDefaultMemAccessConfig (rdc_mem_access_config_t ∗ *config*)

The default configuration is:

```
config->lock = false;
config->baseAddress = 0;
config->endAddress = 0;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                 RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

Parameters

config	Pointer to the policy configuration.

17.4.18 static void RDC_LockMemAccessConfig (RDC_Type ∗ *base*, rdc_mem_t *mem*) [inline], [static]

Once locked, it could not be unlocked until reset. After locked, you can only call RDC_SetMemAccessValid to enable the configuration, but can not disable it or change other settings.

Parameters

base	RDC peripheral base address.
mem	Which memory region to lock.

17.4.19 static void RDC_SetMemAccessValid (RDC_Type * *base*, rdc_mem_t *mem*, bool *valid*) [inline], [static]

Parameters

base	RDC peripheral base address.
mem	Which memory region to operate.
valid	Pass in true to valid, false to invalid.

17.4.20 void RDC_GetMemViolationStatus (RDC_Type * *base*, rdc_mem_t *mem*, rdc_mem_status_t * *status*)

The first access violation is captured. Subsequent violations are ignored until the status register is cleared. Contents are cleared upon reading the register. Clearing of contents occurs only when the status is read by the memory region's associated domain ID(s).

Parameters

base	RDC peripheral base address.
mem	Which memory region to get.
status	The returned status.

17.4.21 static void RDC_ClearMemViolationFlag (RDC_Type * *base*, rdc_mem_t *mem*) [inline], [static]

Parameters

base	RDC peripheral base address.
mem	Which memory region to clear.

17.4.22 static uint8_t RDC_GetCurrentMasterDomainId (RDC_Type ∗ *base*) [inline], [static]

This function returns the domain ID of the current bus master.

Parameters

base	RDC peripheral base address.

Returns

Domain ID of current bus master.

Chapter 18
RDC_SEMA42: Hardware Semaphores Driver

Overview

The MCUXpresso SDK provides a driver for the RDC_SEMA42 module of MCUXpresso SDK devices.

The RDC_SEMA42 driver should be used together with RDC driver.

Before using the RDC_SEMA42, call the RDC_SEMA42_Init() function to initialize the module. Note that this function only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either the RDC_SEMA42_ResetGate() or RDC_SEMA42_ResetAllGates() functions. The function RDC_SEMA42_Deinit() deinitializes the RD-C_SEMA42.

The RDC_SEMA42 provides two functions to lock the RDC_SEMA42 gate. The function RDC_SEM-A42_TryLock() tries to lock the gate. If the gate has been locked by another processor, this function returns an error immediately. The function RDC_SEMA42_Lock() is a blocking method, which waits until the gate is free and locks it.

The RDC_SEMA42_Unlock() unlocks the RDC_SEMA42 gate. The gate can only be unlocked by the processor which locked it. If the gate is not locked by the current processor, this function takes no effect. The function RDC_SEMA42_GetGateStatus() returns a status whether the gate is unlocked and which processor locks the gate. The function RDC_SEMA42_GetLockDomainID() returns the ID of the domain which has locked the gate.

The RDC_SEMA42 gate can be reset to unlock forcefully. The function RDC_SEMA42_ResetGate() resets a specific gate. The function RDC_SEMA42_ResetAllGates() resets all gates.

Macros

- #define RDC_SEMA42_GATE_NUM_RESET_ALL (64U)
  The number to reset all RDC_SEMA42 gates.
- #define RDC_SEMA42_GATEn(base, n) (((volatile uint8_t ∗)(&((base)->GATE0)))[(n)])
  RDC_SEMA42 gate n register address.
- #define RDC_SEMA42_GATE_COUNT (64U)
  RDC_SEMA42 gate count.

Functions

- void RDC_SEMA42_Init (RDC_SEMAPHORE_Type ∗base)
  Initializes the RDC_SEMA42 module.
- void RDC_SEMA42_Deinit (RDC_SEMAPHORE_Type ∗base)
  De-initializes the RDC_SEMA42 module.
- status_t RDC_SEMA42_TryLock (RDC_SEMAPHORE_Type ∗base, uint8_t gateNum, uint8_-t masterIndex, uint8_t domainId)
  Tries to lock the RDC_SEMA42 gate.

- void RDC_SEMA42_Lock (RDC_SEMAPHORE_Type *base, uint8_t gateNum, uint8_t masterIndex, uint8_t domainId)
  Locks the RDC_SEMA42 gate.
- static void RDC_SEMA42_Unlock (RDC_SEMAPHORE_Type *base, uint8_t gateNum)
  Unlocks the RDC_SEMA42 gate.
- static int32_t RDC_SEMA42_GetLockMasterIndex (RDC_SEMAPHORE_Type *base, uint8_t gateNum)
  Gets which master has currently locked the gate.
- int32_t RDC_SEMA42_GetLockDomainID (RDC_SEMAPHORE_Type *base, uint8_t gateNum)
  Gets which domain has currently locked the gate.
- status_t RDC_SEMA42_ResetGate (RDC_SEMAPHORE_Type *base, uint8_t gateNum)
  Resets the RDC_SEMA42 gate to an unlocked status.
- static status_t RDC_SEMA42_ResetAllGates (RDC_SEMAPHORE_Type *base)
  Resets all RDC_SEMA42 gates to an unlocked status.

Driver version

- #define FSL_RDC_SEMA42_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  RDC_SEMA42 driver version.

Macro Definition Documentation

18.2.1 #define RDC_SEMA42_GATE_NUM_RESET_ALL (64U)

18.2.2 #define RDC_SEMA42_GATEn(*base*, *n*) (((volatile uint8_t *)(&((base)->GATE0)))[(n)])

18.2.3 #define RDC_SEMA42_GATE_COUNT (64U)

Function Documentation

18.3.1 void RDC_SEMA42_Init (RDC_SEMAPHORE_Type * *base*)

This function initializes the RDC_SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either RDC_SEMA42_ResetGate or RDC_SEMA42_ResetAllGates function.

Parameters

base	RDC_SEMA42 peripheral base address.

18.3.2 void RDC_SEMA42_Deinit (RDC_SEMAPHORE_Type * *base*)

This function de-initializes the RDC_SEMA42 module. It only disables the clock.

Parameters

base	RDC_SEMA42 peripheral base address.

18.3.3 status_t RDC_SEMA42_TryLock (RDC_SEMAPHORE_Type * *base*, uint8_t *gateNum*, uint8_t *masterIndex*, uint8_t *domainId*)

This function tries to lock the specific RDC_SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

base	RDC_SEMA42 peripheral base address.
gateNum	Gate number to lock.
masterIndex	Current processor master index.
domainId	Current processor domain ID.

Return values

kStatus_Success	Lock the sema42 gate successfully.
kStatus_Failed	Sema42 gate has been locked by another processor.

18.3.4 void RDC_SEMA42_Lock (RDC_SEMAPHORE_Type * *base*, uint8_t *gateNum*, uint8_t *masterIndex*, uint8_t *domainId*)

This function locks the specific RDC_SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

Parameters

base	RDC_SEMA42 peripheral base address.
gateNum	Gate number to lock.
masterIndex	Current processor master index.
domainId	Current processor domain ID.

18.3.5 static void RDC_SEMA42_Unlock (RDC_SEMAPHORE_Type * *base*, uint8_t *gateNum*) [inline], [static]

This function unlocks the specific RDC_SEMA42 gate. It only writes unlock value to the RDC_SEMA42 gate register. However, it does not check whether the RDC_SEMA42 gate is locked by the current processor or not. As a result, if the RDC_SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

base	RDC_SEMA42 peripheral base address.
gateNum	Gate number to unlock.

18.3.6 static int32_t RDC_SEMA42_GetLockMasterIndex (RDC_SEMAPHORE_Type * *base*, uint8_t *gateNum*) [inline], [static]

Parameters

base	RDC_SEMA42 peripheral base address.
gateNum	Gate number.

Returns

Return -1 if the gate is not locked by any master, otherwise return the master index.

18.3.7 int32_t RDC_SEMA42_GetLockDomainID (RDC_SEMAPHORE_Type * *base*, uint8_t *gateNum*)

Parameters

base	RDC_SEMA42 peripheral base address.
gateNum	Gate number.

Returns

Return -1 if the gate is not locked by any domain, otherwise return the domain ID.

18.3.8 status_t RDC_SEMA42_ResetGate (RDC_SEMAPHORE_Type * *base*, uint8_t *gateNum*)

This function resets a RDC_SEMA42 gate to an unlocked status.

Parameters

base	RDC_SEMA42 peripheral base address.
gateNum	Gate number.

Return values

kStatus_Success	RDC_SEMA42 gate is reset successfully.
kStatus_Failed	Some other reset process is ongoing.

18.3.9 static status_t RDC_SEMA42_ResetAllGates (RDC_SEMAPHORE_Type * *base*) [inline], [static]

This function resets all RDC_SEMA42 gate to an unlocked status.

Parameters

base	RDC_SEMA42 peripheral base address.

Return values

kStatus_Success	RDC_SEMA42 is reset successfully.
kStatus_RDC_SEMA42_-Reseting	Some other reset process is ongoing.

Chapter 19
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

19.3.1 SAI Send/receive using an interrupt method

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

19.3.2 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

19.4.1 Overview

Data Structures

- struct sai_config_t
  SAI user configuration structure. More...
- struct sai_transfer_format_t
  sai transfer format More...
- struct sai_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_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_data_pin_state_t {
kSAI_DataPinStateTriState,
kSAI_DataPinStateOutputZero = 1U }
sai data pin state definition
- 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_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.

19.4.2 Data Structure Documentation

19.4.2.1 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.
- sai_bclk_source_t bclkSource
  Bit Clock source.
- sai_master_slave_t masterSlave
  Master or slave.

19.4.2.2 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.
- 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.

19.4.2.2.0.21 Field Documentation

19.4.2.2.0.21.1 bool sai_transfer_format_t::isFrameSyncCompact

19.4.2.3 struct sai_fifo_t

Data Fields

- bool fifoContinueOneError
  fifo continues when error occur
- sai_fifo_packing_t fifoPacking
  fifo packing mode
- uint8_t fifoWatermark
  fifo watermark

19.4.2.4 struct sai_bit_clock_t

Data Fields

- bool bclkSrcSwap
  bit clock source swap
- bool bclkInputDelay
  bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .
- sai_clock_polarity_t bclkPolarity
  bit clock polarity
- sai_bclk_source_t bclkSource
  bit Clock source

19.4.2.4.0.22 Field Documentation

19.4.2.4.0.22.1 bool sai_bit_clock_t::bclkInputDelay

19.4.2.5 struct sai_frame_sync_t

Data Fields

- uint8_t frameSyncWidth
  frame sync width in number of bit clocks
- bool frameSyncEarly
  TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame.
- sai_clock_polarity_t frameSyncPolarity
  frame sync polarity

19.4.2.6 struct sai_serial_data_t

Data Fields

- sai_data_pin_state_t dataMode
  sai data pin state when slots masked or channel disabled
- 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

19.4.2.7 struct sai_transceiver_t

Data Fields

- sai_serial_data_t serialData
  serial data configurations
- sai_frame_sync_t frameSync
  ws configurations
- sai_bit_clock_t bitClock
  bit clock configurations
- sai_fifo_t fifo
  fifo configurations
- sai_master_slave_t masterSlave
  transceiver is master or slave
- sai_sync_mode_t syncMode
  transceiver sync mode
- uint8_t startChannel
  Transfer start channel.
- uint8_t channelMask
  enabled channel mask value, reference _sai_channel_mask
- uint8_t endChannel
  end channel number
- uint8_t channelNums
  Total enabled channel numbers.

19.4.2.8 struct sai_transfer_t

Data Fields

- uint8_t * data
  Data start address to transfer.
- size_t dataSize
  Transfer size.

19.4.2.8.0.23 Field Documentation

19.4.2.8.0.23.1 uint8_t* sai_transfer_t::data

19.4.2.8.0.23.2 size_t sai_transfer_t::dataSize

19.4.2.9 struct _sai_handle

Data Fields

- I2S_Type * base
  base address
- uint32_t state
  Transfer status.
- sai_transfer_callback_t callback
  Callback function called at transfer event.
- void * userData
  Callback parameter passed to callback function.
- uint8_t bitWidth
  Bit width for transfer, 8/16/24/32 bits.
- uint8_t channel
  Transfer start channel.
- uint8_t channelMask
  enabled channel mask value, refernece _sai_channel_mask
- uint8_t endChannel
  end channel number
- uint8_t channelNums
  Total enabled channel numbers.
- sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]
  Transfer queue storing queued transfer.
- size_t transferSize [SAI_XFER_QUEUE_SIZE]
  Data bytes need to transfer.
- volatile uint8_t queueUser
  Index for user to queue transfer.
- volatile uint8_t queueDriver
  Index for driver to get the transfer data and size.
- uint8_t watermark
  Watermark value.

19.4.3 Macro Definition Documentation

19.4.3.1 #define SAI_XFER_QUEUE_SIZE (4U)

19.4.4 Enumeration Type Documentation

19.4.4.1 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.

19.4.4.2 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

19.4.4.3 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.

19.4.4.4 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

19.4.4.5 enum sai_mono_stereo_t

Enumerator

kSAI_Stereo Stereo sound.
kSAI_MonoRight Only Right channel have sound.
kSAI_MonoLeft Only left channel have sound.

19.4.4.6 enum sai_data_order_t

Enumerator

kSAI_DataLSB LSB bit transferred first.
kSAI_DataMSB MSB bit transferred first.

19.4.4.7 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.

19.4.4.8 enum sai_sync_mode_t

Enumerator

kSAI_ModeAsync Asynchronous mode.
kSAI_ModeSync Synchronous mode (with receiver or transmit)

19.4.4.9 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.

19.4.4.10 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.

19.4.4.11 anonymous enum

Enumerator

kSAI_FIFOWarningDMAEnable FIFO warning caused by the DMA request.
kSAI_FIFORequestDMAEnable FIFO request caused by the DMA request.

19.4.4.12 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.

19.4.4.13 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.

19.4.4.14 enum sai_fifo_packing_t

Enumerator

kSAI_FifoPackingDisabled Packing disabled.
kSAI_FifoPacking8bit 8 bit packing enabled
kSAI_FifoPacking16bit 16bit packing enabled

19.4.4.15 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.

19.4.4.16 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.

19.4.4.17 enum sai_data_pin_state_t

Enumerator

kSAI_DataPinStateTriState transmit data pins are tri-stated when slots are masked or channels are disabled
kSAI_DataPinStateOutputZero transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

19.4.4.18 enum sai_transceiver_type_t

Enumerator

kSAI_Transmitter sai transmitter
kSAI_Receiver sai receiver

19.4.4.19 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.

19.4.5 Function Documentation

19.4.5.1 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.

19.4.5.2 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.

19.4.5.3 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

19.4.5.4 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

19.4.5.5 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.

19.4.5.6 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.

19.4.5.7 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

19.4.5.8 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

19.4.5.9 void SAI_TxEnable (I2S_Type * *base,* bool *enable*)

Parameters

base	SAI base pointer.
enable	True means enable SAI Tx, false means disable.

19.4.5.10 void SAI_RxEnable (I2S_Type * *base,* bool *enable*)

Parameters

base	SAI base pointer.
enable	True means enable SAI Rx, false means disable.

19.4.5.11 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.

19.4.5.12 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.

19.4.5.13 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.

19.4.5.14 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.

19.4.5.15 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.

19.4.5.16 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.

19.4.5.17 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.

19.4.5.18 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.

19.4.5.19 void SAI_TxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

base	SAI base pointer.
config	fifo configurations.

19.4.5.20 void SAI_RxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

base	SAI base pointer.
config	fifo configurations.

19.4.5.21 void SAI_TxSetFrameSyncConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_frame_sync_t * *config*)

Parameters

base	SAI base pointer.
masterSlave	master or slave.
config	frame sync configurations, can be NULL in slave mode.

19.4.5.22 void SAI_RxSetFrameSyncConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_frame_sync_t * *config*)

Parameters

base	SAI base pointer.
masterSlave	master or slave.
config	frame sync configurations, can be NULL in slave mode.

19.4.5.23 void SAI_TxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

base	SAI base pointer.
config	serial data configurations.

19.4.5.24 void SAI_RxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

base	SAI base pointer.
config	serial data configurations.

19.4.5.25 void SAI_TxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

base	SAI base pointer.
config	transmitter configurations.

19.4.5.26 void SAI_RxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

base	SAI base pointer.
config	receiver configurations.

19.4.5.27 void SAI_GetClassicI2SConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

config	transceiver configurations.
bitWidth	audio data bitWidth.
mode	audio data channel.
saiChannel-Mask	mask value of the channel to be enable.

19.4.5.28 void SAI_GetLeftJustifiedConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

config	transceiver configurations.
bitWidth	audio data bitWidth.
mode	audio data channel.
saiChannel-Mask	mask value of the channel to be enable.

19.4.5.29 void SAI_GetRightJustifiedConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

config	transceiver configurations.
bitWidth	audio data bitWidth.
mode	audio data channel.
saiChannel-Mask	mask value of the channel to be enable.

19.4.5.30 void SAI_GetTDMConfig (sai_transceiver_t ∗ *config*, sai_frame_sync_len_t *frameSyncWidth*, sai_word_width_t *bitWidth*, uint32_t *dataWordNum*, uint32_t *saiChannelMask*)

Parameters

config	transceiver configurations.
frameSync-Width	length of frame sync.
bitWidth	audio data word width.
dataWordNum	word number in one frame.
saiChannel-Mask	mask value of the channel to be enable.

19.4.5.31 void SAI_GetDSPConfig (sai_transceiver_t ∗ *config*, sai_frame_sync_len_t *frameSyncWidth*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

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.

19.4.5.32 static uint32_t SAI_TxGetStatusFlag (I2S_Type * *base*) [inline], [static]

Parameters

base	SAI base pointer

Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

19.4.5.33 static void SAI_TxClearStatusFlags (I2S_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	State mask. It can be a combination of the following source if defined: • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

19.4.5.34 static uint32_t SAI_RxGetStatusFlag (I2S_Type * *base*) [inline], [static]

Parameters

base	SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

19.4.5.35 static void SAI_RxClearStatusFlags (I2S_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	State mask. It can be a combination of the following sources if defined. • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

19.4.5.36 void SAI_TxSoftwareReset (I2S_Type ∗ *base*, sai_reset_type_t *type*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1∼TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

base	SAI base pointer
type	Reset type, FIFO reset or software reset

19.4.5.37 void SAI_RxSoftwareReset (I2S_Type ∗ *base*, sai_reset_type_t *type*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1∼RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

base	SAI base pointer
type	Reset type, FIFO reset or software reset

19.4.5.38 void SAI_TxSetChannelFIFOMask (I2S_Type * *base*, uint8_t *mask*)

Parameters

base	SAI base pointer
mask	Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

19.4.5.39 void SAI_RxSetChannelFIFOMask (I2S_Type * *base*, uint8_t *mask*)

Parameters

base	SAI base pointer
mask	Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

19.4.5.40 void SAI_TxSetDataOrder (I2S_Type * *base*, sai_data_order_t *order*)

Parameters

base	SAI base pointer
order	Data order MSB or LSB

19.4.5.41 void SAI_RxSetDataOrder (I2S_Type * *base*, sai_data_order_t *order*)

Parameters

base	SAI base pointer

order	Data order MSB or LSB

19.4.5.42 void SAI_TxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
polarity	

19.4.5.43 void SAI_RxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
polarity	

19.4.5.44 void SAI_TxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
polarity	

19.4.5.45 void SAI_RxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
polarity	

19.4.5.46 void SAI_TxSetFIFOPacking (I2S_Type * *base*, sai_fifo_packing_t *pack*)

Parameters

base	SAI base pointer.
pack	FIFO pack type. It is element of sai_fifo_packing_t.

19.4.5.47 void SAI_RxSetFIFOPacking (I2S_Type * *base*, sai_fifo_packing_t *pack*)

Parameters

base	SAI base pointer.
pack	FIFO pack type. It is element of sai_fifo_packing_t.

19.4.5.48 static void SAI_TxSetFIFOErrorContinue (I2S_Type * *base*, bool *isEnabled*) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

base	SAI base pointer.
isEnabled	Is FIFO error continue enabled, true means enable, false means disable.

19.4.5.49 static void SAI_RxSetFIFOErrorContinue (I2S_Type * *base*, bool *isEnabled*) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

base	SAI base pointer.
isEnabled	Is FIFO error continue enabled, true means enable, false means disable.

19.4.5.50 static void SAI_TxEnableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

19.4.5.51 static void SAI_RxEnableInterrupts (I2S_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

19.4.5.52 static void SAI_TxDisableInterrupts (I2S_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

19.4.5.53 static void SAI_RxDisableInterrupts (I2S_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	SAI base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

19.4.5.54 static void SAI_TxEnableDMA (I2S_Type * *base,* uint32_t *mask,* bool *enable*) [inline], [static]

Parameters

base	SAI base pointer
mask	DMA source The parameter can be combination of the following sources if defined. • kSAI_FIFOWarningDMAEnable • kSAI_FIFORequestDMAEnable
enable	True means enable DMA, false means disable DMA.

19.4.5.55 static void SAI_RxEnableDMA (I2S_Type * *base,* uint32_t *mask,* bool *enable*) [inline], [static]

Parameters

base	SAI base pointer

mask	DMA source The parameter can be a combination of the following sources if defined. • kSAI_FIFOWarningDMAEnable • kSAI_FIFORequestDMAEnable
enable	True means enable DMA, false means disable DMA.

19.4.5.56 static uint32_t SAI_TxGetDataRegisterAddress (I2S_Type * *base*, uint32_t *channel*) [inline], [static]

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

Parameters

base	SAI base pointer.
channel	Which data channel used.

Returns

data register address.

19.4.5.57 static uint32_t SAI_RxGetDataRegisterAddress (I2S_Type * *base*, uint32_t *channel*) [inline], [static]

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

Parameters

base	SAI base pointer.
channel	Which data channel used.

Returns

data register address.

19.4.5.58 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.

19.4.5.59 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.

19.4.5.60 void SAI_WriteBlocking (I2S_Type ∗ *base,* uint32_t *channel,* uint32_t *bitWidth,* uint8_t ∗ *buffer,* uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

base	SAI base pointer.
channel	Data channel used.
bitWidth	How many bits in an audio word; usually 8/16/24/32 bits.
buffer	Pointer to the data to be written.
size	Bytes to be written.

19.4.5.61 void SAI_WriteMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

base	SAI base pointer.
channel	Data channel used.
channelMask	channel mask.
bitWidth	How many bits in an audio word; usually 8/16/24/32 bits.
buffer	Pointer to the data to be written.
size	Bytes to be written.

19.4.5.62 static void SAI_WriteData (I2S_Type * *base*, uint32_t *channel*, uint32_t *data*) [inline], [static]

Parameters

base	SAI base pointer.
channel	Data channel used.
data	Data needs to be written.

19.4.5.63 void SAI_ReadBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

base	SAI base pointer.
channel	Data channel used.
bitWidth	How many bits in an audio word; usually 8/16/24/32 bits.
buffer	Pointer to the data to be read.
size	Bytes to be read.

19.4.5.64 void SAI_ReadMultiChannelBlocking (I2S_Type * *base,* uint32_t *channel,* uint32_t *channelMask,* uint32_t *bitWidth,* uint8_t * *buffer,* uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

base	SAI base pointer.
channel	Data channel used.
channelMask	channel mask.
bitWidth	How many bits in an audio word; usually 8/16/24/32 bits.
buffer	Pointer to the data to be read.
size	Bytes to be read.

19.4.5.65 static uint32_t SAI_ReadData (I2S_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	SAI base pointer.
channel	Data channel used.

Returns

Data in SAI FIFO.

19.4.5.66 void SAI_TransferTxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

base	SAI base pointer
handle	SAI handle pointer.
callback	Pointer to the user callback function.
userData	User parameter passed to the callback function

19.4.5.67 void SAI_TransferRxCreateHandle (I2S_Type ∗ *base,* sai_handle_t ∗ *handle,* sai_transfer_callback_t *callback,* void ∗ *userData*)

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

base	SAI base pointer.
handle	SAI handle pointer.
callback	Pointer to the user callback function.
userData	User parameter passed to the callback function.

19.4.5.68 void SAI_TransferTxSetConfig (I2S_Type ∗ *base,* sai_handle_t ∗ *handle,* sai_transceiver_t ∗ *config*)

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

base	SAI base pointer.
handle	SAI handle pointer.
config	tranmitter configurations.

19.4.5.69 void SAI_TransferRxSetConfig (I2S_Type ∗ *base,* sai_handle_t ∗ *handle,* sai_transceiver_t ∗ *config*)

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

base	SAI base pointer.
handle	SAI handle pointer.
config	receiver configurations.

19.4.5.70 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.

19.4.5.71 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.

19.4.5.72 status_t SAI_TransferSendNonBlocking (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*, sai_transfer_t ∗ *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_-SAI_Busy, the transfer is finished.

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure which stores the transfer state.
xfer	Pointer to the sai_transfer_t structure.

Return values

kStatus_Success	Successfully started the data receive.
kStatus_SAI_TxBusy	Previous receive still not finished.
kStatus_InvalidArgument	The input parameter is invalid.

19.4.5.73 status_t SAI_TransferReceiveNonBlocking (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*, sai_transfer_t ∗ *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_-SAI_Busy, the transfer is finished.

Parameters

base	SAI base pointer
handle	Pointer to the sai_handle_t structure which stores the transfer state.
xfer	Pointer to the sai_transfer_t structure.

Return values

kStatus_Success	Successfully started the data receive.
kStatus_SAI_RxBusy	Previous receive still not finished.
kStatus_InvalidArgument	The input parameter is invalid.

19.4.5.74 status_t SAI_TransferGetSendCount (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure which stores the transfer state.
count	Bytes count sent.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

19.4.5.75 status_t SAI_TransferGetReceiveCount (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure which stores the transfer state.
count	Bytes count received.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

19.4.5.76 void SAI_TransferAbortSend (I2S_Type * *base*, sai_handle_t * *handle*)

Note

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

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure which stores the transfer state.

19.4.5.77 void SAI_TransferAbortReceive (I2S_Type * *base*, sai_handle_t * *handle*)

Note

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

Parameters

base	SAI base pointer
handle	Pointer to the sai_handle_t structure which stores the transfer state.

19.4.5.78 void SAI_TransferTerminateSend (I2S_Type * *base*, sai_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.

19.4.5.79 void SAI_TransferTerminateReceive (I2S_Type * *base*, sai_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

base	SAI base pointer.
handle	SAI eDMA handle pointer.

19.4.5.80 void SAI_TransferTxHandleIRQ (I2S_Type * *base*, sai_handle_t * *handle*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure.

19.4.5.81 void SAI_TransferRxHandleIRQ (I2S_Type * *base*, sai_handle_t * *handle*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure.

Chapter 20
SEMA4: Hardware Semaphores Driver

Overview

The MCUXpresso SDK provides a driver for the SEMA4 module of MCUXpresso SDK devices.

Macros

- #define SEMA4_GATE_NUM_RESET_ALL (64U)
  The number to reset all SEMA4 gates.
- #define SEMA4_GATEn(base, n) (((volatile uint8_t *)(&((base)->Gate00)))[(n)])
  SEMA4 gate n register address.

Functions

- void SEMA4_Init (SEMA4_Type *base)
  Initializes the SEMA4 module.
- void SEMA4_Deinit (SEMA4_Type *base)
  De-initializes the SEMA4 module.
- status_t SEMA4_TryLock (SEMA4_Type *base, uint8_t gateNum, uint8_t procNum)
  Tries to lock the SEMA4 gate.
- void SEMA4_Lock (SEMA4_Type *base, uint8_t gateNum, uint8_t procNum)
  Locks the SEMA4 gate.
- static void SEMA4_Unlock (SEMA4_Type *base, uint8_t gateNum)
  Unlocks the SEMA4 gate.
- static int32_t SEMA4_GetLockProc (SEMA4_Type *base, uint8_t gateNum)
  Gets the status of the SEMA4 gate.
- status_t SEMA4_ResetGate (SEMA4_Type *base, uint8_t gateNum)
  Resets the SEMA4 gate to an unlocked status.
- static status_t SEMA4_ResetAllGates (SEMA4_Type *base)
  Resets all SEMA4 gates to an unlocked status.
- static void SEMA4_EnableGateNotifyInterrupt (SEMA4_Type *base, uint8_t procNum, uint32_t mask)
  Enable the gate notification interrupt.
- static void SEMA4_DisableGateNotifyInterrupt (SEMA4_Type *base, uint8_t procNum, uint32_t mask)
  Disable the gate notification interrupt.
- static uint32_t SEMA4_GetGateNotifyStatus (SEMA4_Type *base, uint8_t procNum)
  Get the gate notification flags.
- status_t SEMA4_ResetGateNotify (SEMA4_Type *base, uint8_t gateNum)
  Resets the SEMA4 gate IRQ notification.
- static status_t SEMA4_ResetAllGateNotify (SEMA4_Type *base)
  Resets all SEMA4 gates IRQ notification.

Driver version

- #define FSL_SEMA4_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  SEMA4 driver version.

Macro Definition Documentation

20.2.1 #define SEMA4_GATE_NUM_RESET_ALL (64U)

Function Documentation

20.3.1 void SEMA4_Init (SEMA4_Type * *base*)

This function initializes the SEMA4 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA4_-ResetGate or SEMA4_ResetAllGates function.

Parameters

base	SEMA4 peripheral base address.

20.3.2 void SEMA4_Deinit (SEMA4_Type * *base*)

This function de-initializes the SEMA4 module. It only disables the clock.

Parameters

base	SEMA4 peripheral base address.

20.3.3 status_t SEMA4_TryLock (SEMA4_Type * *base,* uint8_t *gateNum,* uint8_t *procNum*)

This function tries to lock the specific SEMA4 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

base	SEMA4 peripheral base address.

gateNum	Gate number to lock.
procNum	Current processor number.

Return values

kStatus_Success	Lock the sema4 gate successfully.
kStatus_Fail	Sema4 gate has been locked by another processor.

20.3.4 void SEMA4_Lock (SEMA4_Type ∗ *base,* uint8_t *gateNum,* uint8_t *procNum*)

This function locks the specific SEMA4 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

Parameters

base	SEMA4 peripheral base address.
gateNum	Gate number to lock.
procNum	Current processor number.

20.3.5 static void SEMA4_Unlock (SEMA4_Type ∗ *base,* uint8_t *gateNum*) [inline], [static]

This function unlocks the specific SEMA4 gate. It only writes unlock value to the SEMA4 gate register. However, it does not check whether the SEMA4 gate is locked by the current processor or not. As a result, if the SEMA4 gate is not locked by the current processor, this function has no effect.

Parameters

base	SEMA4 peripheral base address.
gateNum	Gate number to unlock.

20.3.6 static int32_t SEMA4_GetLockProc (SEMA4_Type ∗ *base,* uint8_t *gateNum*) [inline], [static]

This function checks the lock status of a specific SEMA4 gate.

Parameters

base	SEMA4 peripheral base address.
gateNum	Gate number.

Returns

Return -1 if the gate is unlocked, otherwise return the processor number which has locked the gate.

20.3.7 status_t SEMA4_ResetGate (SEMA4_Type * *base*, uint8_t *gateNum*)

This function resets a SEMA4 gate to an unlocked status.

Parameters

base	SEMA4 peripheral base address.
gateNum	Gate number.

Return values

kStatus_Success	SEMA4 gate is reset successfully.
kStatus_Fail	Some other reset process is ongoing.

20.3.8 static status_t SEMA4_ResetAllGates (SEMA4_Type * *base*) [inline], [static]

This function resets all SEMA4 gate to an unlocked status.

Parameters

base	SEMA4 peripheral base address.

Return values

kStatus_Success	SEMA4 is reset successfully.

kStatus_Fail	Some other reset process is ongoing.

20.3.9 static void SEMA4_EnableGateNotifyInterrupt (SEMA4_Type ∗ *base*, uint8_t *procNum*, uint32_t *mask*) [inline], [static]

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

base	SEMA4 peripheral base address.
procNum	Current processor number.
mask	OR'ed value of the gate index, for example: (1<<0) | (1<<1) means gate 0 and gate 1.

20.3.10 static void SEMA4_DisableGateNotifyInterrupt (SEMA4_Type ∗ *base*, uint8_t *procNum*, uint32_t *mask*) [inline], [static]

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

base	SEMA4 peripheral base address.
procNum	Current processor number.
mask	OR'ed value of the gate index, for example: (1<<0) | (1<<1) means gate 0 and gate 1.

20.3.11 static uint32_t SEMA4_GetGateNotifyStatus (SEMA4_Type ∗ *base*, uint8_t *procNum*) [inline], [static]

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle. The status flags are cleared automatically when the gate is locked by current core or locked again before the other core.

Parameters

base	SEMA4 peripheral base address.
procNum	Current processor number.

Returns

OR'ed value of the gate index, for example: (1<<0) | (1<<1) means gate 0 and gate 1 flags are pending.

20.3.12 status_t SEMA4_ResetGateNotify (SEMA4_Type ∗ *base,* uint8_t *gateNum*)

This function resets a SEMA4 gate IRQ notification.

Parameters

base	SEMA4 peripheral base address.
gateNum	Gate number.

Return values

kStatus_Success	Reset successfully.
kStatus_Fail	Some other reset process is ongoing.

20.3.13 static status_t SEMA4_ResetAllGateNotify (SEMA4_Type ∗ *base*) [inline], [static]

This function resets all SEMA4 gate IRQ notifications.

Parameters

base	SEMA4 peripheral base address.

Return values

kStatus_Success	Reset successfully.
kStatus_Fail	Some other reset process is ongoing.

Chapter 21
TMU: Thermal Management Unit Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the thermal management unit (TMU) module of MCUXpresso SDK devices.

Typical use case

21.2.1 Monitor and report Configuration

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

Data Structures

- struct tmu_thresold_config_t
  configuration for TMU thresold. More...
- struct tmu_interrupt_status_t
  TMU interrupt status. More...
- struct tmu_config_t
  Configuration for TMU module. More...

Macros

- #define FSL_TMU_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  TMU driver version.

Enumerations

- enum _tmu_interrupt_enable {
  kTMU_ImmediateTemperatureInterruptEnable,
  kTMU_AverageTemperatureInterruptEnable,
  kTMU_AverageTemperatureCriticalInterruptEnable }
  TMU interrupt enable.
- enum _tmu_interrupt_status_flags {
  kTMU_ImmediateTemperatureStatusFlags = TMU_TIDR_ITTE_MASK,
  kTMU_AverageTemperatureStatusFlags = TMU_TIDR_ATTE_MASK,
  kTMU_AverageTemperatureCriticalStatusFlags }
  TMU interrupt status flags.
- enum _tmu_status_flags {
  kTMU_IntervalExceededStatusFlags = TMU_TSR_MIE_MASK,
  kTMU_OutOfLowRangeStatusFlags = TMU_TSR_ORL_MASK,
  kTMU_OutOfHighRangeStatusFlags }

TMU status flags.
- enum tmu_average_low_pass_filter_t {
kTMU_AverageLowPassFilter1_0 = 0U,
kTMU_AverageLowPassFilter0_5 = 1U,
kTMU_AverageLowPassFilter0_25 = 2U,
kTMU_AverageLowPassFilter0_125 = 3U }
  Average low pass filter setting.

Functions

- void TMU_Init (TMU_Type ∗base, const tmu_config_t ∗config)
  Enable the access to TMU registers and Initialize TMU module.
- void TMU_Deinit (TMU_Type ∗base)
  De-initialize TMU module and Disable the access to DCDC registers.
- void TMU_GetDefaultConfig (tmu_config_t ∗config)
  Gets the default configuration for TMU.
- static void TMU_Enable (TMU_Type ∗base, bool enable)
  Enable/Disable the TMU module.
- static void TMU_EnableInterrupts (TMU_Type ∗base, uint32_t mask)
  Enable the TMU interrupts.
- static void TMU_DisableInterrupts (TMU_Type ∗base, uint32_t mask)
  Disable the TMU interrupts.
- void TMU_GetInterruptStatusFlags (TMU_Type ∗base, tmu_interrupt_status_t ∗status)
  Get interrupt status flags.
- void TMU_ClearInterruptStatusFlags (TMU_Type ∗base, uint32_t mask)
  Clear interrupt status flags and corresponding interrupt critical site capture register.
- static uint32_t TMU_GetStatusFlags (TMU_Type ∗base)
  Get TMU status flags.
- status_t TMU_GetHighestTemperature (TMU_Type ∗base, uint32_t ∗temperature)
  Get the highest temperature reached for any enabled monitored site within the temperature sensor range.
- status_t TMU_GetLowestTemperature (TMU_Type ∗base, uint32_t ∗temperature)
  Get the lowest temperature reached for any enabled monitored site within the temperature sensor range.
- status_t TMU_GetImmediateTemperature (TMU_Type ∗base, uint32_t siteIndex, uint32_-t ∗temperature)
  Get the last immediate temperature at site n.
- status_t TMU_GetAverageTemperature (TMU_Type ∗base, uint32_t siteIndex, uint32_t ∗temperature)
  Get the last average temperature at site n.
- void TMU_SetHighTemperatureThresold (TMU_Type ∗base, const tmu_thresold_config_t ∗config)
  Configure the high temperature thresold value and enable/disable relevant thresold.

Data Structure Documentation

21.3.1 struct tmu_thresold_config_t

Data Fields

- bool immediateThresoldEnable
  Enable high temperature immediate threshold.
- bool AverageThresoldEnable

Enable high temperature average threshold.
- bool AverageCriticalThresoldEnable
  Enable high temperature average critical threshold.
- uint8_t immediateThresoldValue
  Range:0U-125U.
- uint8_t averageThresoldValue
  Range:0U-125U.
- uint8_t averageCriticalThresoldValue
  Range:0U-125U.

21.3.1.0.0.24 Field Documentation

21.3.1.0.0.24.1 bool tmu_thresold_config_t::immediateThresoldEnable

21.3.1.0.0.24.2 bool tmu_thresold_config_t::AverageThresoldEnable

21.3.1.0.0.24.3 bool tmu_thresold_config_t::AverageCriticalThresoldEnable

21.3.1.0.0.24.4 uint8_t tmu_thresold_config_t::immediateThresoldValue

Valid when corresponding thresold is enabled. High temperature immediate threshold value. Determines the current upper temperature threshold, for anyenabled monitored site.

21.3.1.0.0.24.5 uint8_t tmu_thresold_config_t::averageThresoldValue

Valid when corresponding thresold is enabled. High temperature average threshold value. Determines the average upper temperature threshold, for any enabled monitored site.

21.3.1.0.0.24.6 uint8_t tmu_thresold_config_t::averageCriticalThresoldValue

Valid when corresponding thresold is enabled. High temperature average critical threshold value. Determines the average upper critical temperature threshold, for any enabled monitored site.

21.3.2 struct tmu_interrupt_status_t

Data Fields

- uint32_t interruptDetectMask
  The mask of interrupt status flags.
- uint16_t immediateInterruptsSiteMask
  The mask of the temperature sensor site associated with a detected ITTE event.
- uint16_t AverageInterruptsSiteMask
  The mask of the temperature sensor site associated with a detected ATTE event.
- uint16_t AverageCriticalInterruptsSiteMask
  The mask of the temperature sensor site associated with a detected ATCTE event.

21.3.2.0.0.25 Field Documentation

21.3.2.0.0.25.1 uint32_t tmu_interrupt_status_t::interruptDetectMask

Refer to "_tmu_interrupt_status_flags" enumeration.

21.3.2.0.0.25.2 uint16_t tmu_interrupt_status_t::immediateInterruptsSiteMask

Please refer to "_tmu_monitor_site" enumeration.

21.3.2.0.0.25.3 uint16_t tmu_interrupt_status_t::AverageInterruptsSiteMask

Please refer to "_tmu_monitor_site" enumeration.

21.3.2.0.0.25.4 uint16_t tmu_interrupt_status_t::AverageCriticalInterruptsSiteMask

Please refer to "_tmu_monitor_site" enumeration.

21.3.3 struct tmu_config_t

Data Fields

- uint8_t monitorInterval
    Temperature monitoring interval in seconds.
- uint16_t monitorSiteSelection
    By setting the select bit for a temperature sensor site, it is enabled and included in all monitoring functions.
- tmu_average_low_pass_filter_t averageLPF
    The average temperature is calculated as: ALPF x Current_Temp + (1 - ALPF) x Average_Temp.

21.3.3.0.0.26 Field Documentation

21.3.3.0.0.26.1 uint8_t tmu_config_t::monitorInterval

Please refer to specific table in RM.

21.3.3.0.0.26.2 uint16_t tmu_config_t::monitorSiteSelection

If no site is selected, site 0 is monitored by default. Refer to "_tmu_monitor_site" enumeration. Please look up relevant table in reference manual.

21.3.3.0.0.26.3 tmu_average_low_pass_filter_t tmu_config_t::averageLPF

For proper operation, this field should only change when monitoring is disabled.

Macro Definition Documentation

21.4.1 #define FSL_TMU_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

Version 2.0.3.

Enumeration Type Documentation

21.5.1 enum _tmu_interrupt_enable

Enumerator

kTMU_ImmediateTemperatureInterruptEnable Immediate temperature threshold exceeded interrupt enable.
kTMU_AverageTemperatureInterruptEnable Average temperature threshold exceeded interrupt enable.
kTMU_AverageTemperatureCriticalInterruptEnable Average temperature critical threshold exceeded interrupt enable. >

21.5.2 enum _tmu_interrupt_status_flags

Enumerator

kTMU_ImmediateTemperatureStatusFlags Immediate temperature threshold exceeded(ITTE).
kTMU_AverageTemperatureStatusFlags Average temperature threshold exceeded(ATTE).
kTMU_AverageTemperatureCriticalStatusFlags Average temperature critical threshold exceeded. (ATCTE)

21.5.3 enum _tmu_status_flags

Enumerator

kTMU_IntervalExceededStatusFlags Monitoring interval exceeded. The time required to perform measurement of all monitored sites has exceeded the monitoring interval as defined by TMTMIR.
kTMU_OutOfLowRangeStatusFlags Out-of-range low temperature measurement detected. A temperature sensor detected a temperature reading below the lowest measurable temperature of 0 °C.
kTMU_OutOfHighRangeStatusFlags Out-of-range high temperature measurement detected. A temperature sensor detected a temperature reading above the highest measurable temperature of 125 °C.

21.5.4 enum tmu_average_low_pass_filter_t

Enumerator

kTMU_AverageLowPassFilter1_0 Average low pass filter = 1.
kTMU_AverageLowPassFilter0_5 Average low pass filter = 0.5.
kTMU_AverageLowPassFilter0_25 Average low pass filter = 0.25.
kTMU_AverageLowPassFilter0_125 Average low pass filter = 0.125.

Function Documentation

21.6.1 void TMU_Init (TMU_Type ∗ *base*, const tmu_config_t ∗ *config*)

Parameters

base	TMU peripheral base address.
config	Pointer to configuration structure. Refer to "tmu_config_t" structure.

21.6.2 void TMU_Deinit (TMU_Type ∗ *base*)

Parameters

base	TMU peripheral base address.

21.6.3 void TMU_GetDefaultConfig (tmu_config_t ∗ *config*)

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

Example:

```
config->monitorInterval = 0U;
config->monitorSiteSelection = 0U;
config->averageLPF = kTMU_AverageLowPassFilter1_0;
```

Parameters

config	Pointer to TMU configuration structure.

21.6.4 static void TMU_Enable (TMU_Type ∗ *base*, bool *enable*) [inline], [static]

Parameters

base	TMU peripheral base address.
enable	Switcher to enable/disable TMU.

21.6.5 static void TMU_EnableInterrupts (TMU_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	TMU peripheral base address.
mask	The interrupt mask. Refer to "_tmu_interrupt_enable" enumeration.

21.6.6 static void TMU_DisableInterrupts (TMU_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	TMU peripheral base address.
mask	The interrupt mask. Refer to "_tmu_interrupt_enable" enumeration.

21.6.7 void TMU_GetInterruptStatusFlags (TMU_Type * *base*, tmu_interrupt_status_t * *status*)

Parameters

base	TMU peripheral base address.
status	The pointer to interrupt status structure. Record the current interrupt status. Please refer to "tmu_interrupt_status_t" structure.

21.6.8 void TMU_ClearInterruptStatusFlags (TMU_Type * *base*, uint32_t *mask*)

Parameters

base	TMU peripheral base address.
mask	The mask of interrupt status flags. Refer to "_tmu_interrupt_status_flags" enumeration.

21.6.9 static uint32_t TMU_GetStatusFlags (TMU_Type * *base*) [inline], [static]

Parameters

base	TMU peripheral base address.

Returns

The mask of status flags. Refer to "_tmu_status_flags" enumeration.

21.6.10 status_t TMU_GetHighestTemperature (TMU_Type * *base*, uint32_t * *temperature*)

Parameters

base	TMU peripheral base address.
temperature	Highest temperature recorded in degrees Celsius by any enabled monitored site.

Returns

Execution status.

Return values

kStatus_Success	Temperature reading is valid.
kStatus_Fail	Temperature reading is not valid due to no measured temperature within the sensor range of 0-125 °C for an enabled monitored site.

21.6.11 status_t TMU_GetLowestTemperature (TMU_Type * *base*, uint32_t * *temperature*)

Parameters

base	TMU peripheral base address.
temperature	Lowest temperature recorded in degrees Celsius by any enabled monitored site.

Returns

Execution status.

Return values

kStatus_Success	Temperature reading is valid.
kStatus_Fail	Temperature reading is not valid due to no measured temperature within the sensor range of 0-125 °C for an enabled monitored site.

21.6.12 status_t TMU_GetImmediateTemperature (TMU_Type * *base*, uint32_t *siteIndex*, uint32_t * *temperature*)

The site must be part of the list of enabled monitored sites as defined by monitorSiteSelection in "tmu_-config_t" structure.

Parameters

base	TMU peripheral base address.
siteIndex	The index of the site user want to read. 0U: site0 ~ 15U: site15.
temperature	Last immediate temperature reading at site n .

Returns

Execution status.

Return values

kStatus_Success	Temperature reading is valid.
kStatus_Fail	Temperature reading is not valid because temperature out of sensor range or first measurement still pending.

21.6.13 status_t TMU_GetAverageTemperature (TMU_Type * *base*, uint32_t *siteIndex*, uint32_t * *temperature*)

The site must be part of the list of enabled monitored sites as defined by monitorSiteSelection in "tmu_-config_t" structure.

Parameters

base	TMU peripheral base address.
siteIndex	The index of the site user want to read. 0U: site0 ∼ 15U: site15.
temperature	Last average temperature reading at site n .

Returns

Execution status.

Return values

kStatus_Success	Temperature reading is valid.
kStatus_Fail	Temperature reading is not valid because temperature out of sensor range or first measurement still pending.

21.6.14 void TMU_SetHighTemperatureThresold (TMU_Type ∗ *base,* const tmu_thresold_config_t ∗ *config*)

Parameters

base	TMU peripheral base address.
config	Pointer to configuration structure. Refer to "tmu_thresold_config_t" structure.

Chapter 22
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...

Enumerations

- enum _wdog_interrupt_enable { kWDOG_InterruptEnable = WDOG_WICR_WIE_MASK }
  WDOG interrupt configuration structure, default settings all disabled.
- enum _wdog_status_flags {
  kWDOG_RunningFlag = WDOG_WCR_WDE_MASK,
  kWDOG_PowerOnResetFlag = WDOG_WRSR_POR_MASK,
  kWDOG_TimeoutResetFlag = WDOG_WRSR_TOUT_MASK,
  kWDOG_SoftwareResetFlag = WDOG_WRSR_SFTW_MASK,
  kWDOG_InterruptFlag = WDOG_WICR_WTIS_MASK }
  WDOG status flags.

Driver version

- #define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  Defines WDOG driver version.

Refresh sequence

- #define **WDOG_REFRESH_KEY** (0xAAAA5555U)

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.
- 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_TriggerSystemSoftwareReset (WDOG_Type *base)
  Trigger the system software reset.
- static void WDOG_TriggerSoftwareSignal (WDOG_Type *base)
  Trigger an output assertion.
- static void WDOG_EnableInterrupts (WDOG_Type *base, uint16_t mask)
  Enables the WDOG interrupt.
- uint16_t WDOG_GetStatusFlags (WDOG_Type *base)
  Gets the WDOG all reset status flags.
- void WDOG_ClearInterruptStatus (WDOG_Type *base, uint16_t mask)
  Clears the WDOG flag.
- static void WDOG_SetTimeoutValue (WDOG_Type *base, uint16_t timeoutCount)
  Sets the WDOG timeout value.
- static void WDOG_SetInterrputTimeoutValue (WDOG_Type *base, uint16_t timeoutCount)
  Sets the WDOG interrupt count timeout value.
- static void WDOG_DisablePowerDownEnable (WDOG_Type *base)
  Disable the WDOG power down enable bit.
- void WDOG_Refresh (WDOG_Type *base)
  Refreshes the WDOG timer.

Data Structure Documentation

22.3.1 struct wdog_work_mode_t

Data Fields

- bool enableWait
  continue or suspend WDOG in wait mode
- bool enableStop
  continue or suspend WDOG in stop mode
- bool enableDebug
  continue or suspend WDOG in debug mode

22.3.2 struct wdog_config_t

Data Fields

- bool enableWdog
  Enables or disables WDOG.
- wdog_work_mode_t workMode
  Configures WDOG work mode in debug stop and wait mode.
- bool enableInterrupt

Enables or disables WDOG interrupt.
- uint16_t timeoutValue
  Timeout value.
- uint16_t interruptTimeValue
  Interrupt count timeout value.
- bool softwareResetExtension
  software reset extension
- bool enablePowerDown
  power down enable bit
- bool enableTimeOutAssert
  Enable WDOG_B timeout assertion.

22.3.2.0.0.27 Field Documentation

22.3.2.0.0.27.1 bool wdog_config_t::enableTimeOutAssert

Enumeration Type Documentation

22.4.1 enum _wdog_interrupt_enable

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

Enumerator

kWDOG_InterruptEnable WDOG timeout generates an interrupt before reset.

22.4.2 enum _wdog_status_flags

This structure contains the WDOG status flags for use in the WDOG functions.

Enumerator

kWDOG_RunningFlag Running flag, set when WDOG is enabled.
kWDOG_PowerOnResetFlag Power On flag, set when reset is the result of a powerOnReset.
kWDOG_TimeoutResetFlag Timeout flag, set when reset is the result of a timeout.
kWDOG_SoftwareResetFlag Software flag, set when reset is the result of a software.
kWDOG_InterruptFlag interrupt flag,whether interrupt has occurred or not

Function Documentation

22.5.1 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->workMode.enableWait = true;
*   wdogConfig->workMode.enableStop = false;
```

```
*    wdogConfig->workMode.enableDebug = false;
*    wdogConfig->enableInterrupt = false;
*    wdogConfig->enablePowerdown = false;
*    wdogConfig->resetExtension = flase;
*    wdogConfig->timeoutValue = 0xFFU;
*    wdogConfig->interruptTimeValue = 0x04u;
*
```

Parameters

config	Pointer to the WDOG configuration structure.

See Also

wdog_config_t

22.5.2 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.

This is an example.

```
*    wdog_config_t config;
*    WDOG_GetDefaultConfig(&config);
*    config.timeoutValue = 0xffU;
*    config->interruptTimeValue = 0x04u;
*    WDOG_Init(wdog_base,&config);
*
```

Parameters

base	WDOG peripheral base address
config	The configuration of WDOG

22.5.3 void WDOG_Deinit (WDOG_Type ∗ *base*)

This function shuts down the WDOG. Watchdog Enable bit is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. This bit(WDE) can be set/reset only in debug mode(exception).

22.5.4 static void WDOG_Enable (WDOG_Type ∗ *base*) [inline], [static]

This function writes a value into the WDOG_WCR register to enable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception.

Parameters

base	WDOG peripheral base address

22.5.5 static void WDOG_Disable (WDOG_Type * *base*) [inline], [static]

This function writes a value into the WDOG_WCR register to disable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write,once the bit is set. only debug mode exception

Parameters

base	WDOG peripheral base address

22.5.6 static void WDOG_TriggerSystemSoftwareReset (WDOG_Type * *base*) [inline], [static]

This function will write to the WCR[SRS] bit to trigger a software system reset. This bit will automatically resets to "1" after it has been asserted to "0". Note: Calling this API will reset the system right now, please using it with more attention.

Parameters

base	WDOG peripheral base address

22.5.7 static void WDOG_TriggerSoftwareSignal (WDOG_Type * *base*) [inline], [static]

This function will write to the WCR[WDA] bit to trigger WDOG_B signal assertion. The WDOG_B signal can be routed to external pin of the chip, the output pin will turn to assertion along with WDOG_B signal. Note: The WDOG_B signal will remain assert until a power on reset occurred, so, please take more attention while calling it.

Parameters

base	WDOG peripheral base address

22.5.8 static void WDOG_EnableInterrupts (WDOG_Type * *base*, uint16_t *mask*) [inline], [static]

This bit is a write once only bit. Once the software does a write access to this bit, it will get locked and cannot be reprogrammed until the next system reset assertion

Parameters

base	WDOG peripheral base address
mask	The interrupts to enable The parameter can be combination of the following source if defined. • kWDOG_InterruptEnable

22.5.9 uint16_t WDOG_GetStatusFlags (WDOG_Type * *base*)

This function gets all reset status flags.

```
* uint16_t status;
* status = WDOG_GetStatusFlags (wdog_base);
*
```

Parameters

base	WDOG peripheral base address

Returns

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

See Also

_wdog_status_flags
- true: a related status flag has been set.
- false: a related status flag is not set.

22.5.10 void WDOG_ClearInterruptStatus (WDOG_Type * *base*, uint16_t *mask*)

This function clears the WDOG status flag.

This is an example for clearing the interrupt flag.

```
*   WDOG_ClearStatusFlags(wdog_base,KWDOG_InterruptFlag);
*
```

Parameters

base	WDOG peripheral base address
mask	The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

22.5.11 static void WDOG_SetTimeoutValue (WDOG_Type ∗ *base,* uint16_t *timeoutCount*) [inline], [static]

This function sets the timeout value. This function writes a value into WCR registers. The time-out value can be written at any point of time but it is loaded to the counter at the time when WDOG is enabled or after the service routine has been performed.

Parameters

base	WDOG peripheral base address
timeoutCount	WDOG timeout value; count of WDOG clock tick.

22.5.12 static void WDOG_SetInterrputTimeoutValue (WDOG_Type ∗ *base,* uint16_t *timeoutCount*) [inline], [static]

This function sets the interrupt count timeout value. This function writes a value into WIC registers which are wirte-once. This field is write once only. Once the software does a write access to this field, it will get locked and cannot be reprogrammed until the next system reset assertion.

Parameters

base	WDOG peripheral base address
timeoutCount	WDOG timeout value; count of WDOG clock tick.

22.5.13 static void WDOG_DisablePowerDownEnable (WDOG_Type ∗ *base*) [inline], [static]

This function disable the WDOG power down enable(PDE). This function writes a value into WMCR registers which are wirte-once. This field is write once only. Once software sets this bit it cannot be reset until the next system reset.

Parameters

base	WDOG peripheral base address

22.5.14 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

Chapter 23
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.

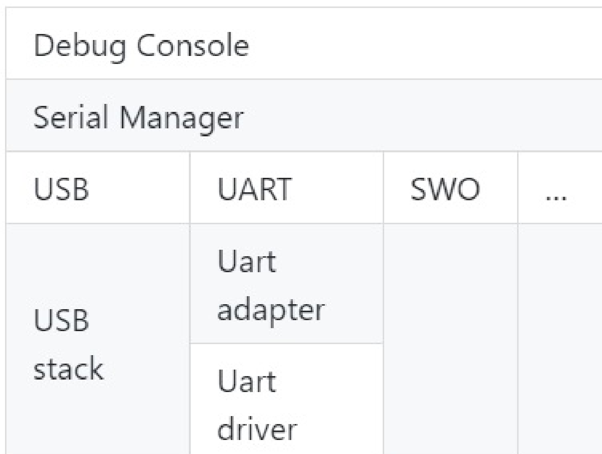

Figure 23.1.1: Debug console overview

Function groups

23.2.1 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);
```

23.2.2 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 */
```

23.2.3 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

23.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

23.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

23.4.3 #define DEBUGCONSOLE_DISABLE 2U

23.4.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

The macro only support to be redefined in project setting.

23.4.5 #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

23.5.1 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

23.5.2 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.

23.5.3 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.

23.5.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

23.5.5 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.

23.5.6 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.

23.5.7 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.

23.5.8 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.

23.5.9 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.

23.5.10 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.

23.5.11 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

23.5.12 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 24
CODEC codec Driver

Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

Modules

- codec common Driver

codec common Driver

24.2.1 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.

24.2.2 Data Structure Documentation

24.2.2.1 struct codec_config_t

Data Fields

- uint32_t codecDevType
  codec type
- void * codecDevConfig
  Codec device specific configuration.

24.2.2.2 struct codec_capability_t

Data Fields

- uint32_t codecModuleCapability
  codec module capability
- uint32_t codecPlayCapability
  codec play capability
- uint32_t codecRecordCapability
  codec record capability

24.2.2.3 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

24.2.3 Macro Definition Documentation

24.2.3.1 #define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

24.2.4 Enumeration Type Documentation

24.2.4.1 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.

24.2.4.2 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.

24.2.4.3 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.

24.2.4.4 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

24.2.4.5 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

24.2.4.6 enum codec_module_ctrl_cmd_t

Enumerator

kCODEC_ModuleSwitchI2SInInterface module digital interface siwtch.

24.2.4.7 anonymous enum

Enumerator

kCODEC_ModuleI2SInInterfacePCM Pcm interface.
kCODEC_ModuleI2SInInterfaceDSD DSD interface.

24.2.4.8 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

24.2.4.9 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

24.2.4.10 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

24.2.4.11 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

24.2.4.12 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

24.2.5 Function Documentation

24.2.5.1 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.

24.2.5.2 status_t CODEC_Deinit (codec_handle_t ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

24.2.5.3 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.

24.2.5.4 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.

24.2.5.5 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 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

24.2.5.6 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.

24.2.5.7 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.

24.2.5.8 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.

24.2.5.9 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.

24.2.5.10 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 25
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 26
Ecspi_cmsis_driver

This section describes the programming interface of the ecspi 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.

Function groups

26.1.1 ECSPI CMSIS GetVersion Operation

This function group will return the ECSPI CMSIS Driver version to user.

26.1.2 ECSPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

26.1.3 ECSPI 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.

26.1.4 ECSPI CMSIS Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

26.1.5 ECSPI CMSIS Status Operation

This function group gets the ecspi transfer status.

26.1.6 ECSPI CMSIS Control Operation

This function can select 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 set other control command.

Typical use case

26.2.1 Master Operation

```
/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*ECSPI master init*/
Driver_SPI0.Initialize(ECSPI_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();
```

26.2.2 Slave Operation

```
/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*DSPI slave init*/
Driver_SPI2.Initialize(ECSPI_SlaveSignalEvent_t);
Driver_SPI2.PowerControl(ARM_POWER_FULL);
Driver_SPI2.Control(ARM_SPI_MODE_SLAVE, false);

/* Start slave transfer */
Driver_SPI2.Transfer(slaveTxData, slaveRxData, TRANSFER_SIZE);

/* slave power off */
Driver_SPI2.PowerControl(ARM_POWER_OFF);

/* slave uninitialize */
Driver_SPI2.Uninitialize();
```

Chapter 27
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

27.1.1 Master Operation in interrupt transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}
/*Init I2C1*/
Driver_I2C1.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C1.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
Driver_I2C1.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transmit*/
Driver_I2C1.MasterTransmit(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

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

27.1.2 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 28
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 please refer to 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.

Function groups

28.1.1 UART CMSIS GetVersion Operation

This function group will return the UART CMSIS Driver version to user.

28.1.2 UART CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

28.1.3 UART CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize the uart instance . And this API must be called before you configure an uart instance or after you Deinit an uart instance.The right steps to start an instance is that you must initialize the instance which been slected firstly,then you can power on the instance.After these all have been done,you can configure the instance by using control operation.If you want to Uninitialize the instance, you must power off the instance first.

28.1.4 UART CMSIS Transfer Operation

This function group controls the transfer, send/receive data.

28.1.5 UART CMSIS Status Operation

This function group gets the UART transfer status.

28.1.6 UART CMSIS Control Operation

This function can configure an instance ,set baudrate for uart, get current baudrate ,set transfer data bits and other control command.

Chapter 29
Ecspi_freertos_driver

Overview

Driver version

- #define FSL_ECSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  ECSPI FreeRTOS driver version.

ECSPI RTOS Operation

- status_t ECSPI_RTOS_Init (ecspi_rtos_handle_t ∗handle, ECSPI_Type ∗base, const ecspi_master-_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes ECSPI.
- status_t ECSPI_RTOS_Deinit (ecspi_rtos_handle_t ∗handle)
  Deinitializes the ECSPI.
- status_t ECSPI_RTOS_Transfer (ecspi_rtos_handle_t ∗handle, ecspi_transfer_t ∗transfer)
  Performs ECSPI transfer.

Macro Definition Documentation

29.2.1 #define FSL_ECSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

Function Documentation

29.3.1 status_t ECSPI_RTOS_Init (ecspi_rtos_handle_t ∗ *handle,* ECSPI_Type ∗ *base,* const ecspi_master_config_t ∗ *masterConfig,* uint32_t *srcClock_Hz*)

This function initializes the ECSPI module and related RTOS context.

Parameters

handle	The RTOS ECSPI handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the ECSPI instance to initialize.
masterConfig	Configuration structure to set-up ECSPI in master mode.

srcClock_Hz	Frequency of input clock of the ECSPI module.

Returns

status of the operation.

29.3.2 status_t ECSPI_RTOS_Deinit (ecspi_rtos_handle_t ∗ *handle*)

This function deinitializes the ECSPI module and related RTOS context.

Parameters

handle	The RTOS ECSPI handle.

29.3.3 status_t ECSPI_RTOS_Transfer (ecspi_rtos_handle_t ∗ *handle*, ecspi_transfer_t ∗ *transfer*)

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

Parameters

handle	The RTOS ECSPI handle.
transfer	Structure specifying the transfer parameters.

Returns

status of the operation.

Chapter 30
I2C FreeRTOS Driver

Overview

Driver version

- #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 7))
  I2C FreeRTOS driver version.

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

30.2.1 #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 7))

Function Documentation

30.3.1 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.

30.3.2 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.

30.3.3 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 31
Wm8524_adapter

Overview

Macros

- #define HAL_CODEC_HANDLER_SIZE (4)
  codec handler size

Enumerations

- enum _codec_type { kCODEC_WM8524 }
  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

31.2.1 enum _codec_type

Enumerator

kCODEC_WM8524 wm8524

Function Documentation

31.3.1 status_t HAL_CODEC_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

31.3.2 status_t HAL_CODEC_Deinit (void * *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

31.3.3 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.

31.3.4 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.

31.3.5 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.

31.3.6 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.

31.3.7 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.

31.3.8 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.

31.3.9 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.

31.3.10 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 32
Wm8524

Overview

Data Structures

- struct wm8524_handle_t
  WM8524 handler. More...

Typedefs

- typedef void(∗ wm8524_setMuteIO)(uint32_t output)
  < mute control io function pointer

Enumerations

- enum wm8524_protocol_t {
  kWM8524_ProtocolLeftJustified = 0x0,
  kWM8524_ProtocolI2S = 0x1,
  kWM8524_ProtocolRightJustified = 0x2 }
  The audio data transfer protocol.
- enum _wm8524_mute_control {
  kWM8524_Mute = 0U,
  kWM8524_Unmute = 1U }
  wm8524 mute operation

Functions

- status_t WM8524_Init (wm8524_handle_t ∗handle, wm8524_config_t ∗config)
  Initializes WM8524.
- void WM8524_ConfigFormat (wm8524_handle_t ∗handle, wm8524_protocol_t protocol)
  Configure WM8524 audio protocol.
- void WM8524_SetMute (wm8524_handle_t ∗handle, bool isMute)
  Sets the codec mute state.

Driver version

- #define FSL_WM8524_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  WM8524 driver version 2.1.1.

Data Structure Documentation

32.2.1 struct wm8524_handle_t

Data Fields

- wm8524_config_t ∗ config
  wm8524 config pointer

Macro Definition Documentation

32.3.1 #define FSL_WM8524_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

Typedef Documentation

32.4.1 typedef void(∗ wm8524_setMuteIO)(uint32_t output)

format control io function pointer

Enumeration Type Documentation

32.5.1 enum wm8524_protocol_t

Enumerator

kWM8524_ProtocolLeftJustified Left justified mode.
kWM8524_ProtocolI2S I2S mode.
kWM8524_ProtocolRightJustified Right justified mode.

32.5.2 enum _wm8524_mute_control

Enumerator

kWM8524_Mute mute left and right channel DAC
kWM8524_Unmute unmute left and right channel DAC

Function Documentation

32.6.1 status_t WM8524_Init (wm8524_handle_t ∗ *handle,* wm8524_config_t ∗ *config*)

Parameters

handle	WM8524 handle structure.
config	WM8524 configure structure.

Returns

kStatus_Success.

32.6.2 void WM8524_ConfigFormat (wm8524_handle_t * *handle*, wm8524_protocol_t *protocol*)

Parameters

handle	WM8524 handle structure.
protocol	WM8524 configuration structure.

32.6.3 void WM8524_SetMute (wm8524_handle_t * *handle*, bool *isMute*)

Parameters

handle	WM8524 handle structure.
isMute	true means mute, false means normal.

Chapter 33
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

33.2.1 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.

33.2.1.0.0.28 Field Documentation

33.2.1.0.0.28.1 uint8_t* serial_manager_config_t::ringBuffer

Besides, the memory space cannot be free during the lifetime of the serial manager module.

33.2.2 struct serial_manager_callback_message_t

Data Fields

- uint8_t ∗ buffer
  Transferred buffer.
- uint32_t length
  Transferred data length.

Macro Definition Documentation

33.3.1 #define SERIAL_MANAGER_TIME_DELAY_DEFAULT_VALUE (1U)

33.3.2 #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_-SIZE_TEMP + 12U)

Definition of serial manager handle size.

33.3.3 #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.

33.3.4 #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.

33.3.5 #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.

33.3.6 #define SERIAL_MANAGER_USE_COMMON_TASK (0U)

33.3.7 #define SERIAL_MANAGER_TASK_PRIORITY (2U)

33.3.8 #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)

Enumeration Type Documentation

33.4.1 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.

33.4.2 enum serial_manager_type_t

Enumerator

kSerialManager_NonBlocking None blocking handle.
kSerialManager_Blocking Blocking handle.

33.4.3 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

33.5.1 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.

33.5.2 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.

33.5.3 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);
*
```

33.5.4 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.

33.5.5 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_HAND-LE_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));
*
```

33.5.6 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.

33.5.7 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.

33.5.8 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.

33.5.9 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.

33.5.10 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 34
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

34.2.1 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

34.3.1 enum serial_port_swo_protocol_t

Enumerator

kSerialManager_SwoProtocolManchester SWO Manchester protocol.
kSerialManager_SwoProtocolNrz SWO UART/NRZ protocol.

Chapter 35
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

35.2.1 enum serial_port_uart_parity_mode_t

Enumerator

kSerialManager_UartParityDisabled Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

35.2.2 enum serial_port_uart_stop_bit_count_t

Enumerator

kSerialManager_UartOneStopBit One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

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

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.