MCUXpresso SDK API Reference Manual

NXP Semiconductors

Document Number: MCUXSDKIMX8MMAPIRM
Rev. 0
Jan 2019

Contents

Contents

Section Number | Title | Page Number

Contents

Contents

Section Number | Title | Page Number

Contents

Contents

Section Number	Title	Page Number

Contents

Contents

Section Number

Contents

Title

Page Number

Contents

Section Number | Title | Page Number

Contents

Section Number | Title | Page Number

Contents

Contents

Contents

Section Number | Title | Page Number

Chapter 1
Introduction

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

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

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

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

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

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

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

Table 2: MCUXpresso SDK Folder Structure

Chapter 2
Driver errors status

- kStatus_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_PDM_Busy = 7200
- kStatus_PDM_CLK_LOW = 7201
- kStatus_PDM_FIFO_ERROR = 7202
- kStatus_PDM_QueueFull = 7203
- kStatus_PDM_Idle = 7204
- 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_SAI_TxBusy = 1900
- kStatus_SAI_RxBusy = 1901
- kStatus_SAI_TxError = 1902
- kStatus_SAI_RxError = 1903
- kStatus_SAI_QueueFull = 1904
- kStatus_SAI_TxIdle = 1905
- kStatus_SAI_RxIdle = 1906
- kStatus_SDMA_ERROR = 7300

- kStatus_SDMA_Busy = 7301

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

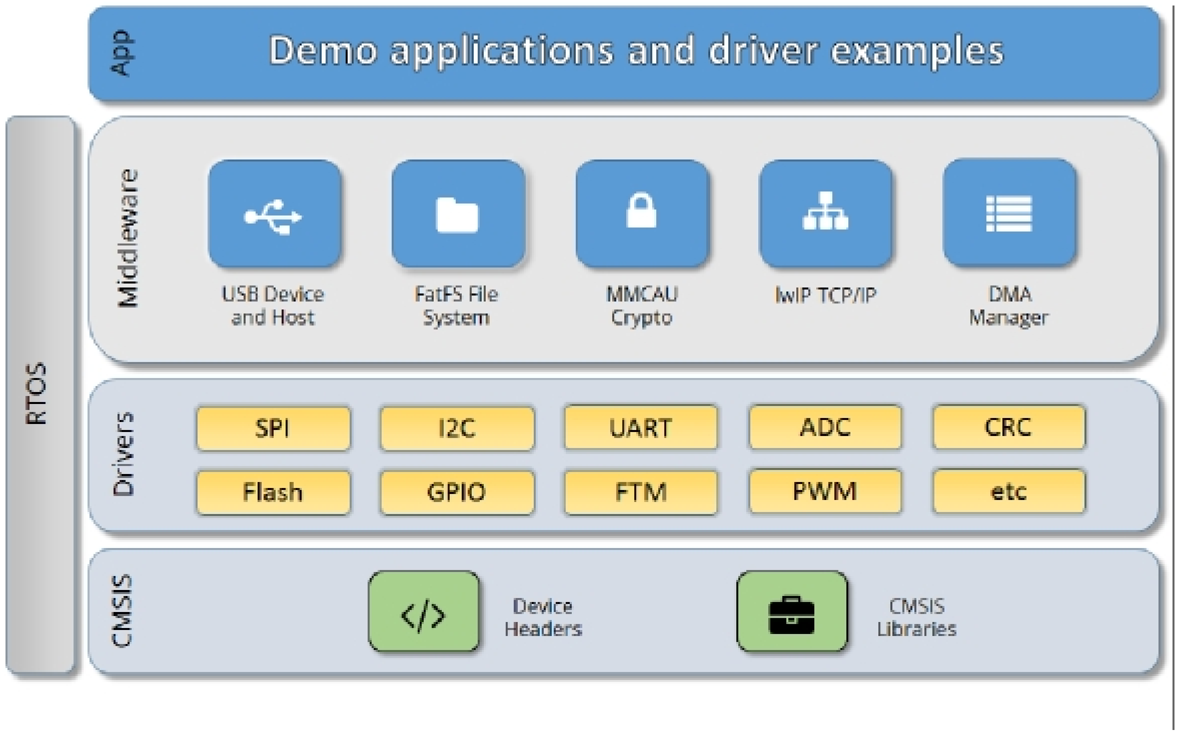

Figure 1: MCUXpresso SDK Block Diagram

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/DM-A driver/eDMA driver to quickly enable the peripherals and perform transfers.

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

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

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

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

Interrupt handling for transactional APIs

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

```
        PUBWEAK SPI0_IRQHandler
        PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

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

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

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

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

Feature Header Files

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

Application

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

Chapter 4
Trademarks

Chapter 5
Clock Driver

5.1 Overview

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

Data Structures

- struct ccm_analog_frac_pll_config_t
  Fractional-N PLL configuration. More...
- struct ccm_analog_integer_pll_config_t
  Integer PLL configuration. More...

Macros

- #define OSC24M_CLK_FREQ 24000000U
  XTAL 24M clock frequency.
- #define CLKPAD_FREQ 0U
  pad clock 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 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 PDM_CLOCKS
  Clock ip name array for PDM.
- #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_GEN_CTRL_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_Iomux0 = CCM_TUPLE(27U, 33U),
kCLOCK_Iomux1 = CCM_TUPLE(28U, 33U),
kCLOCK_Iomux2 = CCM_TUPLE(29U, 33U),
kCLOCK_Iomux3 = CCM_TUPLE(30U, 33U),
kCLOCK_Iomux4 = CCM_TUPLE(31U, 33U),
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_Sim_s = CCM_TUPLE(67U, 32U),

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_RootAudioAhb = (uint32_t)(&(CCM)->ROOT[34].TARGET_ROOT),
kCLOCK_RootAudioIpg = (uint32_t)(&(CCM)->ROOT[35].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),
kCLOCK_RootPdm = (uint32_t)(&(CCM)->ROOT[132].TARGET_ROOT) }
ccm root name used to get clock frequency.
- enum clock_rootmux_m4_clk_sel_t {

kCLOCK_M4RootmuxOsc24M = 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_AxiRootmuxOsc24M = 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_AhbRootmuxOsc24M = 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_audio_ahb_clk_sel_t {
kCLOCK_AudioAhbRootmuxOsc24M = 0U,
kCLOCK_AudioAhbRootmuxSysPll2Div2 = 1U,
kCLOCK_AudioAhbRootmuxSysPll1 = 2U,
kCLOCK_AudioAhbRootmuxSysPll2 = 3U,
kCLOCK_AudioAhbRootmuxSysPll2Div6 = 4U,
kCLOCK_AudioAhbRootmuxSysPll3 = 5U,
kCLOCK_AudioAhbRootmuxAudioPll1 = 6U,
kCLOCK_AudioAhbRootmuxVideoPll1 = 7U }
  Root clock select enumeration for Audio AHB bus.
- enum clock_rootmux_qspi_clk_sel_t {

kCLOCK_QspiRootmuxOsc24M = 0U,
kCLOCK_QspiRootmuxSysPll1Div2 = 1U,
kCLOCK_QspiRootmuxSysPll2Div3 = 2U,
kCLOCK_QspiRootmuxSysPll2Div2 = 3U,
kCLOCK_QspiRootmuxAudioPll2 = 4U,
kCLOCK_QspiRootmuxSysPll1Div3 = 5U,
kCLOCK_QspiRootmuxSysPll3 = 6,
kCLOCK_QspiRootmuxSysPll1Div8 = 7U }
Root clock select enumeration for QSPI peripheral.

- enum clock_rootmux_ecspi_clk_sel_t {
kCLOCK_EcspiRootmuxOsc24M = 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_I2cRootmuxOsc24M = 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_UartRootmuxOsc24M = 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_GptRootmuxOsc24M = 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_WdogRootmuxOsc24M = 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_PwmRootmuxOsc24M = 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_SaiRootmuxOsc24M = 0U,
kCLOCK_SaiRootmuxAudioPll1 = 1U,
kCLOCK_SaiRootmuxAudioPll2 = 2U,
kCLOCK_SaiRootmuxVideoPll1 = 3U,
kCLOCK_SaiRootmuxSysPll1Div6 = 4U,
kCLOCK_SaiRootmuxOsc26m = 5U,
kCLOCK_SaiRootmuxExtClk1 = 6U,
kCLOCK_SaiRootmuxExtClk2 = 7U }
Root clock select enumeration for SAI peripheral.
- enum clock_rootmux_pdm_clk_sel_t {

kCLOCK_PdmRootmuxOsc24M = 0U,
kCLOCK_PdmRootmuxSystemPll2 = 1U,
kCLOCK_PdmRootmuxAudioPll1 = 2U,
kCLOCK_PdmRootmuxSysPll1 = 3U,
kCLOCK_PdmRootmuxSysPll2 = 4U,
kCLOCK_PdmRootmuxSysPll3 = 5U,
kCLOCK_PdmRootmuxExtClk3 = 6U,
kCLOCK_PdmRootmuxAudioPll2 = 7U }
Root clock select enumeration for PDM peripheral.
- enum clock_rootmux_noc_clk_sel_t {
kCLOCK_NocRootmuxOsc24M = 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_DramPllInternalPll1BypassCtrl,
kCLOCK_GpuPLLPwrBypassCtrl,
kCLOCK_VpuPllPwrBypassCtrl,
kCLOCK_ArmPllPwrBypassCtrl,
kCLOCK_SysPll1InternalPll1BypassCtrl,
kCLOCK_SysPll2InternalPll1BypassCtrl,
kCLOCK_SysPll3InternalPll1BypassCtrl }

PLL control names for PLL bypass.

- enum clock_pll_clke_t {
  kCLOCK_AudioPll1Clke,
  kCLOCK_AudioPll2Clke,
  kCLOCK_VideoPll1Clke,
  kCLOCK_DramPllClke,
  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 }

  PLL clock names for clock enable/disable settings.
- enum clock_pll_ctrl_t

  ANALOG Power down override control.
- enum _ccm_analog_pll_ref_clk {
  kANALOG_PllRefOsc24M = 0U,
  kANALOG_PllPadClk = 1U }

  PLL reference clock select.

Driver version

- #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

  CLOCK driver version 2.0.1.

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.

CCM Gate Control

- static void CLOCK_ControlGate (uint32_t ccmGate, clock_gate_value_t control)
  lockrief 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_OverridePllClke (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_integer_pll_config_t ∗config)
  Initializes the ANALOG ARM PLL.
- void CLOCK_DeinitArmPll (void)
  De-initialize the ARM PLL.
- void CLOCK_InitSysPll1 (const ccm_analog_integer_pll_config_t ∗config)
  Initializes the ANALOG SYS PLL1.
- void CLOCK_DeinitSysPll1 (void)
  De-initialize the System PLL1.
- void CLOCK_InitSysPll2 (const ccm_analog_integer_pll_config_t ∗config)
  Initializes the ANALOG SYS PLL2.
- void CLOCK_DeinitSysPll2 (void)
  De-initialize the System PLL2.
- void CLOCK_InitSysPll3 (const ccm_analog_integer_pll_config_t ∗config)
  Initializes the ANALOG SYS PLL3.
- void CLOCK_DeinitSysPll3 (void)
  De-initialize the System PLL3.
- 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_InitIntegerPll (CCM_ANALOG_Type ∗base, const ccm_analog_integer_pll_config-_t ∗config, clock_pll_ctrl_t type)
  Initializes the ANALOG Integer PLL.
- uint32_t CLOCK_GetIntegerPllFreq (CCM_ANALOG_Type ∗base, clock_pll_ctrl_t type, uint32_t refClkFreq, bool pll1Bypass)
  Get the ANALOG Integer 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.

5.2 Data Structure Documentation

5.2.1 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
- uint32_t mainDiv
  Value of the 10-bit programmable main-divider, range must be 64∼1023.
- uint32_t dsm
  Value of 16-bit DSM.
- uint8_t preDiv
  Value of the 6-bit programmable pre-divider, range must be 1∼63.
- uint8_t postDiv
  Value of the 3-bit programmable Scaler, range must be 0∼6.

5.2.2 struct ccm_analog_integer_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
- uint32_t mainDiv
  Value of the 10-bit programmable main-divider, range must be 64∼1023.
- uint8_t preDiv
  Value of the 6-bit programmable pre-divider, range must be 1∼63.
- uint8_t postDiv
  Value of the 3-bit programmable Scaler, range must be 0∼6.

5.3 Macro Definition Documentation

5.3.1 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

5.3.2 #define ECSPI_CLOCKS

Value:

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

5.3.3 #define GPIO_CLOCKS

Value:

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

5.3.4 #define GPT_CLOCKS

Value:

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

5.3.5 #define I2C_CLOCKS

Value:

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

5.3.6 #define IOMUX_CLOCKS

Value:

```
{                                                                                   \
        kCLOCK_Iomuxc0, kCLOCK_Iomuxc1, kCLOCK_Iomuxc2, kCLOCK_Iomuxc3, kCLOCK_Iomuxc4, \
    }
```

5.3.7 #define PWM_CLOCKS

Value:

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

5.3.8 #define RDC_CLOCKS

Value:

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

5.3.9 #define SAI_CLOCKS

Value:

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

5.3.10 #define RDC_SEMA42_CLOCKS

Value:

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

5.3.11 #define UART_CLOCKS

Value:

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

5.3.12 #define USDHC_CLOCKS

Value:

```
{                                                              \
        kCLOCK_IpInvalid, kCLOCK_Usdhc1, kCLOCK_Usdhc2,
      kCLOCK_Usdhc3 \
    }
```

5.3.13 #define WDOG_CLOCKS

Value:

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

5.3.14 #define TMU_CLOCKS

Value:

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

5.3.15 #define SDMA_CLOCKS

Value:

```
{                                            \
        kCLOCK_Sdma1, kCLOCK_Sdma2, kCLOCK_Sdma3 \
    }
```

5.3.16 #define MU_CLOCKS

Value:

```
{                \
        kCLOCK_Mu \
    }
```

5.3.17 #define QSPI_CLOCKS

Value:

```
{                  \
        kCLOCK_Qspi \
    }
```

5.3.18 #define PDM_CLOCKS

Value:

```
{                 \
        kCLOCK_Pdm \
    }
```

5.3.19 #define kCLOCK_CoreSysClk kCLOCK_CoreM4Clk

5.3.20 #define CLOCK_GetCoreSysClkFreq CLOCK_GetCoreM4Freq

5.4 Enumeration Type Documentation

5.4.1 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.

5.4.2 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_Iomux0 IOMUX Clock Gate.
kCLOCK_Iomux1 IOMUX Clock Gate.
kCLOCK_Iomux2 IOMUX Clock Gate.
kCLOCK_Iomux3 IOMUX Clock Gate.
kCLOCK_Iomux4 IOMUX 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_Usdhc1 USDHC1 Clock Gate.
kCLOCK_Usdhc2 USDHC2 Clock Gate.
kCLOCK_Wdog1 WDOG1 Clock Gate.
kCLOCK_Wdog2 WDOG2 Clock Gate.
kCLOCK_Wdog3 WDOG3 Clock Gate.
kCLOCK_Pdm PDM Clock Gate.
kCLOCK_Usdhc3 USDHC3 Clock Gate.
kCLOCK_Sdma3 SDMA3 Clock Gate.
kCLOCK_TempSensor TempSensor Clock Gate.

5.4.3 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_RootAudioAhb Audio AHB Clock control name.
kCLOCK_RootAudioIpg Audio 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.
kCLOCK_RootPdm	PDM Clock control name.

5.4.4 enum clock_rootmux_m4_clk_sel_t

Enumerator

kCLOCK_M4RootmuxOsc24M	ARM Cortex-M4 Clock from OSC 24M.
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.

5.4.5 enum clock_rootmux_axi_clk_sel_t

Enumerator

kCLOCK_AxiRootmuxOsc24M	ARM AXI Clock from OSC 24M.
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.

5.4.6 enum clock_rootmux_ahb_clk_sel_t

Enumerator

kCLOCK_AhbRootmuxOsc24M ARM AHB Clock from OSC 24M.
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.

5.4.7 enum clock_rootmux_audio_ahb_clk_sel_t

Enumerator

kCLOCK_AudioAhbRootmuxOsc24M ARM Audio AHB Clock from OSC 24M.
kCLOCK_AudioAhbRootmuxSysPll2Div2 ARM Audio AHB Clock from SYSTEM PLL2 divided by 2.
kCLOCK_AudioAhbRootmuxSysPll1 ARM Audio AHB Clock from SYSTEM PLL1.
kCLOCK_AudioAhbRootmuxSysPll2 ARM Audio AHB Clock from SYSTEM PLL2.
kCLOCK_AudioAhbRootmuxSysPll2Div6 ARM Audio AHB Clock from SYSTEM PLL2 divided by 6.
kCLOCK_AudioAhbRootmuxSysPll3 ARM Audio AHB Clock from SYSTEM PLL3.
kCLOCK_AudioAhbRootmuxAudioPll1 ARM Audio AHB Clock from AUDIO PLL1.
kCLOCK_AudioAhbRootmuxVideoPll1 ARM Audio AHB Clock from VIDEO PLL1.

5.4.8 enum clock_rootmux_qspi_clk_sel_t

Enumerator

kCLOCK_QspiRootmuxOsc24M ARM QSPI Clock from OSC 24M.
kCLOCK_QspiRootmuxSysPll1Div2 ARM QSPI Clock from SYSTEM PLL1 divided by 2.
kCLOCK_QspiRootmuxSysPll2Div3 ARM QSPI Clock from SYSTEM PLL2 divided by 3.
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.

5.4.9 enum clock_rootmux_ecspi_clk_sel_t

Enumerator

kCLOCK_EcspiRootmuxOsc24M ECSPI Clock from OSC 24M.
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.

5.4.10 enum clock_rootmux_i2c_clk_sel_t

Enumerator

kCLOCK_I2cRootmuxOsc24M I2C Clock from OSC 24M.
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.

5.4.11 enum clock_rootmux_uart_clk_sel_t

Enumerator

kCLOCK_UartRootmuxOsc24M UART Clock from OSC 24M.
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.

5.4.12 enum clock_rootmux_gpt_t

Enumerator

kCLOCK_GptRootmuxOsc24M GPT Clock from OSC 24M.
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.

5.4.13 enum clock_rootmux_wdog_clk_sel_t

Enumerator

kCLOCK_WdogRootmuxOsc24M WDOG Clock from OSC 24M.
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.

5.4.14 enum clock_rootmux_Pwm_clk_sel_t

Enumerator

kCLOCK_PwmRootmuxOsc24M PWM Clock from OSC 24M.
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.

5.4.15 enum clock_rootmux_sai_clk_sel_t

Enumerator

kCLOCK_SaiRootmuxOsc24M SAI Clock from OSC 24M.
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_SaiRootmuxOsc26m SAI Clock from OSC HDMI 26M.
kCLOCK_SaiRootmuxExtClk1 SAI Clock from External Clock1, External Clock2, External Clock3.
kCLOCK_SaiRootmuxExtClk2 SAI Clock from External Clock2, External Clock3, External Clock4.

5.4.16 enum clock_rootmux_pdm_clk_sel_t

Enumerator

kCLOCK_PdmRootmuxOsc24M GPT Clock from OSC 24M.
kCLOCK_PdmRootmuxSystemPll2 GPT Clock from SYSTEM PLL2 divided by 10.
kCLOCK_PdmRootmuxAudioPll1 GPT Clock from SYSTEM PLL1 divided by 2.
kCLOCK_PdmRootmuxSysPll1 GPT Clock from SYSTEM PLL1 divided by 20.
kCLOCK_PdmRootmuxSysPll2 GPT Clock from VIDEO PLL1.
kCLOCK_PdmRootmuxSysPll3 GPT Clock from SYSTEM PLL1 divided by 10.
kCLOCK_PdmRootmuxExtClk3 GPT Clock from AUDIO PLL1.
kCLOCK_PdmRootmuxAudioPll2 GPT Clock from External Clock1, External Clock2, External Clock3.

5.4.17 enum clock_rootmux_noc_clk_sel_t

Enumerator

kCLOCK_NocRootmuxOsc24M NOC Clock from OSC 24M.
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.

5.4.18 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.

5.4.19 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.

5.4.20 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_DramPllInternalPll1BypassCtrl	CCM DRAM PLL 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 bypass Control.
kCLOCK_SysPll2InternalPll1BypassCtrl	CCM System PLL2 bypass Control.
kCLOCK_SysPll3InternalPll1BypassCtrl	CCM System PLL3 bypass Control.

5.4.21 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_DramPllClke	Dram pll 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.

5.4.22 enum _ccm_analog_pll_ref_clk

Enumerator

kANALOG_PllRefOsc24M	reference OSC 24M
kANALOG_PllPadClk	reference PAD CLK

5.5 Function Documentation

5.5.1 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).

5.5.2 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).

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

Parameters

base	CCM base pointer.
rootClk	Root clock control (see clock_root_control_t enumeration)

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

Parameters

base	CCM base pointer.
rootClk	Root control (see clock_root_control_t enumeration)

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

Parameters

base	CCM base pointer.
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.

5.5.6 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)
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)

5.5.7 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)

5.5.8 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.

5.5.9 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.

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

base CCM base pointer.

Parameters

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

5.5.11 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).

5.5.12 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).

5.5.13 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)

5.5.14 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)

5.5.15 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.

5.5.16 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.

5.5.17 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.

5.5.18 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)

5.5.19 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)

5.5.20 static void CLOCK_OverridePllClke (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.

5.5.21 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.

5.5.22 void CLOCK_InitArmPll (const ccm_analog_integer_pll_config_t ∗ *config*)

Parameters

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

Note

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

5.5.23 void CLOCK_InitSysPll1 (const ccm_analog_integer_pll_config_t ∗ *config*)

Parameters

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

Note

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

5.5.24 void CLOCK_InitSysPll2 (const ccm_analog_integer_pll_config_t ∗ *config*)

Parameters

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

Note

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

5.5.25 void CLOCK_InitSysPll3 (const ccm_analog_integer_pll_config_t ∗ *config*)

Parameters

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

Note

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

5.5.26 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.

5.5.27 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.

5.5.28 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).

5.5.29 void CLOCK_InitIntegerPll (CCM_ANALOG_Type ∗ *base,* const ccm_analog_integer_pll_config_t ∗ *config,* clock_pll_ctrl_t *type*)

Parameters

base	CCM ANALOG base address

config	Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).
type	integer pll type

5.5.30 uint32_t CLOCK_GetIntegerPllFreq (CCM_ANALOG_Type ∗ *base*, clock_pll_ctrl_t *type*, uint32_t *refClkFreq*, bool *pll1Bypass*)

Parameters

base	CCM ANALOG base address.
type	integer pll type
pll1Bypass	pll1 bypass flag

Returns

Clock frequency

5.5.31 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.

5.5.32 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.
fractional	pll reference clock frequency

Returns

Clock frequency

5.5.33 uint32_t CLOCK_GetPllFreq (clock_pll_ctrl_t *pll*)

Parameters

type	fractional pll type.

Returns

Clock frequency

5.5.34 uint32_t CLOCK_GetPllRefClkFreq (clock_pll_ctrl_t *ctrl*)

Parameters

type	fractional pll type.

Returns

Clock frequency

5.5.35 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

5.5.36 uint32_t CLOCK_GetCoreM4Freq (void)

Returns

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

5.5.37 uint32_t CLOCK_GetAxiFreq (void)

Returns

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

5.5.38 uint32_t CLOCK_GetAhbFreq (void)

Returns

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

Chapter 6
ECSPI: Serial Peripheral Interface Driver

6.1 Overview

Modules

- ECSPI Driver
- ECSPI FreeRTOS Driver
- ECSPI SDMA Driver

6.2 ECSPI Driver

6.2.1 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.

6.2.2 Typical use case

6.2.2.1 SPI master transfer using polling method

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

6.2.2.2 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.

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 _ecspi_status {
  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) }
  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 _ecspi_interrupt_enable {
  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 _ecspi_flags {

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 _ecspi_dma_enable_t {
kECSPI_TxDmaEnable = ECSPI_DMAREG_TEDEN_MASK,
kECSPI_RxDmaEnable = ECSPI_DMAREG_RXDEN_MASK,
kECSPI_DmaAllEnable = (ECSPI_DMAREG_TEDEN_MASK | ECSPI_DMAREG_RXDEN_M-ASK) }
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_wave_form_t {
kECSPI_WaveFormSingle = 0x0U,
kECSPI_WaveFormMultiple }
ECSPI wave form 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, 0, 0))
  ECSPI driver version 2.0.0.

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.
- void 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.

6.2.3 Data Structure Documentation

6.2.3.1 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_wave_form_t waveForm
  Wave form.
- ecspi_clock_polarity_t polarity
  Clock polarity.
- ecspi_clock_phase_t phase
  Clock phase.

6.2.3.2 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.

6.2.3.2.0.1 Field Documentation

6.2.3.2.0.1.1 bool ecspi_master_config_t::enableLoopback

6.2.3.3 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.

6.2.3.4 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.

6.2.3.5 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.

6.2.4 Macro Definition Documentation

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

6.2.4.2 #define ECSPI_DUMMYDATA (0xFFFFFFFFU)

6.2.5 Enumeration Type Documentation

6.2.5.1 enum _ecspi_status

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.

6.2.5.2 enum ecspi_clock_polarity_t

Enumerator

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

6.2.5.3 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.

6.2.5.4 enum _ecspi_interrupt_enable

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.

6.2.5.5 enum _ecspi_flags

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.

6.2.5.6 enum _ecspi_dma_enable_t

Enumerator

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

6.2.5.7 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.

6.2.5.8 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.

6.2.5.9 enum ecspi_master_slave_mode_t

Enumerator

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

6.2.5.10 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.

6.2.5.11 enum ecspi_clock_inactive_state_t

Enumerator

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

6.2.5.12 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.

6.2.5.13 enum ecspi_wave_form_t

Enumerator

kECSPI_WaveFormSingle The wave form for signal burst.
kECSPI_WaveFormMultiple The wave form for multiple burst.

6.2.5.14 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).

6.2.6 Function Documentation

6.2.6.1 uint32_t ECSPI_GetInstance (ECSPI_Type ∗ *base*)

Parameters

base	ECSPI base address

6.2.6.2 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

6.2.6.3 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.

6.2.6.4 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

6.2.6.5 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

6.2.6.6 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

6.2.6.7 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

6.2.6.8 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.

6.2.6.9 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.

6.2.6.10 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

6.2.6.11 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

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

Parameters

base	ECSPI base pointer

6.2.6.13 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

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

Parameters

base	ECSPI base pointer
source	ECSPI DMA source.
enable	True means enable DMA, false means disable DMA

6.2.6.15 static uint8_t ECSPI_GetTxFifoCount (ECSPI_Type ∗ *base*) `[inline]`, `[static]`

Parameters

base	ECSPI base pointer.

Returns

the number of words in Tx FIFO buffer.

6.2.6.16 static uint8_t ECSPI_GetRxFifoCount (ECSPI_Type ∗ *base*) `[inline]`, `[static]`

Parameters

base	ECSPI base pointer.

Returns

the number of words in Rx FIFO buffer.

6.2.6.17 static void ECSPI_SetChannelSelect (ECSPI_Type ∗ *base*, ecspi_channel_source_t *channel*) `[inline]`, `[static]`

Parameters

base	ECSPI base pointer

channel	Channel source.

6.2.6.18 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

6.2.6.19 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.

6.2.6.20 void 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

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

Parameters

base	ECSPI base pointer
data	Data needs to be write.

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

Parameters

base	ECSPI base pointer

Returns

Data in the register.

6.2.6.23 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.

6.2.6.24 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.

6.2.6.25 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.

6.2.6.26 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_NoTransferIn-Progress	There is not a non-blocking transaction currently in progress.

6.2.6.27 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.

6.2.6.28 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.

6.2.6.29 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.

6.2.6.30 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.

6.2.6.31 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_NoTransferIn-Progress	There is not a non-blocking transaction currently in progress.

6.2.6.32 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.

6.2.6.33 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

6.3 ECSPI FreeRTOS Driver

6.3.1 Overview

Driver version

- #define FSL_ECSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  ECSPI freertos driver version 2.0.0.

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.

6.3.2 Macro Definition Documentation

6.3.2.1 #define FSL_ECSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

6.3.3 Function Documentation

6.3.3.1 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.

6.3.3.2 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.

6.3.3.3 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.

6.4 ECSPI SDMA Driver

6.4.1 Overview

Data Structures

- struct ecspi_sdma_handle_t
  ECSPI SDMA transfer handle, users should not touch the content of the handle. More...

Typedefs

- typedef void(∗ ecspi_sdma_callback_t)(ECSPI_Type ∗base, ecspi_sdma_handle_t ∗handle, status-_t status, void ∗userData)
  ECSPI SDMA callback called at the end of transfer.

Driver version

- #define FSL_ECSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  ECSPI freertos driver version 2.0.1.

DMA Transactional

- void ECSPI_MasterTransferCreateHandleSDMA (ECSPI_Type ∗base, ecspi_sdma_handle_-t ∗handle, ecspi_sdma_callback_t callback, void ∗userData, sdma_handle_t ∗txHandle, sdma-_handle_t ∗rxHandle, uint32_t eventSourceTx, uint32_t eventSourceRx, uint32_t TxChannel, uint32_t RxChannel)
  Initialize the ECSPI master SDMA handle.
- void ECSPI_SlaveTransferCreateHandleSDMA (ECSPI_Type ∗base, ecspi_sdma_handle_-t ∗handle, ecspi_sdma_callback_t callback, void ∗userData, sdma_handle_t ∗txHandle, sdma-_handle_t ∗rxHandle, uint32_t eventSourceTx, uint32_t eventSourceRx, uint32_t TxChannel, uint32_t RxChannel)
  Initialize the ECSPI Slave SDMA handle.
- status_t ECSPI_MasterTransferSDMA (ECSPI_Type ∗base, ecspi_sdma_handle_t ∗handle, ecspi_-transfer_t ∗xfer)
  Perform a non-blocking ECSPI master transfer using SDMA.
- status_t ECSPI_SlaveTransferSDMA (ECSPI_Type ∗base, ecspi_sdma_handle_t ∗handle, ecspi_-transfer_t ∗xfer)
  Perform a non-blocking ECSPI slave transfer using SDMA.
- void ECSPI_MasterTransferAbortSDMA (ECSPI_Type ∗base, ecspi_sdma_handle_t ∗handle)
  Abort a ECSPI master transfer using SDMA.
- void ECSPI_SlaveTransferAbortSDMA (ECSPI_Type ∗base, ecspi_sdma_handle_t ∗handle)
  Abort a ECSPI slave transfer using SDMA.

6.4.2 Data Structure Documentation

6.4.2.1 struct _ecspi_sdma_handle

Data Fields

- bool txInProgress
  Send transfer finished.
- bool rxInProgress
  Receive transfer finished.
- sdma_handle_t ∗ txSdmaHandle
  DMA handler for ECSPI send.
- sdma_handle_t ∗ rxSdmaHandle
  DMA handler for ECSPI receive.
- ecspi_sdma_callback_t callback
  Callback for ECSPI SDMA transfer.
- void ∗ userData
  User Data for ECSPI SDMA callback.
- uint32_t state
  Internal state of ECSPI SDMA transfer.
- uint32_t ChannelTx
  Channel for send handle.
- uint32_t ChannelRx
  Channel for receive handler.

6.4.3 Macro Definition Documentation

6.4.3.1 #define FSL_ECSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

6.4.4 Typedef Documentation

6.4.4.1 typedef void(∗ ecspi_sdma_callback_t)(ECSPI_Type ∗base, ecspi_sdma_handle_t ∗handle, status_t status, void ∗userData)

6.4.5 Function Documentation

6.4.5.1 void ECSPI_MasterTransferCreateHandleSDMA (ECSPI_Type ∗ *base*, ecspi_sdma_handle_t ∗ *handle*, ecspi_sdma_callback_t *callback*, void ∗ *userData*, sdma_handle_t ∗ *txHandle*, sdma_handle_t ∗ *rxHandle*, uint32_t *eventSourceTx*, uint32_t *eventSourceRx*, uint32_t *TxChannel*, uint32_t *RxChannel*)

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

Parameters

base	ECSPI peripheral base address.
handle	ECSPI handle pointer.
callback	User callback function called at the end of a transfer.
userData	User data for callback.
txHandle	SDMA handle pointer for ECSPI Tx, the handle shall be static allocated by users.
rxHandle	SDMA handle pointer for ECSPI Rx, the handle shall be static allocated by users.
eventSourceTx	event source for ECSPI send, which can be found in SDMA mapping.
eventSourceRx	event source for ECSPI receive, which can be found in SDMA mapping.
TxChannel	SDMA channel for ECSPI send.
RxChannel	SDMA channel for ECSPI receive.

6.4.5.2 void ECSPI_SlaveTransferCreateHandleSDMA (ECSPI_Type ∗ *base*, ecspi_sdma_handle_t ∗ *handle*, ecspi_sdma_callback_t *callback*, void ∗ *userData*, sdma_handle_t ∗ *txHandle*, sdma_handle_t ∗ *rxHandle*, uint32_t *eventSourceTx*, uint32_t *eventSourceRx*, uint32_t *TxChannel*, uint32_t *RxChannel*)

This function initializes the ECSPI Slave SDMA handle which can be used for other SPI Slave transactional APIs. Usually, for a specified ECSPI instance, user need only call this API once to get the initialized handle.

Parameters

base	ECSPI peripheral base address.
handle	ECSPI handle pointer.
callback	User callback function called at the end of a transfer.
userData	User data for callback.
txHandle	SDMA handle pointer for ECSPI Tx, the handle shall be static allocated by users.
rxHandle	SDMA handle pointer for ECSPI Rx, the handle shall be static allocated by users.
eventSourceTx	event source for ECSPI send, which can be found in SDMA mapping.
eventSourceRx	event source for ECSPI receive, which can be found in SDMA mapping.

TxChannel	SDMA channel for ECSPI send.
RxChannel	SDMA channel for ECSPI receive.

6.4.5.3 status_t ECSPI_MasterTransferSDMA (ECSPI_Type ∗ *base*, ecspi_sdma_handle_t ∗ *handle*, ecspi_transfer_t ∗ *xfer*)

Note

This interface returned immediately after transfer initiates.

Parameters

base	ECSPI peripheral base address.
handle	ECSPI SDMA handle pointer.
xfer	Pointer to sdma transfer structure.

Return values

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

6.4.5.4 status_t ECSPI_SlaveTransferSDMA (ECSPI_Type ∗ *base*, ecspi_sdma_handle_t ∗ *handle*, ecspi_transfer_t ∗ *xfer*)

Note

This interface returned immediately after transfer initiates.

Parameters

base	ECSPI peripheral base address.
handle	ECSPI SDMA handle pointer.
xfer	Pointer to sdma transfer structure.

Return values

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

6.4.5.5 void ECSPI_MasterTransferAbortSDMA (ECSPI_Type ∗ *base*, ecspi_sdma_handle_t ∗ *handle*)

Parameters

base	ECSPI peripheral base address.
handle	ECSPI SDMA handle pointer.

6.4.5.6 void ECSPI_SlaveTransferAbortSDMA (ECSPI_Type ∗ *base*, ecspi_sdma_handle_t ∗ *handle*)

Parameters

base	ECSPI peripheral base address.
handle	ECSPI SDMA handle pointer.

Chapter 7
GPT: General Purpose Timer

7.1 Overview

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

7.2 Function groups

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

7.2.1 Initialization and deinitialization

The function GPT_Init() initializes the gpt with specified configurations. The function GPT_GetDefault-Config() 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.

7.3 Typical use case

7.3.1 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, 0))
  Version 2.0.0.

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.

7.4 Data Structure Documentation

7.4.1 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.

7.4.1.0.0.2 Field Documentation

7.4.1.0.0.2.1 gpt_clock_source_t gpt_config_t::clockSource

7.4.1.0.0.2.2 uint32_t gpt_config_t::divider

7.4.1.0.0.2.3 bool gpt_config_t::enableFreeRun

7.4.1.0.0.2.4 bool gpt_config_t::enableRunInWait

7.4.1.0.0.2.5 bool gpt_config_t::enableRunInStop

7.4.1.0.0.2.6 bool gpt_config_t::enableRunInDoze

7.4.1.0.0.2.7 bool gpt_config_t::enableRunInDbg

7.4.1.0.0.2.8 bool gpt_config_t::enableMode

7.5 Enumeration Type Documentation

7.5.1 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.

7.5.2 enum gpt_input_capture_channel_t

Enumerator

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

7.5.3 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.

7.5.4 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.

7.5.5 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.

7.5.6 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.

7.5.7 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.

7.6 Function Documentation

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

Parameters

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

7.6.2 void GPT_Deinit (GPT_Type ∗ *base*)

Parameters

base	GPT peripheral base address.

7.6.3 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 = true;
*    config->enableMode  = true;
*
```

Parameters

config	Pointer to the user configuration structure.

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

Parameters

base	GPT peripheral base address.

7.6.5 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).

7.6.6 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).

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

Parameters

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

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

Parameters

base	GPT peripheral base address.

Returns

clock divider in GPT module (1-4096).

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

Parameters

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

7.6.10 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).

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

Parameters

base	GPT peripheral base address.

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

Parameters

base	GPT peripheral base address.

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

Parameters

base	GPT peripheral base address.

Returns

Current GPT counter value.

7.6.14 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).

7.6.15 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).

7.6.16 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.

7.6.17 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).

7.6.18 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).

7.6.19 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.

7.6.20 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.

7.6.21 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).

7.6.22 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

7.6.23 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

7.6.24 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

7.6.25 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.

7.6.26 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 8
GPIO: General-Purpose Input/Output Driver

8.1 Overview

Modules

- GPIO Driver

8.2 GPIO Driver

8.2.1 Overview

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

8.2.2 Typical use case

8.2.2.1 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, 1))
  GPIO driver version 2.0.1.

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.

8.2.3 Data Structure Documentation

8.2.3.1 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.

8.2.3.1.0.3 Field Documentation

8.2.3.1.0.3.1 gpio_pin_direction_t gpio_pin_config_t::direction

8.2.3.1.0.3.2 gpio_interrupt_mode_t gpio_pin_config_t::interruptMode

8.2.4 Macro Definition Documentation

8.2.4.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

8.2.5 Enumeration Type Documentation

8.2.5.1 enum gpio_pin_direction_t

Enumerator

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

8.2.5.2 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.

8.2.6 Function Documentation

8.2.6.1 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
initConfig	pointer to a gpio_pin_config_t structure that contains the configuration information.

8.2.6.2 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.

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

8.2.6.4 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

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

8.2.6.6 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

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

8.2.6.8 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

8.2.6.9 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.

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

8.2.6.11 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.

8.2.6.12 static uint8_t GPIO_ReadPadStatus (GPIO_Type ∗ *base,* uint32_t *pin*) `[inline]`, `[static]`

8.2.6.13 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.

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

8.2.6.15 static void GPIO_PortEnableInterrupts (GPIO_Type ∗ *base,* uint32_t *mask*) `[inline]`, `[static]`

Parameters

base	GPIO base pointer.
mask	GPIO pin number macro.

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

Parameters

base	GPIO base pointer.
mask	GPIO pin number macro.

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

Parameters

base	GPIO base pointer.
mask	GPIO pin number macro.

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

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

Parameters

base	GPIO base pointer.

Return values

current	pin interrupt status flag.

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

Parameters

base	GPIO base pointer.

Return values

current	pin interrupt status flag.

8.2.6.21 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.

8.2.6.22 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 9
I2C: Inter-Integrated Circuit Driver

9.1 Overview

Modules

- I2C Driver
- I2C FreeRTOS Driver

9.2 I2C Driver

9.2.1 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.

9.2.2 Typical use case

9.2.2.1 Master Operation in functional method

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

9.2.2.2 Master Operation in interrupt transactional method

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

9.2.2.3 Slave Operation in functional method

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

9.2.2.4 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_WAIT_TIMEOUT 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Timeout 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 _i2c_status {
  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, 4))
  I2C driver version 2.0.4.

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.

9.2.3 Data Structure Documentation

9.2.3.1 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.

9.2.3.1.0.4 Field Documentation

9.2.3.1.0.4.1 bool i2c_master_config_t::enableMaster

9.2.3.1.0.4.2 uint32_t i2c_master_config_t::baudRate_Bps

9.2.3.2 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.

9.2.3.2.0.5 Field Documentation

9.2.3.2.0.5.1 uint32_t i2c_master_transfer_t::flags

9.2.3.2.0.5.2 uint8_t i2c_master_transfer_t::slaveAddress

9.2.3.2.0.5.3 i2c_direction_t i2c_master_transfer_t::direction

9.2.3.2.0.5.4 uint32_t i2c_master_transfer_t::subaddress

Transferred MSB first.

9.2.3.2.0.5.5 uint8_t i2c_master_transfer_t::subaddressSize

9.2.3.2.0.5.6 uint8_t∗ volatile i2c_master_transfer_t::data

9.2.3.2.0.5.7 volatile size_t i2c_master_transfer_t::dataSize

9.2.3.3 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.

9.2.3.3.0.6 Field Documentation

9.2.3.3.0.6.1 i2c_master_transfer_t i2c_master_handle_t::transfer

9.2.3.3.0.6.2 size_t i2c_master_handle_t::transferSize

9.2.3.3.0.6.3 uint8_t i2c_master_handle_t::state

9.2.3.3.0.6.4 i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback

9.2.3.3.0.6.5 void∗ i2c_master_handle_t::userData

9.2.3.4 struct i2c_slave_config_t

Data Fields

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

9.2.3.4.0.7 Field Documentation

9.2.3.4.0.7.1 bool i2c_slave_config_t::enableSlave

9.2.3.4.0.7.2 uint16_t i2c_slave_config_t::slaveAddress

9.2.3.5 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.

9.2.3.5.0.8 Field Documentation

9.2.3.5.0.8.1 i2c_slave_transfer_event_t i2c_slave_transfer_t::event

9.2.3.5.0.8.2 uint8_t∗ volatile i2c_slave_transfer_t::data

9.2.3.5.0.8.3 volatile size_t i2c_slave_transfer_t::dataSize

9.2.3.5.0.8.4 status_t i2c_slave_transfer_t::completionStatus

Only applies for kI2C_SlaveCompletionEvent.

9.2.3.5.0.8.5 size_t i2c_slave_transfer_t::transferredCount

9.2.3.6 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.

9.2.3.6.0.9 Field Documentation

9.2.3.6.0.9.1 volatile uint8_t i2c_slave_handle_t::state

9.2.3.6.0.9.2 i2c_slave_transfer_t i2c_slave_handle_t::transfer

9.2.3.6.0.9.3 uint32_t i2c_slave_handle_t::eventMask

9.2.3.6.0.9.4 i2c_slave_transfer_callback_t i2c_slave_handle_t::callback

9.2.3.6.0.9.5 void∗ i2c_slave_handle_t::userData

9.2.4 Macro Definition Documentation

9.2.4.1 #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

9.2.4.2 #define I2C_WAIT_TIMEOUT 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/

9.2.5 Typedef Documentation

9.2.5.1 typedef void(∗ i2c_master_transfer_callback_t)(I2C_Type ∗base, i2c_master_handle_t ∗handle, status_t status, void ∗userData)

9.2.5.2 typedef void(∗ i2c_slave_transfer_callback_t)(I2C_Type ∗base, i2c_slave_transfer_t ∗xfer, void ∗userData)

9.2.6 Enumeration Type Documentation

9.2.6.1 enum _i2c_status

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 poling status flags.
kStatus_I2C_Addr_Nak	NAK received during the address probe.

9.2.6.2 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.

9.2.6.3 enum _i2c_interrupt_enable

Enumerator

kI2C_GlobalInterruptEnable I2C global interrupt.

9.2.6.4 enum i2c_direction_t

Enumerator

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

9.2.6.5 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.

9.2.6.6 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.

9.2.7 Function Documentation

9.2.7.1 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

9.2.7.2 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

9.2.7.3 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.

9.2.7.4 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
srcClock_Hz	I2C peripheral clock frequency in Hz

9.2.7.5 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

9.2.7.6 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.

9.2.7.7 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.

9.2.7.8 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.

9.2.7.9 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_IntPendingFlagFlag

9.2.7.10 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.

9.2.7.11 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

9.2.7.12 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

9.2.7.13 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

9.2.7.14 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

9.2.7.15 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.

9.2.7.16 status_t I2C_MasterStop (I2C_Type ∗ *base*)

Return values

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

9.2.7.17 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.

9.2.7.18 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.

9.2.7.19 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.

9.2.7.20 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.

9.2.7.21 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.

9.2.7.22 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_Arbitration-Lost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.

9.2.7.23 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.

9.2.7.24 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_MasterGet-TransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_I2C_Busy, the transfer is finished.

Parameters

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

Return values

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

9.2.7.25 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.

9.2.7.26 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.

9.2.7.27 void I2C_MasterTransferHandleIRQ (I2C_Type ∗ *base*, void ∗ *i2cHandle*)

Parameters

base	I2C base pointer.
i2cHandle	pointer to i2c_master_handle_t structure.

9.2.7.28 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.

9.2.7.29 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.

9.2.7.30 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.

9.2.7.31 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.

9.2.7.32 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

9.3 I2C FreeRTOS Driver

9.3.1 Overview

Driver version

- #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  I2C freertos driver version 2.0.3.

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.

9.3.2 Macro Definition Documentation

9.3.2.1 #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

9.3.3 Function Documentation

9.3.3.1 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.

9.3.3.2 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.

9.3.3.3 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 10
TMU: Thermal Management Unit Driver

10.1 Overview

The MCUXpresso SDK provides a peripheral driver for the thermal management unit (TMU) module of MCUXpresso SDK devices.

10.2 Typical use case

10.2.1 Monitor and report Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/tmu-_1

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, 0))
  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_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.
- enum tmu_amplifier_gain_t {
kTMU_AmplifierGain6_34 = 0U,
kTMU_AmplifierGain6_485 = 1U,
kTMU_AmplifierGain6_63 = 2U,
kTMU_AmplifierGain6_775 = 3U,
kTMU_AmplifierGain6_92 = 4U,
kTMU_AmplifierGain7_065 = 5U,
kTMU_AmplifierGain7_21 = 6U,
kTMU_AmplifierGain7_355 = 7U,
kTMU_AmplifierGain7_5 = 8U,
kTMU_AmplifierGain7_645 = 9U,
kTMU_AmplifierGain7_79 = 10U,
kTMU_AmplifierGain7_935 = 11U,
kTMU_AmplifierGain8_08 = 12U,
kTMU_AmplifierGain8_225 = 13U,
kTMU_AmplifierGain8_37 = 14U,
kTMU_AmplifierGain8_515 = 15U }
Amplifier gain setting.
- enum tmu_amplifier_reference_voltage_t {

kTMU_AmplifierReferenceVoltage510 = 0U,
kTMU_AmplifierReferenceVoltage517_5 = 1U,
kTMU_AmplifierReferenceVoltage525 = 2U,
kTMU_AmplifierReferenceVoltage532_5 = 3U,
kTMU_AmplifierReferenceVoltage540 = 4U,
kTMU_AmplifierReferenceVoltage547_5 = 5U,
kTMU_AmplifierReferenceVoltage555 = 6U,
kTMU_AmplifierReferenceVoltage562_5 = 7U,
kTMU_AmplifierReferenceVoltage570 = 8U,
kTMU_AmplifierReferenceVoltage577_5 = 9U,
kTMU_AmplifierReferenceVoltage585 = 10U,
kTMU_AmplifierReferenceVoltage592_5 = 11U,
kTMU_AmplifierReferenceVoltage600 = 12U,
kTMU_AmplifierReferenceVoltage607_5 = 13U,
kTMU_AmplifierReferenceVoltage615 = 14U,
kTMU_AmplifierReferenceVoltage622_5 = 15U,
kTMU_AmplifierReferenceVoltage630 = 16U,
kTMU_AmplifierReferenceVoltage637_5 = 17U,
kTMU_AmplifierReferenceVoltage645 = 18U,
kTMU_AmplifierReferenceVoltage652_5 = 19U,
kTMU_AmplifierReferenceVoltage660 = 20U,
kTMU_AmplifierReferenceVoltage667_5 = 21U,
kTMU_AmplifierReferenceVoltage675 = 22U,
kTMU_AmplifierReferenceVoltage682_5 = 23U,
kTMU_AmplifierReferenceVoltage690 = 24U,
kTMU_AmplifierReferenceVoltage697_5 = 25U,
kTMU_AmplifierReferenceVoltage705 = 26U,
kTMU_AmplifierReferenceVoltage712_5 = 27U,
kTMU_AmplifierReferenceVoltage720 = 28U,
kTMU_AmplifierReferenceVoltage727_5 = 29U,
kTMU_AmplifierReferenceVoltage735 = 30U,
kTMU_AmplifierReferenceVoltage742_5 = 31U }

Amplifier reference voltage 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 monitoring the temperature sensor.
- 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.
- status_t TMU_GetImmediateTemperature (TMU_Type ∗base, uint32_t ∗temperature)
  Get the last immediate temperature at site.
- status_t TMU_GetAverageTemperature (TMU_Type ∗base, uint32_t ∗temperature)
  Get the last average temperature at site.
- void TMU_SetHighTemperatureThresold (TMU_Type ∗base, const tmu_thresold_config_t ∗config)
  Configure the high temperature thresold value and enable/disable relevant thresold.

Variables

- bool tmu_thresold_config_t::immediateThresoldEnable
  Enable high temperature immediate threshold.
- bool tmu_thresold_config_t::AverageThresoldEnable
  Enable high temperature average threshold.
- bool tmu_thresold_config_t::AverageCriticalThresoldEnable
  Enable high temperature average critical threshold.
- uint8_t tmu_thresold_config_t::immediateThresoldValue
  Range:10U-125U.
- uint8_t tmu_thresold_config_t::averageThresoldValue
  Range:10U-125U.
- uint8_t tmu_thresold_config_t::averageCriticalThresoldValue
  Range:10U-125U.
- uint32_t tmu_interrupt_status_t::interruptDetectMask
  The mask of interrupt status flags.
- tmu_average_low_pass_filter_t tmu_config_t::averageLPF
  The average temperature is calculated as: ALPF x Current_Temp + (1 - ALPF) x Average_Temp.
- tmu_amplifier_gain_t tmu_config_t::amplifierGain
  Amplifier gain setting.
- tmu_amplifier_reference_voltage_t tmu_config_t::amplifierVref
  Amplifier reference voltage setting.

10.3 Data Structure Documentation

10.3.1 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:10U-125U.
- uint8_t averageThresoldValue

Range:10U-125U.
- uint8_t averageCriticalThresoldValue
  Range:10U-125U.

10.3.2 struct tmu_interrupt_status_t

Data Fields

- uint32_t interruptDetectMask
  The mask of interrupt status flags.

10.3.3 struct tmu_config_t

Data Fields

- tmu_average_low_pass_filter_t averageLPF
  The average temperature is calculated as: ALPF x Current_Temp + (1 - ALPF) x Average_Temp.
- tmu_amplifier_gain_t amplifierGain
  Amplifier gain setting.
- tmu_amplifier_reference_voltage_t amplifierVref
  Amplifier reference voltage setting.

10.4 Macro Definition Documentation

10.4.1 #define FSL_TMU_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

Version 2.0.0.

10.5 Enumeration Type Documentation

10.5.1 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. >

10.5.2 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)

10.5.3 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.

10.5.4 enum tmu_amplifier_gain_t

Enumerator

kTMU_AmplifierGain6_34 TMU amplifier gain voltage 6.34mV.
kTMU_AmplifierGain6_485 TMU amplifier gain voltage 6.485mV.
kTMU_AmplifierGain6_63 TMU amplifier gain voltage 6.63mV.
kTMU_AmplifierGain6_775 TMU amplifier gain voltage 6.775mV.
kTMU_AmplifierGain6_92 TMU amplifier gain voltage 6.92mV.
kTMU_AmplifierGain7_065 TMU amplifier gain voltage 7.065mV.
kTMU_AmplifierGain7_21 TMU amplifier gain voltage 7.21mV.
kTMU_AmplifierGain7_355 TMU amplifier gain voltage 7.355mV.
kTMU_AmplifierGain7_5 TMU amplifier gain voltage 7.5mV.
kTMU_AmplifierGain7_645 TMU amplifier gain voltage 7.645mV.
kTMU_AmplifierGain7_79 TMU amplifier gain voltage 7.79mV.
kTMU_AmplifierGain7_935 TMU amplifier gain voltage 7.935mV.
kTMU_AmplifierGain8_08 TMU amplifier gain voltage 8.08mV(default).
kTMU_AmplifierGain8_225 TMU amplifier gain voltage 8.225mV.
kTMU_AmplifierGain8_37 TMU amplifier gain voltage 8.37mV.
kTMU_AmplifierGain8_515 TMU amplifier gain voltage 8.515mV.

10.5.5 enum tmu_amplifier_reference_voltage_t

Enumerator

kTMU_AmplifierReferenceVoltage510 TMU amplifier reference voltage 510mV.
kTMU_AmplifierReferenceVoltage517_5 TMU amplifier reference voltage 517.5mV.
kTMU_AmplifierReferenceVoltage525 TMU amplifier reference voltage 525mV.
kTMU_AmplifierReferenceVoltage532_5 TMU amplifier reference voltage 532.5mV.
kTMU_AmplifierReferenceVoltage540 TMU amplifier reference voltage 540mV.
kTMU_AmplifierReferenceVoltage547_5 TMU amplifier reference voltage 547.5mV.
kTMU_AmplifierReferenceVoltage555 TMU amplifier reference voltage 555mV.
kTMU_AmplifierReferenceVoltage562_5 TMU amplifier reference voltage 562.5mV.
kTMU_AmplifierReferenceVoltage570 TMU amplifier reference voltage 570mV.
kTMU_AmplifierReferenceVoltage577_5 TMU amplifier reference voltage 577.5mV.
kTMU_AmplifierReferenceVoltage585 TMU amplifier reference voltage 585mV.
kTMU_AmplifierReferenceVoltage592_5 TMU amplifier reference voltage 592.5mV.
kTMU_AmplifierReferenceVoltage600 TMU amplifier reference voltage 600mV.
kTMU_AmplifierReferenceVoltage607_5 TMU amplifier reference voltage 607.5mV.
kTMU_AmplifierReferenceVoltage615 TMU amplifier reference voltage 615mV.
kTMU_AmplifierReferenceVoltage622_5 TMU amplifier reference voltage 622.5mV.
kTMU_AmplifierReferenceVoltage630 TMU amplifier reference voltage 630mV.
kTMU_AmplifierReferenceVoltage637_5 TMU amplifier reference voltage 637.5mV.
kTMU_AmplifierReferenceVoltage645 TMU amplifier reference voltage 645mV.
kTMU_AmplifierReferenceVoltage652_5 TMU amplifier reference voltage 652.5mV(default).
kTMU_AmplifierReferenceVoltage660 TMU amplifier reference voltage 660mV.
kTMU_AmplifierReferenceVoltage667_5 TMU amplifier reference voltage 667.5mV.
kTMU_AmplifierReferenceVoltage675 TMU amplifier reference voltage 675mV.
kTMU_AmplifierReferenceVoltage682_5 TMU amplifier reference voltage 682.5mV.
kTMU_AmplifierReferenceVoltage690 TMU amplifier reference voltage 690mV.
kTMU_AmplifierReferenceVoltage697_5 TMU amplifier reference voltage 697.5mV.
kTMU_AmplifierReferenceVoltage705 TMU amplifier reference voltage 705mV.
kTMU_AmplifierReferenceVoltage712_5 TMU amplifier reference voltage 712.5mV.
kTMU_AmplifierReferenceVoltage720 TMU amplifier reference voltage 720mV.
kTMU_AmplifierReferenceVoltage727_5 TMU amplifier reference voltage 727.5mV.
kTMU_AmplifierReferenceVoltage735 TMU amplifier reference voltage 735mV.
kTMU_AmplifierReferenceVoltage742_5 TMU amplifier reference voltage 742.5mV.

10.6 Function Documentation

10.6.1 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.

10.6.2 void TMU_Deinit (TMU_Type * *base*)

Parameters

base	TMU peripheral base address.

10.6.3 void TMU_GetDefaultConfig (tmu_config_t * *config*)

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

Example:

```
config->amplifierGain = 12U;
config->amplifierRef = 19U;
config->averageLPF = kTMU_AverageLowPassFilter0_5;
```

Parameters

config	Pointer to TMU configuration structure.

10.6.4 static void TMU_Enable (TMU_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	TMU peripheral base address.
enable	Switcher to enable/disable TMU.

10.6.5 static void TMU_EnableInterrupts (TMU_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

bas	TMU peripheral base address.
mask	The interrupt mask. Refer to "_tmu_interrupt_enable" enumeration.

10.6.6 static void TMU_DisableInterrupts (TMU_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

bas	TMU peripheral base address.
mask	The interrupt mask. Refer to "_tmu_interrupt_enable" enumeration.

10.6.7 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.

10.6.8 void TMU_ClearInterruptStatusFlags (TMU_Type ∗ *base*, uint32_t *mask*)

Parameters

base	TMU peripheral base address.
The	mask of interrupt status flags. Refer to "_tmu_interrupt_status_flags" enumeration.

10.6.9 status_t TMU_GetImmediateTemperature (TMU_Type ∗ *base*, uint32_t ∗ *temperature*)

Parameters

base	TMU peripheral base address.
temperature	Last immediate temperature reading at site when V=1.

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.

10.6.10 status_t TMU_GetAverageTemperature (TMU_Type ∗ *base,* uint32_t ∗ *temperature*)

Parameters

base	TMU peripheral base address.
temperature	Last average temperature reading at site.

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.

10.6.11 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.

10.7 Variable Documentation

10.7.1 bool tmu_thresold_config_t::immediateThresoldEnable

10.7.2 bool tmu_thresold_config_t::AverageThresoldEnable

10.7.3 bool tmu_thresold_config_t::AverageCriticalThresoldEnable

10.7.4 uint8_t tmu_thresold_config_t::immediateThresoldValue

Valid when corresponding threshold is enabled. High temperature immediate threshold value. Determines the current upper temperature threshold, for any enabled monitored site.

10.7.5 uint8_t tmu_thresold_config_t::averageThresoldValue

Valid when corresponding threshold is enabled. High temperature average threshold value. Determines the average upper temperature threshold, for any enabled monitored site.

10.7.6 uint8_t tmu_thresold_config_t::averageCriticalThresoldValue

Valid when corresponding threshold is enabled. High temperature average critical threshold value. Determines the average upper critical temperature threshold, for any enabled monitored site.

10.7.7 uint32_t tmu_interrupt_status_t::interruptDetectMask

Refer to "_tmu_interrupt_status_flags" enumeration.

10.7.8 tmu_average_low_pass_filter_t tmu_config_t::averageLPF

For proper operation, this field should only change when monitoring is disabled.

10.7.9 tmu_amplifier_gain_t tmu_config_t::amplifierGain

10.7.10 tmu_amplifier_reference_voltage_t tmu_config_t::amplifierVref

Chapter 11
PDM: Microphone Interface

11.1 Overview

Modules

- PDM Driver
- PDM SDMA Driver

11.2 PDM Driver

11.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Microphone Interface (PDM) module of MCUXpresso SDK devices.

PDM driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for PDM initialization, configuration, and operation for the optimization and customization purpose. Using the functional API requires the knowledge of the PDM peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. PDM 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. Initialize the handle by calling the PDM_TransferCreateHandle() API.

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

11.2.2 Typical use case

11.2.2.1 PDM receive using an interrupt method

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

11.2.2.2 PDM receive using a DMA method

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

11.2.2.3 PDM receive using a transactional method

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

Data Structures

- struct pdm_channel_config_t
  PDM channel configurations. More...

- struct pdm_config_t
  PDM user configuration structure. More...
- struct pdm_hwvad_config_t
  PDM voice activity detector user configuration structure. More...
- struct pdm_hwvad_noise_filter_t
  PDM voice activity detector noise filter user configuration structure. More...
- struct pdm_hwvad_zero_cross_detector_t
  PDM voice activity detector zero cross detector configuration structure. More...
- struct pdm_transfer_t
  PDM SDMA transfer structure. More...
- struct pdm_handle_t
  PDM handle structure. More...

Macros

- #define PDM_XFER_QUEUE_SIZE (4)
  PDM XFER QUEUE SIZE.

Typedefs

- typedef void(∗ pdm_transfer_callback_t)(PDM_Type ∗base, pdm_handle_t ∗handle, status_t status, void ∗userData)
  PDM transfer callback prototype.

Enumerations

- enum _pdm_status {
  kStatus_PDM_Busy = MAKE_STATUS(kStatusGroup_PDM, 0),
  kStatus_PDM_CLK_LOW = MAKE_STATUS(kStatusGroup_PDM, 1),
  kStatus_PDM_FIFO_ERROR = MAKE_STATUS(kStatusGroup_PDM, 2),
  kStatus_PDM_QueueFull = MAKE_STATUS(kStatusGroup_PDM, 3),
  kStatus_PDM_Idle = MAKE_STATUS(kStatusGroup_PDM, 4) }
  PDM return status.
- enum _pdm_interrupt_enable {
  kPDM_ErrorInterruptEnable = PDM_CTRL_1_ERREN_MASK,
  kPDM_FIFOInterruptEnable = PDM_CTRL_1_DISEL(2U) }
  The PDM interrupt enable flag.
- enum _pdm_internal_status {

kPDM_StatusDfBusyFlag = PDM_STAT_BSY_FIL_MASK,
kPDM_StatusFIRFilterReady = PDM_STAT_FIR_RDY_MASK,
kPDM_StatusFrequencyLow = PDM_STAT_LOWFREQF_MASK,
kPDM_StatusCh0FifoDataAvaliable = PDM_STAT_CH0F_MASK,
kPDM_StatusCh1FifoDataAvaliable = PDM_STAT_CH1F_MASK,
kPDM_StatusCh2FifoDataAvaliable = PDM_STAT_CH2F_MASK,
kPDM_StatusCh3FifoDataAvaliable = PDM_STAT_CH3F_MASK,
kPDM_StatusCh4FifoDataAvaliable = PDM_STAT_CH4F_MASK,
kPDM_StatusCh5FifoDataAvaliable = PDM_STAT_CH5F_MASK,
kPDM_StatusCh6FifoDataAvaliable = PDM_STAT_CH6F_MASK,
kPDM_StatusCh7FifoDataAvaliable = PDM_STAT_CH7F_MASK }
The PDM status.

- enum _pdm_channel_enable_mask {
kPDM_EnableChannel0 = PDM_STAT_CH0F_MASK,
kPDM_EnableChannel1 = PDM_STAT_CH1F_MASK,
kPDM_EnableChannel2 = PDM_STAT_CH2F_MASK,
kPDM_EnableChannel3 = PDM_STAT_CH3F_MASK,
kPDM_EnableChannel4 = PDM_STAT_CH4F_MASK,
kPDM_EnableChannel5 = PDM_STAT_CH5F_MASK,
kPDM_EnableChannel6 = PDM_STAT_CH6F_MASK,
kPDM_EnableChannel7 = PDM_STAT_CH7F_MASK }
PDM channel enable mask.
- enum _pdm_fifo_status {
kPDM_FifoStatusUnderflowCh0 = PDM_FIFO_STAT_FIFOUND0_MASK,
kPDM_FifoStatusUnderflowCh1 = PDM_FIFO_STAT_FIFOUND1_MASK,
kPDM_FifoStatusUnderflowCh2 = PDM_FIFO_STAT_FIFOUND2_MASK,
kPDM_FifoStatusUnderflowCh3 = PDM_FIFO_STAT_FIFOUND3_MASK,
kPDM_FifoStatusUnderflowCh4 = PDM_FIFO_STAT_FIFOUND4_MASK,
kPDM_FifoStatusUnderflowCh5 = PDM_FIFO_STAT_FIFOUND5_MASK,
kPDM_FifoStatusUnderflowCh6 = PDM_FIFO_STAT_FIFOUND6_MASK,
kPDM_FifoStatusUnderflowCh7 = PDM_FIFO_STAT_FIFOUND6_MASK,
kPDM_FifoStatusOverflowCh0 = PDM_FIFO_STAT_FIFOOVF0_MASK,
kPDM_FifoStatusOverflowCh1 = PDM_FIFO_STAT_FIFOOVF1_MASK,
kPDM_FifoStatusOverflowCh2 = PDM_FIFO_STAT_FIFOOVF2_MASK,
kPDM_FifoStatusOverflowCh3 = PDM_FIFO_STAT_FIFOOVF3_MASK,
kPDM_FifoStatusOverflowCh4 = PDM_FIFO_STAT_FIFOOVF4_MASK,
kPDM_FifoStatusOverflowCh5 = PDM_FIFO_STAT_FIFOOVF5_MASK,
kPDM_FifoStatusOverflowCh6 = PDM_FIFO_STAT_FIFOOVF6_MASK,
kPDM_FifoStatusOverflowCh7 = PDM_FIFO_STAT_FIFOOVF7_MASK }
The PDM fifo status.
- enum _pdm_output_status {

kPDM_OutputStatusUnderFlowCh0 = PDM_OUT_STAT_OUTUNF0_MASK,
kPDM_OutputStatusUnderFlowCh1 = PDM_OUT_STAT_OUTUNF1_MASK,
kPDM_OutputStatusUnderFlowCh2 = PDM_OUT_STAT_OUTUNF2_MASK,
kPDM_OutputStatusUnderFlowCh3 = PDM_OUT_STAT_OUTUNF3_MASK,
kPDM_OutputStatusUnderFlowCh4 = PDM_OUT_STAT_OUTUNF4_MASK,
kPDM_OutputStatusUnderFlowCh5 = PDM_OUT_STAT_OUTUNF5_MASK,
kPDM_OutputStatusUnderFlowCh6 = PDM_OUT_STAT_OUTUNF6_MASK,
kPDM_OutputStatusUnderFlowCh7 = PDM_OUT_STAT_OUTUNF7_MASK,
kPDM_OutputStatusOverFlowCh0 = PDM_OUT_STAT_OUTOVF0_MASK,
kPDM_OutputStatusOverFlowCh1 = PDM_OUT_STAT_OUTOVF1_MASK,
kPDM_OutputStatusOverFlowCh2 = PDM_OUT_STAT_OUTOVF2_MASK,
kPDM_OutputStatusOverFlowCh3 = PDM_OUT_STAT_OUTOVF3_MASK,
kPDM_OutputStatusOverFlowCh4 = PDM_OUT_STAT_OUTOVF4_MASK,
kPDM_OutputStatusOverFlowCh5 = PDM_OUT_STAT_OUTOVF5_MASK,
kPDM_OutputStatusOverFlowCh6 = PDM_OUT_STAT_OUTOVF6_MASK,
kPDM_OutputStatusOverFlowCh7 = PDM_OUT_STAT_OUTOVF7_MASK }
  The PDM output status.

- enum pdm_dc_remover_t {
  kPDM_DcRemoverCutOff21Hz = 0U,
  kPDM_DcRemoverCutOff83Hz = 1U,
  kPDM_DcRemoverCutOff152Hz = 2U,
  kPDM_DcRemoverBypass = 3U }
  PDM DC remover configurations.
- enum pdm_df_quality_mode_t {
  kPDM_QualityModeMedium = 0U,
  kPDM_QualityModeHigh = 1U,
  kPDM_QualityModeLow = 7U,
  kPDM_QualityModeVeryLow0 = 6U,
  kPDM_QualityModeVeryLow1 = 5U,
  kPDM_QualityModeVeryLow2 = 4U }
  PDM decimation filter quality mode.
- enum _pdm_qulaity_mode_k_factor {
  kPDM_QualityModeHighKFactor = 1U,
  kPDM_QualityModeMediumKFactor = 2U,
  kPDM_QualityModeLowKFactor = 4U,
  kPDM_QualityModeVeryLow2KFactor = 8U }
  PDM quality mode K factor.
- enum pdm_df_output_gain_t {

kPDM_DfOutputGain0 = 0U,
kPDM_DfOutputGain1 = 1U,
kPDM_DfOutputGain2 = 2U,
kPDM_DfOutputGain3 = 3U,
kPDM_DfOutputGain4 = 4U,
kPDM_DfOutputGain5 = 5U,
kPDM_DfOutputGain6 = 6U,
kPDM_DfOutputGain7 = 7U,
kPDM_DfOutputGain8 = 8U,
kPDM_DfOutputGain9 = 9U,
kPDM_DfOutputGain10 = 0xAU,
kPDM_DfOutputGain11 = 0xBU,
kPDM_DfOutputGain12 = 0xCU,
kPDM_DfOutputGain13 = 0xDU,
kPDM_DfOutputGain14 = 0xEU,
kPDM_DfOutputGain15 = 0xFU }
PDM decimation filter output gain.

- enum _pdm_hwvad_interrupt_enable {
kPDM_HwvadErrorInterruptEnable = PDM_VAD0_CTRL_1_VADERIE_MASK,
kPDM_HwvadInterruptEnable = PDM_VAD0_CTRL_1_VADIE_MASK }
PDM voice activity detector interrupt type.
- enum _pdm_hwvad_int_status {
kPDM_HwvadStatusInputSaturation = PDM_VAD0_STAT_VADINSATF_MASK,
kPDM_HwvadStatusVoiceDetectFlag = PDM_VAD0_STAT_VADIF_MASK }
The PDM hwvad interrupt status flag.
- enum pdm_hwvad_hpf_config_t {
kPDM_HwvadHpfBypassed = 0x0U,
kPDM_HwvadHpfCutOffFreq1750Hz = 0x1U,
kPDM_HwvadHpfCutOffFreq215Hz = 0x2U }
High pass filter configure cut-off frequency.
- enum pdm_hwvad_filter_status_t {
kPDM_HwvadInternalFilterNormalOperation = 0U,
kPDM_HwvadInternalFilterInitial = PDM_VAD0_CTRL_1_VADST10_MASK }
HWVAD internal filter status.
- enum pdm_hwvad_zcd_result_t {
kPDM_HwvadResultOREnergyBasedDetection,
kPDM_HwvadResultANDEnergyBasedDetection }
PDM voice activity detector zero cross detector result.

Driver version

- #define FSL_PDM_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
Version 2.0.1.

Initialization and deinitialization

- void PDM_Init (PDM_Type ∗base, const pdm_config_t ∗config)
  Initializes the PDM peripheral.
- void PDM_Deinit (PDM_Type ∗base)
  De-initializes the PDM peripheral.
- static void PDM_Reset (PDM_Type ∗base)
  Resets the PDM module.
- static void PDM_Enable (PDM_Type ∗base, bool enable)
  Enables/disables PDM interface.
- static void PDM_EnableDoze (PDM_Type ∗base, bool enable)
  Enables/disables DOZE.
- static void PDM_EnableDebugMode (PDM_Type ∗base, bool enable)
  Enables/disables debug mode for PDM.
- static void PDM_EnableInDebugMode (PDM_Type ∗base, bool enable)
  Enables/disables PDM interface in debug mode.
- static void PDM_EnterLowLeakageMode (PDM_Type ∗base, bool enable)
  Enables/disables PDM interface disable/Low Leakage mode.
- static void PDM_EnableChannel (PDM_Type ∗base, uint8_t channel, bool enable)
  Enables/disables the PDM channel.
- void PDM_SetChannelConfig (PDM_Type ∗base, uint32_t channel, const pdm_channel_config_t ∗config)
  PDM one channel configurations.
- status_t PDM_SetSampleRate (PDM_Type ∗base, uint32_t enableChannelMask, pdm_df_quality_-mode_t qualityMode, uint8_t osr, uint32_t clkDiv)
  PDM set sample rate.

Status

- static uint32_t PDM_GetStatus (PDM_Type ∗base)
  Gets the PDM internal status flag.
- static uint32_t PDM_GetFifoStatus (PDM_Type ∗base)
  Gets the PDM FIFO status flag.
- static uint32_t PDM_GetOutputStatus (PDM_Type ∗base)
  Gets the PDM output status flag.
- static void PDM_ClearStatus (PDM_Type ∗base, uint32_t mask)
  Clears the PDM Tx status.
- static void PDM_ClearFIFOStatus (PDM_Type ∗base, uint32_t mask)
  Clears the PDM Tx status.
- static void PDM_ClearOutputStatus (PDM_Type ∗base, uint32_t mask)
  Clears the PDM output status.

Interrupts

- void PDM_EnableInterrupts (PDM_Type ∗base, uint32_t mask)
  Enables the PDM interrupt requests.
- static void PDM_DisableInterrupts (PDM_Type ∗base, uint32_t mask)
  Disables the PDM interrupt requests.

DMA Control

- static void PDM_EnableDMA (PDM_Type ∗base, bool enable)
  Enables/disables the PDM DMA requests.
- static uint32_t PDM_GetDataRegisterAddress (PDM_Type ∗base, uint32_t channel)
  Gets the PDM data register address.

Bus Operations

- static int16_t PDM_ReadData (PDM_Type ∗base, uint32_t channel)
  Reads data from the PDM FIFO.

Voice Activity Detector

- void PDM_SetHwvadConfig (PDM_Type ∗base, const pdm_hwvad_config_t ∗config)
  Configure voice activity detector.
- static void PDM_ForceHwvadOutputDisable (PDM_Type ∗base, bool enable)
  PDM hwvad force output disable.
- static void PDM_ResetHwvad (PDM_Type ∗base)
  PDM hwvad reset.
- static void PDM_EnableHwvad (PDM_Type ∗base, bool enable)
  Enable/Disable Voice activity detector.
- static void PDM_EnableHwvadInterrupts (PDM_Type ∗base, uint32_t mask)
  Enables the PDM Voice Detector interrupt requests.
- static void PDM_DisableHwvadInterrupts (PDM_Type ∗base, uint32_t mask)
  Disables the PDM Voice Detector interrupt requests.
- static void PDM_ClearHwvadInterruptStatusFlags (PDM_Type ∗base, uint32_t mask)
  Clears the PDM voice activity detector status flags.
- static uint32_t PDM_GetHwvadInterruptStatusFlags (PDM_Type ∗base)
  Clears the PDM voice activity detector status flags.
- static uint32_t PDM_GetHwvadInitialFlag (PDM_Type ∗base)
  Get the PDM voice activity detector initial flags.
- static uint32_t PDM_GetHwvadVoiceDetectedFlag (PDM_Type ∗base)
  Get the PDM voice activity detector voice detected flags.
- static void PDM_EnableHwvadSignalFilter (PDM_Type ∗base, bool enable)
  Enables/disables voice activity detector signal filter.
- void PDM_SetHwvadSignalFilterConfig (PDM_Type ∗base, bool enableMaxBlock, uint32_t signalGain)
  Configure voice activity detector signal filter.
- void PDM_SetHwvadNoiseFilterConfig (PDM_Type ∗base, const pdm_hwvad_noise_filter_t ∗config)
  Configure voice activity detector noise filter.
- static void PDM_EnableHwvadZeroCrossDetector (PDM_Type ∗base, bool enable)
  Enables/disables voice activity detector zero cross detector.
- void PDM_SetHwvadZeroCrossDetectorConfig (PDM_Type ∗base, const pdm_hwvad_zero_cross-_detector_t ∗config)
  Configure voice activity detector zero cross detector.
- static uint16_t PDM_GetNoiseData (PDM_Type ∗base)

Reads noise data.
- static void PDM_SetHwvadInternalFilterStatus (PDM_Type ∗base, pdm_hwvad_filter_status_t status)
  set hwvad internal filter status .

Transactional

- void PDM_TransferCreateHandle (PDM_Type ∗base, pdm_handle_t ∗handle, pdm_transfer_callback_t callback, void ∗userData)
  Initializes the PDM handle.
- void PDM_ReadNonBlocking (PDM_Type ∗base, uint32_t startChannel, uint32_t channelNums, int16_t ∗buffer, size_t size)
  PDM read data non blocking.
- status_t PDM_TransferReceiveNonBlocking (PDM_Type ∗base, pdm_handle_t ∗handle, pdm_transfer_t ∗xfer)
  Performs an interrupt non-blocking receive transfer on PDM.
- void PDM_TransferAbortReceive (PDM_Type ∗base, pdm_handle_t ∗handle)
  Aborts the current IRQ receive.
- void PDM_TransferHandleIRQ (PDM_Type ∗base, pdm_handle_t ∗handle)
  Tx interrupt handler.

11.2.3 Data Structure Documentation

11.2.3.1 struct pdm_channel_config_t

Data Fields

- pdm_dc_remover_t cutOffFreq
  DC remover cut off frequency.
- pdm_df_output_gain_t gain
  Decimation Filter Output Gain.

11.2.3.2 struct pdm_config_t

Data Fields

- bool enableDoze
  This module will enter disable/low leakage mode if DOZEN is active with ipg_doze is asserted.
- uint8_t fifoWatermark
  Watermark value for FIFO.
- pdm_df_quality_mode_t qualityMode
  Quality mode.
- uint8_t cicOverSampleRate
  CIC filter over sampling rate.

11.2.3.3 struct pdm_hwvad_config_t

Data Fields

- uint8_t channel
  Which channel uses voice activity detector.
- uint8_t initializeTime
  Number of frames or samples to initialize voice activity detector.
- uint8_t cicOverSampleRate
  CIC filter over sampling rate.
- uint8_t inputGain
  Voice activity detector input gain.
- uint32_t frameTime
  Voice activity frame time.
- pdm_hwvad_hpf_config_t cutOffFreq
  High pass filter cut off frequency.
- bool enableFrameEnergy
  If frame energy enabled, true means enable.
- bool enablePreFilter
  If pre-filter enabled.

11.2.3.3.0.10 Field Documentation

11.2.3.3.0.10.1 uint8_t pdm_hwvad_config_t::initializeTime

11.2.3.4 struct pdm_hwvad_noise_filter_t

Data Fields

- bool enableAutoNoiseFilter
  If noise fileter automatically activated, true means enable.
- bool enableNoiseMin
  If Noise minimum block enabled, true means enabled.
- bool enableNoiseDecimation
  If enable noise input decimation.
- bool enableNoiseDetectOR
  Enables a OR logic in the output of minimum noise estimator block.
- uint32_t noiseFilterAdjustment
  The adjustment value of the noise filter.
- uint32_t noiseGain
  Gain value for the noise energy or envelope estimated.

11.2.3.5 struct pdm_hwvad_zero_cross_detector_t

Data Fields

- bool enableAutoThreshold
  If ZCD auto-threshold enabled, true means enabled.
- pdm_hwvad_zcd_result_t zcdAnd
  Is ZCD result is AND'ed with energy-based detection, false means OR'ed.

- uint32_t threshold
  The adjustment value of the noise filter.
- uint32_t adjustmentThreshold
  Gain value for the noise energy or envelope estimated.

11.2.3.5.0.11 Field Documentation

11.2.3.5.0.11.1 bool pdm_hwvad_zero_cross_detector_t::enableAutoThreshold

11.2.3.6 struct pdm_transfer_t

Data Fields

- volatile uint8_t ∗ data
  Data start address to transfer.
- volatile size_t dataSize
  Total Transfer bytes size.

11.2.3.6.0.12 Field Documentation

11.2.3.6.0.12.1 volatile uint8_t∗ pdm_transfer_t::data

11.2.3.6.0.12.2 volatile size_t pdm_transfer_t::dataSize

11.2.3.7 struct _pdm_handle

PDM handle.

Data Fields

- uint32_t state
  Transfer status.
- pdm_transfer_callback_t callback
  Callback function called at transfer event.
- void ∗ userData
  Callback parameter passed to callback function.
- pdm_transfer_t pdmQueue [PDM_XFER_QUEUE_SIZE]
  Transfer queue storing queued transfer.
- size_t transferSize [PDM_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.
- uint8_t startChannel
  end channel
- uint8_t channelNums
  Enabled channel number.

11.2.4 Enumeration Type Documentation

11.2.4.1 enum _pdm_status

Enumerator

kStatus_PDM_Busy	PDM is busy.
kStatus_PDM_CLK_LOW	PDM clock frequency low.
kStatus_PDM_FIFO_ERROR	PDM FIFO underrun or overflow.
kStatus_PDM_QueueFull	PDM FIFO underrun or overflow.
kStatus_PDM_Idle	PDM is idle.

11.2.4.2 enum _pdm_interrupt_enable

Enumerator

kPDM_ErrorInterruptEnable	PDM channel error interrupt enable.
kPDM_FIFOInterruptEnable	PDM channel FIFO interrupt.

11.2.4.3 enum _pdm_internal_status

Enumerator

kPDM_StatusDfBusyFlag	Decimation filter is busy processing data.
kPDM_StatusFIRFilterReady	FIR filter data is ready.
kPDM_StatusFrequencyLow	Mic app clock frequency not high enough.
kPDM_StatusCh0FifoDataAvaliable	channel 0 fifo data reached watermark level
kPDM_StatusCh1FifoDataAvaliable	channel 1 fifo data reached watermark level
kPDM_StatusCh2FifoDataAvaliable	channel 2 fifo data reached watermark level
kPDM_StatusCh3FifoDataAvaliable	channel 3 fifo data reached watermark level
kPDM_StatusCh4FifoDataAvaliable	channel 4 fifo data reached watermark level
kPDM_StatusCh5FifoDataAvaliable	channel 5 fifo data reached watermark level
kPDM_StatusCh6FifoDataAvaliable	channel 6 fifo data reached watermark level
kPDM_StatusCh7FifoDataAvaliable	channel 7 fifo data reached watermark level

11.2.4.4 enum _pdm_channel_enable_mask

Enumerator

kPDM_EnableChannel0	channgel 0 enable mask
kPDM_EnableChannel1	channgel 1 enable mask
kPDM_EnableChannel2	channgel 2 enable mask
kPDM_EnableChannel3	channgel 3 enable mask

kPDM_EnableChannel4	channgel 4 enable mask
kPDM_EnableChannel5	channgel 5 enable mask
kPDM_EnableChannel6	channgel 6 enable mask
kPDM_EnableChannel7	channgel 7 enable mask

11.2.4.5 enum _pdm_fifo_status

Enumerator

kPDM_FifoStatusUnderflowCh0	channel0 fifo status underflow
kPDM_FifoStatusUnderflowCh1	channel1 fifo status underflow
kPDM_FifoStatusUnderflowCh2	channel2 fifo status underflow
kPDM_FifoStatusUnderflowCh3	channel3 fifo status underflow
kPDM_FifoStatusUnderflowCh4	channel4 fifo status underflow
kPDM_FifoStatusUnderflowCh5	channel5 fifo status underflow
kPDM_FifoStatusUnderflowCh6	channel6 fifo status underflow
kPDM_FifoStatusUnderflowCh7	channel7 fifo status underflow
kPDM_FifoStatusOverflowCh0	channel0 fifo status overflow
kPDM_FifoStatusOverflowCh1	channel1 fifo status overflow
kPDM_FifoStatusOverflowCh2	channel2 fifo status overflow
kPDM_FifoStatusOverflowCh3	channel3 fifo status overflow
kPDM_FifoStatusOverflowCh4	channel4 fifo status overflow
kPDM_FifoStatusOverflowCh5	channel5 fifo status overflow
kPDM_FifoStatusOverflowCh6	channel6 fifo status overflow
kPDM_FifoStatusOverflowCh7	channel7 fifo status overflow

11.2.4.6 enum _pdm_output_status

Enumerator

kPDM_OutputStatusUnderFlowCh0	channel0 output status underflow
kPDM_OutputStatusUnderFlowCh1	channel1 output status underflow
kPDM_OutputStatusUnderFlowCh2	channel2 output status underflow
kPDM_OutputStatusUnderFlowCh3	channel3 output status underflow
kPDM_OutputStatusUnderFlowCh4	channel4 output status underflow
kPDM_OutputStatusUnderFlowCh5	channel5 output status underflow
kPDM_OutputStatusUnderFlowCh6	channel6 output status underflow
kPDM_OutputStatusUnderFlowCh7	channel7 output status underflow
kPDM_OutputStatusOverFlowCh0	channel0 output status overflow
kPDM_OutputStatusOverFlowCh1	channel1 output status overflow
kPDM_OutputStatusOverFlowCh2	channel2 output status overflow
kPDM_OutputStatusOverFlowCh3	channel3 output status overflow
kPDM_OutputStatusOverFlowCh4	channel4 output status overflow

kPDM_OutputStatusOverFlowCh5	channel5 output status overflow
kPDM_OutputStatusOverFlowCh6	channel6 output status overflow
kPDM_OutputStatusOverFlowCh7	channel7 output status overflow

11.2.4.7 enum pdm_dc_remover_t

Enumerator

kPDM_DcRemoverCutOff21Hz	DC remover cut off 21HZ.
kPDM_DcRemoverCutOff83Hz	DC remover cut off 83HZ.
kPDM_DcRemoverCutOff152Hz	DC remover cut off 152HZ.
kPDM_DcRemoverBypass	DC remover bypass.

11.2.4.8 enum pdm_df_quality_mode_t

Enumerator

kPDM_QualityModeMedium	quality mode memdium
kPDM_QualityModeHigh	quality mode high
kPDM_QualityModeLow	quality mode low
kPDM_QualityModeVeryLow0	quality mode very low0
kPDM_QualityModeVeryLow1	quality mode very low1
kPDM_QualityModeVeryLow2	quality mode very low2

11.2.4.9 enum _pdm_qulaity_mode_k_factor

Enumerator

kPDM_QualityModeHighKFactor	high quality mode K factor = 1 / 2
kPDM_QualityModeMediumKFactor	medium/very low0 quality mode K factor = 2 / 2
kPDM_QualityModeLowKFactor	low/very low1 quality mode K factor = 4 / 2
kPDM_QualityModeVeryLow2KFactor	very low2 quality mode K factor = 8 / 2

11.2.4.10 enum pdm_df_output_gain_t

Enumerator

kPDM_DfOutputGain0	Decimation filter output gain 0.
kPDM_DfOutputGain1	Decimation filter output gain 1.
kPDM_DfOutputGain2	Decimation filter output gain 2.
kPDM_DfOutputGain3	Decimation filter output gain 3.
kPDM_DfOutputGain4	Decimation filter output gain 4.

kPDM_DfOutputGain5 Decimation filter output gain 5.
kPDM_DfOutputGain6 Decimation filter output gain 6.
kPDM_DfOutputGain7 Decimation filter output gain 7.
kPDM_DfOutputGain8 Decimation filter output gain 8.
kPDM_DfOutputGain9 Decimation filter output gain 9.
kPDM_DfOutputGain10 Decimation filter output gain 10.
kPDM_DfOutputGain11 Decimation filter output gain 11.
kPDM_DfOutputGain12 Decimation filter output gain 12.
kPDM_DfOutputGain13 Decimation filter output gain 13.
kPDM_DfOutputGain14 Decimation filter output gain 14.
kPDM_DfOutputGain15 Decimation filter output gain 15.

11.2.4.11 enum _pdm_hwvad_interrupt_enable

Enumerator

kPDM_HwvadErrorInterruptEnable PDM channel HWVAD error interrupt enable.
kPDM_HwvadInterruptEnable PDM channel HWVAD interrupt.

11.2.4.12 enum _pdm_hwvad_int_status

Enumerator

kPDM_HwvadStatusInputSaturation HWVAD saturation condition.
kPDM_HwvadStatusVoiceDetectFlag HWVAD voice detect interrupt triggered.

11.2.4.13 enum pdm_hwvad_hpf_config_t

Enumerator

kPDM_HwvadHpfBypassed High-pass filter bypass.
kPDM_HwvadHpfCutOffFreq1750Hz High-pass filter cut off frequency 1750HZ.
kPDM_HwvadHpfCutOffFreq215Hz High-pass filter cut off frequency 215HZ.

11.2.4.14 enum pdm_hwvad_filter_status_t

Enumerator

kPDM_HwvadInternalFilterNormalOperation internal filter ready for normal operation
kPDM_HwvadInternalFilterInitial interla filter are initial

11.2.4.15 enum pdm_hwvad_zcd_result_t

Enumerator

kPDM_HwvadResultOREnergyBasedDetection zero cross detector result will be OR with energy based detection

kPDM_HwvadResultANDEnergyBasedDetection zero cross detector result will be AND with energy based detection

11.2.5 Function Documentation

11.2.5.1 void PDM_Init (PDM_Type ∗ *base,* const pdm_config_t ∗ *config*)

Ungates the PDM clock, resets the module, and configures PDM with a configuration structure. The configuration structure can be custom filled or set with default values by PDM_GetDefaultConfig().

Note

This API should be called at the beginning of the application to use the PDM driver. Otherwise, accessing the PDM module can cause a hard fault because the clock is not enabled.

Parameters

base	PDM base pointer
config	PDM configuration structure.

11.2.5.2 void PDM_Deinit (PDM_Type ∗ *base*)

This API gates the PDM clock. The PDM module can't operate unless PDM_Init is called to enable the clock.

Parameters

base	PDM base pointer

11.2.5.3 static void PDM_Reset (PDM_Type ∗ *base*) `[inline]`, `[static]`

Parameters

base	PDM base pointer

11.2.5.4 static void PDM_Enable (PDM_Type ∗ *base,* bool *enable*) `[inline]`, `[static]`

Parameters

base	PDM base pointer
enable	True means PDM interface is enabled, false means PDM interface is disabled.

11.2.5.5 static void PDM_EnableDoze (PDM_Type ∗ *base,* bool *enable*) `[inline]`, `[static]`

Parameters

base	PDM base pointer
enable	True means the module will enter Disable/Low Leakage mode when ipg_doze is asserted, false means the module will not enter Disable/Low Leakage mode when ipg_doze is asserted.

11.2.5.6 static void PDM_EnableDebugMode (PDM_Type ∗ *base,* bool *enable*) `[inline]`, `[static]`

The PDM interface cannot enter debug mode once in Disable/Low Leakage or Low Power mode.

Parameters

base	PDM base pointer
enable	True means PDM interface enter debug mode, false means PDM interface in normal mode.

11.2.5.7 static void PDM_EnableInDebugMode (PDM_Type ∗ *base,* bool *enable*) `[inline]`, `[static]`

Parameters

base	PDM base pointer
enable	True means PDM interface is enabled debug mode, false means PDM interface is disabled after after completing the current frame in debug mode.

11.2.5.8 static void PDM_EnterLowLeakageMode (PDM_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	PDM base pointer
enable	True means PDM interface is in disable/low leakage mode, False means PDM interface is in normal mode.

11.2.5.9 static void PDM_EnableChannel (PDM_Type ∗ *base,* uint8_t *channel,* bool *enable*) [inline], [static]

Parameters

base	PDM base pointer
channel	PDM channel number need to enable or disable.
enable	True means enable PDM channel, false means disable.

11.2.5.10 void PDM_SetChannelConfig (PDM_Type ∗ *base,* uint32_t *channel,* const pdm_channel_config_t ∗ *config*)

Parameters

base	PDM base pointer
config	PDM channel configurations.
channel	channel number. after completing the current frame in debug mode.

11.2.5.11 status_t PDM_SetSampleRate (PDM_Type ∗ *base,* uint32_t *enableChannelMask,* pdm_df_quality_mode_t *qualityMode,* uint8_t *osr,* uint32_t *clkDiv*)

Parameters

base	PDM base pointer
enable-ChannelMask	PDM channel enable mask.
qualityMode	quality mode.
osr	cic oversample rate
clkDiv	clock divider after completing the current frame in debug mode.

11.2.5.12 static uint32_t PDM_GetStatus (PDM_Type ∗ *base*) `[inline]`, `[static]`

Use the Status Mask in _pdm_internal_status to get the status value needed

Parameters

base	PDM base pointer

Returns

PDM status flag value.

11.2.5.13 static uint32_t PDM_GetFifoStatus (PDM_Type ∗ *base*) `[inline]`, `[static]`

Use the Status Mask in _pdm_fifo_status to get the status value needed

Parameters

base	PDM base pointer

Returns

FIFO status.

11.2.5.14 static uint32_t PDM_GetOutputStatus (PDM_Type ∗ *base*) `[inline]`, `[static]`

Use the Status Mask in _pdm_output_status to get the status value needed

Parameters

base	PDM base pointer

Returns

output status.

11.2.5.15 static void PDM_ClearStatus (PDM_Type ∗ *base*, uint32_t *mask*) `[inline]`, `[static]`

Parameters

base	PDM base pointer
mask	State mask. It can be a combination of the status between kPDM_StatusFrequency-Low and kPDM_StatusCh7FifoDataAvaliable.

11.2.5.16 static void PDM_ClearFIFOStatus (PDM_Type ∗ *base*, uint32_t *mask*) `[inline]`, `[static]`

Parameters

base	PDM base pointer
mask	State mask.It can be a combination of the status in _pdm_fifo_status.

11.2.5.17 static void PDM_ClearOutputStatus (PDM_Type ∗ *base*, uint32_t *mask*) `[inline]`, `[static]`

Parameters

base	PDM base pointer
mask	State mask. It can be a combination of the status in _pdm_output_status.

11.2.5.18 void PDM_EnableInterrupts (PDM_Type ∗ *base*, uint32_t *mask*)

Parameters

base	PDM base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kPDM_ErrorInterruptEnable • kPDM_FIFOInterruptEnable

11.2.5.19 static void PDM_DisableInterrupts (PDM_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	PDM base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kPDM_ErrorInterruptEnable • kPDM_FIFOInterruptEnable

11.2.5.20 static void PDM_EnableDMA (PDM_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	PDM base pointer
enable	True means enable DMA, false means disable DMA.

11.2.5.21 static uint32_t PDM_GetDataRegisterAddress (PDM_Type ∗ *base,* uint32_t *channel*) [inline], [static]

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

Parameters

base	PDM base pointer.
channel	Which data channel used.

Returns

data register address.

11.2.5.22 static int16_t PDM_ReadData (PDM_Type ∗ *base,* uint32_t *channel*) [inline], [static]

Parameters

base	PDM base pointer.
channel	Data channel used.

Returns

Data in PDM FIFO.

11.2.5.23 void PDM_SetHwvadConfig (PDM_Type ∗ *base,* const pdm_hwvad_config_t ∗ *config*)

Parameters

base	PDM base pointer
config	Voice activity detector configure structure pointer .

11.2.5.24 static void PDM_ForceHwvadOutputDisable (PDM_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	PDM base pointer
enable,true	is output force disable, false is output not force.

11.2.5.25 static void PDM_ResetHwvad (PDM_Type ∗ *base*) [inline], [static]

It will reset VADNDATA register and will clean all internal buffers, should be called when the PDM isn't running.

Parameters

base	PDM base pointer

11.2.5.26 static void PDM_EnableHwvad (PDM_Type ∗ *base*, bool *enable*) [inline], [static]

Should be called when the PDM isn't running.

Parameters

base	PDM base pointer.
enable	True means enable voice activity detector, false means disable.

11.2.5.27 static void PDM_EnableHwvadInterrupts (PDM_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PDM base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kPDM_HWVADErrorInterruptEnable • kPDM_HWVADInterruptEnable

11.2.5.28 static void PDM_DisableHwvadInterrupts (PDM_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	PDM base pointer
mask	interrupt source The parameter can be a combination of the following sources if defined. • kPDM_HWVADErrorInterruptEnable • kPDM_HWVADInterruptEnable

11.2.5.29 static void PDM_ClearHwvadInterruptStatusFlags (PDM_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	PDM base pointer
mask	State mask,reference _pdm_hwvad_int_status.

11.2.5.30 static uint32_t PDM_GetHwvadInterruptStatusFlags (PDM_Type ∗ *base*) [inline], [static]

Parameters

base	PDM base pointer

Returns

status, reference _pdm_hwvad_int_status

11.2.5.31 static uint32_t PDM_GetHwvadInitialFlag (PDM_Type ∗ *base*) [inline], [static]

Parameters

base	PDM base pointer

Returns

initial flag.

11.2.5.32 static uint32_t PDM_GetHwvadVoiceDetectedFlag (PDM_Type ∗ *base*) [inline], [static]

NOte: this flag is auto cleared when voice gone.

Parameters

base	PDM base pointer

Returns

voice detected flag.

11.2.5.33 static void PDM_EnableHwvadSignalFilter (PDM_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	PDM base pointer
enable	True means enable signal filter, false means disable.

11.2.5.34 void PDM_SetHwvadSignalFilterConfig (PDM_Type ∗ *base,* bool *enableMaxBlock,* uint32_t *signalGain*)

Parameters

base	PDM base pointer
enableMax-Block	If signal maximum block enabled.
signalGain	Gain value for the signal energy.

11.2.5.35 void PDM_SetHwvadNoiseFilterConfig (PDM_Type ∗ *base,* const pdm_hwvad_noise_filter_t ∗ *config*)

Parameters

base	PDM base pointer
config	Voice activity detector noise filter configure structure pointer .

11.2.5.36 static void PDM_EnableHwvadZeroCrossDetector (PDM_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	PDM base pointer
enable	True means enable zero cross detector, false means disable.

11.2.5.37 void PDM_SetHwvadZeroCrossDetectorConfig (PDM_Type ∗ *base,* const pdm_hwvad_zero_cross_detector_t ∗ *config*)

Parameters

base	PDM base pointer
config	Voice activity detector zero cross detector configure structure pointer .

11.2.5.38 static uint16_t PDM_GetNoiseData (PDM_Type ∗ *base*) [inline], [static]

Parameters

base	PDM base pointer.

Returns

Data in PDM noise data register.

11.2.5.39 static void PDM_SetHwvadInternalFilterStatus (PDM_Type ∗ *base*, pdm_hwvad_filter_status_t *status*) [inline], [static]

Note: filter initial status should be asserted for two more cycles, then set it to normal operation.

Parameters

base	PDM base pointer.
status	internal filter status.

11.2.5.40 void PDM_TransferCreateHandle (PDM_Type ∗ *base*, pdm_handle_t ∗ *handle*, pdm_transfer_callback_t *callback*, void ∗ *userData*)

This function initializes the handle for the PDM transactional APIs. Call this function once to get the handle initialized.

Parameters

base	PDM base pointer.

handle	PDM handle pointer.
callback	Pointer to the user callback function.
userData	User parameter passed to the callback function.

11.2.5.41 void PDM_ReadNonBlocking (PDM_Type ∗ *base,* uint32_t *startChannel,* uint32_t *channelNums,* int16_t ∗ *buffer,* size_t *size*)

So the actually read data byte size in this function is (size ∗ 2 ∗ channelNums).

Parameters

base	PDM base pointer.
startChannel	start channel number.
channelNums	total enabled channelnums.
buffer	received buffer address.
size	number of 16bit data to read.

11.2.5.42 status_t PDM_TransferReceiveNonBlocking (PDM_Type ∗ *base,* pdm_handle_t ∗ *handle,* pdm_transfer_t ∗ *xfer*)

Note

This API returns immediately after the transfer initiates. Call the PDM_RxGetTransferStatusIR-Q to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_PDM_Busy, the transfer is finished.

Parameters

base	PDM base pointer
handle	Pointer to the pdm_handle_t structure which stores the transfer state.
xfer	Pointer to the pdm_transfer_t structure.

Return values

kStatus_Success	Successfully started the data receive.

kStatus_PDM_Busy	Previous receive still not finished.

11.2.5.43 void PDM_TransferAbortReceive (PDM_Type ∗ *base*, pdm_handle_t ∗ *handle*)

Note

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

Parameters

base	PDM base pointer
handle	Pointer to the pdm_handle_t structure which stores the transfer state.

11.2.5.44 void PDM_TransferHandleIRQ (PDM_Type ∗ *base*, pdm_handle_t ∗ *handle*)

Parameters

base	PDM base pointer.
handle	Pointer to the pdm_handle_t structure.

11.3 PDM SDMA Driver

11.3.1 Overview

Data Structures

- struct pdm_sdma_handle_t
  PDM DMA transfer handle, users should not touch the content of the handle. More...

Typedefs

- typedef void(∗ pdm_sdma_callback_t)(PDM_Type ∗base, pdm_sdma_handle_t ∗handle, status_t status, void ∗userData)
  PDM eDMA transfer callback function for finish and error.

Driver version

- #define FSL_PDM_SDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  Version 2.0.0.

eDMA Transactional

- void PDM_TransferCreateHandleSDMA (PDM_Type ∗base, pdm_sdma_handle_t ∗handle, pdm_sdma_callback_t callback, void ∗userData, sdma_handle_t ∗dmaHandle, uint32_t eventSource)
  Initializes the PDM eDMA handle.
- status_t PDM_TransferReceiveSDMA (PDM_Type ∗base, pdm_sdma_handle_t ∗handle, pdm_transfer_t ∗xfer)
  Performs a non-blocking PDM receive using eDMA.
- void PDM_TransferAbortReceiveSDMA (PDM_Type ∗base, pdm_sdma_handle_t ∗handle)
  Aborts a PDM receive using eDMA.
- void PDM_SetChannelConfigSDMA (PDM_Type ∗base, pdm_sdma_handle_t ∗handle, uint32_t channel, const pdm_channel_config_t ∗config)
  PDM channel configurations.

11.3.2 Data Structure Documentation

11.3.2.1 struct _pdm_sdma_handle

Data Fields

- sdma_handle_t ∗ dmaHandle
  DMA handler for PDM send.
- uint8_t nbytes
  eDMA minor byte transfer count initially configured.

- uint8_t fifoWidth
  fifo width
- uint8_t endChannel
  The last enabled channel.
- uint8_t channelNums
  total channel numbers
- uint8_t count
  The transfer data count in a DMA request.
- uint32_t state
  Internal state for PDM eDMA transfer.
- uint32_t eventSource
  PDM event source number.
- pdm_sdma_callback_t callback
  Callback for users while transfer finish or error occurs.
- void ∗ userData
  User callback parameter.
- sdma_buffer_descriptor_t bdPool [PDM_XFER_QUEUE_SIZE]
  BD pool for SDMA transfer.
- pdm_transfer_t pdmQueue [PDM_XFER_QUEUE_SIZE]
  Transfer queue storing queued transfer.
- size_t transferSize [PDM_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.

11.3.2.1.0.13 Field Documentation

11.3.2.1.0.13.1 uint8_t pdm_sdma_handle_t::nbytes

11.3.2.1.0.13.2 sdma_buffer_descriptor_t pdm_sdma_handle_t::bdPool[PDM_XFER_QUEUE_SIZE]

11.3.2.1.0.13.3 pdm_transfer_t pdm_sdma_handle_t::pdmQueue[PDM_XFER_QUEUE_SIZE]

11.3.2.1.0.13.4 volatile uint8_t pdm_sdma_handle_t::queueUser

11.3.3 Function Documentation

11.3.3.1 void PDM_TransferCreateHandleSDMA (PDM_Type ∗ *base,* pdm_sdma_handle_t ∗ *handle,* pdm_sdma_callback_t *callback,* void ∗ *userData,* sdma_handle_t ∗ *dmaHandle,* uint32_t *eventSource*)

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

Parameters

base	PDM base pointer.
handle	PDM eDMA handle pointer.
base	PDM peripheral base address.
callback	Pointer to user callback function.
userData	User parameter passed to the callback function.
dmaHandle	eDMA handle pointer, this handle shall be static allocated by users.
dma	request source.

11.3.3.2 status_t PDM_TransferReceiveSDMA (PDM_Type ∗ *base*, pdm_sdma_handle_t ∗ *handle*, pdm_transfer_t ∗ *xfer*)

Note

This interface returns immediately after the transfer initiates. Call the PDM_GetReceiveRemaining-Bytes to poll the transfer status and check whether the PDM transfer is finished.

Parameters

base	PDM base pointer
handle	PDM eDMA handle pointer.
xfer	Pointer to DMA transfer structure.

Return values

kStatus_Success	Start a PDM eDMA receive successfully.
kStatus_InvalidArgument	The input argument is invalid.
kStatus_RxBusy	PDM is busy receiving data.

11.3.3.3 void PDM_TransferAbortReceiveSDMA (PDM_Type ∗ *base*, pdm_sdma_handle_t ∗ *handle*)

Parameters

base	PDM base pointer
handle	PDM eDMA handle pointer.

11.3.3.4 void PDM_SetChannelConfigSDMA (PDM_Type ∗ *base*, pdm_sdma_handle_t ∗ *handle*, uint32_t *channel*, const pdm_channel_config_t ∗ *config*)

Parameters

base	PDM base pointer.
handle	PDM eDMA handle pointer.
channel	channel number.
config	channel configurations.

Chapter 12
PWM: Pulse Width Modulation Driver

12.1 Overview

Modules

- PWM Driver

12.2 PWM Driver

12.2.1 Overview

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

The function PWM_Init() initializes the PWM with a specified configurations. The function PWM_Get-DefaultConfig() 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.

12.2.2 Typical use case

12.2.2.1 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 = (1U << 6) }
  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.

12.2.3 Enumeration Type Documentation

12.2.3.1 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.

12.2.3.2 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.

12.2.3.3 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

12.2.3.4 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.

12.2.3.5 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.

12.2.3.6 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.

12.2.3.7 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.

12.2.3.8 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.

12.2.3.9 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

12.2.4 Function Documentation

12.2.4.1 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.

12.2.4.2 void PWM_Deinit (PWM_Type ∗ *base*)

Parameters

base	PWM peripheral base address

12.2.4.3 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.

12.2.4.4 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

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

Parameters

base	PWM peripheral base address

12.2.4.6 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

12.2.4.7 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

12.2.4.8 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

12.2.4.9 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

12.2.4.10 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

12.2.4.11 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

12.2.4.12 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

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

Parameters

base	PWM peripheral base address
mask	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.

12.2.4.14 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.

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

Parameters

base	PWM peripheral base address
mask	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)

12.2.4.16 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.

12.2.4.17 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 13
UART: Universal Asynchronous Receiver/Transmitter Driver

13.1 Overview

Modules

- UART Driver
- UART FreeRTOS Driver
- UART SDMA Driver

13.2 UART Driver

13.2.1 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.

13.2.2 Typical use case

13.2.2.1 UART Send/receive using a polling method

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

13.2.2.2 UART Send/receive using an interrupt method

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

13.2.2.3 UART Receive using the ringbuffer feature

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

13.2.2.4 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...

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 _uart_status {
  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) }
  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 _uart_flags {

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.

Variables

- uint32_t uart_config_t::baudRate_Bps
  UART baud rate.
- uart_parity_mode_t uart_config_t::parityMode
  Parity error check mode of this module.
- uart_data_bits_t uart_config_t::dataBitsCount
  Data bits count, eight (default), seven.
- uart_stop_bit_count_t uart_config_t::stopBitCount
  Number of stop bits in one frame.
- uint8_t uart_config_t::txFifoWatermark
  TX FIFO watermark.
- uint8_t uart_config_t::rxFifoWatermark
  RX FIFO watermark.
- bool uart_config_t::enableAutoBaudRate
  Enable automatic baud rate detection.
- bool uart_config_t::enableTx
  Enable TX.
- bool uart_config_t::enableRx
  Enable RX.
- uint8_t ∗ uart_transfer_t::data
  The buffer of data to be transfer.
- size_t uart_transfer_t::dataSize
  The byte count to be transfer.
- uint8_t ∗volatile uart_handle_t::txData
  Address of remaining data to send.
- volatile size_t uart_handle_t::txDataSize
  Size of the remaining data to send.
- size_t uart_handle_t::txDataSizeAll
  Size of the data to send out.
- uint8_t ∗volatile uart_handle_t::rxData
  Address of remaining data to receive.
- volatile size_t uart_handle_t::rxDataSize
  Size of the remaining data to receive.
- size_t uart_handle_t::rxDataSizeAll
  Size of the data to receive.
- uint8_t ∗ uart_handle_t::rxRingBuffer
  Start address of the receiver ring buffer.
- size_t uart_handle_t::rxRingBufferSize
  Size of the ring buffer.
- volatile uint16_t uart_handle_t::rxRingBufferHead
  Index for the driver to store received data into ring buffer.
- volatile uint16_t uart_handle_t::rxRingBufferTail
  Index for the user to get data from the ring buffer.
- uart_transfer_callback_t uart_handle_t::callback
  Callback function.
- void ∗ uart_handle_t::userData
  UART callback function parameter.
- volatile uint8_t uart_handle_t::txState
  TX transfer state.
- volatile uint8_t uart_handle_t::rxState

RX transfer state.

Driver version

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

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.
- void 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_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.

13.2.3 Data Structure Documentation

13.2.3.1 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.
- bool enableAutoBaudRate
  Enable automatic baud rate detection.
- bool enableTx
  Enable TX.
- bool enableRx
  Enable RX.

13.2.3.2 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.

13.2.3.3 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.

13.2.4 Typedef Documentation

13.2.4.1 typedef void(∗ uart_transfer_callback_t)(UART_Type ∗base, uart_handle_t ∗handle, status_t status, void ∗userData)

13.2.5 Enumeration Type Documentation

13.2.5.1 enum _uart_status

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.

13.2.5.2 enum uart_data_bits_t

Enumerator

kUART_SevenDataBits	Seven data bit.
kUART_EightDataBits	Eight data bit.

13.2.5.3 enum uart_parity_mode_t

Enumerator

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

13.2.5.4 enum uart_stop_bit_count_t

Enumerator

kUART_OneStopBit	One stop bit.
kUART_TwoStopBit	Two stop bits.

13.2.5.5 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.

13.2.5.6 enum _uart_interrupt_enable

13.2.5.7 enum _uart_flags

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.

13.2.6 Function Documentation

13.2.6.1 uint32_t UART_GetInstance (UART_Type ∗ *base*)

Parameters

base	UART peripheral base address.

Returns

UART instance.

13.2.6.2 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.

13.2.6.3 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_GetDefault-Config() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the 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

13.2.6.4 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.

13.2.6.5 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.

13.2.6.6 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.

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

Parameters

base	UART base pointer.

13.2.6.8 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.

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

Parameters

base	UART base pointer.

13.2.6.10 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.

13.2.6.11 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.

13.2.6.12 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.

13.2.6.13 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.

13.2.6.14 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.

13.2.6.15 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.

13.2.6.16 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.

13.2.6.17 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.

13.2.6.18 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.

13.2.6.19 void 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.

Note

> This function does not check whether all data is sent out to the bus. Before disabling the TX, check kUART_TransmissionCompleteFlag to ensure that the TX is finished.

Parameters

base	UART peripheral base address.
data	Start address of the data to write.
length	Size of the data to write.

13.2.6.20 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_Success	Successfully received all data.

13.2.6.21 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.

13.2.6.22 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 `ring-BufferSize` 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.

13.2.6.23 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.

13.2.6.24 size_t UART_TransferGetRxRingBufferLength (uart_handle_t ∗ *handle*)

Parameters

handle	UART handle pointer.

Returns

Length of received data in RX ring buffer.

13.2.6.25 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.

13.2.6.26 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.

13.2.6.27 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_NoTransferIn-Progress	No send in progress.
kStatus_InvalidArgument	The parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

13.2.6.28 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 k-Status_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 `received-Bytes` 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.

13.2.6.29 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.

13.2.6.30 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`;

13.2.6.31 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.

13.2.6.32 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.

13.2.6.33 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.

13.2.6.34 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.

13.2.6.35 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.

13.2.6.36 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.

13.2.6.37 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.

13.2.7 Variable Documentation

13.2.7.1 uint32_t uart_config_t::baudRate_Bps

13.2.7.2 uart_parity_mode_t uart_config_t::parityMode

13.2.7.3 uart_stop_bit_count_t uart_config_t::stopBitCount

13.2.7.4 uint8_t* uart_transfer_t::data

13.2.7.5 size_t uart_transfer_t::dataSize

13.2.7.6 uint8_t* volatile uart_handle_t::txData

13.2.7.7 volatile size_t uart_handle_t::txDataSize

13.2.7.8 size_t uart_handle_t::txDataSizeAll

13.2.7.9 uint8_t* volatile uart_handle_t::rxData

13.2.7.10 volatile size_t uart_handle_t::rxDataSize

13.2.7.11 size_t uart_handle_t::rxDataSizeAll

13.2.7.12 uint8_t* uart_handle_t::rxRingBuffer

13.2.7.13 size_t uart_handle_t::rxRingBufferSize

13.2.7.14 volatile uint16_t uart_handle_t::rxRingBufferHead

13.2.7.15 volatile uint16_t uart_handle_t::rxRingBufferTail

13.2.7.16 uart_transfer_callback_t uart_handle_t::callback

13.2.7.17 void* uart_handle_t::userData

13.2.7.18 volatile uint8_t uart_handle_t::txState

13.3 UART FreeRTOS Driver

13.3.1 Overview

Data Structures

- struct uart_rtos_config_t
  UART configuration structure. More...

Driver version

- #define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  UART freertos driver version 2.0.0.

UART RTOS Operation

- int UART_RTOS_Init (uart_rtos_handle_t ∗handle, uart_handle_t ∗t_handle, const uart_rtos_config_t ∗cfg)
  Initializes a UART instance for operation in RTOS.
- int UART_RTOS_Deinit (uart_rtos_handle_t ∗handle)
  Deinitializes a UART instance for operation.

UART transactional Operation

- int UART_RTOS_Send (uart_rtos_handle_t ∗handle, const 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.

13.3.2 Data Structure Documentation

13.3.2.1 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.

13.3.3 Macro Definition Documentation

13.3.3.1 #define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

13.3.4 Function Documentation

13.3.4.1 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.

13.3.4.2 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.

13.3.4.3 int UART_RTOS_Send (uart_rtos_handle_t ∗ *handle,* const uint8_t ∗ *buffer,* uint32_t *length*)

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

handle	The RTOS UART handle.
buffer	The pointer to the buffer to send.
length	The number of bytes to send.

13.3.4.4 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.

13.4 UART SDMA Driver

13.4.1 Overview

Data Structures

- struct uart_sdma_handle_t
  UART sDMA handle. More...

Typedefs

- typedef void(∗ uart_sdma_transfer_callback_t)(UART_Type ∗base, uart_sdma_handle_t ∗handle, status_t status, void ∗userData)
  UART transfer callback function.

Driver version

- #define FSL_UART_SDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  UART SDMA driver version 2.0.1.

sDMA transactional

- void UART_TransferCreateHandleSDMA (UART_Type ∗base, uart_sdma_handle_t ∗handle, uart_sdma_transfer_callback_t callback, void ∗userData, sdma_handle_t ∗txSdmaHandle, sdma_handle_t ∗rxSdmaHandle, uint32_t eventSourceTx, uint32_t eventSourceRx)
  Initializes the UART handle which is used in transactional functions.
- status_t UART_SendSDMA (UART_Type ∗base, uart_sdma_handle_t ∗handle, uart_transfer_t ∗xfer)
  Sends data using sDMA.
- status_t UART_ReceiveSDMA (UART_Type ∗base, uart_sdma_handle_t ∗handle, uart_transfer_t ∗xfer)
  Receives data using sDMA.
- void UART_TransferAbortSendSDMA (UART_Type ∗base, uart_sdma_handle_t ∗handle)
  Aborts the sent data using sDMA.
- void UART_TransferAbortReceiveSDMA (UART_Type ∗base, uart_sdma_handle_t ∗handle)
  Aborts the receive data using sDMA.

13.4.2 Data Structure Documentation

13.4.2.1 struct _uart_sdma_handle

Data Fields

- uart_sdma_transfer_callback_t callback

Callback function.
- void ∗ userData
  UART callback function parameter.
- size_t rxDataSizeAll
  Size of the data to receive.
- size_t txDataSizeAll
  Size of the data to send out.
- sdma_handle_t ∗ txSdmaHandle
  The sDMA TX channel used.
- sdma_handle_t ∗ rxSdmaHandle
  The sDMA RX channel used.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.

13.4.2.1.0.14 Field Documentation

13.4.2.1.0.14.1 uart_sdma_transfer_callback_t uart_sdma_handle_t::callback

13.4.2.1.0.14.2 void∗ uart_sdma_handle_t::userData

13.4.2.1.0.14.3 size_t uart_sdma_handle_t::rxDataSizeAll

13.4.2.1.0.14.4 size_t uart_sdma_handle_t::txDataSizeAll

13.4.2.1.0.14.5 sdma_handle_t∗ uart_sdma_handle_t::txSdmaHandle

13.4.2.1.0.14.6 sdma_handle_t∗ uart_sdma_handle_t::rxSdmaHandle

13.4.2.1.0.14.7 volatile uint8_t uart_sdma_handle_t::txState

13.4.3 Macro Definition Documentation

13.4.3.1 #define FSL_UART_SDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

13.4.4 Typedef Documentation

13.4.4.1 typedef void(∗ uart_sdma_transfer_callback_t)(UART_Type ∗base, uart_sdma_handle_t ∗handle, status_t status, void ∗userData)

13.4.5 Function Documentation

13.4.5.1 void UART_TransferCreateHandleSDMA (UART_Type ∗ *base,* uart_sdma_handle_t ∗ *handle,* uart_sdma_transfer_callback_t *callback,* void ∗ *userData,* sdma_handle_t ∗ *txSdmaHandle,* sdma_handle_t ∗ *rxSdmaHandle,* uint32_t *eventSourceTx,* uint32_t *eventSourceRx*)

Parameters

base	UART peripheral base address.
handle	Pointer to the uart_sdma_handle_t structure.
callback	UART callback, NULL means no callback.
userData	User callback function data.
rxSdmaHandle	User-requested DMA handle for RX DMA transfer.
txSdmaHandle	User-requested DMA handle for TX DMA transfer.
eventSourceTx	Eventsource for TX DMA transfer.
eventSourceRx	Eventsource for RX DMA transfer.

13.4.5.2 status_t UART_SendSDMA (UART_Type ∗ *base,* uart_sdma_handle_t ∗ *handle,* uart_transfer_t ∗ *xfer*)

This function sends data using sDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

base	UART peripheral base address.
handle	UART handle pointer.
xfer	UART sDMA transfer structure. See uart_transfer_t.

Return values

kStatus_Success	if succeeded; otherwise failed.
kStatus_UART_TxBusy	Previous transfer ongoing.
kStatus_InvalidArgument	Invalid argument.

13.4.5.3 status_t UART_ReceiveSDMA (UART_Type ∗ *base,* uart_sdma_handle_t ∗ *handle,* uart_transfer_t ∗ *xfer*)

This function receives data using sDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

base	UART peripheral base address.
handle	Pointer to the uart_sdma_handle_t structure.
xfer	UART sDMA transfer structure. See uart_transfer_t.

Return values

kStatus_Success	if succeeded; otherwise failed.
kStatus_UART_RxBusy	Previous transfer ongoing.
kStatus_InvalidArgument	Invalid argument.

13.4.5.4 void UART_TransferAbortSendSDMA (UART_Type ∗ *base*, uart_sdma_handle_t ∗ *handle*)

This function aborts sent data using sDMA.

Parameters

base	UART peripheral base address.
handle	Pointer to the uart_sdma_handle_t structure.

13.4.5.5 void UART_TransferAbortReceiveSDMA (UART_Type ∗ *base*, uart_sdma_handle_t ∗ *handle*)

This function aborts receive data using sDMA.

Parameters

base	UART peripheral base address.
handle	Pointer to the uart_sdma_handle_t structure.

Chapter 14
MU: Messaging Unit

14.1 Overview

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

14.2 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.

14.2.1 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.

14.2.2 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.

14.2.3 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.

14.2.4 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.

14.2.5 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 }
  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 = (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_NmiInterruptTrigger = MU_CR_NMI_MASK,
  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, 2))
  MU driver version 2.0.2.

Function description

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 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.
- static void MU_ClearNmi (MU_Type ∗base)
  Clear non-maskable interrupt (NMI) sent by the other core.

MU misc functions

- void MU_BootCoreB (MU_Type ∗base, mu_core_boot_mode_t mode)
  Boots the core at B side.
- static void MU_HoldCoreBReset (MU_Type ∗base)
  Holds the core reset of B side.
- void MU_BootOtherCore (MU_Type ∗base, mu_core_boot_mode_t mode)
  Boots the other core.
- static void MU_HoldOtherCoreReset (MU_Type ∗base)

Holds the other core reset.
- static void MU_ResetBothSides (MU_Type ∗base)
  Resets the MU for both A side and B side.
- void MU_HardwareResetOtherCore (MU_Type ∗base, bool waitReset, bool holdReset, mu_core_-boot_mode_t bootMode)
  Hardware reset the other core.
- static void MU_SetClockOnOtherCoreEnable (MU_Type ∗base, bool enable)
  Enables or disables the clock on the other core.
- static mu_power_mode_t MU_GetOtherCorePowerMode (MU_Type ∗base)
  Gets the power mode of the other core.

14.3 Macro Definition Documentation

14.3.1 #define FSL_MU_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

14.4 Enumeration Type Documentation

14.4.1 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.

14.4.2 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.

14.4.3 enum _mu_interrupt_trigger

Enumerator

kMU_NmiInterruptTrigger	NMI interrupt.
kMU_GenInt0InterruptTrigger	General purpose interrupt 0.
kMU_GenInt1InterruptTrigger	General purpose interrupt 1.
kMU_GenInt2InterruptTrigger	General purpose interrupt 2.
kMU_GenInt3InterruptTrigger	General purpose interrupt 3.

14.5 Function Documentation

14.5.1 void MU_Init (MU_Type ∗ *base*)

This function enables the MU clock only.

Parameters

base	MU peripheral base address.

14.5.2 void MU_Deinit (MU_Type ∗ *base*)

This function disables the MU clock only.

Parameters

base	MU peripheral base address.

14.5.3 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.

14.5.4 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.

14.5.5 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.

14.5.6 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.

14.5.7 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.

14.5.8 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.

14.5.9 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.

14.5.10 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.

14.5.11 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

14.5.12 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.

14.5.13 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.

14.5.14 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.

14.5.15 static void MU_ClearNmi (MU_Type ∗ *base*) `[inline]`, `[static]`

This function clears non-maskable interrupt (NMI) sent by the other core.

Parameters

base	MU peripheral base address.

14.5.16 void MU_BootCoreB (MU_Type ∗ *base,* mu_core_boot_mode_t *mode*)

This function sets the B side core's boot configuration and releases the core from reset.

Parameters

base	MU peripheral base address.
mode	Core B boot mode.

Note

Only MU side A can use this function.

14.5.17 static void MU_HoldCoreBReset (MU_Type ∗ *base*) `[inline]`, `[static]`

This function causes the core of B side to be held in reset following any reset event.

Parameters

base	MU peripheral base address.

Note

Only A side could call this function.

14.5.18 void MU_BootOtherCore (MU_Type ∗ *base,* mu_core_boot_mode_t *mode*)

This function boots the other core with a boot configuration.

Parameters

base	MU peripheral base address.
mode	The other core boot mode.

14.5.19 static void MU_HoldOtherCoreReset (MU_Type ∗ *base*) [inline], [static]

This function causes the other core to be held in reset following any reset event.

Parameters

base	MU peripheral base address.

14.5.20 static void MU_ResetBothSides (MU_Type ∗ *base*) [inline], [static]

This function resets the MU for both A side and B side. Before reset, it is recommended to interrupt processor B, because this function may affect the ongoing processor B programs.

Parameters

base	MU peripheral base address.

Note

For some platforms, only MU side A could use this function, check reference manual for details.

14.5.21 void MU_HardwareResetOtherCore (MU_Type ∗ *base*, bool *waitReset*, bool *holdReset*, mu_core_boot_mode_t *bootMode*)

This function resets the other core, the other core could mask the hardware reset by calling MU_Mask-HardwareReset. The hardware reset mask feature is only available for some platforms. This function could be used together with MU_BootOtherCore to control the other core reset workflow.

Example 1: Reset the other core, and no hold reset

```
* MU_HardwareResetOtherCore(MU_A, true, false, bootMode);
*
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later.

```
* // Here the other core enters reset, and the reset is hold
* MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);
* // Current core boot the other core when necessary.
* MU_BootOtherCore(MU_A, bootMode);
*
```

Parameters

base	MU peripheral base address.
waitReset	Wait the other core enters reset. • true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked. • false: Don't wait the reset.

holdReset	Hold the other core reset or not. • true: Hold the other core in reset, this function returns directly when the other core enters reset. • false: Don't hold the other core in reset, this function waits until the other core out of reset.
bootMode	Boot mode of the other core, if `holdReset` is true, this parameter is useless.

14.5.22 static void MU_SetClockOnOtherCoreEnable (MU_Type ∗ *base*, bool *enable*) [inline], [static]

This function enables or disables the platform clock on the other core when that core enters a stop mode. If disabled, the platform clock for the other core is disabled when it enters stop mode. If enabled, the platform clock keeps running on the other core in stop mode, until this core also enters stop mode.

Parameters

base	MU peripheral base address.
enable	Enable or disable the clock on the other core.

14.5.23 static mu_power_mode_t MU_GetOtherCorePowerMode (MU_Type ∗ *base*) [inline], [static]

This function gets the power mode of the other core.

Parameters

base	MU peripheral base address.

Returns

Power mode of the other core.

Chapter 15
RDC: Resource Domain Controller

15.1 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...

Macros

- #define FSL_RDC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  Version 2.0.0.

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.
- static 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.
- static void RDC_SetMasterDomainAssignment (RDC_Type ∗base, rdc_master_t master, const rdc-_domain_assignment_t ∗assignment)
  Set master domain assignment.
- static void RDC_GetDefaultMasterDomainAssignment (rdc_domain_assignment_t ∗assignment)
  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.

15.2 Data Structure Documentation

15.2.1 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.

15.2.1.0.0.15 Field Documentation

15.2.1.0.0.15.1 uint32_t rdc_hardware_config_t::domainNumber

15.2.1.0.0.15.2 uint32_t rdc_hardware_config_t::masterNumber

15.2.1.0.0.15.3 uint32_t rdc_hardware_config_t::periphNumber

15.2.1.0.0.15.4 uint32_t rdc_hardware_config_t::memNumber

15.2.2 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.

15.2.2.0.0.16 Field Documentation

15.2.2.0.0.16.1 uint32_t rdc_domain_assignment_t::domainId

15.2.2.0.0.16.2 uint32_t rdc_domain_assignment_t::__pad0__

15.2.2.0.0.16.3 uint32_t rdc_domain_assignment_t::lock

15.2.3 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.

15.2.3.0.0.17 Field Documentation

15.2.3.0.0.17.1 rdc_periph_t rdc_periph_access_config_t::periph

15.2.3.0.0.17.2 bool rdc_periph_access_config_t::lock

15.2.3.0.0.17.3 bool rdc_periph_access_config_t::enableSema

15.2.3.0.0.17.4 uint16_t rdc_periph_access_config_t::policy

15.2.4 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.
- uint32_t baseAddress
  Start address of the memory region.
- uint32_t endAddress
  End address of the memory region.
- uint16_t policy
  Access policy.

15.2.4.0.0.18 Field Documentation

15.2.4.0.0.18.1 rdc_mem_t rdc_mem_access_config_t::mem

15.2.4.0.0.18.2 bool rdc_mem_access_config_t::lock

15.2.4.0.0.18.3 uint32_t rdc_mem_access_config_t::baseAddress

15.2.4.0.0.18.4 uint32_t rdc_mem_access_config_t::endAddress

15.2.4.0.0.18.5 uint16_t rdc_mem_access_config_t::policy

15.2.5 struct rdc_mem_status_t

Data Fields

- bool hasViolation
  Violating happens or not.
- uint8_t domainID
  Violating Domain ID.
- uint32_t address
  Violating Address.

15.2.5.0.0.19 Field Documentation

15.2.5.0.0.19.1 bool rdc_mem_status_t::hasViolation

15.2.5.0.0.19.2 uint8_t rdc_mem_status_t::domainID

15.2.5.0.0.19.3 uint32_t rdc_mem_status_t::address

15.3 Macro Definition Documentation

15.3.1 #define FSL_RDC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

15.4 Enumeration Type Documentation

15.4.1 enum _rdc_interrupts

Enumerator

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

15.4.2 enum _rdc_flags

Enumerator

kRDC_PowerDownDomainOn Power down domain is ON.

15.4.3 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.

15.5 Function Documentation

15.5.1 void RDC_Init (RDC_Type ∗ *base*)

This function enables the RDC clock.

Parameters

base	RDC peripheral base address.

15.5.2 void RDC_Deinit (RDC_Type ∗ *base*)

This function disables the RDC clock.

Parameters

base	RDC peripheral base address.

15.5.3 static void RDC_GetHardwareConfig (RDC_Type ∗ *base*, rdc_hardware_config_t ∗ *config*) `[inline]`, `[static]`

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.

15.5.4 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.

15.5.5 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.

15.5.6 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.

15.5.7 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.

15.5.8 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.

15.5.9 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.

15.5.10 static void RDC_SetMasterDomainAssignment (RDC_Type ∗ *base*, rdc_master_t *master*, const rdc_domain_assignment_t ∗ *assignment*) [inline], [static]

Parameters

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

assignment	Pointer to the assignment.

15.5.11 static void RDC_GetDefaultMasterDomainAssignment (rdc_domain_assignment_t ∗ *assignment*) [inline], [static]

The default configuration is:

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

Parameters

assignment	Pointer to the assignment.

15.5.12 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.

15.5.13 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.

15.5.14 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.

15.5.15 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.

15.5.16 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.

15.5.17 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.

15.5.18 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_SetMemAccess-Valid 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.

15.5.19 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.

15.5.20 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.

15.5.21 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.

15.5.22 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 16
RDC_SEMA42: Hardware Semaphores Driver

16.1 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) (∗(&((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, 0))
  RDC_SEMA42 driver version.

16.2 Macro Definition Documentation

16.2.1 #define RDC_SEMA42_GATE_NUM_RESET_ALL (64U)

16.2.2 #define RDC_SEMA42_GATEn(*base,* *n*) (∗(&((base)->GATE0) + (n)))

16.2.3 #define RDC_SEMA42_GATE_COUNT (64U)

16.3 Function Documentation

16.3.1 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.

16.3.2 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.

16.3.3 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.

16.3.4 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.

16.3.5 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_SEM-A42 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.

16.3.6 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.

16.3.7 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.

16.3.8 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.

16.3.9 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 17
SAI: Serial Audio Interface

17.1 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.

17.2 Typical use case

17.2.1 SAI Send/receive using an interrupt method

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

17.2.2 SAI Send/receive using a DMA method

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

Modules

- SAI DMA Driver
- SAI Driver
- SAI EDMA Driver
- SAI SDMA Driver

17.3 SAI Driver

17.3.1 Overview

Data Structures

- struct sai_config_t
  SAI user configuration structure. More...
- struct sai_transfer_format_t
  SAI transfer format. More...
- struct sai_transfer_t
  SAI transfer structure. More...
- struct sai_handle_t
  SAI handle structure. More...

Macros

- #define SAI_XFER_QUEUE_SIZE (4)
  SAI transfer queue size, user can refine it according to use case.

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 _sai_status_t {
  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 return status.
- enum _sai_channel_mask {

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 }

- 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 }
  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 }
  SAI clock polarity, active high or low.
- enum sai_sync_mode_t {
  kSAI_ModeAsync = 0x0U,
  kSAI_ModeSync,
  kSAI_ModeSyncWithOtherTx,
  kSAI_ModeSyncWithOtherRx }
  Synchronous or asynchronous mode.
- enum sai_mclk_source_t {
  kSAI_MclkSourceSysclk = 0x0U,
  kSAI_MclkSourceSelect1,
  kSAI_MclkSourceSelect2,
  kSAI_MclkSourceSelect3 }
  Mater clock source.
- enum sai_bclk_source_t {

kSAI_BclkSourceBusclk = 0x0U,
kSAI_BclkSourceMclkOption1 = 0x1U,
kSAI_BclkSourceMclkOption2 = 0x2U,
kSAI_BclkSourceMclkOption3 = 0x3U,
kSAI_BclkSourceMclkDiv = 0x1U,
kSAI_BclkSourceOtherSai0 = 0x2U,
kSAI_BclkSourceOtherSai1 = 0x3U }
Bit clock source.

- enum _sai_interrupt_enable_t {
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 }
The SAI interrupt enable flag.
- enum _sai_dma_enable_t {
kSAI_FIFOWarningDMAEnable = I2S_TCSR_FWDE_MASK,
kSAI_FIFORequestDMAEnable = I2S_TCSR_FRDE_MASK }
The DMA request sources.
- enum _sai_flags {
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 }
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.

Driver version

- #define FSL_SAI_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))
  Version 2.1.7.

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_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.

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.
- 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.

17.3.2 Data Structure Documentation

17.3.2.1 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.

17.3.2.2 struct sai_transfer_format_t

Data Fields

- uint32_t sampleRate_Hz
  Sample rate of audio data.
- uint32_t bitWidth

Data length of audio data, usually 8/16/24/32 bits.
- sai_mono_stereo_t stereo
  Mono or stereo.
- uint32_t masterClockHz
  Master clock frequency in Hz.
- uint8_t watermark
  Watermark value.
- uint8_t channel
  Transfer start channel.
- uint8_t channelMask
  enabled channel mask value, reference _sai_channel_mask
- uint8_t endChannel
  end channel number
- uint8_t channelNums
  Total enabled channel numbers.
- sai_protocol_t protocol
  Which audio protocol used.
- bool isFrameSyncCompact
  True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

17.3.2.2.0.20 Field Documentation

17.3.2.2.0.20.1 bool sai_transfer_format_t::isFrameSyncCompact

17.3.2.3 struct sai_transfer_t

Data Fields

- uint8_t ∗ data
  Data start address to transfer.
- size_t dataSize
  Transfer size.

17.3.2.3.0.21 Field Documentation

17.3.2.3.0.21.1 uint8_t∗ sai_transfer_t::data

17.3.2.3.0.21.2 size_t sai_transfer_t::dataSize

17.3.2.4 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.

17.3.3 Macro Definition Documentation

17.3.3.1 #define SAI_XFER_QUEUE_SIZE (4)

17.3.4 Enumeration Type Documentation

17.3.4.1 enum _sai_status_t

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.

17.3.4.2 enum _sai_channel_mask

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

17.3.4.3 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.

17.3.4.4 enum sai_master_slave_t

Enumerator

kSAI_Master	Master mode.
kSAI_Slave	Slave mode.

17.3.4.5 enum sai_mono_stereo_t

Enumerator

kSAI_Stereo	Stereo sound.
kSAI_MonoRight	Only Right channel have sound.
kSAI_MonoLeft	Only left channel have sound.

17.3.4.6 enum sai_data_order_t

Enumerator

kSAI_DataLSB	LSB bit transferred first.
kSAI_DataMSB	MSB bit transferred first.

17.3.4.7 enum sai_clock_polarity_t

Enumerator

kSAI_PolarityActiveHigh Clock active high.
kSAI_PolarityActiveLow Clock active low.

17.3.4.8 enum sai_sync_mode_t

Enumerator

kSAI_ModeAsync Asynchronous mode.
kSAI_ModeSync Synchronous mode (with receiver or transmit)
kSAI_ModeSyncWithOtherTx Synchronous with another SAI transmit.
kSAI_ModeSyncWithOtherRx Synchronous with another SAI receiver.

17.3.4.9 enum sai_mclk_source_t

Enumerator

kSAI_MclkSourceSysclk Master clock from the system clock.
kSAI_MclkSourceSelect1 Master clock from source 1.
kSAI_MclkSourceSelect2 Master clock from source 2.
kSAI_MclkSourceSelect3 Master clock from source 3.

17.3.4.10 enum sai_bclk_source_t

Enumerator

kSAI_BclkSourceBusclk Bit clock using bus clock.
kSAI_BclkSourceMclkOption1 Bit clock MCLK option 1.
kSAI_BclkSourceMclkOption2 Bit clock MCLK option2.
kSAI_BclkSourceMclkOption3 Bit clock MCLK option3.
kSAI_BclkSourceMclkDiv Bit clock using master clock divider.
kSAI_BclkSourceOtherSai0 Bit clock from other SAI device.
kSAI_BclkSourceOtherSai1 Bit clock from other SAI device.

17.3.4.11 enum _sai_interrupt_enable_t

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.

17.3.4.12 enum _sai_dma_enable_t

Enumerator

kSAI_FIFOWarningDMAEnable FIFO warning caused by the DMA request.
kSAI_FIFORequestDMAEnable FIFO request caused by the DMA request.

17.3.4.13 enum _sai_flags

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.

17.3.4.14 enum sai_reset_type_t

Enumerator

kSAI_ResetTypeSoftware Software reset, reset the logic state.
kSAI_ResetTypeFIFO FIFO reset, reset the FIFO read and write pointer.
kSAI_ResetAll All reset.

17.3.4.15 enum sai_fifo_packing_t

Enumerator

kSAI_FifoPackingDisabled Packing disabled.
kSAI_FifoPacking8bit 8 bit packing enabled
kSAI_FifoPacking16bit 16bit packing enabled

17.3.4.16 enum sai_sample_rate_t

Enumerator

kSAI_SampleRate8KHz	Sample rate 8000 Hz.
kSAI_SampleRate11025Hz	Sample rate 11025 Hz.
kSAI_SampleRate12KHz	Sample rate 12000 Hz.
kSAI_SampleRate16KHz	Sample rate 16000 Hz.
kSAI_SampleRate22050Hz	Sample rate 22050 Hz.
kSAI_SampleRate24KHz	Sample rate 24000 Hz.
kSAI_SampleRate32KHz	Sample rate 32000 Hz.
kSAI_SampleRate44100Hz	Sample rate 44100 Hz.
kSAI_SampleRate48KHz	Sample rate 48000 Hz.
kSAI_SampleRate96KHz	Sample rate 96000 Hz.
kSAI_SampleRate192KHz	Sample rate 192000 Hz.
kSAI_SampleRate384KHz	Sample rate 384000 Hz.

17.3.4.17 enum sai_word_width_t

Enumerator

kSAI_WordWidth8bits	Audio data width 8 bits.
kSAI_WordWidth16bits	Audio data width 16 bits.
kSAI_WordWidth24bits	Audio data width 24 bits.
kSAI_WordWidth32bits	Audio data width 32 bits.

17.3.5 Function Documentation

17.3.5.1 void SAI_TxInit (I2S_Type ∗ *base*, const sai_config_t ∗ *config*)

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.

17.3.5.2 void SAI_RxInit (I2S_Type ∗ *base,* const sai_config_t ∗ *config*)

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.

17.3.5.3 void SAI_TxGetDefaultConfig (sai_config_t ∗ *config*)

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

17.3.5.4 void SAI_RxGetDefaultConfig (sai_config_t ∗ *config*)

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

17.3.5.5 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

17.3.5.6 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

17.3.5.7 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

17.3.5.8 void SAI_TxEnable (I2S_Type ∗ *base,* bool *enable*)

Parameters

base	SAI base pointer
enable	True means enable SAI Tx, false means disable.

17.3.5.9 void SAI_RxEnable (I2S_Type ∗ *base,* bool *enable*)

Parameters

base	SAI base pointer
enable	True means enable SAI Rx, false means disable.

17.3.5.10 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.

17.3.5.11 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

17.3.5.12 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.

17.3.5.13 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

17.3.5.14 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

17.3.5.15 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

17.3.5.16 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.

17.3.5.17 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.

17.3.5.18 void SAI_TxSetDataOrder (I2S_Type ∗ *base*, sai_data_order_t *order*)

Parameters

base	SAI base pointer
order	Data order MSB or LSB

17.3.5.19 void SAI_RxSetDataOrder (I2S_Type ∗ *base*, sai_data_order_t *order*)

Parameters

base	SAI base pointer
order	Data order MSB or LSB

17.3.5.20 void SAI_TxSetBitClockPolarity (I2S_Type ∗ *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
order	Data order MSB or LSB

17.3.5.21 void SAI_RxSetBitClockPolarity (I2S_Type ∗ *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
order	Data order MSB or LSB

17.3.5.22 void SAI_TxSetFrameSyncPolarity (I2S_Type ∗ *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
order	Data order MSB or LSB

17.3.5.23 void SAI_RxSetFrameSyncPolarity (I2S_Type ∗ *base*, sai_clock_polarity_t *polarity*)

Parameters

base	SAI base pointer
order	Data order MSB or LSB

17.3.5.24 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.

17.3.5.25 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.

17.3.5.26 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.

17.3.5.27 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.

17.3.5.28 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

17.3.5.29 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

17.3.5.30 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

17.3.5.31 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

17.3.5.32 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.

17.3.5.33 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.

17.3.5.34 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.

17.3.5.35 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.

17.3.5.36 void SAI_TxSetFormat (I2S_Type ∗ *base,* sai_transfer_format_t ∗ *format,* uint32_t *mclkSourceClockHz,* uint32_t *bclkSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

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.

17.3.5.37 void SAI_RxSetFormat (I2S_Type ∗ *base,* sai_transfer_format_t ∗ *format,* uint32_t *mclkSourceClockHz,* uint32_t *bclkSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

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.

17.3.5.38 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.

17.3.5.39 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.

17.3.5.40 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.

17.3.5.41 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.

17.3.5.42 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.

17.3.5.43 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.

17.3.5.44 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

17.3.5.45 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.

17.3.5.46 status_t SAI_TransferTxSetFormat (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*, sai_transfer_format_t ∗ *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

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.

17.3.5.47 status_t SAI_TransferRxSetFormat (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*, sai_transfer_format_t ∗ *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

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.

17.3.5.48 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.

17.3.5.49 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.

17.3.5.50 status_t SAI_TransferGetSendCount (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure which stores the transfer state.
count	Bytes count sent.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferIn-Progress	There is not a non-blocking transaction currently in progress.

17.3.5.51 status_t SAI_TransferGetReceiveCount (I2S_Type ∗ *base*, sai_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure which stores the transfer state.
count	Bytes count received.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferIn-Progress	There is not a non-blocking transaction currently in progress.

17.3.5.52 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.

17.3.5.53 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.

17.3.5.54 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.

17.3.5.55 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.

17.3.5.56 void SAI_TransferTxHandleIRQ (I2S_Type ∗ *base,* sai_handle_t ∗ *handle*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure.

17.3.5.57 void SAI_TransferRxHandleIRQ (I2S_Type ∗ *base,* sai_handle_t ∗ *handle*)

Parameters

base	SAI base pointer.
handle	Pointer to the sai_handle_t structure.

17.4 SAI DMA Driver

17.5 SAI EDMA Driver

17.6 SAI SDMA Driver

17.6.1 Overview

Data Structures

- struct sai_sdma_handle_t
  SAI DMA transfer handle, users should not touch the content of the handle. More...

Typedefs

- typedef void(∗ sai_sdma_callback_t)(I2S_Type ∗base, sai_sdma_handle_t ∗handle, status_t status, void ∗userData)
  SAI SDMA transfer callback function for finish and error.

Driver version

- #define FSL_SAI_SDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))
  Version 2.1.5.

SDMA Transactional

- void SAI_TransferTxCreateHandleSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_sdma_callback_t callback, void ∗userData, sdma_handle_t ∗dmaHandle, uint32_t eventSource)
  Initializes the SAI SDMA handle.
- void SAI_TransferRxCreateHandleSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_sdma_callback_t callback, void ∗userData, sdma_handle_t ∗dmaHandle, uint32_t eventSource)
  Initializes the SAI Rx SDMA handle.
- void SAI_TransferTxSetFormatSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_transfer_format_t ∗format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  Configures the SAI Tx audio format.
- void SAI_TransferRxSetFormatSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_transfer_format_t ∗format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
  Configures the SAI Rx audio format.
- status_t SAI_TransferSendSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_transfer_t ∗xfer)
  Performs a non-blocking SAI transfer using DMA.
- status_t SAI_TransferReceiveSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle, sai_transfer_t ∗xfer)
  Performs a non-blocking SAI receive using SDMA.
- void SAI_TransferAbortSendSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle)
  Aborts a SAI transfer using SDMA.
- void SAI_TransferAbortReceiveSDMA (I2S_Type ∗base, sai_sdma_handle_t ∗handle)
  Aborts a SAI receive using SDMA.

17.6.2 Data Structure Documentation

17.6.2.1 struct _sai_sdma_handle

Data Fields

- sdma_handle_t ∗ dmaHandle
  DMA handler for SAI send.
- uint8_t bytesPerFrame
  Bytes in a frame.
- uint8_t channel
  start data channel
- uint8_t channelNums
  total transfer channel numbers, used for multififo
- uint8_t channelMask
  enabled channel mask value, refernece _sai_channel_mask
- uint8_t fifoOffset
  fifo address offset between multifo
- uint8_t count
  The transfer data count in a DMA request.
- uint32_t state
  Internal state for SAI SDMA transfer.
- uint32_t eventSource
  SAI event source number.
- sai_sdma_callback_t callback
  Callback for users while transfer finish or error occurs.
- void ∗ userData
  User callback parameter.
- sdma_buffer_descriptor_t bdPool [SAI_XFER_QUEUE_SIZE]
  BD pool for SDMA transfer.
- sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]
  Transfer queue storing queued transfer.
- size_t transferSize [SAI_XFER_QUEUE_SIZE]
  Data bytes need to transfer.
- volatile uint8_t queueUser
  Index for user to queue transfer.
- volatile uint8_t queueDriver
  Index for driver to get the transfer data and size.

17.6.2.1.0.22 Field Documentation

17.6.2.1.0.22.1 sdma_buffer_descriptor_t sai_sdma_handle_t::bdPool[SAI_XFER_QUEUE_SIZE]

17.6.2.1.0.22.2 sai_transfer_t sai_sdma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]

17.6.2.1.0.22.3 volatile uint8_t sai_sdma_handle_t::queueUser

17.6.3 Function Documentation

17.6.3.1 void SAI_TransferTxCreateHandleSDMA (I2S_Type ∗ *base*, sai_sdma_handle_t ∗ *handle*, sai_sdma_callback_t *callback*, void ∗ *userData*, sdma_handle_t ∗ *dmaHandle*, uint32_t *eventSource*)

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

Parameters

base	SAI base pointer.
handle	SAI SDMA handle pointer.
base	SAI peripheral base address.
callback	Pointer to user callback function.
userData	User parameter passed to the callback function.
dmaHandle	SDMA handle pointer, this handle shall be static allocated by users.

17.6.3.2 void SAI_TransferRxCreateHandleSDMA (I2S_Type ∗ *base,* sai_sdma_handle_t ∗ *handle,* sai_sdma_callback_t *callback,* void ∗ *userData,* sdma_handle_t ∗ *dmaHandle,* uint32_t *eventSource*)

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

Parameters

base	SAI base pointer.
handle	SAI SDMA handle pointer.
base	SAI peripheral base address.
callback	Pointer to user callback function.
userData	User parameter passed to the callback function.
dmaHandle	SDMA handle pointer, this handle shall be static allocated by users.

17.6.3.3 void SAI_TransferTxSetFormatSDMA (I2S_Type ∗ *base,* sai_sdma_handle_t ∗ *handle,* sai_transfer_format_t ∗ *format,* uint32_t *mclkSourceClockHz,* uint32_t *bclkSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the SDMA parameter according to formatting requirements.

Parameters

base	SAI base pointer.

handle	SAI SDMA handle pointer.
format	Pointer to SAI audio data format structure.
mclkSource-ClockHz	SAI master clock source frequency in Hz.
bclkSource-ClockHz	SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format.

Return values

kStatus_Success	Audio format set successfully.
kStatus_InvalidArgument	The input argument is invalid.

17.6.3.4 void SAI_TransferRxSetFormatSDMA (I2S_Type ∗ *base,* sai_sdma_handle_t ∗ *handle,* sai_transfer_format_t ∗ *format,* uint32_t *mclkSourceClockHz,* uint32_t *bclkSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the SDMA parameter according to formatting requirements.

Parameters

base	SAI base pointer.
handle	SAI SDMA handle pointer.
format	Pointer to SAI audio data format structure.
mclkSource-ClockHz	SAI master clock source frequency in Hz.
bclkSource-ClockHz	SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format.

Return values

kStatus_Success	Audio format set successfully.
kStatus_InvalidArgument	The input argument is invalid.

17.6.3.5 status_t SAI_TransferSendSDMA (I2S_Type ∗ *base,* sai_sdma_handle_t ∗ *handle,* sai_transfer_t ∗ *xfer*)

Note

This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

base	SAI base pointer.
handle	SAI SDMA handle pointer.
xfer	Pointer to the DMA transfer structure.

Return values

kStatus_Success	Start a SAI SDMA send successfully.
kStatus_InvalidArgument	The input argument is invalid.
kStatus_TxBusy	SAI is busy sending data.

17.6.3.6 status_t SAI_TransferReceiveSDMA (I2S_Type ∗ *base,* sai_sdma_handle_t ∗ *handle,* sai_transfer_t ∗ *xfer*)

Note

This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemaining-Bytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

base	SAI base pointer
handle	SAI SDMA handle pointer.
xfer	Pointer to DMA transfer structure.

Return values

kStatus_Success	Start a SAI SDMA receive successfully.
kStatus_InvalidArgument	The input argument is invalid.
kStatus_RxBusy	SAI is busy receiving data.

17.6.3.7 void SAI_TransferAbortSendSDMA (I2S_Type ∗ *base,* sai_sdma_handle_t ∗ *handle*)

Parameters

base	SAI base pointer.
handle	SAI SDMA handle pointer.

17.6.3.8 void SAI_TransferAbortReceiveSDMA (I2S_Type ∗ *base*, sai_sdma_handle_t ∗ *handle*)

Parameters

base	SAI base pointer
handle	SAI SDMA handle pointer.

Chapter 18
SDMA: Smart Direct Memory Access (SDMA) Controller Driver

18.1 Overview

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

18.2 Typical use case

18.2.1 SDMA Operation

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

Data Structures

- struct sdma_config_t
  SDMA global configuration structure. More...
- struct sdma_multi_fifo_config_t
  SDMA multi fifo configurations. More...
- struct sdma_sw_done_config_t
  SDMA sw done configurations. More...
- struct sdma_transfer_config_t
  SDMA transfer configuration. More...
- struct sdma_buffer_descriptor_t
  SDMA buffer descriptor structure. More...
- struct sdma_channel_control_t
  SDMA channel control descriptor structure. More...
- struct sdma_context_data_t
  SDMA context structure for each channel. More...
- struct sdma_handle_t
  SDMA transfer handle structure. More...

Typedefs

- typedef void(∗ sdma_callback)(struct _sdma_handle ∗handle, void ∗userData, bool transferDone, uint32_t bdIndex)
  Define callback function for SDMA.

Enumerations

- enum sdma_transfer_size_t {
  kSDMA_TransferSize1Bytes = 0x1U,
  kSDMA_TransferSize2Bytes = 0x2U,
  kSDMA_TransferSize4Bytes = 0x0U }

SDMA transfer configuration.

- enum sdma_bd_status_t {
kSDMA_BDStatusDone = 0x1U,
kSDMA_BDStatusWrap = 0x2U,
kSDMA_BDStatusContinuous = 0x4U,
kSDMA_BDStatusInterrupt = 0x8U,
kSDMA_BDStatusError = 0x10U,
kSDMA_BDStatusLast,
kSDMA_BDStatusExtend = 0x80 }
SDMA buffer descriptor status.
- enum sdma_bd_command_t {
kSDMA_BDCommandSETDM = 0x1U,
kSDMA_BDCommandGETDM = 0x2U,
kSDMA_BDCommandSETPM = 0x4U,
kSDMA_BDCommandGETPM = 0x6U,
kSDMA_BDCommandSETCTX = 0x7U,
kSDMA_BDCommandGETCTX = 0x3U }
SDMA buffer descriptor command.
- enum sdma_context_switch_mode_t {
kSDMA_ContextSwitchModeStatic = 0x0U,
kSDMA_ContextSwitchModeDynamicLowPower,
kSDMA_ContextSwitchModeDynamicWithNoLoop,
kSDMA_ContextSwitchModeDynamic }
SDMA context switch mode.
- enum sdma_clock_ratio_t {
kSDMA_HalfARMClockFreq = 0x0U,
kSDMA_ARMClockFreq }
SDMA core clock frequency ratio to the ARM DMA interface.
- enum sdma_transfer_type_t {
kSDMA_MemoryToMemory = 0x0U,
kSDMA_PeripheralToMemory,
kSDMA_MemoryToPeripheral }
SDMA transfer type.
- enum sdma_peripheral_t {
kSDMA_PeripheralTypeMemory = 0x0,
kSDMA_PeripheralTypeUART,
kSDMA_PeripheralTypeUART_SP,
kSDMA_PeripheralTypeSPDIF,
kSDMA_PeripheralNormal,
kSDMA_PeripheralNormal_SP,
kSDMA_PeripheralMultiFifoPDM,
kSDMA_PeripheralMultiFifoSaiRX,
kSDMA_PeripheralMultiFifoSaiTX }
Peripheral type use SDMA.
- enum _sdma_transfer_status {
kStatus_SDMA_ERROR = MAKE_STATUS(kStatusGroup_SDMA, 0),

kStatus_SDMA_Busy = MAKE_STATUS(kStatusGroup_SDMA, 1) }
SDMA transfer status.
- enum _sdma_multi_fifo_mask {
kSDMA_MultiFifoWatermarkLevelMask = 0xFFU,
kSDMA_MultiFifoNumsMask = 0xFU,
kSDMA_MultiFifoOffsetMask = 0xFU,
kSDMA_MultiFifoSwDoneMask = 0x1U,
kSDMA_MultiFifoSwDoneSelectorMask = 0xFU }
SDMA multi fifo mask.
- enum _sdma_multi_fifo_shift {
kSDMA_MultiFifoWatermarkLevelShift = 0U,
kSDMA_MultiFifoNumsShift = 8U,
kSDMA_MultiFifoOffsetShift = 16U,
kSDMA_MultiFifoSwDoneShift = 23U,
kSDMA_MultiFifoSwDoneSelectorShift = 24U }
SDMA multi fifo shift.

Driver version

- #define FSL_SDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
SDMA driver version.

SDMA initialization and de-initialization

- void SDMA_Init (SDMAARM_Type ∗base, const sdma_config_t ∗config)
Initializes the SDMA peripheral.
- void SDMA_Deinit (SDMAARM_Type ∗base)
Deinitializes the SDMA peripheral.
- void SDMA_GetDefaultConfig (sdma_config_t ∗config)
Gets the SDMA default configuration structure.
- void SDMA_ResetModule (SDMAARM_Type ∗base)
Sets all SDMA core register to reset status.

SDMA Channel Operation

- static void SDMA_EnableChannelErrorInterrupts (SDMAARM_Type ∗base, uint32_t channel)
Enables the interrupt source for the SDMA error.
- static void SDMA_DisableChannelErrorInterrupts (SDMAARM_Type ∗base, uint32_t channel)
Disables the interrupt source for the SDMA error.

SDMA Buffer Descriptor Operation

- void SDMA_ConfigBufferDescriptor (sdma_buffer_descriptor_t ∗bd, uint32_t srcAddr, uint32_-t destAddr, sdma_transfer_size_t busWidth, size_t bufferSize, bool isLast, bool enableInterrupt, bool isWrap, sdma_transfer_type_t type)
Sets buffer descriptor contents.

SDMA Channel Transfer Operation

- static void SDMA_SetChannelPriority (SDMAARM_Type ∗base, uint32_t channel, uint8_t priority)
  Set SDMA channel priority.
- static void SDMA_SetSourceChannel (SDMAARM_Type ∗base, uint32_t source, uint32_t channelMask)
  Set SDMA request source mapping channel.
- static void SDMA_StartChannelSoftware (SDMAARM_Type ∗base, uint32_t channel)
  Start a SDMA channel by software trigger.
- static void SDMA_StartChannelEvents (SDMAARM_Type ∗base, uint32_t channel)
  Start a SDMA channel by hardware events.
- static void SDMA_StopChannel (SDMAARM_Type ∗base, uint32_t channel)
  Stop a SDMA channel.
- void SDMA_SetContextSwitchMode (SDMAARM_Type ∗base, sdma_context_switch_mode_t mode)
  Set the SDMA context switch mode.

SDMA Channel Status Operation

- static uint32_t SDMA_GetChannelInterruptStatus (SDMAARM_Type ∗base)
  Gets the SDMA interrupt status of all channels.
- static void SDMA_ClearChannelInterruptStatus (SDMAARM_Type ∗base, uint32_t mask)
  Clear the SDMA channel interrupt status of specific channels.
- static uint32_t SDMA_GetChannelStopStatus (SDMAARM_Type ∗base)
  Gets the SDMA stop status of all channels.
- static void SDMA_ClearChannelStopStatus (SDMAARM_Type ∗base, uint32_t mask)
  Clear the SDMA channel stop status of specific channels.
- static uint32_t SDMA_GetChannelPendStatus (SDMAARM_Type ∗base)
  Gets the SDMA channel pending status of all channels.
- static void SDMA_ClearChannelPendStatus (SDMAARM_Type ∗base, uint32_t mask)
  Clear the SDMA channel pending status of specific channels.
- static uint32_t SDMA_GetErrorStatus (SDMAARM_Type ∗base)
  Gets the SDMA channel error status.
- bool SDMA_GetRequestSourceStatus (SDMAARM_Type ∗base, uint32_t source)
  Gets the SDMA request source pending status.

SDMA Transactional Operation

- void SDMA_CreateHandle (sdma_handle_t ∗handle, SDMAARM_Type ∗base, uint32_t channel, sdma_context_data_t ∗context)
  Creates the SDMA handle.
- void SDMA_InstallBDMemory (sdma_handle_t ∗handle, sdma_buffer_descriptor_t ∗BDPool, uint32_t BDCount)
  Installs the BDs memory pool into the SDMA handle.
- void SDMA_SetCallback (sdma_handle_t ∗handle, sdma_callback callback, void ∗userData)
  Installs a callback function for the SDMA transfer.
- void SDMA_SetMultiFifoConfig (sdma_transfer_config_t ∗config, uint32_t fifoNums, uint32_t fifoOffset)
  multi fifo configurations.

- void SDMA_EnableSwDone (SDMAARM_Type ∗base, sdma_transfer_config_t ∗config, uint8_t sel, sdma_peripheral_t type)
  enable sdma sw done feature.
- void SDMA_LoadScript (SDMAARM_Type ∗base, uint32_t destAddr, void ∗srcAddr, size_-t bufferSizeBytes)
  load script to sdma program memory.
- void SDMA_DumpScript (SDMAARM_Type ∗base, uint32_t srcAddr, void ∗destAddr, size_-t bufferSizeBytes)
  dump script from sdma program memory.
- void SDMA_PrepareTransfer (sdma_transfer_config_t ∗config, uint32_t srcAddr, uint32_t destAddr, uint32_t srcWidth, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferSize, uint32_t eventSource, sdma_peripheral_t peripheral, sdma_transfer_type_t type)
  Prepares the SDMA transfer structure.
- void SDMA_SubmitTransfer (sdma_handle_t ∗handle, const sdma_transfer_config_t ∗config)
  Submits the SDMA transfer request.
- void SDMA_StartTransfer (sdma_handle_t ∗handle)
  SDMA starts transfer.
- void SDMA_StopTransfer (sdma_handle_t ∗handle)
  SDMA stops transfer.
- void SDMA_AbortTransfer (sdma_handle_t ∗handle)
  SDMA aborts transfer.
- uint32_t SDMA_GetTransferredBytes (sdma_handle_t ∗handle)
  Get transferred bytes while not using BD pools.
- bool SDMA_IsPeripheralInSPBA (uint32_t addr)
  Judge if address located in SPBA.
- void SDMA_HandleIRQ (sdma_handle_t ∗handle)
  SDMA IRQ handler for complete a buffer descriptor transfer.

18.3 Data Structure Documentation

18.3.1 struct sdma_config_t

Data Fields

- bool enableRealTimeDebugPin
  If enable real-time debug pin, default is closed to reduce power consumption.
- bool isSoftwareResetClearLock
  If software reset clears the LOCK bit which prevent writing SDMA scripts into SDMA.
- sdma_clock_ratio_t ratio
  SDMA core clock ratio to ARM platform DMA interface.

18.3.1.0.0.23 Field Documentation

18.3.1.0.0.23.1 bool sdma_config_t::enableRealTimeDebugPin

18.3.1.0.0.23.2 bool sdma_config_t::isSoftwareResetClearLock

18.3.2 struct sdma_multi_fifo_config_t

Data Fields

- uint8_t fifoNums
  fifo numbers
- uint8_t fifoOffset
  offset between multi fifo data register address

18.3.3 struct sdma_sw_done_config_t

Data Fields

- bool enableSwDone
  fifo numbers
- uint8_t swDoneSel
  offset between multi fifo data register address

18.3.4 struct sdma_transfer_config_t

This structure configures the source/destination transfer attribute.

Data Fields

- uint32_t srcAddr
  Source address of the transfer.
- uint32_t destAddr
  Destination address of the transfer.
- sdma_transfer_size_t srcTransferSize
  Source data transfer size.
- sdma_transfer_size_t destTransferSize
  Destination data transfer size.
- uint32_t bytesPerRequest
  Bytes to transfer in a minor loop.
- uint32_t transferSzie
  Bytes to transfer for this descriptor.
- uint32_t scriptAddr
  SDMA script address located in SDMA ROM.
- uint32_t eventSource

Event source number for the channel.
- bool isEventIgnore
  True means software trigger, false means hardware trigger.
- bool isSoftTriggerIgnore
  If ignore the HE bit, 1 means use hardware events trigger, 0 means software trigger.
- sdma_transfer_type_t type
  Transfer type, transfer type used to decide the SDMA script.
- sdma_multi_fifo_config_t multiFifo
  multi fifo configurations
- sdma_sw_done_config_t swDone
  sw done selector

18.3.4.0.0.24 Field Documentation

18.3.4.0.0.24.1 sdma_transfer_size_t sdma_transfer_config_t::srcTransferSize

18.3.4.0.0.24.2 sdma_transfer_size_t sdma_transfer_config_t::destTransferSize

18.3.4.0.0.24.3 uint32_t sdma_transfer_config_t::scriptAddr

18.3.4.0.0.24.4 uint32_t sdma_transfer_config_t::eventSource

0 means no event, use software trigger

18.3.4.0.0.24.5 sdma_transfer_type_t sdma_transfer_config_t::type

18.3.5 struct sdma_buffer_descriptor_t

This structure is a buffer descriptor, this structure describes the buffer start address and other options

Data Fields

- uint32_t count: 16
  Bytes of the buffer length for this buffer descriptor.
- uint32_t status: 8
  E,R,I,C,W,D status bits stored here.
- uint32_t command: 8
  command mostlky used for channel 0
- uint32_t bufferAddr
  Buffer start address for this descriptor.
- uint32_t extendBufferAddr
  External buffer start address, this is an optional for a transfer.

18.3.5.0.0.25 Field Documentation

18.3.5.0.0.25.1 uint32_t sdma_buffer_descriptor_t::count

18.3.5.0.0.25.2 uint32_t sdma_buffer_descriptor_t::bufferAddr

18.3.5.0.0.25.3 uint32_t sdma_buffer_descriptor_t::extendBufferAddr

18.3.6 struct sdma_channel_control_t

Data Fields

- uint32_t currentBDAddr
  Address of current buffer descriptor processed.
- uint32_t baseBDAddr
  The start address of the buffer descriptor array.
- uint32_t channelDesc
  Optional for transfer.
- uint32_t status
  Channel status.

18.3.7 struct sdma_context_data_t

This structure can be load into SDMA core, with this structure, SDMA scripts can start work.

Data Fields

- uint32_t GeneralReg [8]
  8 general regsiters used for SDMA RISC core

18.3.8 struct sdma_handle_t

Data Fields

- sdma_callback callback
  Callback function for major count exhausted.
- void ∗ userData
  Callback function parameter.
- SDMAARM_Type ∗ base
  SDMA peripheral base address.
- sdma_buffer_descriptor_t ∗ BDPool
  Pointer to memory stored BD arrays.
- uint32_t bdCount
  How many buffer descriptor.
- uint32_t bdIndex

How many buffer descriptor.
- uint32_t eventSource
  Event source count for the channel.
- sdma_context_data_t ∗ context
  Channel context to exectute in SDMA.
- uint8_t channel
  SDMA channel number.
- uint8_t priority
  SDMA channel priority.
- uint8_t flags
  The status of the current channel.

18.3.8.0.0.26 Field Documentation

18.3.8.0.0.26.1 sdma_callback sdma_handle_t::callback

18.3.8.0.0.26.2 void∗ sdma_handle_t::userData

18.3.8.0.0.26.3 SDMAARM_Type∗ sdma_handle_t::base

18.3.8.0.0.26.4 sdma_buffer_descriptor_t∗ sdma_handle_t::BDPool

18.3.8.0.0.26.5 uint8_t sdma_handle_t::channel

18.3.8.0.0.26.6 uint8_t sdma_handle_t::flags

18.4 Macro Definition Documentation

18.4.1 #define FSL_SDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

Version 2.1.0.

18.5 Typedef Documentation

18.5.1 typedef void(∗ sdma_callback)(struct _sdma_handle ∗handle, void ∗userData, bool transferDone, uint32_t bdIndex)

18.6 Enumeration Type Documentation

18.6.1 enum sdma_transfer_size_t

Enumerator

kSDMA_TransferSize1Bytes	Source/Destination data transfer size is 1 byte every time.
kSDMA_TransferSize2Bytes	Source/Destination data transfer size is 2 bytes every time.
kSDMA_TransferSize4Bytes	Source/Destination data transfer size is 4 bytes every time.

18.6.2 enum sdma_bd_status_t

Enumerator

kSDMA_BDStatusDone BD ownership, 0 means ARM core owns the BD, while 1 means SDMA owns BD.
kSDMA_BDStatusWrap While this BD is last one, the next BD will be the first one.
kSDMA_BDStatusContinuous Buffer is allowed to transfer/receive to/from multiple buffers.
kSDMA_BDStatusInterrupt While this BD finished, send an interrupt.
kSDMA_BDStatusError Error occurred on buffer descriptor command.
kSDMA_BDStatusLast This BD is the last BD in this array. It means the transfer ended after this buffer
kSDMA_BDStatusExtend Buffer descriptor extend status for SDMA scripts.

18.6.3 enum sdma_bd_command_t

Enumerator

kSDMA_BDCommandSETDM Load SDMA data memory from ARM core memory buffer.
kSDMA_BDCommandGETDM Copy SDMA data memory to ARM core memory buffer.
kSDMA_BDCommandSETPM Load SDMA program memory from ARM core memory buffer.
kSDMA_BDCommandGETPM Copy SDMA program memory to ARM core memory buffer.
kSDMA_BDCommandSETCTX Load context for one channel into SDMA RAM from ARM platform memory buffer.
kSDMA_BDCommandGETCTX Copy context for one channel from SDMA RAM to ARM platform memory buffer.

18.6.4 enum sdma_context_switch_mode_t

Enumerator

kSDMA_ContextSwitchModeStatic SDMA context switch mode static.
kSDMA_ContextSwitchModeDynamicLowPower SDMA context switch mode dynamic with low power.
kSDMA_ContextSwitchModeDynamicWithNoLoop SDMA context switch mode dynamic with no loop.
kSDMA_ContextSwitchModeDynamic SDMA context switch mode dynamic.

18.6.5 enum sdma_clock_ratio_t

Enumerator

kSDMA_HalfARMClockFreq SDMA core clock frequency half of ARM platform.

kSDMA_ARMClockFreq SDMA core clock frequency equals to ARM platform.

18.6.6 enum sdma_transfer_type_t

Enumerator

kSDMA_MemoryToMemory	Transfer from memory to memory.
kSDMA_PeripheralToMemory	Transfer from peripheral to memory.
kSDMA_MemoryToPeripheral	Transfer from memory to peripheral.

18.6.7 enum sdma_peripheral_t

Enumerator

kSDMA_PeripheralTypeMemory	Peripheral DDR memory.
kSDMA_PeripheralTypeUART	UART use SDMA.
kSDMA_PeripheralTypeUART_SP	UART instance in SPBA use SDMA.
kSDMA_PeripheralTypeSPDIF	SPDIF use SDMA.
kSDMA_PeripheralNormal	Normal peripheral use SDMA.
kSDMA_PeripheralNormal_SP	Normal peripheral in SPBA use SDMA.
kSDMA_PeripheralMultiFifoPDM	multi fifo PDM
kSDMA_PeripheralMultiFifoSaiRX	multi fifo sai rx use SDMA
kSDMA_PeripheralMultiFifoSaiTX	multi fifo sai tx use SDMA

18.6.8 enum _sdma_transfer_status

Enumerator

kStatus_SDMA_ERROR	SDMA context error.
kStatus_SDMA_Busy	Channel is busy and can't handle the transfer request.

18.6.9 enum _sdma_multi_fifo_mask

Enumerator

kSDMA_MultiFifoWatermarkLevelMask	multi fifo watermark level mask
kSDMA_MultiFifoNumsMask	multi fifo nums mask
kSDMA_MultiFifoOffsetMask	multi fifo offset mask
kSDMA_MultiFifoSwDoneMask	multi fifo sw done mask
kSDMA_MultiFifoSwDoneSelectorMask	multi fifo sw done selector mask

18.6.10 enum _sdma_multi_fifo_shift

Enumerator

kSDMA_MultiFifoWatermarkLevelShift	multi fifo watermark level shift
kSDMA_MultiFifoNumsShift	multi fifo nums shift
kSDMA_MultiFifoOffsetShift	multi fifo offset shift
kSDMA_MultiFifoSwDoneShift	multi fifo sw done shift
kSDMA_MultiFifoSwDoneSelectorShift	multi fifo sw done selector shift

18.7 Function Documentation

18.7.1 void SDMA_Init (SDMAARM_Type ∗ *base*, const sdma_config_t ∗ *config*)

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

Parameters

base	SDMA peripheral base address.
config	A pointer to the configuration structure, see "sdma_config_t".

Note

This function enables the minor loop map feature.

18.7.2 void SDMA_Deinit (SDMAARM_Type ∗ *base*)

This function gates the SDMA clock.

Parameters

base	SDMA peripheral base address.

18.7.3 void SDMA_GetDefaultConfig (sdma_config_t ∗ *config*)

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

```
*   config.enableRealTimeDebugPin = false;
*   config.isSoftwareResetClearLock = true;
*   config.ratio = kSDMA_HalfARMClockFreq;
*
```

Parameters

config	A pointer to the SDMA configuration structure.

18.7.4 void SDMA_ResetModule (SDMAARM_Type ∗ *base*)

If only reset ARM core, SDMA register cannot return to reset value, shall call this function to reset all SDMA register to reset vaule. But the internal status cannot be reset.

Parameters

base	SDMA peripheral base address.

18.7.5 static void SDMA_EnableChannelErrorInterrupts (SDMAARM_Type ∗ *base,* uint32_t *channel*) `[inline]`, `[static]`

Enable this will trigger an interrupt while SDMA occurs error while executing scripts.

Parameters

base	SDMA peripheral base address.
channel	SDMA channel number.

18.7.6 static void SDMA_DisableChannelErrorInterrupts (SDMAARM_Type ∗ *base,* uint32_t *channel*) `[inline]`, `[static]`

Parameters

base	SDMA peripheral base address.
channel	SDMA channel number.

18.7.7 void SDMA_ConfigBufferDescriptor (sdma_buffer_descriptor_t ∗ *bd,* uint32_t *srcAddr,* uint32_t *destAddr,* sdma_transfer_size_t *busWidth,* size_t *bufferSize,* bool *isLast,* bool *enableInterrupt,* bool *isWrap,* sdma_transfer_type_t *type*)

This function sets the descriptor contents such as source, dest address and status bits.

Parameters

bd	Pointer to the buffer descriptor structure.
srcAddr	Source address for the buffer descriptor.
destAddr	Destination address for the buffer descriptor.
busWidth	The transfer width, it only can be a member of sdma_transfer_size_t.
bufferSize	Buffer size for this descriptor, this number shall less than 0xFFFF. If need to transfer a big size, shall divide into several buffer descriptors.
isLast	Is the buffer descriptor the last one for the channel to transfer. If only one descriptor used for the channel, this bit shall set to TRUE.
enableInterrupt	If trigger an interrupt while this buffer descriptor transfer finished.
isWrap	Is the buffer descriptor need to be wrapped. While this bit set to true, it will automatically wrap to the first buffer descrtiptor to do transfer.
type	Transfer type, memory to memory, peripheral to memory or memory to peripheral.

18.7.8 static void SDMA_SetChannelPriority (SDMAARM_Type ∗ *base,* uint32_t *channel,* uint8_t *priority*) `[inline]`, `[static]`

This function sets the channel priority. The default value is 0 for all channels, priority 0 will prevents channel from starting, so the priority must be set before start a channel.

Parameters

base	SDMA peripheral base address.
channel	SDMA channel number.
priority	SDMA channel priority.

18.7.9 static void SDMA_SetSourceChannel (SDMAARM_Type ∗ *base,* uint32_t *source,* uint32_t *channelMask*) `[inline]`, `[static]`

This function sets which channel will be triggered by the dma request source.

Parameters

base	SDMA peripheral base address.
source	SDMA dma request source number.
channelMask	SDMA channel mask. 1 means channel 0, 2 means channel 1, 4 means channel 3. SDMA supports an event trigger multi-channel. A channel can also be triggered by several source events.

18.7.10 static void SDMA_StartChannelSoftware (SDMAARM_Type ∗ *base*, uint32_t *channel*) `[inline]`, `[static]`

This function start a channel.

Parameters

base	SDMA peripheral base address.
channel	SDMA channel number.

18.7.11 static void SDMA_StartChannelEvents (SDMAARM_Type ∗ *base*, uint32_t *channel*) `[inline]`, `[static]`

This function start a channel.

Parameters

base	SDMA peripheral base address.
channel	SDMA channel number.

18.7.12 static void SDMA_StopChannel (SDMAARM_Type ∗ *base*, uint32_t *channel*) `[inline]`, `[static]`

This function stops a channel.

Parameters

base	SDMA peripheral base address.

channel	SDMA channel number.

18.7.13 void SDMA_SetContextSwitchMode (SDMAARM_Type ∗ *base*, sdma_context_switch_mode_t *mode*)

Parameters

base	SDMA peripheral base address.
mode	SDMA context switch mode.

18.7.14 static uint32_t SDMA_GetChannelInterruptStatus (SDMAARM_Type ∗ *base*) [inline], [static]

Parameters

base	SDMA peripheral base address.

Returns

The interrupt status for all channels. Check the relevant bits for specific channel.

18.7.15 static void SDMA_ClearChannelInterruptStatus (SDMAARM_Type ∗ *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SDMA peripheral base address.
mask	The interrupt status need to be cleared.

18.7.16 static uint32_t SDMA_GetChannelStopStatus (SDMAARM_Type ∗ *base*) [inline], [static]

Parameters

base	SDMA peripheral base address.

Returns

The stop status for all channels. Check the relevant bits for specific channel.

18.7.17 static void SDMA_ClearChannelStopStatus (SDMAARM_Type $*$ *base*, uint32_t *mask*) `[inline]`, `[static]`

Parameters

base	SDMA peripheral base address.
mask	The stop status need to be cleared.

18.7.18 static uint32_t SDMA_GetChannelPendStatus (SDMAARM_Type $*$ *base*) `[inline]`, `[static]`

Parameters

base	SDMA peripheral base address.

Returns

The pending status for all channels. Check the relevant bits for specific channel.

18.7.19 static void SDMA_ClearChannelPendStatus (SDMAARM_Type $*$ *base*, uint32_t *mask*) `[inline]`, `[static]`

Parameters

base	SDMA peripheral base address.

mask	The pending status need to be cleared.

18.7.20 static uint32_t SDMA_GetErrorStatus (SDMAARM_Type ∗ *base*) [inline], [static]

SDMA channel error flag is asserted while an incoming DMA request was detected and it triggers a channel that is already pending or being serviced. This probably means there is an overflow of data for that channel.

Parameters

base	SDMA peripheral base address.

Returns

The error status for all channels. Check the relevant bits for specific channel.

18.7.21 bool SDMA_GetRequestSourceStatus (SDMAARM_Type ∗ *base,* uint32_t *source*)

Parameters

base	SDMA peripheral base address.
source	DMA request source number.

Returns

True means the request source is pending, otherwise not pending.

18.7.22 void SDMA_CreateHandle (sdma_handle_t ∗ *handle,* SDMAARM_Type ∗ *base,* uint32_t *channel,* sdma_context_data_t ∗ *context*)

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

Parameters

handle	SDMA handle pointer. The SDMA handle stores callback function and parameters.
base	SDMA peripheral base address.
channel	SDMA channel number.
context	Context structure for the channel to download into SDMA. Users shall make sure the context located in a non-cacheable memory, or it will cause SDMA run fail. Users shall not touch the context contents, it only be filled by SDMA driver in SDMA_-SubmitTransfer function.

18.7.23 void SDMA_InstallBDMemory (sdma_handle_t ∗ *handle,* sdma_buffer_descriptor_t ∗ *BDPool,* uint32_t *BDCount*)

This function is called after the SDMA_CreateHandle to use multi-buffer feature.

Parameters

handle	SDMA handle pointer.
BDPool	A memory pool to store BDs. It must be located in non-cacheable address.
BDCount	The number of BD slots.

18.7.24 void SDMA_SetCallback (sdma_handle_t ∗ *handle,* sdma_callback *callback,* void ∗ *userData*)

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

Parameters

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

18.7.25 void SDMA_SetMultiFifoConfig (sdma_transfer_config_t ∗ *config,* uint32_t *fifoNums,* uint32_t *fifoOffset*)

This api is used to support multi fifo for SDMA, if user want to get multi fifo data, then this api shoule be called before submit transfer.

Parameters

config	transfer configurations.
fifoNums	fifo numbers that multi fifo operation perform.
fifoOffset	offset between multififo address.

18.7.26 void SDMA_EnableSwDone (SDMAARM_Type ∗ *base*, sdma_-transfer_config_t ∗ *config*, uint8_t *sel*, sdma_peripheral_t *type*)

Parameters

base	SDMA base.
config	transfer configurations.
sel	sw done selector.
type	peripheral type is used to determine the corresponding peripheral sw done selector bit.

18.7.27 void SDMA_LoadScript (SDMAARM_Type ∗ *base*, uint32_t *destAddr*, void ∗ *srcAddr*, size_t *bufferSizeBytes*)

Parameters

base	SDMA base.
destAddr	dest script address, should be SDMA program memory address.
srcAddr	source address of target script.
bufferSizeBytes	bytes size of script.

18.7.28 void SDMA_DumpScript (SDMAARM_Type ∗ *base*, uint32_t *srcAddr*, void ∗ *destAddr*, size_t *bufferSizeBytes*)

Parameters

base	SDMA base.
srcAddr	should be SDMA program memory address.
destAddr	address to store scripts.
bufferSizeBytes	bytes size of script.

18.7.29 void SDMA_PrepareTransfer (sdma_transfer_config_t ∗ *config*, uint32_t *srcAddr*, uint32_t *destAddr*, uint32_t *srcWidth*, uint32_t *destWidth*, uint32_t *bytesEachRequest*, uint32_t *transferSize*, uint32_t *eventSource*, sdma_peripheral_t *peripheral*, sdma_transfer_type_t *type*)

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

Parameters

config	The user configuration structure of type sdma_transfer_t.
srcAddr	SDMA transfer source address.
destAddr	SDMA transfer destination address.
srcWidth	SDMA transfer source address width(bytes).
destWidth	SDMA transfer destination address width(bytes).
bytesEach-Request	SDMA transfer bytes per channel request.
transferSize	SDMA transfer bytes to be transferred.
eventSource	Event source number for the transfer, if use software trigger, just write 0.
peripheral	Peripheral type, used to decide if need to use some special scripts.
type	SDMA transfer type. Used to decide the correct SDMA script address in SDMA ROM.

Note

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

18.7.30 void SDMA_SubmitTransfer (sdma_handle_t ∗ *handle*, const sdma_transfer_config_t ∗ *config*)

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

Parameters

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

18.7.31 void SDMA_StartTransfer (sdma_handle_t ∗ *handle*)

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

Parameters

handle	SDMA handle pointer.

18.7.32 void SDMA_StopTransfer (sdma_handle_t ∗ *handle*)

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

Parameters

handle	SDMA handle pointer.

18.7.33 void SDMA_AbortTransfer (sdma_handle_t ∗ *handle*)

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

Parameters

handle	DMA handle pointer.

18.7.34 uint32_t SDMA_GetTransferredBytes (sdma_handle_t ∗ *handle*)

This function returns the buffer descriptor count value if not using buffer descriptor. While do a simple transfer, which only uses one descriptor, the SDMA driver inside handle the buffer descriptor. In uart receive case, it can tell users how many data already received, also it can tells users how many data transfferd while error occurred. Notice, the count would not change while transfer is on-going using default SDMA script.

Parameters

handle	DMA handle pointer.

Returns

Transferred bytes.

18.7.35 bool SDMA_IsPeripheralInSPBA (uint32_t *addr*)

Parameters

addr	Address which need to judge.
return	True means located in SPBA, false means not.

18.7.36 void SDMA_HandleIRQ (sdma_handle_t ∗ *handle*)

This function clears the interrupt flags and also handle the CCB for the channel.

Parameters

handle	SDMA handle pointer.

Chapter 19
SEMA4: Hardware Semaphores Driver

19.1 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) (∗(&((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, 0))
  SEMA4 driver version.

19.2 Macro Definition Documentation

19.2.1 #define SEMA4_GATE_NUM_RESET_ALL (64U)

19.3 Function Documentation

19.3.1 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.

19.3.2 void SEMA4_Deinit (SEMA4_Type ∗ *base*)

This function de-initializes the SEMA4 module. It only disables the clock.

Parameters

base	SEMA4 peripheral base address.

19.3.3 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.

19.3.4 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.

19.3.5 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.

19.3.6 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.

19.3.7 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.

19.3.8 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.

19.3.9 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.

19.3.10 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.

19.3.11 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.

19.3.12 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.

19.3.13 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 20
WDOG: Watchdog Timer Driver

20.1 Overview

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

20.2 Typical use case

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

Data Structures

- struct wdog_work_mode_t
  Defines WDOG work mode. More...
- struct wdog_config_t
  Describes WDOG configuration structure. More...

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, 0))
  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.

20.3 Data Structure Documentation

20.3.1 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

20.3.2 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.

20.3.2.0.0.27 Field Documentation

20.3.2.0.0.27.1 bool wdog_config_t::enableTimeOutAssert

20.4 Enumeration Type Documentation

20.4.1 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.

20.4.2 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

20.5 Function Documentation

20.5.1 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

20.5.2 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

20.5.3 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).

20.5.4 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

20.5.5 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

20.5.6 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

20.5.7 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

20.5.8 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

20.5.9 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.

20.5.10 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

20.5.11 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.

20.5.12 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.

20.5.13 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

20.5.14 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 21
Debug Console

21.1 Overview

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data.

21.2 Function groups

21.2.1 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);
```

Selects 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);
```

21.2.2 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(const char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE    /* Select printf, scanf, putchar, getchar of SDK version. */
#define PRINTF                 DbgConsole_Printf
#define SCANF                  DbgConsole_Scanf
#define PUTCHAR                DbgConsole_Putchar
#define GETCHAR                DbgConsole_Getchar
#else                   /* Select printf, scanf, putchar, getchar of toolchain. */
#define PRINTF                 printf
#define SCANF                  scanf
#define PUTCHAR                putchar
#define GETCHAR                getchar
#endif /* SDK_DEBUGCONSOLE */
```

21.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalents 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

Print out failure messages using MCUXpresso SDK __assert_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
    PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file ,
      line, func);
    for (;;)
    {}
}
```

Note:

To use ’printf’ and ’scanf’ for GNUC Base, add file **’fsl_sbrk.c’** in path: **..\{package}\devices\{subset}\utilities\fsl-_sbrk.c** to your project.

Modules

- SWO
  /!*
- Semihosting

Macros

- #define SDK_DEBUGCONSOLE 1U
  Definition to select sdk or toolchain printf, scanf.
- #define SDK_DEBUGCONSOLE_UART
  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.

Initialization

- status_t DbgConsole_Init (uint8_t instance, uint32_t baudRate, serial_port_type_t device, uint32_t clkSrcFreq)
  Initializes the peripheral used for debug messages.
- status_t DbgConsole_Deinit (void)
  De-initializes the peripheral used for debug messages.
- int DbgConsole_Printf (const char ∗formatString,...)
  Writes formatted output to the standard output stream.
- int DbgConsole_Putchar (int ch)
  Writes a character to stdout.
- int DbgConsole_Scanf (char ∗formatString,...)
  Reads formatted data from the standard input stream.
- int DbgConsole_Getchar (void)
  Reads a character from standard input.
- status_t DbgConsole_Flush (void)
  Debug console flush.

21.4 Macro Definition Documentation

21.4.1 #define SDK_DEBUGCONSOLE 1U

21.4.2 #define SDK_DEBUGCONSOLE_UART

21.5 Function Documentation

21.5.1 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.
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

21.5.2 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.

21.5.3 int DbgConsole_Printf (const char ∗ *formatString,* *...*)

Call this function to write a formatted output to the standard output stream.

Parameters

formatString	Format control string.

Returns

Returns the number of characters printed or a negative value if an error occurs.

21.5.4 int DbgConsole_Putchar (int *ch*)

Call this function to write a character to stdout.

Parameters

ch	Character to be written.

Returns

Returns the character written.

21.5.5 int DbgConsole_Scanf (char * *formatString*, ...)

Call this function to read formatted data from the standard input stream.

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG_CONSOLE_TRANSFER_NON_B-LOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole_TryGetchar to get the input char.

Parameters

formatString	Format control string.

Returns

Returns the number of fields successfully converted and assigned.

21.5.6 int DbgConsole_Getchar (void)

Call this function to read a character from standard input.

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG_CONSOLE_TRANSFER_NON_B-LOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole_TryGetchar to get the input char.

Returns

Returns the character read.

21.5.7 status_t DbgConsole_Flush (void)

Call this function to wait the tx buffer empty. If interrupt transfer is using, make sure the global IRQ is enable before call this function This function should be called when 1, before enter power down mode 2, log is required to print to terminal immediately

Returns

Indicates whether wait idle was successful or not.

21.5.8 int StrFormatPrintf (const char ∗ *fmt,* va_list *ap,* char ∗ *buf,* printfCb *cb*)

Note

I/O is performed by calling given function pointer using following (∗func_ptr)(c);

Parameters

in	*fmt*	Format string for printf.
in	*ap*	Arguments to printf.
in	*buf*	pointer to the buffer
	cb	print callbck function pointer

Returns

Number of characters to be print

21.5.9 int StrFormatScanf (const char ∗ *line_ptr,* char ∗ *format,* va_list *args_ptr*)

Parameters

in	*line_ptr*	The input line of ASCII data.
in	*format*	Format first points to the format string.
in	*args_ptr*	The list of parameters.

Returns

Number of input items converted and assigned.

Return values

IO_EOF	When line_ptr is empty string "".

21.6 Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism can be used, for example, to enable functions in the C library, such as printf() and scanf(), to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

21.6.1 Guide Semihosting for IAR

NOTE: After the setting both "printf" and "scanf" are available for debugging, if you want use PRINTF with semihosting, please make sure the SDK_DEBUGCONSOLE is disabled.

Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

Step 2: Building the project

1. Compile and link the project by choosing Project>Make or F7.
2. Alternatively, click the Make button on the tool bar. The Make command compiles and links those files that have been modified.

Step 3: Starting semihosting

1. Choose "Semihosting_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting.

1. Make sure the SDK_DEBUGCONSOLE_UART is not defined, remove the default definition in fsl_debug_console.h.

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

21.6.2 Guide Semihosting for Keil µVision

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

Step 1: Setting up the environment

1. Make sure the SDK_DEBUGCONSOLE_UART is not defined, remove the default definition in fsl_debug_console.h..
2. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
3. Open Project>Options for target or using Alt+F7 or click.
4. Select “Debug” tab, select “J-Link/J-Trace Cortex” and click “Setting button”.
5. Select “Debug” tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click O-K, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

Step 3: Building the project

1. Compile and link the project by choosing Project>Build Target or using F7.

Step 4: Building the project

1. Choose “Debug” on menu bar or Ctrl F5.
2. In menu bar, choose "Serial Window" and click to "Debug (printf) Viewer".
3. Run line by line to see result in Console Window.

21.6.3 Guide Semihosting for MCUXpresso IDE

Step 1: Setting up the environment

1. To set debugger options, choose Project>Properties. select the setting category.
2. Select Tool Settings, unfold MCU C Compile.
3. Select Preprocessor item.
4. Set SDK_DEBUGCONSOLE=0, if set SDK_DEBUGCONSOLE=1, the log will be redirect to the UART.

Step 2: Building the project

1. Compile and link the project.

Step 3: Starting semihosting

1. Download and debug the project.
2. When the project runs successfully, the result can be seen in the Console window.

Semihosting can also be selected through the "Quick settings" menu in the left bottom window, Quick settings->SDK Debug Console->Semihost console.

21.6.4 Guide Semihosting for ARMGCC

Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
   - "Host Name (or IP address)" : localhost
   - "Port" :2333
   - "Connection type" : Telet.
   - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

Add to "CMakeLists.txt"

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__heap_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__heap_size__=0x2000")

Step 2: Building the project

1. Change "CMakeLists.txt":
   Change "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=nano.specs")"
   to "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=rdimon.specs")"
   Replace paragraph
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")
To
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --specs=rdimon.specs ")
Remove
target_link_libraries(semihosting_ARMGCC.elf debug nosys)

2. Run "build_debug.bat" to build project

Step 3: Starting semihosting

(a) Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

(b) After the setting, press "enter". The PuTTY window now shows the printf() output.

21.7 SWO

/∗!

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

21.7.1 Guide SWO for SDK

NOTE: After the setting both "printf" and "PRINTF" are available for debugging, JlinkSWOViewer can be used to capture the output log.

Step 1: Setting up the environment

1. Define DEBUG_CONSOLE_IO_SWO in your project settings.
2. Prepare code, the port and baudrate can be decided by application, clkSrcFreq should be mcu core clock frequency:

   ```
   DbgConsole_Init(port, baudrate, DEBUG_CONSOLE_DEVICE_TYPE_SWO, clkSrcFreq);
   ```

3. Use PRINTF or printf to print some thing in application.

   #### Step 2: Building the project

Step 3: Download and run project

21.7.1.1 Guide SWO for IAR

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

Step 1: Setting up the environment

1. Choose project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via SWO.
4. To configure the hardware's generation of trace data, click the SWO Configuration button available in the SWO Configuration dialog box. The value of the CPU clock option must reflect the frequency of the CPU clock speed at which the application executes. Note also that the settings you make are preserved between debug sessions. To decrease the amount of transmissions on the communication channel, you can disable the Timestamp option. Alternatively, set a lower rate for PC Sampling or use a higher SWO clock frequency.
5. Open the SWO Trace window from J-LINK,and click the Activate button to enable trace data collection.
6. Make sure the SDK_DEBUGCONSOLE_UART is not defined, remove the default definition in fsl_debug_console.h..
7. Start the project by choosing Project>Download and Debug.

Step 2: Building the project

Step 3: Starting swo

1. Download and debug application.
2. Choose View -> Terminal I/O to display the output from the I/O operations.
3. Run application.

21.7.2 Guide SWO for Keil µVision

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

Step 1: Setting up the environment

1. Make sure the SDK_DEBUGCONSOLE_UART is not defined, remove the default definition in fsl_debug_console.h.
2. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
3. Open Project>Options for target or using Alt+F7 or click.
4. Select “Debug” tab, select “J-Link/J-Trace Cortex” and click “Setting button”.
5. Select “Debug” tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click O-K, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

Step 3: Building the project

1. Compile and link the project by choosing Project>Build Target or using F7.

Step 4: Run the project

1. Choose “Debug” on menu bar or Ctrl F5.
2. In menu bar, choose "Serial Window" and click to "Debug (printf) Viewer".
3. Run line by line to see result in Console Window.

21.7.3 Guide SWO for MCUXpresso IDE

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

21.7.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.