Document Number: MCUXSDKAPIRM
Rev 2.11.0
Jan 2022

MCUXpresso SDK API Reference Manual

NXP Semiconductors

Contents

Chapter 1
Introduction

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

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

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

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

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

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

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

MCUXpresso SDK Folder Structure

Chapter 2
Trademarks

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

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

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

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

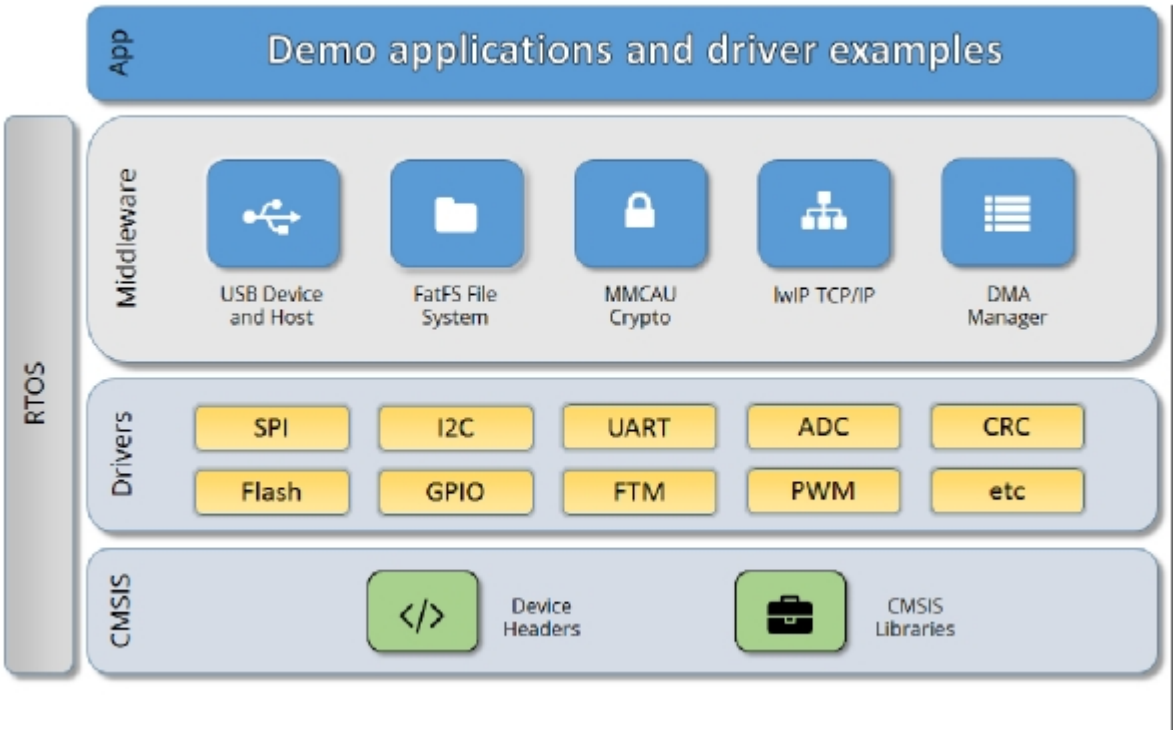
Chapter 3
Architectural Overview

This chapter provides the architectural overview for the MCUXpresso Software Development Kit (MCUXpresso SDK). It describes each layer within the architecture and its associated components.

Overview

The MCUXpresso SDK architecture consists of five key components listed below.

1. The Arm Cortex Microcontroller Software Interface Standard (CMSIS) CORE compliance device-specific header files, SOC Header, and CMSIS math/DSP libraries.
2. Peripheral Drivers
3. Real-time Operating Systems (RTOS)
4. Stacks and Middleware that integrate with the MCUXpresso SDK
5. Demo Applications based on the MCUXpresso SDK



MCUXpresso SDK Block Diagram

MCU header files

Each supported MCU device in the MCUXpresso SDK has an overall System-on Chip (SoC) memory-

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

CMSIS Support

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

MCUXpresso SDK Peripheral Drivers

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

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

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

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

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

Interrupt handling for transactional APIs

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

```
        PUBWEAK SPI0_IRQHandler
        PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

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

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

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

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

Feature Header Files

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

Application

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

Chapter 4
Clock Driver

4.1 Overview

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

The clock driver supports:

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

Files

- file fsl_clock.h

Data Structures

- struct pll_config_t
  PLL configuration structure. *More...*
- struct pll_setup_t
  PLL0 setup structure This structure can be used to pre-build a PLL setup configuration at run-time and quickly set the PLL to the configuration. *More...*

Macros

- #define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0
  Configure whether driver controls clock.
- #define CLOCK_USR_CFG_PLL_CONFIG_CACHE_COUNT 2U
  User-defined the size of cache for CLOCK_PllGetConfig() function.
- #define ROM_CLOCKS
  Clock ip name array for ROM.
- #define SRAM_CLOCKS
  Clock ip name array for SRAM.
- #define FLASH_CLOCKS
  Clock ip name array for FLASH.
- #define FMC_CLOCKS
  Clock ip name array for FMC.
- #define INPUTMUX_CLOCKS
  Clock ip name array for INPUTMUX.
- #define IOCON_CLOCKS
  Clock ip name array for IOCON.
- #define GPIO_CLOCKS
  Clock ip name array for GPIO.
- #define PINT_CLOCKS
  Clock ip name array for PINT.

- #define GINT_CLOCKS
  Clock ip name array for GINT.
- #define DMA_CLOCKS
  Clock ip name array for DMA.
- #define CRC_CLOCKS
  Clock ip name array for CRC.
- #define WWDT_CLOCKS
  Clock ip name array for WWDT.
- #define RTC_CLOCKS
  Clock ip name array for RTC.
- #define MAILBOX_CLOCKS
  Clock ip name array for Mailbox.
- #define LPADC_CLOCKS
  Clock ip name array for LPADC.
- #define MRT_CLOCKS
  Clock ip name array for MRT.
- #define OSTIMER_CLOCKS
  Clock ip name array for OSTIMER.
- #define SCT_CLOCKS
  Clock ip name array for SCT0.
- #define MCAN_CLOCKS
  Clock ip name array for MCAN.
- #define UTICK_CLOCKS
  Clock ip name array for UTICK.
- #define FLEXCOMM_CLOCKS
  Clock ip name array for FLEXCOMM.
- #define LPUART_CLOCKS
  Clock ip name array for LPUART.
- #define BI2C_CLOCKS
  Clock ip name array for BI2C.
- #define LPSPI_CLOCKS
  Clock ip name array for LSPI.
- #define FLEXI2S_CLOCKS
  Clock ip name array for FLEXI2S.
- #define CTIMER_CLOCKS
  Clock ip name array for CTIMER.
- #define EZHA_CLOCKS
  Clock ip name array for EZHA.
- #define EZHB_CLOCKS
  Clock ip name array for EZHB.
- #define COMP_CLOCKS
  Clock ip name array for COMP.
- #define USB1CLK_CLOCKS
  Clock ip name array for USB1CLK.
- #define FREQME_CLOCKS
  Clock ip name array for FREQME.
- #define USBRAM_CLOCKS
  Clock ip name array for USBRAM.
- #define CDOG_CLOCKS
  Clock ip name array for CDOG.
- #define RNG_CLOCKS

Clock ip name array for RNG.

- #define USBHMR0_CLOCKS
  Clock ip name array for USBHMR0.
- #define USBHSL0_CLOCKS
  Clock ip name array for USBHSL0.
- #define HASHCRYPT_CLOCKS
  Clock ip name array for HashCrypt.
- #define PLULUT_CLOCKS
  Clock ip name array for PLULUT.
- #define PUF_CLOCKS
  Clock ip name array for PUF.
- #define CASPER_CLOCKS
  Clock ip name array for CASPER.
- #define ANALOGCTRL_CLOCKS
  Clock ip name array for ANALOGCTRL.
- #define HS_LSPI_CLOCKS
  Clock ip name array for HS_LSPI.
- #define GPIO_SEC_CLOCKS
  Clock ip name array for GPIO_SEC.
- #define GPIO_SEC_INT_CLOCKS
  Clock ip name array for GPIO_SEC_INT.
- #define USBD_CLOCKS
  Clock ip name array for USBD.
- #define USBH_CLOCKS
  Clock ip name array for USBH.
- #define CLK_GATE_REG_OFFSET_SHIFT 8U
  Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.
- #define BUS_CLK kCLOCK_BusClk
  Peripherals clock source definition.
- #define CLK_ATTACH_ID(mux, sel, pos) ((((uint32_t)(mux) << 0U) | (((uint32_t)(sel) + 1U) & 0xFU) << 8U) << ((uint32_t)(pos)*12U))
  Clock Mux Switches The encoding is as follows each connection identified is 32bits wide while 24bits are valuable starting from LSB upwards.
- #define PLL_CONFIGFLAG_USEINRATE (1U << 0U)
  PLL configuration structure flags for 'flags' field These flags control how the PLL configuration function sets up the PLL setup structure.
- #define PLL_CONFIGFLAG_FORCENOFRACT (1U << 2U)
  Force non-fractional output mode, PLL output will not use the fractional, automatic bandwidth, or SS hardware.
- #define PLL_SETUPFLAG_POWERUP (1U << 0U)
  PLL setup structure flags for 'flags' field These flags control how the PLL setup function sets up the PLL.
- #define PLL_SETUPFLAG_WAITLOCK (1U << 1U)
  Setup will wait for PLL lock, implies the PLL will be pwoered on.
- #define PLL_SETUPFLAG_ADGVOLT (1U << 2U)
  Optimize system voltage for the new PLL rate.
- #define PLL_SETUPFLAG_USEFEEDBACKDIV2 (1U << 3U)
  Use feedback divider by 2 in divider path.

Enumerations

- enum clock_ip_name_t {
  kCLOCK_IpInvalid = 0U,
  kCLOCK_Rom = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 1),
  kCLOCK_Sram1 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 3),
  kCLOCK_Sram2 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 4),
  kCLOCK_Flash = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 7),
  kCLOCK_Fmc = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 8),
  kCLOCK_InputMux = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 11),
  kCLOCK_Iocon = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 13),
  kCLOCK_Gpio0 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 14),
  kCLOCK_Gpio1 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 15),
  kCLOCK_Pint = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 18),
  kCLOCK_Gint = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 19),
  kCLOCK_Dma0 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 20),
  kCLOCK_Crc = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 21),
  kCLOCK_Wwdt = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 22),
  kCLOCK_Rtc = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 23),
  kCLOCK_Mailbox = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 26),
  kCLOCK_Adc0 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, 27),
  kCLOCK_Mrt = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 0),
  kCLOCK_OsTimer0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 1),
  kCLOCK_Sct0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 2),
  kCLOCK_Mcan = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 7),
  kCLOCK_Utick0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 10),
  kCLOCK_FlexComm0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 11),
  kCLOCK_FlexComm1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 12),
  kCLOCK_FlexComm2 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 13),
  kCLOCK_FlexComm3 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 14),
  kCLOCK_FlexComm4 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 15),
  kCLOCK_FlexComm5 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 16),
  kCLOCK_FlexComm6 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 17),
  kCLOCK_FlexComm7 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 18),
  kCLOCK_MinUart0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 11),
  kCLOCK_MinUart1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 12),
  kCLOCK_MinUart2 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 13),
  kCLOCK_MinUart3 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 14),
  kCLOCK_MinUart4 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 15),
  kCLOCK_MinUart5 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 16),
  kCLOCK_MinUart6 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 17),
  kCLOCK_MinUart7 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 18),
  kCLOCK_LSpi0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 11),
  kCLOCK_LSpi1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 12),
  kCLOCK_LSpi2 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 13),
  kCLOCK_LSpi3 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 14),
  kCLOCK_LSpi4 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 15),
  kCLOCK_LSpi5 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 16),
  kCLOCK_LSpi6 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 17),
  kCLOCK_LSpi7 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, 18),

kCLOCK_Gpio_Sec_Int = CLK_GATE_DEFINE(AHB_CLK_CTRL2, 30) }
Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

- enum clock_name_t {
  kCLOCK_CoreSysClk,
  kCLOCK_BusClk,
  kCLOCK_ClockOut,
  kCLOCK_FroHf,
  kCLOCK_Pll1Out,
  kCLOCK_Mclk,
  kCLOCK_Fro12M,
  kCLOCK_Fro1M,
  kCLOCK_ExtClk,
  kCLOCK_Pll0Out,
  kCLOCK_FlexI2S }
  Clock name used to get clock frequency.
- enum clock_attach_id_t {
  kFRO12M_to_MAIN_CLK = MUX_A(CM_MAINCLKSELA, 0) | MUX_B(CM_MAINCLKSELB, 0, 0),
  kEXT_CLK_to_MAIN_CLK = MUX_A(CM_MAINCLKSELA, 1) | MUX_B(CM_MAINCLKSELB, 0, 0),
  kFRO1M_to_MAIN_CLK = MUX_A(CM_MAINCLKSELA, 2) | MUX_B(CM_MAINCLKSELB, 0, 0),
  kFRO_HF_to_MAIN_CLK = MUX_A(CM_MAINCLKSELA, 3) | MUX_B(CM_MAINCLKSELB, 0, 0),
  kPLL0_to_MAIN_CLK = MUX_A(CM_MAINCLKSELA, 0) | MUX_B(CM_MAINCLKSELB, 1, 0),
  kPLL1_to_MAIN_CLK = MUX_A(CM_MAINCLKSELA, 0) | MUX_B(CM_MAINCLKSELB, 2, 0),
  kOSC32K_to_MAIN_CLK = MUX_A(CM_MAINCLKSELA, 0) | MUX_B(CM_MAINCLKSE-

LB, 3, 0),
kMAIN_CLK_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 0),
kPLL0_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 1),
kEXT_CLK_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 2),
kFRO_HF_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 3),
kFRO1M_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 4),
kPLL1_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 5),
kOSC32K_to_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 6),
kNONE_to_SYS_CLKOUT = MUX_A(CM_CLKOUTCLKSEL, 7),
kFRO12M_to_PLL0 = MUX_A(CM_PLL0CLKSEL, 0),
kEXT_CLK_to_PLL0 = MUX_A(CM_PLL0CLKSEL, 1),
kFRO1M_to_PLL0 = MUX_A(CM_PLL0CLKSEL, 2),
kOSC32K_to_PLL0 = MUX_A(CM_PLL0CLKSEL, 3),
kNONE_to_PLL0 = MUX_A(CM_PLL0CLKSEL, 7),
kMCAN_DIV_to_MCAN = MUX_A(CM_MCANCLKSEL, 0),
kFRO1M_to_MCAN = MUX_A(CM_MCANCLKSEL, 1),
kOSC32K_to_MCAN = MUX_A(CM_MCANCLKSEL, 2),
kNONE_to_MCAN = MUX_A(CM_MCANCLKSEL, 7),
kMAIN_CLK_to_ADC_CLK = MUX_A(CM_ADCASYNCCLKSEL, 0),
kPLL0_to_ADC_CLK = MUX_A(CM_ADCASYNCCLKSEL, 1),
kFRO_HF_to_ADC_CLK = MUX_A(CM_ADCASYNCCLKSEL, 2),
kEXT_CLK_to_ADC_CLK = MUX_A(CM_ADCASYNCCLKSEL, 4),
kNONE_to_ADC_CLK = MUX_A(CM_ADCASYNCCLKSEL, 7),
kMAIN_CLK_to_USB0_CLK = MUX_A(CM_USB0CLKSEL, 0),
kPLL0_to_USB0_CLK = MUX_A(CM_USB0CLKSEL, 1),
kFRO_HF_to_USB0_CLK = MUX_A(CM_USB0CLKSEL, 3),
kPLL1_to_USB0_CLK = MUX_A(CM_USB0CLKSEL, 5),
kNONE_to_USB0_CLK = MUX_A(CM_USB0CLKSEL, 7),
kOSC32K_to_CLK32K = MUX_A(CM_CLK32KCLKSEL, 0),
kFRO1MDIV_to_CLK32K = MUX_A(CM_CLK32KCLKSEL, 1),
kNONE_to_CLK32K = MUX_A(CM_CLK32KCLKSEL, 7),
kMAIN_CLK_to_FLEXCOMM0 = MUX_A(CM_FXCOMCLKSEL0, 0),
kPLL0_DIV_to_FLEXCOMM0 = MUX_A(CM_FXCOMCLKSEL0, 1),
kFRO12M_to_FLEXCOMM0 = MUX_A(CM_FXCOMCLKSEL0, 2),
kFRO_HF_DIV_to_FLEXCOMM0 = MUX_A(CM_FXCOMCLKSEL0, 3),
kFRO1M_to_FLEXCOMM0 = MUX_A(CM_FXCOMCLKSEL0, 4),
kMCLK_to_FLEXCOMM0 = MUX_A(CM_FXCOMCLKSEL0, 5),
kOSC32K_to_FLEXCOMM0 = MUX_A(CM_FXCOMCLKSEL0, 6),
kNONE_to_FLEXCOMM0 = MUX_A(CM_FXCOMCLKSEL0, 7),
kMAIN_CLK_to_FLEXCOMM1 = MUX_A(CM_FXCOMCLKSEL1, 0),
kPLL0_DIV_to_FLEXCOMM1 = MUX_A(CM_FXCOMCLKSEL1, 1),
kFRO12M_to_FLEXCOMM1 = MUX_A(CM_FXCOMCLKSEL1, 2),
kFRO_HF_DIV_to_FLEXCOMM1 = MUX_A(CM_FXCOMCLKSEL1, 3),
kFRO1M_to_FLEXCOMM1 = MUX_A(CM_FXCOMCLKSEL1, 4),
kMCLK_to_FLEXCOMM1 = MUX_A(CM_FXCOMCLKSEL1, 5),
kOSC32K_to_FLEXCOMM1 = MUX_A(CM_FXCOMCLKSEL1, 6),
kNONE_to_FLEXCOMM1 = MUX_A(CM_FXCOMCLKSEL1, 7),
kMAIN_CLK_to_FLEXCOMM2 = MUX_A(CM_FXCOMCLKSEL2, 0),
kPLL0_DIV_to_FLEXCOMM2 = MUX_A(CM_FXCOMCLKSEL2, 1),

kNONE_to_NONE = (int)0x80000000U }
The enumerator of clock attach Id.

- enum clock_div_name_t {
  kCLOCK_DivSystickClk0 = 0,
  kCLOCK_DivArmTrClkDiv = 2,
  kCLOCK_DivCanClk = 3,
  kCLOCK_DivFlexFrg0 = 8,
  kCLOCK_DivFlexFrg1 = 9,
  kCLOCK_DivFlexFrg2 = 10,
  kCLOCK_DivFlexFrg3 = 11,
  kCLOCK_DivFlexFrg4 = 12,
  kCLOCK_DivFlexFrg5 = 13,
  kCLOCK_DivFlexFrg6 = 14,
  kCLOCK_DivFlexFrg7 = 15,
  kCLOCK_DivAhbClk = 32,
  kCLOCK_DivClkOut = 33,
  kCLOCK_DivFrohfClk = 34,
  kCLOCK_DivWdtClk = 35,
  kCLOCK_DivAdcAsyncClk = 37,
  kCLOCK_DivUsb0Clk = 38,
  kCLOCK_DivFro1mClk = 40,
  kCLOCK_DivMClk = 43,
  kCLOCK_DivSctClk = 45,
  kCLOCK_DivPll0Clk = 49 }
  Clock dividers.
- enum ss_progmodfm_t {
  kSS_MF_512 = (0 << 20),
  kSS_MF_384 = (1 << 20),
  kSS_MF_256 = (2 << 20),
  kSS_MF_128 = (3 << 20),
  kSS_MF_64 = (4 << 20),
  kSS_MF_32 = (5 << 20),
  kSS_MF_24 = (6 << 20),
  kSS_MF_16 = (7 << 20) }
  PLL Spread Spectrum (SS) Programmable modulation frequency See (MF) field in the PLL0SSCG1 register in the UM.
- enum ss_progmoddp_t {
  kSS_MR_K0 = (0 << 23),
  kSS_MR_K1 = (1 << 23),
  kSS_MR_K1_5 = (2 << 23),
  kSS_MR_K2 = (3 << 23),
  kSS_MR_K3 = (4 << 23),
  kSS_MR_K4 = (5 << 23),
  kSS_MR_K6 = (6 << 23),
  kSS_MR_K8 = (7 << 23) }

PLL Spread Spectrum (SS) Programmable frequency modulation depth See (MR) field in the PLL0SSCG1 register in the UM.

- enum ss_modwvctrl_t {
  kSS_MC_NOC = (0 << 26),
  kSS_MC_RECC = (2 << 26),
  kSS_MC_MAXC = (3 << 26) }
  PLL Spread Spectrum (SS) Modulation waveform control See (MC) field in the PLL0SSCG1 register in the UM.
- enum pll_error_t {
  kStatus_PLL_Success = MAKE_STATUS(kStatusGroup_Generic, 0),
  kStatus_PLL_OutputTooLow = MAKE_STATUS(kStatusGroup_Generic, 1),
  kStatus_PLL_OutputTooHigh = MAKE_STATUS(kStatusGroup_Generic, 2),
  kStatus_PLL_InputTooLow = MAKE_STATUS(kStatusGroup_Generic, 3),
  kStatus_PLL_InputTooHigh = MAKE_STATUS(kStatusGroup_Generic, 4),
  kStatus_PLL_OutsideIntLimit = MAKE_STATUS(kStatusGroup_Generic, 5),
  kStatus_PLL_CCOTooLow = MAKE_STATUS(kStatusGroup_Generic, 6),
  kStatus_PLL_CCOTooHigh = MAKE_STATUS(kStatusGroup_Generic, 7) }
  PLL status definitions.
- enum clock_usbfs_src_t {
  kCLOCK_UsbfsSrcFro = (uint32_t)kCLOCK_FroHf,
  kCLOCK_UsbfsSrcPll0 = (uint32_t)kCLOCK_Pll0Out,
  kCLOCK_UsbfsSrcMainClock = (uint32_t)kCLOCK_CoreSysClk,
  kCLOCK_UsbfsSrcPll1 = (uint32_t)kCLOCK_Pll1Out,
  kCLOCK_UsbfsSrcNone }
  USB FS clock source definition.
- enum clock_usbhs_src_t { kCLOCK_UsbSrcUnused = 0xFFFFFFFFU }
  USBhs clock source definition.
- enum clock_usb_phy_src_t { kCLOCK_UsbPhySrcExt = 0U }
  Source of the USB HS PHY.

Functions

- static void CLOCK_EnableClock (clock_ip_name_t clk)
  Enable the clock for specific IP.
- static void CLOCK_DisableClock (clock_ip_name_t clk)
  Disable the clock for specific IP.
- status_t CLOCK_SetupFROClocking (uint32_t iFreq)
  Initialize the Core clock to given frequency (12, 48 or 96 MHz). Turns on FRO and uses default CCO, if freq is 12000000, then high speed output is off, else high speed output is enabled.
- void CLOCK_SetFLASHAccessCyclesForFreq (uint32_t system_freq_hz)
  Set the flash wait states for the input freuqency.
- status_t CLOCK_SetupExtClocking (uint32_t iFreq)
  Initialize the external osc clock to given frequency.
- status_t CLOCK_SetupI2SMClkClocking (uint32_t iFreq)
  Initialize the I2S MCLK clock to given frequency.
- status_t CLOCK_SetupPLUClkInClocking (uint32_t iFreq)
  Initialize the PLU CLKIN clock to given frequency.
- void CLOCK_AttachClk (clock_attach_id_t connection)
  Configure the clock selection muxes.

- clock_attach_id_t CLOCK_GetClockAttachId (clock_attach_id_t attachId)
  Get the actual clock attach id. This fuction uses the offset in input attach id, then it reads the actual source value in the register and combine the offset to obtain an actual attach id.
- void CLOCK_SetClkDiv (clock_div_name_t div_name, uint32_t divided_by_value, bool reset)
  Setup peripheral clock dividers.
- void CLOCK_SetRtc1khzClkDiv (uint32_t divided_by_value)
  Setup rtc 1khz clock divider.
- void CLOCK_SetRtc1hzClkDiv (uint32_t divided_by_value)
  Setup rtc 1hz clock divider.
- uint32_t CLOCK_SetFlexCommClock (uint32_t id, uint32_t freq)
  Set the flexcomm output frequency.
- uint32_t CLOCK_GetFlexCommInputClock (uint32_t id)
  Return Frequency of flexcomm input clock.
- uint32_t CLOCK_GetFreq (clock_name_t clockName)
  Return Frequency of selected clock.
- uint32_t CLOCK_GetFro12MFreq (void)
  Return Frequency of FRO 12MHz.
- uint32_t CLOCK_GetFro1MFreq (void)
  Return Frequency of FRO 1MHz.
- uint32_t CLOCK_GetClockOutClkFreq (void)
  Return Frequency of ClockOut.
- uint32_t CLOCK_GetMCanClkFreq (void)
  Return Frequency of Can Clock.
- uint32_t CLOCK_GetAdcClkFreq (void)
  Return Frequency of Adc Clock.
- uint32_t CLOCK_GetUsb0ClkFreq (void)
  Return Frequency of Usb0 Clock.
- uint32_t CLOCK_GetUsb1ClkFreq (void)
  Return Frequency of Usb1 Clock.
- uint32_t CLOCK_GetMclkClkFreq (void)
  Return Frequency of MClk Clock.
- uint32_t CLOCK_GetSctClkFreq (void)
  Return Frequency of SCTimer Clock.
- uint32_t CLOCK_GetExtClkFreq (void)
  Return Frequency of External Clock.
- uint32_t CLOCK_GetWdtClkFreq (void)
  Return Frequency of Watchdog.
- uint32_t CLOCK_GetFroHfFreq (void)
  Return Frequency of High-Freq output of FRO.
- uint32_t CLOCK_GetPll0OutFreq (void)
  Return Frequency of PLL.
- uint32_t CLOCK_GetPll1OutFreq (void)
  Return Frequency of USB PLL.
- uint32_t CLOCK_GetOsc32KFreq (void)
  Return Frequency of 32kHz osc.
- uint32_t CLOCK_GetCoreSysClkFreq (void)
  Return Frequency of Core System.
- uint32_t CLOCK_GetI2SMClkFreq (void)
  Return Frequency of I2S MCLK Clock.
- uint32_t CLOCK_GetPLUClkInFreq (void)
  Return Frequency of PLU CLKIN Clock.

- uint32_t CLOCK_GetFlexCommClkFreq (uint32_t id)
  Return Frequency of FlexComm Clock.
- uint32_t CLOCK_GetHsLspiClkFreq (void)
  Return Frequency of High speed SPI Clock.
- uint32_t CLOCK_GetCTimerClkFreq (uint32_t id)
  Return Frequency of CTimer functional Clock.
- uint32_t CLOCK_GetSystickClkFreq (uint32_t id)
  Return Frequency of SystickClock.
- uint32_t CLOCK_GetPLL0InClockRate (void)
  Return PLL0 input clock rate.
- uint32_t CLOCK_GetPLL1InClockRate (void)
  Return PLL1 input clock rate.
- uint32_t CLOCK_GetPLL0OutClockRate (bool recompute)
  Return PLL0 output clock rate.
- uint32_t CLOCK_GetPLL1OutClockRate (bool recompute)
  Return PLL1 output clock rate.
- __STATIC_INLINE void CLOCK_SetBypassPLL0 (bool bypass)
  Enables and disables PLL0 bypass mode.
- __STATIC_INLINE void CLOCK_SetBypassPLL1 (bool bypass)
  Enables and disables PLL1 bypass mode.
- __STATIC_INLINE bool CLOCK_IsPLL0Locked (void)
  Check if PLL is locked or not.
- __STATIC_INLINE bool CLOCK_IsPLL1Locked (void)
  Check if PLL1 is locked or not.
- void CLOCK_SetStoredPLL0ClockRate (uint32_t rate)
  Store the current PLL0 rate.
- uint32_t CLOCK_GetPLL0OutFromSetup (pll_setup_t *pSetup)
  Return PLL0 output clock rate from setup structure.
- uint32_t CLOCK_GetPLL1OutFromSetup (pll_setup_t *pSetup)
  Return PLL1 output clock rate from setup structure.
- pll_error_t CLOCK_SetupPLL0Data (pll_config_t *pControl, pll_setup_t *pSetup)
  Set PLL0 output based on the passed PLL setup data.
- pll_error_t CLOCK_SetupPLL0Prec (pll_setup_t *pSetup, uint32_t flagcfg)
  Set PLL output from PLL setup structure (precise frequency)
- pll_error_t CLOCK_SetPLL0Freq (const pll_setup_t *pSetup)
  Set PLL output from PLL setup structure (precise frequency)
- pll_error_t CLOCK_SetPLL1Freq (const pll_setup_t *pSetup)
  Set PLL output from PLL setup structure (precise frequency)
- void CLOCK_SetupPLL0Mult (uint32_t multiply_by, uint32_t input_freq)
  Set PLL0 output based on the multiplier and input frequency.
- static void CLOCK_DisableUsbDevicefs0Clock (clock_ip_name_t clk)
  Disable USB clock.
- bool CLOCK_EnableUsbfs0DeviceClock (clock_usbfs_src_t src, uint32_t freq)
  Enable USB Device FS clock.
- bool CLOCK_EnableUsbfs0HostClock (clock_usbfs_src_t src, uint32_t freq)
  Enable USB HOST FS clock.
- bool CLOCK_EnableUsbhs0PhyPllClock (clock_usb_phy_src_t src, uint32_t freq)
  Enable USB phy clock.
- bool CLOCK_EnableUsbhs0DeviceClock (clock_usbhs_src_t src, uint32_t freq)
  Enable USB Device HS clock.
- bool CLOCK_EnableUsbhs0HostClock (clock_usbhs_src_t src, uint32_t freq)

Enable USB HOST HS clock.
- void CLOCK_EnableOstimer32kClock (void)
  Enable the OSTIMER 32k clock.

Driver version

- #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 3, 7))
  CLOCK driver version 2.3.7.

4.2 Data Structure Documentation

4.2.1 struct pll_config_t

This structure can be used to configure the settings for a PLL setup structure. Fill in the desired configuration for the PLL and call the PLL setup function to fill in a PLL setup structure.

Data Fields

- uint32_t desiredRate
  Desired PLL rate in Hz.
- uint32_t inputRate
  PLL input clock in Hz, only used if PLL_CONFIGFLAG_USEINRATE flag is set.
- uint32_t flags
  PLL configuration flags, Or'ed value of PLL_CONFIGFLAG_∗ definitions.
- ss_progmodfm_t ss_mf
  SS Programmable modulation frequency, only applicable when not using PLL_CONFIGFLAG_FORCENOFRACT flag.
- ss_progmoddp_t ss_mr
  SS Programmable frequency modulation depth, only applicable when not using PLL_CONFIGFLAG_FORCENOFRACT flag.
- ss_modwvctrl_t ss_mc
  SS Modulation waveform control, only applicable when not using PLL_CONFIGFLAG_FORCENOFRACT flag.
- bool mfDither
  false for fixed modulation frequency or true for dithering, only applicable when not using PLL_CONFIGFLAG_FORCENOFRACT flag

4.2.2 struct pll_setup_t

It can be populated with the PLL setup function. If powering up or waiting for PLL lock, the PLL input clock source should be configured prior to PLL setup.

Data Fields

- uint32_t pllctrl

PLL control register PLL0CTRL.
- uint32_t pllndec
  PLL NDEC register PLL0NDEC.
- uint32_t pllpdec
  PLL PDEC register PLL0PDEC.
- uint32_t pllmdec
  PLL MDEC registers PLL0PDEC.
- uint32_t pllsscg [2]
  PLL SSCTL registers PLL0SSCG.
- uint32_t pllRate
  Acutal PLL rate.
- uint32_t flags
  PLL setup flags, Or'ed value of PLL_SETUPFLAG_∗ definitions.

4.3 Macro Definition Documentation

4.3.1 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 3, 7))

4.3.2 #define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0

When set to 0, peripheral drivers will enable clock in initialize function and disable clock in de-initialize function. When set to 1, peripheral driver will not control the clock, application could control the clock out of the driver.

Note

All drivers share this feature switcher. If it is set to 1, application should handle clock enable and disable for all drivers.

4.3.3 #define CLOCK_USR_CFG_PLL_CONFIG_CACHE_COUNT 2U

Once define this MACRO to be non-zero value, CLOCK_PllGetConfig() function would cache the recent calulation and accelerate the execution to get the right settings.

4.3.4 #define ROM_CLOCKS

Value:

```
{                \
        kCLOCK_Rom \
    }
```

4.3.5 #define SRAM_CLOCKS

Value:

```
{                                   \
        kCLOCK_Sram1, kCLOCK_Sram2 \
    }
```

4.3.6 #define FLASH_CLOCKS

Value:

```
{                 \
        kCLOCK_Flash \
    }
```

4.3.7 #define FMC_CLOCKS

Value:

```
{               \
        kCLOCK_Fmc \
    }
```

4.3.8 #define INPUTMUX_CLOCKS

Value:

```
{                     \
        kCLOCK_InputMux0 \
    }
```

4.3.9 #define IOCON_CLOCKS

Value:

```
{                 \
        kCLOCK_Iocon \
    }
```

4.3.10 #define GPIO_CLOCKS

Value:

```
{                                   \
        kCLOCK_Gpio0, kCLOCK_Gpio1 \
    }
```

4.3.11 #define PINT_CLOCKS

Value:

```
{               \
        kCLOCK_Pint \
    }
```

4.3.12 #define GINT_CLOCKS

Value:

```
{                           \
        kCLOCK_Gint, kCLOCK_Gint \
    }
```

4.3.13 #define DMA_CLOCKS

Value:

```
{                           \
        kCLOCK_Dma0, kCLOCK_Dma1 \
    }
```

4.3.14 #define CRC_CLOCKS

Value:

```
{               \
        kCLOCK_Crc \
    }
```

4.3.15 #define WWDT_CLOCKS

Value:

```
{                \
        kCLOCK_Wwdt \
    }
```

4.3.16 #define RTC_CLOCKS

Value:

```
{               \
        kCLOCK_Rtc \
    }
```

4.3.17 #define MAILBOX_CLOCKS

Value:

```
{                   \
        kCLOCK_Mailbox \
    }
```

4.3.18 #define LPADC_CLOCKS

Value:

```
{                  \
        kCLOCK_Adc0  \
    }
```

4.3.19 #define MRT_CLOCKS

Value:

```
{               \
        kCLOCK_Mrt \
    }
```

4.3.20 #define OSTIMER_CLOCKS

Value:

```
{                   \
        kCLOCK_OsTimer0 \
    }
```

4.3.21 #define SCT_CLOCKS

Value:

```
{                \
        kCLOCK_Sct0 \
    }
```

4.3.22 #define MCAN_CLOCKS

Value:

```
{                \
        kCLOCK_Mcan \
    }
```

4.3.23 #define UTICK_CLOCKS

Value:

```
{                  \
        kCLOCK_Utick0 \
    }
```

4.3.24 #define FLEXCOMM_CLOCKS

Value:

```
{
          \
        kCLOCK_FlexComm0, kCLOCK_FlexComm1,
      kCLOCK_FlexComm2, kCLOCK_FlexComm3,
      kCLOCK_FlexComm4, kCLOCK_FlexComm5, \
            kCLOCK_FlexComm6, kCLOCK_FlexComm7,
      kCLOCK_Hs_Lspi                                                  \
    }
```

4.3.25 #define LPUART_CLOCKS

Value:

```
{
      \
        kCLOCK_MinUart0, kCLOCK_MinUart1,
      kCLOCK_MinUart2, kCLOCK_MinUart3, kCLOCK_MinUart4,
      kCLOCK_MinUart5, \
            kCLOCK_MinUart6, kCLOCK_MinUart7
                                                \
    }
```

4.3.26 #define BI2C_CLOCKS

Value:

```
{
              \
        kCLOCK_BI2c0, kCLOCK_BI2c1, kCLOCK_BI2c2,
      kCLOCK_BI2c3, kCLOCK_BI2c4, kCLOCK_BI2c5,
      kCLOCK_BI2c6, kCLOCK_BI2c7 \
    }
```

4.3.27 #define LPSPI_CLOCKS

Value:

```
{
              \
        kCLOCK_LSpi0, kCLOCK_LSpi1, kCLOCK_LSpi2,
      kCLOCK_LSpi3, kCLOCK_LSpi4, kCLOCK_LSpi5,
      kCLOCK_LSpi6, kCLOCK_LSpi7 \
    }
```

4.3.28 #define FLEXI2S_CLOCKS

Value:

```
{
      \
        kCLOCK_FlexI2s0, kCLOCK_FlexI2s1,
      kCLOCK_FlexI2s2, kCLOCK_FlexI2s3, kCLOCK_FlexI2s4,
      kCLOCK_FlexI2s5, \
            kCLOCK_FlexI2s6, kCLOCK_FlexI2s7
                                                \
    }
```

4.3.29 #define CTIMER_CLOCKS

Value:

```
{                                                                       \
        kCLOCK_Timer0, kCLOCK_Timer1,
      kCLOCK_Timer2, kCLOCK_Timer3, kCLOCK_Timer4 \
    }
```

4.3.30 #define USB1CLK_CLOCKS

Value:

```
{                   \
        kCLOCK_Usb1Clk \
    }
```

4.3.31 #define FREQME_CLOCKS

Value:

```
{                   \
        kCLOCK_Freqme \
    }
```

4.3.32 #define USBRAM_CLOCKS

Value:

```
{                   \
        kCLOCK_UsbRam1 \
    }
```

4.3.33 #define CDOG_CLOCKS

Value:

```
{                 \
        kCLOCK_Cdog \
    }
```

4.3.34 #define RNG_CLOCKS

Value:

```
{               \
        kCLOCK_Rng \
    }
```

4.3.35 #define USBHMR0_CLOCKS

Value:

```
{                   \
        kCLOCK_Usbhmr0 \
    }
```

4.3.36 #define USBHSL0_CLOCKS

Value:

```
{                   \
        kCLOCK_Usbhsl0 \
    }
```

4.3.37 #define HASHCRYPT_CLOCKS

Value:

```
{                     \
        kCLOCK_HashCrypt \
    }
```

4.3.38 #define PLULUT_CLOCKS

Value:

```
{                   \
        kCLOCK_PluLut \
    }
```

4.3.39 #define PUF_CLOCKS

Value:

```
{                \
        kCLOCK_Puf \
    }
```

4.3.40 #define CASPER_CLOCKS

Value:

```
{                   \
        kCLOCK_Casper \
    }
```

4.3.41 #define ANALOGCTRL_CLOCKS

Value:

```
{                      \
        kCLOCK_AnalogCtrl \
    }
```

4.3.42 #define HS_LSPI_CLOCKS

Value:

```
{                    \
        kCLOCK_Hs_Lspi \
    }
```

4.3.43 #define GPIO_SEC_CLOCKS

Value:

```
{                     \
        kCLOCK_Gpio_Sec \
    }
```

4.3.44 #define GPIO_SEC_INT_CLOCKS

Value:

```
{                       \
        kCLOCK_Gpio_Sec_Int \
    }
```

4.3.45 #define USBD_CLOCKS

Value:

```
{                                           \
        kCLOCK_Usbd0, kCLOCK_Usbh1, kCLOCK_Usbd1 \
    }
```

4.3.46 #define USBH_CLOCKS

Value:

```
{                   \
        kCLOCK_Usbh1 \
    }
```

4.3.47 #define CLK_GATE_REG_OFFSET_SHIFT 8U

4.3.48 #define BUS_CLK kCLOCK_BusClk

4.3.49 #define CLK_ATTACH_ID(*mux, sel, pos*) ((((uint32_t)(mux) << 0U) | (((uint32_t)(sel) + 1U) & 0xFU) << 8U) << ((uint32_t)(pos)*12U))

[4 bits for choice, 0 means invalid choice] [8 bits mux ID]*

4.3.50 #define PLL_CONFIGFLAG_USEINRATE (1U << 0U)

When the PLL_CONFIGFLAG_USEINRATE flag is selected, the 'InputRate' field in the configuration structure must be assigned with the expected PLL frequency. If the PLL_CONFIGFLAG_USEINRATE is not used, 'InputRate' is ignored in the configuration function and the driver will determine the PLL rate from the currently selected PLL source. This flag might be used to configure the PLL input clock more accurately when using the WDT oscillator or a more dyanmic CLKIN source.

When the PLL_CONFIGFLAG_FORCENOFRACT flag is selected, the PLL hardware for the automatic bandwidth selection, Spread Spectrum (SS) support, and fractional M-divider are not used.

Flag to use InputRate in PLL configuration structure for setup

4.3.51 #define PLL_SETUPFLAG_POWERUP (1U << 0U)

Setup will power on the PLL after setup

4.4 Enumeration Type Documentation

4.4.1 enum clock_ip_name_t

Enumerator

kCLOCK_IpInvalid Invalid Ip Name.
kCLOCK_Rom Clock gate name: Rom.
kCLOCK_Sram1 Clock gate name: Sram1.
kCLOCK_Sram2 Clock gate name: Sram2.
kCLOCK_Flash Clock gate name: Flash.
kCLOCK_Fmc Clock gate name: Fmc.
kCLOCK_InputMux Clock gate name: InputMux.
kCLOCK_Iocon Clock gate name: Iocon.
kCLOCK_Gpio0 Clock gate name: Gpio0.
kCLOCK_Gpio1 Clock gate name: Gpio1.
kCLOCK_Pint Clock gate name: Pint.
kCLOCK_Gint Clock gate name: Gint.
kCLOCK_Dma0 Clock gate name: Dma0.
kCLOCK_Crc Clock gate name: Crc.
kCLOCK_Wwdt Clock gate name: Wwdt.
kCLOCK_Rtc Clock gate name: Rtc.
kCLOCK_Mailbox Clock gate name: Mailbox.
kCLOCK_Adc0 Clock gate name: Adc0.
kCLOCK_Mrt Clock gate name: Mrt.
kCLOCK_OsTimer0 Clock gate name: OsTimer0.
kCLOCK_Sct0 Clock gate name: Sct0.
kCLOCK_Mcan Clock gate name: Mcan.
kCLOCK_Utick0 Clock gate name: Utick0.
kCLOCK_FlexComm0 Clock gate name: FlexComm0.
kCLOCK_FlexComm1 Clock gate name: FlexComm1.
kCLOCK_FlexComm2 Clock gate name: FlexComm2.
kCLOCK_FlexComm3 Clock gate name: FlexComm3.
kCLOCK_FlexComm4 Clock gate name: FlexComm4.
kCLOCK_FlexComm5 Clock gate name: FlexComm5.
kCLOCK_FlexComm6 Clock gate name: FlexComm6.

kCLOCK_FlexComm7 Clock gate name: FlexComm7.
kCLOCK_MinUart0 Clock gate name: MinUart0.
kCLOCK_MinUart1 Clock gate name: MinUart1.
kCLOCK_MinUart2 Clock gate name: MinUart2.
kCLOCK_MinUart3 Clock gate name: MinUart3.
kCLOCK_MinUart4 Clock gate name: MinUart4.
kCLOCK_MinUart5 Clock gate name: MinUart5.
kCLOCK_MinUart6 Clock gate name: MinUart6.
kCLOCK_MinUart7 Clock gate name: MinUart7.
kCLOCK_LSpi0 Clock gate name: LSpi0.
kCLOCK_LSpi1 Clock gate name: LSpi1.
kCLOCK_LSpi2 Clock gate name: LSpi2.
kCLOCK_LSpi3 Clock gate name: LSpi3.
kCLOCK_LSpi4 Clock gate name: LSpi4.
kCLOCK_LSpi5 Clock gate name: LSpi5.
kCLOCK_LSpi6 Clock gate name: LSpi6.
kCLOCK_LSpi7 Clock gate name: LSpi7.
kCLOCK_BI2c0 Clock gate name: BI2c0.
kCLOCK_BI2c1 Clock gate name: BI2c1.
kCLOCK_BI2c2 Clock gate name: BI2c2.
kCLOCK_BI2c3 Clock gate name: BI2c3.
kCLOCK_BI2c4 Clock gate name: BI2c4.
kCLOCK_BI2c5 Clock gate name: BI2c5.
kCLOCK_BI2c6 Clock gate name: BI2c6.
kCLOCK_BI2c7 Clock gate name: BI2c7.
kCLOCK_FlexI2s0 Clock gate name: FlexI2s0.
kCLOCK_FlexI2s1 Clock gate name: FlexI2s1.
kCLOCK_FlexI2s2 Clock gate name: FlexI2s2.
kCLOCK_FlexI2s3 Clock gate name: FlexI2s3.
kCLOCK_FlexI2s4 Clock gate name: FlexI2s4.
kCLOCK_FlexI2s5 Clock gate name: FlexI2s5.
kCLOCK_FlexI2s6 Clock gate name: FlexI2s6.
kCLOCK_FlexI2s7 Clock gate name: FlexI2s7.
kCLOCK_Timer2 Clock gate name: Timer2.
kCLOCK_Usbd0 Clock gate name: Usbd0.
kCLOCK_Timer0 Clock gate name: Timer0.
kCLOCK_Timer1 Clock gate name: Timer1.
kCLOCK_Ezha Clock gate name: Ezha.
kCLOCK_Ezhb Clock gate name: Ezhb.
kCLOCK_Dma1 Clock gate name: Dma1.
kCLOCK_Comp Clock gate name: Comp.
kCLOCK_Usbh1 Clock gate name: Usbh1.
kCLOCK_Usbd1 Clock gate name: Usbd1.
kCLOCK_UsbRam1 Clock gate name: UsbRam1.
kCLOCK_Usb1Clk Clock gate name: Usb1Clk.

kCLOCK_Freqme Clock gate name: Freqme.
kCLOCK_Cdog Clock gate name: Cdog.
kCLOCK_Rng Clock gate name: Rng.
kCLOCK_Sysctl Clock gate name: Sysctl.
kCLOCK_Usbhmr0 Clock gate name: Usbhmr0.
kCLOCK_Usbhsl0 Clock gate name: Usbhsl0.
kCLOCK_HashCrypt Clock gate name: HashCrypt.
kCLOCK_PluLut Clock gate name: PluLut.
kCLOCK_Timer3 Clock gate name: Timer3.
kCLOCK_Timer4 Clock gate name: Timer4.
kCLOCK_Puf Clock gate name: Puf.
kCLOCK_Casper Clock gate name: Casper.
kCLOCK_AnalogCtrl Clock gate name: AnalogCtrl.
kCLOCK_Hs_Lspi Clock gate name: Lspi.
kCLOCK_Gpio_Sec Clock gate name: GPIO Sec.
kCLOCK_Gpio_Sec_Int Clock gate name: GPIO SEC Int.

4.4.2 enum clock_name_t

Enumerator

kCLOCK_CoreSysClk Core/system clock (aka MAIN_CLK)
kCLOCK_BusClk Bus clock (AHB clock)
kCLOCK_ClockOut CLOCKOUT.
kCLOCK_FroHf FRO48/96.
kCLOCK_Pll1Out PLL1 Output.
kCLOCK_Mclk MCLK.
kCLOCK_Fro12M FRO12M.
kCLOCK_Fro1M FRO1M.
kCLOCK_ExtClk External Clock.
kCLOCK_Pll0Out PLL0 Output.
kCLOCK_FlexI2S FlexI2S clock.

4.4.3 enum clock_attach_id_t

Enumerator

kFRO12M_to_MAIN_CLK Attach FRO12M to MAIN_CLK.
kEXT_CLK_to_MAIN_CLK Attach EXT_CLK to MAIN_CLK.
kFRO1M_to_MAIN_CLK Attach FRO1M to MAIN_CLK.
kFRO_HF_to_MAIN_CLK Attach FRO_HF to MAIN_CLK.
kPLL0_to_MAIN_CLK Attach PLL0 to MAIN_CLK.
kPLL1_to_MAIN_CLK Attach PLL1 to MAIN_CLK.

kOSC32K_to_MAIN_CLK Attach OSC32K to MAIN_CLK.
kMAIN_CLK_to_CLKOUT Attach MAIN_CLK to CLKOUT.
kPLL0_to_CLKOUT Attach PLL0 to CLKOUT.
kEXT_CLK_to_CLKOUT Attach EXT_CLK to CLKOUT.
kFRO_HF_to_CLKOUT Attach FRO_HF to CLKOUT.
kFRO1M_to_CLKOUT Attach FRO1M to CLKOUT.
kPLL1_to_CLKOUT Attach PLL1 to CLKOUT.
kOSC32K_to_CLKOUT Attach OSC32K to CLKOUT.
kNONE_to_SYS_CLKOUT Attach NONE to SYS_CLKOUT.
kFRO12M_to_PLL0 Attach FRO12M to PLL0.
kEXT_CLK_to_PLL0 Attach EXT_CLK to PLL0.
kFRO1M_to_PLL0 Attach FRO1M to PLL0.
kOSC32K_to_PLL0 Attach OSC32K to PLL0.
kNONE_to_PLL0 Attach NONE to PLL0.
kMCAN_DIV_to_MCAN Attach MCAN_DIV to MCAN.
kFRO1M_to_MCAN Attach FRO1M to MCAN.
kOSC32K_to_MCAN Attach OSC32K to MCAN.
kNONE_to_MCAN Attach NONE to MCAN.
kMAIN_CLK_to_ADC_CLK Attach MAIN_CLK to ADC_CLK.
kPLL0_to_ADC_CLK Attach PLL0 to ADC_CLK.
kFRO_HF_to_ADC_CLK Attach FRO_HF to ADC_CLK.
kEXT_CLK_to_ADC_CLK Attach EXT_CLK to ADC_CLK.
kNONE_to_ADC_CLK Attach NONE to ADC_CLK.
kMAIN_CLK_to_USB0_CLK Attach MAIN_CLK to USB0_CLK.
kPLL0_to_USB0_CLK Attach PLL0 to USB0_CLK.
kFRO_HF_to_USB0_CLK Attach FRO_HF to USB0_CLK.
kPLL1_to_USB0_CLK Attach PLL1 to USB0_CLK.
kNONE_to_USB0_CLK Attach NONE to USB0_CLK.
kOSC32K_to_CLK32K Attach OSC32K to CLK32K.
kFRO1MDIV_to_CLK32K Attach FRO1MDIV to CLK32K.
kNONE_to_CLK32K Attach NONE to CLK32K.
kMAIN_CLK_to_FLEXCOMM0 Attach MAIN_CLK to FLEXCOMM0.
kPLL0_DIV_to_FLEXCOMM0 Attach PLL0_DIV to FLEXCOMM0.
kFRO12M_to_FLEXCOMM0 Attach FRO12M to FLEXCOMM0.
kFRO_HF_DIV_to_FLEXCOMM0 Attach FRO_HF_DIV to FLEXCOMM0.
kFRO1M_to_FLEXCOMM0 Attach FRO1M to FLEXCOMM0.
kMCLK_to_FLEXCOMM0 Attach MCLK to FLEXCOMM0.
kOSC32K_to_FLEXCOMM0 Attach OSC32K to FLEXCOMM0.
kNONE_to_FLEXCOMM0 Attach NONE to FLEXCOMM0.
kMAIN_CLK_to_FLEXCOMM1 Attach MAIN_CLK to FLEXCOMM1.
kPLL0_DIV_to_FLEXCOMM1 Attach PLL0_DIV to FLEXCOMM1.
kFRO12M_to_FLEXCOMM1 Attach FRO12M to FLEXCOMM1.
kFRO_HF_DIV_to_FLEXCOMM1 Attach FRO_HF_DIV to FLEXCOMM1.
kFRO1M_to_FLEXCOMM1 Attach FRO1M to FLEXCOMM1.
kMCLK_to_FLEXCOMM1 Attach MCLK to FLEXCOMM1.

kOSC32K_to_FLEXCOMM1 Attach OSC32K to FLEXCOMM1.
kNONE_to_FLEXCOMM1 Attach NONE to FLEXCOMM1.
kMAIN_CLK_to_FLEXCOMM2 Attach MAIN_CLK to FLEXCOMM2.
kPLL0_DIV_to_FLEXCOMM2 Attach PLL0_DIV to FLEXCOMM2.
kFRO12M_to_FLEXCOMM2 Attach FRO12M to FLEXCOMM2.
kFRO_HF_DIV_to_FLEXCOMM2 Attach FRO_HF_DIV to FLEXCOMM2.
kFRO1M_to_FLEXCOMM2 Attach FRO1M to FLEXCOMM2.
kMCLK_to_FLEXCOMM2 Attach MCLK to FLEXCOMM2.
kOSC32K_to_FLEXCOMM2 Attach OSC32K to FLEXCOMM2.
kNONE_to_FLEXCOMM2 Attach NONE to FLEXCOMM2.
kMAIN_CLK_to_FLEXCOMM3 Attach MAIN_CLK to FLEXCOMM3.
kPLL0_DIV_to_FLEXCOMM3 Attach PLL0_DIV to FLEXCOMM3.
kFRO12M_to_FLEXCOMM3 Attach FRO12M to FLEXCOMM3.
kFRO_HF_DIV_to_FLEXCOMM3 Attach FRO_HF_DIV to FLEXCOMM3.
kFRO1M_to_FLEXCOMM3 Attach FRO1M to FLEXCOMM3.
kMCLK_to_FLEXCOMM3 Attach MCLK to FLEXCOMM3.
kOSC32K_to_FLEXCOMM3 Attach OSC32K to FLEXCOMM3.
kNONE_to_FLEXCOMM3 Attach NONE to FLEXCOMM3.
kMAIN_CLK_to_FLEXCOMM4 Attach MAIN_CLK to FLEXCOMM4.
kPLL0_DIV_to_FLEXCOMM4 Attach PLL0_DIV to FLEXCOMM4.
kFRO12M_to_FLEXCOMM4 Attach FRO12M to FLEXCOMM4.
kFRO_HF_DIV_to_FLEXCOMM4 Attach FRO_HF_DIV to FLEXCOMM4.
kFRO1M_to_FLEXCOMM4 Attach FRO1M to FLEXCOMM4.
kMCLK_to_FLEXCOMM4 Attach MCLK to FLEXCOMM4.
kOSC32K_to_FLEXCOMM4 Attach OSC32K to FLEXCOMM4.
kNONE_to_FLEXCOMM4 Attach NONE to FLEXCOMM4.
kMAIN_CLK_to_FLEXCOMM5 Attach MAIN_CLK to FLEXCOMM5.
kPLL0_DIV_to_FLEXCOMM5 Attach PLL0_DIV to FLEXCOMM5.
kFRO12M_to_FLEXCOMM5 Attach FRO12M to FLEXCOMM5.
kFRO_HF_DIV_to_FLEXCOMM5 Attach FRO_HF_DIV to FLEXCOMM5.
kFRO1M_to_FLEXCOMM5 Attach FRO1M to FLEXCOMM5.
kMCLK_to_FLEXCOMM5 Attach MCLK to FLEXCOMM5.
kOSC32K_to_FLEXCOMM5 Attach OSC32K to FLEXCOMM5.
kNONE_to_FLEXCOMM5 Attach NONE to FLEXCOMM5.
kMAIN_CLK_to_FLEXCOMM6 Attach MAIN_CLK to FLEXCOMM6.
kPLL0_DIV_to_FLEXCOMM6 Attach PLL0_DIV to FLEXCOMM6.
kFRO12M_to_FLEXCOMM6 Attach FRO12M to FLEXCOMM6.
kFRO_HF_DIV_to_FLEXCOMM6 Attach FRO_HF_DIV to FLEXCOMM6.
kFRO1M_to_FLEXCOMM6 Attach FRO1M to FLEXCOMM6.
kMCLK_to_FLEXCOMM6 Attach MCLK to FLEXCOMM6.
kOSC32K_to_FLEXCOMM6 Attach OSC32K to FLEXCOMM6.
kNONE_to_FLEXCOMM6 Attach NONE to FLEXCOMM6.
kMAIN_CLK_to_FLEXCOMM7 Attach MAIN_CLK to FLEXCOMM7.
kPLL0_DIV_to_FLEXCOMM7 Attach PLL0_DIV to FLEXCOMM7.
kFRO12M_to_FLEXCOMM7 Attach FRO12M to FLEXCOMM7.

kFRO_HF_DIV_to_FLEXCOMM7 Attach FRO_HF_DIV to FLEXCOMM7.
kFRO1M_to_FLEXCOMM7 Attach FRO1M to FLEXCOMM7.
kMCLK_to_FLEXCOMM7 Attach MCLK to FLEXCOMM7.
kOSC32K_to_FLEXCOMM7 Attach OSC32K to FLEXCOMM7.
kNONE_to_FLEXCOMM7 Attach NONE to FLEXCOMM7.
kMAIN_CLK_to_HSLSPI Attach MAIN_CLK to HSLSPI.
kPLL0_DIV_to_HSLSPI Attach PLL0_DIV to HSLSPI.
kFRO12M_to_HSLSPI Attach FRO12M to HSLSPI.
kFRO_HF_DIV_to_HSLSPI Attach FRO_HF_DIV to HSLSPI.
kFRO1M_to_HSLSPI Attach FRO1M to HSLSPI.
kOSC32K_to_HSLSPI Attach OSC32K to HSLSPI.
kNONE_to_HSLSPI Attach NONE to HSLSPI.
kFRO_HF_to_MCLK Attach FRO_HF to MCLK.
kPLL0_to_MCLK Attach PLL0 to MCLK.
kNONE_to_MCLK Attach NONE to MCLK.
kMAIN_CLK_to_SCT_CLK Attach MAIN_CLK to SCT_CLK.
kPLL0_to_SCT_CLK Attach PLL0 to SCT_CLK.
kEXT_CLK_to_SCT_CLK Attach EXT_CLK to SCT_CLK.
kFRO_HF_to_SCT_CLK Attach FRO_HF to SCT_CLK.
kMCLK_to_SCT_CLK Attach MCLK to SCT_CLK.
kNONE_to_SCT_CLK Attach NONE to SCT_CLK.
kFRO32K_to_OSC32K Attach FRO32K to OSC32K.
kXTAL32K_to_OSC32K Attach XTAL32K to OSC32K.
kOSC32K_to_OSTIMER Attach OSC32K to OSTIMER.
kFRO1M_to_OSTIMER Attach FRO1M to OSTIMER.
kMAIN_CLK_to_OSTIMER Attach MAIN_CLK to OSTIMER.
kTRACE_DIV_to_TRACE Attach TRACE_DIV to TRACE.
kFRO1M_to_TRACE Attach FRO1M to TRACE.
kOSC32K_to_TRACE Attach OSC32K to TRACE.
kNONE_to_TRACE Attach NONE to TRACE.
kSYSTICK_DIV0_to_SYSTICK0 Attach SYSTICK_DIV0 to SYSTICK0.
kFRO1M_to_SYSTICK0 Attach FRO1M to SYSTICK0.
kOSC32K_to_SYSTICK0 Attach OSC32K to SYSTICK0.
kNONE_to_SYSTICK0 Attach NONE to SYSTICK0.
kFRO12M_to_PLL1 Attach FRO12M to PLL1.
kEXT_CLK_to_PLL1 Attach EXT_CLK to PLL1.
kFRO1M_to_PLL1 Attach FRO1M to PLL1.
kOSC32K_to_PLL1 Attach OSC32K to PLL1.
kNONE_to_PLL1 Attach NONE to PLL1.
kMAIN_CLK_to_CTIMER0 Attach MAIN_CLK to CTIMER0.
kPLL0_to_CTIMER0 Attach PLL0 to CTIMER0.
kFRO_HF_to_CTIMER0 Attach FRO_HF to CTIMER0.
kFRO1M_to_CTIMER0 Attach FRO1M to CTIMER0.
kMCLK_to_CTIMER0 Attach MCLK to CTIMER0.
kOSC32K_to_CTIMER0 Attach OSC32K to CTIMER0.

kNONE_to_CTIMER0 Attach NONE to CTIMER0.
kMAIN_CLK_to_CTIMER1 Attach MAIN_CLK to CTIMER1.
kPLL0_to_CTIMER1 Attach PLL0 to CTIMER1.
kFRO_HF_to_CTIMER1 Attach FRO_HF to CTIMER1.
kFRO1M_to_CTIMER1 Attach FRO1M to CTIMER1.
kMCLK_to_CTIMER1 Attach MCLK to CTIMER1.
kOSC32K_to_CTIMER1 Attach OSC32K to CTIMER1.
kNONE_to_CTIMER1 Attach NONE to CTIMER1.
kMAIN_CLK_to_CTIMER2 Attach MAIN_CLK to CTIMER2.
kPLL0_to_CTIMER2 Attach PLL0 to CTIMER2.
kFRO_HF_to_CTIMER2 Attach FRO_HF to CTIMER2.
kFRO1M_to_CTIMER2 Attach FRO1M to CTIMER2.
kMCLK_to_CTIMER2 Attach MCLK to CTIMER2.
kOSC32K_to_CTIMER2 Attach OSC32K to CTIMER2.
kNONE_to_CTIMER2 Attach NONE to CTIMER2.
kMAIN_CLK_to_CTIMER3 Attach MAIN_CLK to CTIMER3.
kPLL0_to_CTIMER3 Attach PLL0 to CTIMER3.
kFRO_HF_to_CTIMER3 Attach FRO_HF to CTIMER3.
kFRO1M_to_CTIMER3 Attach FRO1M to CTIMER3.
kMCLK_to_CTIMER3 Attach MCLK to CTIMER3.
kOSC32K_to_CTIMER3 Attach OSC32K to CTIMER3.
kNONE_to_CTIMER3 Attach NONE to CTIMER3.
kMAIN_CLK_to_CTIMER4 Attach MAIN_CLK to CTIMER4.
kPLL0_to_CTIMER4 Attach PLL0 to CTIMER4.
kFRO_HF_to_CTIMER4 Attach FRO_HF to CTIMER4.
kFRO1M_to_CTIMER4 Attach FRO1M to CTIMER4.
kMCLK_to_CTIMER4 Attach MCLK to CTIMER4.
kOSC32K_to_CTIMER4 Attach OSC32K to CTIMER4.
kNONE_to_CTIMER4 Attach NONE to CTIMER4.
kNONE_to_NONE Attach NONE to NONE.

4.4.4 enum clock_div_name_t

Enumerator

kCLOCK_DivSystickClk0 Systick Clk0 Divider.
kCLOCK_DivArmTrClkDiv Arm Tr Clk Div Divider.
kCLOCK_DivCanClk Can Clock Divider.
kCLOCK_DivFlexFrg0 Flex Frg0 Divider.
kCLOCK_DivFlexFrg1 Flex Frg1 Divider.
kCLOCK_DivFlexFrg2 Flex Frg2 Divider.
kCLOCK_DivFlexFrg3 Flex Frg3 Divider.
kCLOCK_DivFlexFrg4 Flex Frg4 Divider.
kCLOCK_DivFlexFrg5 Flex Frg5 Divider.

kCLOCK_DivFlexFrg6 Flex Frg6 Divider.
kCLOCK_DivFlexFrg7 Flex Frg7 Divider.
kCLOCK_DivAhbClk Ahb Clock Divider.
kCLOCK_DivClkOut Clk Out Divider.
kCLOCK_DivFrohfClk Frohf Clock Divider.
kCLOCK_DivWdtClk Wdt Clock Divider.
kCLOCK_DivAdcAsyncClk Adc Async Clock Divider.
kCLOCK_DivUsb0Clk Usb0 Clock Divider.
kCLOCK_DivFro1mClk Fro1m Clock Divider.
kCLOCK_DivMClk I2S MCLK Clock Divider.
kCLOCK_DivSctClk Sct Clock Divider.
kCLOCK_DivPll0Clk PLL clock divider.

4.4.5 enum ss_progmodfm_t

Enumerator

kSS_MF_512 Nss = 512 (fm ? 3.9 - 7.8 kHz)
kSS_MF_384 Nss ?= 384 (fm ? 5.2 - 10.4 kHz)
kSS_MF_256 Nss = 256 (fm ? 7.8 - 15.6 kHz)
kSS_MF_128 Nss = 128 (fm ? 15.6 - 31.3 kHz)
kSS_MF_64 Nss = 64 (fm ? 32.3 - 64.5 kHz)
kSS_MF_32 Nss = 32 (fm ? 62.5- 125 kHz)
kSS_MF_24 Nss ?= 24 (fm ? 83.3- 166.6 kHz)
kSS_MF_16 Nss = 16 (fm ? 125- 250 kHz)

4.4.6 enum ss_progmoddp_t

Enumerator

kSS_MR_K0 k = 0 (no spread spectrum)
kSS_MR_K1 k = 1
kSS_MR_K1_5 k = 1.5
kSS_MR_K2 k = 2
kSS_MR_K3 k = 3
kSS_MR_K4 k = 4
kSS_MR_K6 k = 6
kSS_MR_K8 k = 8

4.4.7 enum ss_modwvctrl_t

Compensation for low pass filtering of the PLL to get a triangular modulation at the output of the PLL, giving a flat frequency spectrum.

Enumerator

kSS_MC_NOC no compensation
kSS_MC_RECC recommended setting
kSS_MC_MAXC max. compensation

4.4.8 enum pll_error_t

Enumerator

kStatus_PLL_Success PLL operation was successful.
kStatus_PLL_OutputTooLow PLL output rate request was too low.
kStatus_PLL_OutputTooHigh PLL output rate request was too high.
kStatus_PLL_InputTooLow PLL input rate is too low.
kStatus_PLL_InputTooHigh PLL input rate is too high.
kStatus_PLL_OutsideIntLimit Requested output rate isn't possible.
kStatus_PLL_CCOTooLow Requested CCO rate isn't possible.
kStatus_PLL_CCOTooHigh Requested CCO rate isn't possible.

4.4.9 enum clock_usbfs_src_t

Enumerator

kCLOCK_UsbfsSrcFro Use FRO 96 MHz.
kCLOCK_UsbfsSrcPll0 Use PLL0 output.
kCLOCK_UsbfsSrcMainClock Use Main clock.
kCLOCK_UsbfsSrcPll1 Use PLL1 clock.
kCLOCK_UsbfsSrcNone this may be selected in order to reduce power when no output is needed.

4.4.10 enum clock_usbhs_src_t

Enumerator

kCLOCK_UsbSrcUnused Used when the function does not care the clock source.

4.4.11 enum clock_usb_phy_src_t

Enumerator

kCLOCK_UsbPhySrcExt Use external crystal.

4.5 Function Documentation

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

Parameters

clk	: Clock to be enabled.

Returns

Nothing

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

Parameters

clk	: Clock to be Disabled.

Returns

Nothing

4.5.3 status_t CLOCK_SetupFROClocking (uint32_t *iFreq*)

Parameters

iFreq	: Desired frequency (must be one of CLK_FRO_12MHZ or CLK_FRO_48MHZ or CLK_FRO_96MHZ)

Returns

returns success or fail status.

4.5.4 void CLOCK_SetFLASHAccessCyclesForFreq (uint32_t *system_freq_hz*)

Parameters

system_freq_hz	: Input frequency

Returns

Nothing

4.5.5 status_t CLOCK_SetupExtClocking (uint32_t *iFreq*)

Parameters

iFreq	: Desired frequency (must be equal to exact rate in Hz)

Returns

returns success or fail status.

4.5.6 status_t CLOCK_SetupI2SMClkClocking (uint32_t *iFreq*)

Parameters

iFreq	: Desired frequency (must be equal to exact rate in Hz)

Returns

returns success or fail status.

4.5.7 status_t CLOCK_SetupPLUClkInClocking (uint32_t *iFreq*)

Parameters

iFreq	: Desired frequency (must be equal to exact rate in Hz)

Returns

returns success or fail status.

4.5.8 void CLOCK_AttachClk (clock_attach_id_t *connection*)

Parameters

connection	: Clock to be configured.

Returns

Nothing

4.5.9 clock_attach_id_t CLOCK_GetClockAttachId (clock_attach_id_t *attachId*)

Parameters

attachId	: Clock attach id to get.

Returns

Clock source value.

4.5.10 void CLOCK_SetClkDiv (clock_div_name_t *div_name,* uint32_t *divided_by_value,* bool *reset*)

Parameters

div_name	: Clock divider name
divided_by_-value,:	Value to be divided
reset	: Whether to reset the divider counter.

Returns

Nothing

4.5.11 void CLOCK_SetRtc1khzClkDiv (uint32_t *divided_by_value*)

Parameters

divided_by_value,:	Value to be divided

Returns

Nothing

4.5.12 void CLOCK_SetRtc1hzClkDiv (uint32_t *divided_by_value*)

Parameters

divided_by_value,:	Value to be divided

Returns

Nothing

4.5.13 uint32_t CLOCK_SetFlexCommClock (uint32_t *id,* uint32_t *freq*)

Parameters

id	: flexcomm instance id
freq	: output frequency

Returns

0 : the frequency range is out of range. 1 : switch successfully.

4.5.14 uint32_t CLOCK_GetFlexCommInputClock (uint32_t *id*)

Parameters

id	: flexcomm instance id

Returns

Frequency value

4.5.15 uint32_t CLOCK_GetFreq (clock_name_t *clockName*)

Returns

Frequency of selected clock

4.5.16 uint32_t CLOCK_GetFro12MFreq (void)

Returns

Frequency of FRO 12MHz

4.5.17 uint32_t CLOCK_GetFro1MFreq (void)

Returns

Frequency of FRO 1MHz

4.5.18 uint32_t CLOCK_GetClockOutClkFreq (void)

Returns

Frequency of ClockOut

4.5.19 uint32_t CLOCK_GetMCanClkFreq (void)

Returns

Frequency of Can.

4.5.20 uint32_t CLOCK_GetAdcClkFreq (void)

Returns

Frequency of Adc.

4.5.21 uint32_t CLOCK_GetUsb0ClkFreq (void)

Returns

Frequency of Usb0 Clock.

4.5.22 uint32_t CLOCK_GetUsb1ClkFreq (void)

Returns

Frequency of Usb1 Clock.

4.5.23 uint32_t CLOCK_GetMclkClkFreq (void)

Returns

Frequency of MClk Clock.

4.5.24 uint32_t CLOCK_GetSctClkFreq (void)

Returns

Frequency of SCTimer Clock.

4.5.25 uint32_t CLOCK_GetExtClkFreq (void)

Returns

Frequency of External Clock. If no external clock is used returns 0.

4.5.26 uint32_t CLOCK_GetWdtClkFreq (void)

Returns

Frequency of Watchdog

4.5.27 uint32_t CLOCK_GetFroHfFreq (void)

Returns

Frequency of High-Freq output of FRO

4.5.28 uint32_t CLOCK_GetPll0OutFreq (void)

Returns

Frequency of PLL

4.5.29 uint32_t CLOCK_GetPll1OutFreq (void)

Returns

Frequency of PLL

4.5.30 uint32_t CLOCK_GetOsc32KFreq (void)

Returns

Frequency of 32kHz osc

4.5.31 uint32_t CLOCK_GetCoreSysClkFreq (void)

Returns

Frequency of Core System

4.5.32 uint32_t CLOCK_GetI2SMClkFreq (void)

Returns

Frequency of I2S MCLK Clock

4.5.33 uint32_t CLOCK_GetPLUClkInFreq (void)

Returns

Frequency of PLU CLKIN Clock

4.5.34 uint32_t CLOCK_GetFlexCommClkFreq (uint32_t *id*)

Returns

Frequency of FlexComm Clock

4.5.35 uint32_t CLOCK_GetHsLspiClkFreq (void)

Returns

Frequency of High speed SPI Clock

4.5.36 uint32_t CLOCK_GetCTimerClkFreq (uint32_t *id*)

Returns

Frequency of CTimer functional Clock

4.5.37 uint32_t CLOCK_GetSystickClkFreq (uint32_t *id*)

Returns

Frequency of Systick Clock

4.5.38 uint32_t CLOCK_GetPLL0InClockRate (void)

Returns

PLL0 input clock rate

4.5.39 uint32_t CLOCK_GetPLL1InClockRate (void)

Returns

PLL1 input clock rate

4.5.40 uint32_t CLOCK_GetPLL0OutClockRate (bool *recompute*)

Parameters

recompute	: Forces a PLL rate recomputation if true

Returns

PLL0 output clock rate

Note

The PLL rate is cached in the driver in a variable as the rate computation function can take some time to perform. It is recommended to use 'false' with the 'recompute' parameter.

4.5.41 uint32_t CLOCK_GetPLL1OutClockRate (bool *recompute*)

Parameters

recompute	: Forces a PLL rate recomputation if true

Returns

PLL1 output clock rate

Note

The PLL rate is cached in the driver in a variable as the rate computation function can take some time to perform. It is recommended to use 'false' with the 'recompute' parameter.

4.5.42 __STATIC_INLINE void CLOCK_SetBypassPLL0 (bool *bypass*)

bypass : true to bypass PLL0 (PLL0 output = PLL0 input, false to disable bypass

Returns

PLL0 output clock rate

4.5.43 __STATIC_INLINE void CLOCK_SetBypassPLL1 (bool *bypass*)

bypass : true to bypass PLL1 (PLL1 output = PLL1 input, false to disable bypass

Returns

PLL1 output clock rate

4.5.44 __STATIC_INLINE bool CLOCK_IsPLL0Locked (void)

Returns

true if the PLL is locked, false if not locked

4.5.45 __STATIC_INLINE bool CLOCK_IsPLL1Locked (void)

Returns

true if the PLL1 is locked, false if not locked

4.5.46 void CLOCK_SetStoredPLL0ClockRate (uint32_t *rate*)

Parameters

rate,:	Current rate of the PLL0

Returns

Nothing

4.5.47 uint32_t CLOCK_GetPLL0OutFromSetup (pll_setup_t * *pSetup*)

Parameters

pSetup	: Pointer to a PLL setup structure

Returns

System PLL output clock rate the setup structure will generate

4.5.48 uint32_t CLOCK_GetPLL1OutFromSetup (pll_setup_t * *pSetup*)

Parameters

pSetup	: Pointer to a PLL setup structure

Returns

PLL0 output clock rate the setup structure will generate

4.5.49 pll_error_t CLOCK_SetupPLL0Data (pll_config_t * *pControl,* pll_setup_t * *pSetup*)

Parameters

pControl	: Pointer to populated PLL control structure to generate setup with
pSetup	: Pointer to PLL setup structure to be filled

Returns

PLL_ERROR_SUCCESS on success, or PLL setup error code

Note

Actual frequency for setup may vary from the desired frequency based on the accuracy of input clocks, rounding, non-fractional PLL mode, etc.

4.5.50 pll_error_t CLOCK_SetupPLL0Prec (pll_setup_t * *pSetup,* uint32_t *flagcfg*)

Parameters

pSetup	: Pointer to populated PLL setup structure
flagcfg	: Flag configuration for PLL config structure

Returns

PLL_ERROR_SUCCESS on success, or PLL setup error code

Note

This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the PLL, wait for PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main systen clock) that may use the PLL, so these should be setup prior to and after exiting the function.

4.5.51 pll_error_t CLOCK_SetPLL0Freq (const pll_setup_t * *pSetup*)

Parameters

pSetup	: Pointer to populated PLL setup structure

Returns

kStatus_PLL_Success on success, or PLL setup error code

Note

This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the PLL, wait for PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main systen clock) that may use the PLL, so these should be setup prior to and after exiting the function.

4.5.52 pll_error_t CLOCK_SetPLL1Freq (const pll_setup_t * *pSetup*)

Parameters

pSetup	: Pointer to populated PLL setup structure

Returns

kStatus_PLL_Success on success, or PLL setup error code

Note

This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the PLL, wait for PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main systen clock) that may use the PLL, so these should be setup prior to and after exiting the function.

4.5.53 void CLOCK_SetupPLL0Mult (uint32_t *multiply_by,* uint32_t *input_freq*)

Parameters

multiply_by	: multiplier
input_freq	: Clock input frequency of the PLL

Returns

Nothing

Note

Unlike the Chip_Clock_SetupSystemPLLPrec() function, this function does not disable or enable PLL power, wait for PLL lock, or adjust system voltages. These must be done in the application. The function will not alter any source clocks (ie, main systen clock) that may use the PLL, so these should be setup prior to and after exiting the function.

4.5.54 static void CLOCK_DisableUsbDevicefs0Clock (clock_ip_name_t *clk*) `[inline], [static]`

Disable USB clock.

4.5.55 bool CLOCK_EnableUsbfs0DeviceClock (clock_usbfs_src_t *src,* uint32_t *freq*)

Parameters

src	: clock source
freq,:	clock frequency Enable USB Device Full Speed clock.

4.5.56 bool CLOCK_EnableUsbfs0HostClock (clock_usbfs_src_t *src*, uint32_t *freq*)

Parameters

src	: clock source
freq,:	clock frequency Enable USB HOST Full Speed clock.

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

Enable USB phy clock.

4.5.58 bool CLOCK_EnableUsbhs0DeviceClock (clock_usbhs_src_t *src*, uint32_t *freq*)

Enable USB Device High Speed clock.

4.5.59 bool CLOCK_EnableUsbhs0HostClock (clock_usbhs_src_t *src*, uint32_t *freq*)

Enable USB HOST High Speed clock.

4.5.60 void CLOCK_EnableOstimer32kClock (void)

Returns

Nothing

Chapter 5
Power Driver

5.1 Overview

Power driver provides APIs to control peripherals power and control the system power mode.

Macros

- #define LOWPOWER_SRAMRETCTRL_RETEN_RAMX0 (1UL << 0)
  SRAM instances retention control during low power modes.
- #define LOWPOWER_SRAMRETCTRL_RETEN_RAMX1 (1UL << 1)
  Enable SRAMX_1 retention when entering in Low power modes.
- #define LOWPOWER_SRAMRETCTRL_RETEN_RAMX2 (1UL << 2)
  Enable SRAMX_2 retention when entering in Low power modes.
- #define LOWPOWER_SRAMRETCTRL_RETEN_RAMX3 (1UL << 3)
  Enable SRAMX_3 retention when entering in Low power modes.
- #define LOWPOWER_SRAMRETCTRL_RETEN_RAM00 (1UL << 4)
  Enable SRAM0_0 retention when entering in Low power modes.
- #define LOWPOWER_SRAMRETCTRL_RETEN_RAM10 (1UL << 6)
  Enable SRAM1_0 retention when entering in Low power modes.
- #define LOWPOWER_SRAMRETCTRL_RETEN_RAM20 (1UL << 7)
  Enable SRAM2_0 retention when entering in Low power modes.
- #define LOWPOWER_SRAMRETCTRL_RETEN_RAM_USB_HS (1UL << 14)
  Enable SRAM USB HS retention when entering in Low power modes.
- #define WAKEUP_SYS (1ULL << 0) /*!< [SLEEP, DEEP SLEEP] */ /* WWDT0_IRQ and BOD_IRQ*/
  Low Power Modes Wake up sources.
- #define WAKEUP_SDMA0 (1ULL << 1)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_GPIO_GLOBALINT0 (1ULL << 2)
  [SLEEP, DEEP SLEEP, POWER DOWN]
- #define WAKEUP_GPIO_GLOBALINT1 (1ULL << 3)
  [SLEEP, DEEP SLEEP, POWER DOWN]
- #define WAKEUP_GPIO_INT0_0 (1ULL << 4)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_GPIO_INT0_1 (1ULL << 5)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_GPIO_INT0_2 (1ULL << 6)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_GPIO_INT0_3 (1ULL << 7)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_UTICK (1ULL << 8)
  [SLEEP,]
- #define WAKEUP_MRT (1ULL << 9)
  [SLEEP,]
- #define WAKEUP_CTIMER0 (1ULL << 10)

[SLEEP, DEEP SLEEP]
- #define WAKEUP_CTIMER1 (1ULL << 11)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_SCT (1ULL << 12)
  [SLEEP,]
- #define WAKEUP_CTIMER3 (1ULL << 13)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_FLEXCOMM0 (1ULL << 14)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_FLEXCOMM1 (1ULL << 15)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_FLEXCOMM2 (1ULL << 16)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_FLEXCOMM3 (1ULL << 17)
  [SLEEP, DEEP SLEEP, POWER DOWN]
- #define WAKEUP_FLEXCOMM4 (1ULL << 18)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_FLEXCOMM5 (1ULL << 19)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_FLEXCOMM6 (1ULL << 20)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_FLEXCOMM7 (1ULL << 21)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_ADC (1ULL << 22)
  [SLEEP,]
- #define WAKEUP_ACMP (1ULL << 24)
  [SLEEP, DEEP SLEEP, POWER DOWN]
- #define WAKEUP_USB0_NEEDCLK (1ULL << 27)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_USB0 (1ULL << 28)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_RTC_LITE_ALARM_WAKEUP (1ULL << 29)
  [SLEEP, DEEP SLEEP, POWER DOWN, DEEP POWER DOWN]
- #define WAKEUP_GPIO_INT0_4 (1ULL << 32)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_GPIO_INT0_5 (1ULL << 33)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_GPIO_INT0_6 (1ULL << 34)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_GPIO_INT0_7 (1ULL << 35)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_CTIMER2 (1ULL << 36)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_CTIMER4 (1ULL << 37)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_OS_EVENT_TIMER (1ULL << 38)
  [SLEEP, DEEP SLEEP, POWER DOWN, DEEP POWER DOWN]
- #define CAN0_INT0 (1ULL << 43)
  [SLEEP,]
- #define CAN1_INT0 (1ULL << 44)
  [SLEEP,]

- #define WAKEUP_USB1 (1ULL << 47)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_USB1_NEEDCLK (1ULL << 48)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_SEC_HYPERVISOR_CALL (1ULL << 49)
  [SLEEP,]
- #define WAKEUP_SEC_GPIO_INT0_0 (1ULL << 50)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_SEC_GPIO_INT0_1 (1ULL << 51)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_PLU (1ULL << 52)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_SHA (1ULL << 54)
  [SLEEP,]
- #define WAKEUP_CASPER (1ULL << 55)
  [SLEEP,]
- #define WAKEUP_PUF (1ULL << 56)
  [SLEEP,]
- #define WAKEUP_SDMA1 (1ULL << 58)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_LSPI_HS (1ULL << 59)
  [SLEEP, DEEP SLEEP]
- #define WAKEUP_ALLWAKEUPIOS (1ULL << 63)
  [, DEEP POWER DOWN]
- #define LOWPOWER_HWWAKE_FORCED (1UL << 0)
  Sleep Postpone.
- #define LOWPOWER_HWWAKE_PERIPHERALS (1UL << 1)
  Wake for Flexcomms.
- #define LOWPOWER_HWWAKE_SDMA0 (1UL << 3)
  Wake for DMA0.
- #define LOWPOWER_HWWAKE_SDMA1 (1UL << 5)
  Wake for DMA1.
- #define LOWPOWER_HWWAKE_ENABLE_FRO192M (1UL << 31)
  Need to be set if FRO192M is disable - via PDCTRL0 - in Deep Sleep mode and any of \ LOWPOWER_HWWAKE_PERIPHERALS, LOWPOWER_HWWAKE_SDMA0 or LOWPOWER_HWWAKE_SDMA1 is set.
- #define LOWPOWER_CPURETCTRL_ENA_DISABLE 0
  In POWER DOWN mode, CPU Retention is disabled.
- #define LOWPOWER_CPURETCTRL_ENA_ENABLE 1
  In POWER DOWN mode, CPU Retention is enabled.
- #define LOWPOWER_WAKEUPIOSRC_PIO0_INDEX 0
  Wake up I/O sources.
- #define LOWPOWER_WAKEUPIOSRC_PIO1_INDEX 2
  Pin P0(28)
- #define LOWPOWER_WAKEUPIOSRC_PIO2_INDEX 4
  Pin P1(18)
- #define LOWPOWER_WAKEUPIOSRC_PIO3_INDEX 6
  Pin P1(30)
- #define LOWPOWER_WAKEUPIOSRC_DISABLE 0
  Wake up is disable.
- #define LOWPOWER_WAKEUPIOSRC_RISING 1

Wake up on rising edge.
- #define LOWPOWER_WAKEUPIOSRC_FALLING 2
  Wake up on falling edge.
- #define LOWPOWER_WAKEUPIOSRC_RISING_FALLING 3
  Wake up on both rising or falling edges.
- #define LOWPOWER_WAKEUPIOSRC_PIO0MODE_INDEX 12
  Pin P1(1)
- #define LOWPOWER_WAKEUPIOSRC_PIO1MODE_INDEX 14
  Pin P0(28)
- #define LOWPOWER_WAKEUPIOSRC_PIO2MODE_INDEX 16
  Pin P1(18)
- #define LOWPOWER_WAKEUPIOSRC_PIO3MODE_INDEX 18
  Pin P1(30)
- #define LOWPOWER_WAKEUPIOSRC_IO_MODE_PLAIN 0
  Wake up Pad is plain input.
- #define LOWPOWER_WAKEUPIOSRC_IO_MODE_PULLDOWN 1
  Wake up Pad is pull-down.
- #define LOWPOWER_WAKEUPIOSRC_IO_MODE_PULLUP 2
  Wake up Pad is pull-up.
- #define LOWPOWER_WAKEUPIOSRC_IO_MODE_REPEATER 3
  Wake up Pad is in repeater.
- #define LOWPOWER_WAKEUPIO_PIO0_PULLUPDOWN_INDEX 8
  Wake-up I/O 0 pull-up/down configuration index.
- #define LOWPOWER_WAKEUPIO_PIO1_PULLUPDOWN_INDEX 9
  Wake-up I/O 1 pull-up/down configuration index.
- #define LOWPOWER_WAKEUPIO_PIO2_PULLUPDOWN_INDEX 10
  Wake-up I/O 2 pull-up/down configuration index.
- #define LOWPOWER_WAKEUPIO_PIO3_PULLUPDOWN_INDEX 11
  Wake-up I/O 3 pull-up/down configuration index.
- #define LOWPOWER_WAKEUPIO_PIO0_PULLUPDOWN_MASK (1UL << LOWPOWER_-WAKEUPIO_PIO0_PULLUPDOWN_INDEX)
  Wake-up I/O 0 pull-up/down mask.
- #define LOWPOWER_WAKEUPIO_PIO1_PULLUPDOWN_MASK (1UL << LOWPOWER_-WAKEUPIO_PIO1_PULLUPDOWN_INDEX)
  Wake-up I/O 1 pull-up/down mask.
- #define LOWPOWER_WAKEUPIO_PIO2_PULLUPDOWN_MASK (1UL << LOWPOWER_-WAKEUPIO_PIO2_PULLUPDOWN_INDEX)
  Wake-up I/O 2 pull-up/down mask.
- #define LOWPOWER_WAKEUPIO_PIO3_PULLUPDOWN_MASK (1UL << LOWPOWER_-WAKEUPIO_PIO3_PULLUPDOWN_INDEX)
  Wake-up I/O 3 pull-up/down mask.
- #define LOWPOWER_WAKEUPIO_PULLDOWN 0
  Select pull-down.
- #define LOWPOWER_WAKEUPIO_PULLUP 1
  Select pull-up.
- #define LOWPOWER_WAKEUPIO_PIO0_DISABLEPULLUPDOWN_INDEX 12
  Wake-up I/O 0 pull-up/down disable/enable control index.
- #define LOWPOWER_WAKEUPIO_PIO1_DISABLEPULLUPDOWN_INDEX 13
  Wake-up I/O 1 pull-up/down disable/enable control index.
- #define LOWPOWER_WAKEUPIO_PIO2_DISABLEPULLUPDOWN_INDEX 14
  Wake-up I/O 2 pull-up/down disable/enable control index.

- #define LOWPOWER_WAKEUPIO_PIO3_DISABLEPULLUPDOWN_INDEX 15
  Wake-up I/O 3 pull-up/down disable/enable control index.
- #define LOWPOWER_WAKEUPIO_PIO0_DISABLEPULLUPDOWN_MASK (1UL << LOWPOWER_WAKEUPIO_PIO0_DISABLEPULLUPDOWN_INDEX)
  Wake-up I/O 0 pull-up/down disable/enable mask.
- #define LOWPOWER_WAKEUPIO_PIO1_DISABLEPULLUPDOWN_MASK (1UL << LOWPOWER_WAKEUPIO_PIO1_DISABLEPULLUPDOWN_INDEX)
  Wake-up I/O 1 pull-up/down disable/enable mask.
- #define LOWPOWER_WAKEUPIO_PIO2_DISABLEPULLUPDOWN_MASK (1UL << LOWPOWER_WAKEUPIO_PIO2_DISABLEPULLUPDOWN_INDEX)
  Wake-up I/O 2 pull-up/down disable/enable mask.
- #define LOWPOWER_WAKEUPIO_PIO3_DISABLEPULLUPDOWN_MASK (1UL << LOWPOWER_WAKEUPIO_PIO3_DISABLEPULLUPDOWN_INDEX)
  Wake-up I/O 3 pull-up/down disable/enable mask.
- #define LOWPOWER_WAKEUPIO_PIO0_USEEXTERNALPULLUPDOWN_INDEX (16)
  Wake-up I/O 0 use external pull-up/down disable/enable control index.
- #define LOWPOWER_WAKEUPIO_PIO1_USEEXTERNALPULLUPDOWN_INDEX (17)
  Wake-up I/O 1 use external pull-up/down disable/enable control index.
- #define LOWPOWER_WAKEUPIO_PIO2_USEEXTERNALPULLUPDOWN_INDEX (18)
  Wake-up I/O 2 use external pull-up/down disable/enable control index.
- #define LOWPOWER_WAKEUPIO_PIO3_USEEXTERNALPULLUPDOWN_INDEX (19)
  Wake-up I/O 3 use external pull-up/down disable/enable control index.
- #define LOWPOWER_WAKEUPIO_PIO0_USEEXTERNALPULLUPDOWN_MASK (1UL << LOWPOWER_WAKEUPIO_PIO0_USEEXTERNALPULLUPDOWN_INDEX)
  Wake-up I/O 0 use external pull-up/down \ disable/enable mask, 0: disable, 1: enable.
- #define LOWPOWER_WAKEUPIO_PIO1_USEEXTERNALPULLUPDOWN_MASK (1UL << LOWPOWER_WAKEUPIO_PIO1_USEEXTERNALPULLUPDOWN_INDEX)
  Wake-up I/O 1 use external pull-up/down \ disable/enable mask, 0: disable, 1: enable.
- #define LOWPOWER_WAKEUPIO_PIO2_USEEXTERNALPULLUPDOWN_MASK (1UL << LOWPOWER_WAKEUPIO_PIO2_USEEXTERNALPULLUPDOWN_INDEX)
  Wake-up I/O 2 use external pull-up/down \ disable/enable mask, 0: disable, 1: enable.
- #define LOWPOWER_WAKEUPIO_PIO3_USEEXTERNALPULLUPDOWN_MASK (1UL << LOWPOWER_WAKEUPIO_PIO3_USEEXTERNALPULLUPDOWN_INDEX)
  Wake-up I/O 3 use external pull-up/down \ disable/enable mask, 0: disable, 1: enable.

Enumerations

- enum pd_bit_t
  Analog components power modes control during low power modes.
- enum power_bod_vbat_level_t {

kPOWER_BodVbatLevel1000mv = 0,
kPOWER_BodVbatLevel1100mv = 1,
kPOWER_BodVbatLevel1200mv = 2,
kPOWER_BodVbatLevel1300mv = 3,
kPOWER_BodVbatLevel1400mv = 4,
kPOWER_BodVbatLevel1500mv = 5,
kPOWER_BodVbatLevel1600mv = 6,
kPOWER_BodVbatLevel1650mv = 7,
kPOWER_BodVbatLevel1700mv = 8,
kPOWER_BodVbatLevel1750mv = 9,
kPOWER_BodVbatLevel1800mv = 10,
kPOWER_BodVbatLevel1900mv = 11,
kPOWER_BodVbatLevel2000mv = 12,
kPOWER_BodVbatLevel2100mv = 13,
kPOWER_BodVbatLevel2200mv = 14,
kPOWER_BodVbatLevel2300mv = 15,
kPOWER_BodVbatLevel2400mv = 16,
kPOWER_BodVbatLevel2500mv = 17,
kPOWER_BodVbatLevel2600mv = 18,
kPOWER_BodVbatLevel2700mv = 19,
kPOWER_BodVbatLevel2806mv = 20,
kPOWER_BodVbatLevel2900mv = 21,
kPOWER_BodVbatLevel3000mv = 22,
kPOWER_BodVbatLevel3100mv = 23,
kPOWER_BodVbatLevel3200mv = 24,
kPOWER_BodVbatLevel3300mv = 25 }

BOD VBAT level.

- enum power_bod_hyst_t {
  kPOWER_BodHystLevel25mv = 0U,
  kPOWER_BodHystLevel50mv = 1U,
  kPOWER_BodHystLevel75mv = 2U,
  kPOWER_BodHystLevel100mv = 3U }

  BOD Hysteresis control.
- enum power_bod_core_level_t {
  kPOWER_BodCoreLevel600mv = 0,
  kPOWER_BodCoreLevel650mv = 1,
  kPOWER_BodCoreLevel700mv = 2,
  kPOWER_BodCoreLevel750mv = 3,
  kPOWER_BodCoreLevel800mv = 4,
  kPOWER_BodCoreLevel850mv = 5,
  kPOWER_BodCoreLevel900mv = 6,
  kPOWER_BodCoreLevel950mv = 7 }

  BOD core level.
- enum power_device_reset_cause_t {

kRESET_CAUSE_POR = 0UL,
kRESET_CAUSE_PADRESET = 1UL,
kRESET_CAUSE_BODRESET = 2UL,
kRESET_CAUSE_ARMSYSTEMRESET = 3UL,
kRESET_CAUSE_WDTRESET = 4UL,
kRESET_CAUSE_SWRRESET = 5UL,
kRESET_CAUSE_CDOGRESET = 6UL,
kRESET_CAUSE_DPDRESET_WAKEUPIO = 7UL,
kRESET_CAUSE_DPDRESET_RTC = 8UL,
kRESET_CAUSE_DPDRESET_OSTIMER = 9UL,
kRESET_CAUSE_DPDRESET_WAKEUPIO_RTC = 10UL,
kRESET_CAUSE_DPDRESET_WAKEUPIO_OSTIMER = 11UL,
kRESET_CAUSE_DPDRESET_RTC_OSTIMER = 12UL,
kRESET_CAUSE_DPDRESET_WAKEUPIO_RTC_OSTIMER = 13UL,
kRESET_CAUSE_NOT_RELEVANT,
kRESET_CAUSE_NOT_DETERMINISTIC = 15UL }
Device Reset Causes.

- enum power_device_boot_mode_t {
  kBOOT_MODE_POWER_UP,
  kBOOT_MODE_LP_DEEP_SLEEP = 1UL,
  kBOOT_MODE_LP_POWER_DOWN = 2UL,
  kBOOT_MODE_LP_DEEP_POWER_DOWN = 4UL }
  Device Boot Modes.

Functions

- static void POWER_EnablePD (pd_bit_t en)
  API to enable PDRUNCFG bit in the Syscon.
- static void POWER_DisablePD (pd_bit_t en)
  API to disable PDRUNCFG bit in the Syscon.
- void POWER_SetBodVbatLevel (power_bod_vbat_level_t level, power_bod_hyst_t hyst, bool enBodVbatReset)
  set BOD VBAT level.
- static void POWER_EnableDeepSleep (void)
  API to enable deep sleep bit in the ARM Core.
- static void POWER_DisableDeepSleep (void)
  API to disable deep sleep bit in the ARM Core.
- void POWER_CycleCpuAndFlash (void)
  Shut off the Flash and execute the _WFI(), then power up the Flash after wake-up event This MUST BE EXECUTED outside the Flash: either from ROM or from SRAM. The rest could stay in Flash. But, for consistency, it is preferable to have all functions defined in this file implemented in ROM.
- void POWER_EnterDeepSleep (uint32_t exclude_from_pd, uint32_t sram_retention_ctrl, uint64_t wakeup_interrupts, uint32_t hardware_wake_ctrl)
  Configures and enters in DEEP-SLEEP low power mode.
- void POWER_EnterPowerDown (uint32_t exclude_from_pd, uint32_t sram_retention_ctrl, uint64_t wakeup_interrupts, uint32_t cpu_retention_ctrl)
  Configures and enters in POWERDOWN low power mode.

- void POWER_EnterDeepPowerDown (uint32_t exclude_from_pd, uint32_t sram_retention_ctrl, uint64_t wakeup_interrupts, uint32_t wakeup_io_ctrl)
  Configures and enters in DEEPPOWERDOWN low power mode.
- void POWER_EnterSleep (void)
  Configures and enters in SLEEP low power mode.
- void POWER_SetVoltageForFreq (uint32_t system_freq_hz)
  Power Library API to choose normal regulation and set the voltage for the desired operating frequency.
- void POWER_Xtal16mhzCapabankTrim (int32_t pi32_16MfXtalIecLoadpF_x100, int32_t pi32_16MfXtalPPcbParCappF_x100, int32_t pi32_16MfXtalNPcbParCappF_x100)
  Sets board-specific trim values for 16MHz XTAL.
- void POWER_Xtal32khzCapabankTrim (int32_t pi32_32kfXtalIecLoadpF_x100, int32_t pi32_32kfXtalPPcbParCappF_x100, int32_t pi32_32kfXtalNPcbParCappF_x100)
  Sets board-specific trim values for 32kHz XTAL.
- void POWER_SetXtal16mhzLdo (void)
  Enables and sets LDO for 16MHz XTAL.
- void POWER_GetWakeUpCause (power_device_reset_cause_t *p_reset_cause, power_device_boot_mode_t *p_boot_mode, uint32_t *p_wakeupio_cause)
  Return some key information related to the device reset causes / wake-up sources, for all power modes.

Driver version

- #define FSL_POWER_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))
  power driver version 1.0.0.

5.2 Macro Definition Documentation

5.2.1 #define FSL_POWER_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))

5.2.2 #define LOWPOWER_SRAMRETCTRL_RETEN_RAMX0 (1UL << 0)

Enable SRAMX_0 retention when entering in Low power modes

5.2.3 #define LOWPOWER_HWWAKE_FORCED (1UL << 0)

Force peripheral clocking to stay on during deep-sleep mode.

5.2.4 #define LOWPOWER_HWWAKE_PERIPHERALS (1UL << 1)

Any Flexcomm FIFO reaching the level specified by its own TXLVL will cause \ peripheral clocking to wake up temporarily while the related status is asserted

5.2.5 #define LOWPOWER_HWWAKE_SDMA0 (1UL << 3)

DMA0 being busy will cause peripheral clocking to remain running until DMA \ completes. Used in conjonction with LOWPOWER_HWWAKE_PERIPHERALS

5.2.6 #define LOWPOWER_HWWAKE_SDMA1 (1UL << 5)

DMA0 being busy will cause peripheral clocking to remain running until DMA \ completes. Used in conjonction with LOWPOWER_HWWAKE_PERIPHERALS

5.2.7 #define LOWPOWER_WAKEUPIOSRC_PIO0_INDEX 0

Pin P1(1)

5.3 Enumeration Type Documentation

5.3.1 enum power_bod_vbat_level_t

Enumerator

kPOWER_BodVbatLevel1000mv	Brown out detector VBAT level 1V.
kPOWER_BodVbatLevel1100mv	Brown out detector VBAT level 1.1V.
kPOWER_BodVbatLevel1200mv	Brown out detector VBAT level 1.2V.
kPOWER_BodVbatLevel1300mv	Brown out detector VBAT level 1.3V.
kPOWER_BodVbatLevel1400mv	Brown out detector VBAT level 1.4V.
kPOWER_BodVbatLevel1500mv	Brown out detector VBAT level 1.5V.
kPOWER_BodVbatLevel1600mv	Brown out detector VBAT level 1.6V.
kPOWER_BodVbatLevel1650mv	Brown out detector VBAT level 1.65V.
kPOWER_BodVbatLevel1700mv	Brown out detector VBAT level 1.7V.
kPOWER_BodVbatLevel1750mv	Brown out detector VBAT level 1.75V.
kPOWER_BodVbatLevel1800mv	Brown out detector VBAT level 1.8V.
kPOWER_BodVbatLevel1900mv	Brown out detector VBAT level 1.9V.
kPOWER_BodVbatLevel2000mv	Brown out detector VBAT level 2V.
kPOWER_BodVbatLevel2100mv	Brown out detector VBAT level 2.1V.
kPOWER_BodVbatLevel2200mv	Brown out detector VBAT level 2.2V.
kPOWER_BodVbatLevel2300mv	Brown out detector VBAT level 2.3V.
kPOWER_BodVbatLevel2400mv	Brown out detector VBAT level 2.4V.
kPOWER_BodVbatLevel2500mv	Brown out detector VBAT level 2.5V.
kPOWER_BodVbatLevel2600mv	Brown out detector VBAT level 2.6V.
kPOWER_BodVbatLevel2700mv	Brown out detector VBAT level 2.7V.
kPOWER_BodVbatLevel2806mv	Brown out detector VBAT level 2.806V.
kPOWER_BodVbatLevel2900mv	Brown out detector VBAT level 2.9V.
kPOWER_BodVbatLevel3000mv	Brown out detector VBAT level 3.0V.

kPOWER_BodVbatLevel3100mv Brown out detector VBAT level 3.1V.
kPOWER_BodVbatLevel3200mv Brown out detector VBAT level 3.2V.
kPOWER_BodVbatLevel3300mv Brown out detector VBAT level 3.3V.

5.3.2 enum power_bod_hyst_t

Enumerator

kPOWER_BodHystLevel25mv BOD Hysteresis control level 25mv.
kPOWER_BodHystLevel50mv BOD Hysteresis control level 50mv.
kPOWER_BodHystLevel75mv BOD Hysteresis control level 75mv.
kPOWER_BodHystLevel100mv BOD Hysteresis control level 100mv.

5.3.3 enum power_bod_core_level_t

Enumerator

kPOWER_BodCoreLevel600mv Brown out detector core level 600mV.
kPOWER_BodCoreLevel650mv Brown out detector core level 650mV.
kPOWER_BodCoreLevel700mv Brown out detector core level 700mV.
kPOWER_BodCoreLevel750mv Brown out detector core level 750mV.
kPOWER_BodCoreLevel800mv Brown out detector core level 800mV.
kPOWER_BodCoreLevel850mv Brown out detector core level 850mV.
kPOWER_BodCoreLevel900mv Brown out detector core level 900mV.
kPOWER_BodCoreLevel950mv Brown out detector core level 950mV.

5.3.4 enum power_device_reset_cause_t

Enumerator

kRESET_CAUSE_POR Power On Reset.
kRESET_CAUSE_PADRESET Hardware Pin Reset.
kRESET_CAUSE_BODRESET Brown-out Detector reset (either BODVBAT or BODCORE)
kRESET_CAUSE_ARMSYSTEMRESET ARM System Reset.
kRESET_CAUSE_WDTRESET Watchdog Timer Reset.
kRESET_CAUSE_SWRRESET Software Reset.
kRESET_CAUSE_CDOGRESET Code Watchdog Reset.
kRESET_CAUSE_DPDRESET_WAKEUPIO Any of the 4 wake-up pins.
kRESET_CAUSE_DPDRESET_RTC Real Time Counter (RTC)
kRESET_CAUSE_DPDRESET_OSTIMER OS Event Timer (OSTIMER)
kRESET_CAUSE_DPDRESET_WAKEUPIO_RTC Any of the 4 wake-up pins and RTC (it is not possible to distinguish which of these 2 events occured first)

kRESET_CAUSE_DPDRESET_WAKEUPIO_OSTIMER Any of the 4 wake-up pins and OSTIMER (it is not possible to distinguish which of these 2 events occured first)

kRESET_CAUSE_DPDRESET_RTC_OSTIMER Real Time Counter or OS Event Timer (it is not possible to distinguish which of these 2 events occured first)

kRESET_CAUSE_DPDRESET_WAKEUPIO_RTC_OSTIMER Any of the 4 wake-up pins (it is not possible to distinguish which of these 3 events occured first)

kRESET_CAUSE_NOT_RELEVANT No reset cause (for example, this code is used when waking up from DEEP-SLEEP low power mode)

kRESET_CAUSE_NOT_DETERMINISTIC Unknown Reset Cause. Should be treated like "-Hardware Pin Reset" from an application point of view.

5.3.5 enum power_device_boot_mode_t

Enumerator

kBOOT_MODE_POWER_UP All non Low Power Mode wake up (Power On Reset, Pin Reset, BoD Reset, ARM System Reset ...)

kBOOT_MODE_LP_DEEP_SLEEP Wake up from DEEP-SLEEP Low Power mode.

kBOOT_MODE_LP_POWER_DOWN Wake up from POWER-DOWN Low Power mode.

kBOOT_MODE_LP_DEEP_POWER_DOWN Wake up from DEEP-POWER-DOWN Low Power mode.

5.4 Function Documentation

5.4.1 static void POWER_EnablePD (pd_bit_t *en*) [inline], [static]

Note that enabling the bit powers down the peripheral

Parameters

en	peripheral for which to enable the PDRUNCFG bit

Returns

none

5.4.2 static void POWER_DisablePD (pd_bit_t *en*) [inline], [static]

Note that disabling the bit powers up the peripheral

Parameters

en	peripheral for which to disable the PDRUNCFG bit

Returns

none

5.4.3 void POWER_SetBodVbatLevel (power_bod_vbat_level_t *level*, power_bod_hyst_t *hyst*, bool *enBodVbatReset*)

Parameters

level	BOD detect level
hyst	BoD Hysteresis control
enBodVbat-Reset	VBAT brown out detect reset

5.4.4 static void POWER_EnableDeepSleep (void) [inline], [static]

Returns

none

5.4.5 static void POWER_DisableDeepSleep (void) [inline], [static]

Returns

none

5.4.6 void POWER_CycleCpuAndFlash (void)

Returns

Nothing

5.4.7 void POWER_EnterDeepSleep (uint32_t *exclude_from_pd*, uint32_t *sram_retention_ctrl*, uint64_t *wakeup_interrupts*, uint32_t *hardware_wake_ctrl*)

Parameters

exclude_from_pd,:	
sram_retention_ctrl,:	
wakeup_interrupts,:	
hardware_wake_ctrl,:	

Returns

Nothing

```
!!! IMPORTANT NOTES :
```

0 - CPU0 & System CLock frequency is switched to FRO12MHz and is NOT restored back by the API. 1 - CPU0 Interrupt Enable registers (NVIC->ISER) are modified by this function. They are restored back in case of CPU retention or if POWERDOWN is not taken (for instance because an interrupt is pending). 2 - The Non Maskable Interrupt (NMI) is disabled and its configuration before calling this function will be restored back if POWERDOWN is not taken (for instance because an RTC or OSTIMER interrupt is pending). 3 - The HARD FAULT handler should execute from SRAM. (The Hard fault handler should initiate a full chip reset) reset)

5.4.8 void POWER_EnterPowerDown (uint32_t *exclude_from_pd,* uint32_t *sram_retention_ctrl,* uint64_t *wakeup_interrupts,* uint32_t *cpu_retention_ctrl*)

Parameters

exclude_from_pd,:	
sram_retention_ctrl,:	
wakeup_interrupts,:	

cpu_retention-_ctrl,:	0 = CPU retention is disable / 1 = CPU retention is enabled, all other values are RESERVED.

Returns

Nothing

```
!!! IMPORTANT NOTES :
```

0 - CPU0 & System CLock frequency is switched to FRO12MHz and is NOT restored back by the API. 1 - CPU0 Interrupt Enable registers (NVIC->ISER) are modified by this function. They are restored back in case of CPU retention or if POWERDOWN is not taken (for instance because an interrupt is pending). 2 - The Non Maskable Interrupt (NMI) is disabled and its configuration before calling this function will be restored back if POWERDOWN is not taken (for instance because an RTC or OSTIMER interrupt is pending). 3 - In case of CPU retention, it is the responsability of the user to make sure that SRAM instance containing the stack used to call this function WILL BE preserved during low power (via parameter "sram_retention_ctrl") 4 - The HARD FAULT handler should execute from SRAM. (The Hard fault handler should initiate a full chip reset) reset)

5.4.9 void POWER_EnterDeepPowerDown (uint32_t *exclude_from_pd,* uint32_t *sram_retention_ctrl,* uint64_t *wakeup_interrupts,* uint32_t *wakeup_io_ctrl*)

Parameters

exclude_from_-pd,:	
sram_-retention_ctrl,:	
wakeup_-interrupts,:	
wakeup_io_-ctrl,:	

Returns

Nothing

```
!!! IMPORTANT NOTES :
```

0 - CPU0 & System CLock frequency is switched to FRO12MHz and is NOT restored back by the API. 1 - CPU0 Interrupt Enable registers (NVIC->ISER) are modified by this function. They are restored back if DEEPPOWERDOWN is not taken (for instance because an RTC or OSTIMER interrupt is pending). 2 - The Non Maskable Interrupt (NMI) is disabled and its configuration before calling this function will be restored back if DEEPPOWERDOWN is not taken (for instance because an RTC or OSTIMER interrupt is pending). 3 - The HARD FAULT handler should execute from SRAM. (The Hard fault handler should initiate a full chip reset)

5.4.10 void POWER_EnterSleep (void)

Returns

Nothing

5.4.11 void POWER_SetVoltageForFreq (uint32_t *system_freq_hz*)

Parameters

system_freq_hz	- The desired frequency (in Hertz) at which the part would like to operate, note that the voltage and flash wait states should be set before changing frequency

Returns

none

5.4.12 void POWER_Xtal16mhzCapabankTrim (int32_t *pi32_16MfXtalIecLoadpF_x100,* int32_t *pi32_16MfXtalPPcbParCappF_x100,* int32_t *pi32_16MfXtalNPcbParCappF_x100*)

Parameters

pi32_16MfXtalIecLoadpF_x100	Load capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120
pi32_16MfXtalPPcbParCappF_x100	PCB +ve parasitic capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120
pi32_16MfXtalNPcbParCappF_x100	PCB -ve parasitic capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120

Returns

none

Note

Following default Values can be used: pi32_32MfXtalIecLoadpF_x100 Load capacitance, pF x 100 : 600 pi32_32MfXtalPPcbParCappF_x100 PCB +ve parasitic capacitance, pF x 100 : 20 pi32_32MfXtalNPcbParCappF_x100 PCB -ve parasitic capacitance, pF x 100 : 40

5.4.13 void POWER_Xtal32khzCapabankTrim (int32_t *pi32_32kfXtallecLoadpF_x100,* int32_t *pi32_32kfXtalPPcbParCappF_x100,* int32_t *pi32_32kfXtalNPcbParCappF_x100*)

Parameters

pi32_32kfXtalIecLoadpF_x100	Load capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120
pi32_32kfXtalPPcbParCappF_x100	PCB +ve parasitic capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120
pi32_32kfXtalNPcbParCappF_x100	PCB -ve parasitic capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120

Returns

none

Note

Following default Values can be used: pi32_32kfXtalIecLoadpF_x100 Load capacitance, pF x 100 : 600 pi32_32kfXtalPPcbParCappF_x100 PCB +ve parasitic capacitance, pF x 100 : 40 pi32_32kfXtalNPcbParCappF_x100 PCB -ve parasitic capacitance, pF x 100 : 40

5.4.14 void POWER_SetXtal16mhzLdo (void)

Returns

none

5.4.15 void POWER_GetWakeUpCause (power_device_reset_cause_t * *p_reset_cause,* power_device_boot_mode_t * *p_boot_mode,* uint32_t * *p_wakeupio_cause*)

Parameters

p_reset_cause	: the device reset cause, according to the definition of power_device_reset_cause_t type.

p_boot_mode	: the device boot mode, according to the definition of power_device_boot_mode_t type.
p_wakeupio_-cause,:	the wake-up pin sources, according to the definition of register PMC->WAKEIOC-AUSE[3:0].

Returns

Nothing

```
!!!  IMPORTANT ERRATA - IMPORTANT ERRATA - IMPORTANT ERRATA     !!!
!!!   valid ONLY for LPC55S69 (not for LPC55S16 and LPC55S06)   !!!
!!!   when FALLING EDGE DETECTION is enabled on wake-up pins:   !!!
- 1. p_wakeupio_cause is NOT ACCURATE
- 2. Spurious kRESET_CAUSE_DPDRESET_WAKEUPIO* event is reported when
     several wake-up sources are enabled during DEEP-POWER-DOWN
     (like enabling wake-up on RTC and Falling edge wake-up pins)
```

Chapter 6
Reset Driver

6.1 Overview

Reset driver supports peripheral reset and system reset.

Macros

- #define ADC_RSTS

Enumerations

- enum SYSCON_RSTn_t {
  kROM_RST_SHIFT_RSTn = 0 | 1U,
  kSRAM1_RST_SHIFT_RSTn = 0 | 3U,
  kSRAM2_RST_SHIFT_RSTn = 0 | 4U,
  kFLASH_RST_SHIFT_RSTn = 0 | 7U,
  kFMC_RST_SHIFT_RSTn = 0 | 8U,
  kMUX0_RST_SHIFT_RSTn = 0 | 11U,
  kIOCON_RST_SHIFT_RSTn = 0 | 13U,
  kGPIO0_RST_SHIFT_RSTn = 0 | 14U,
  kGPIO1_RST_SHIFT_RSTn = 0 | 15U,
  kPINT_RST_SHIFT_RSTn = 0 | 18U,
  kGINT_RST_SHIFT_RSTn = 0 | 19U,
  kDMA0_RST_SHIFT_RSTn = 0 | 20U,
  kCRC_RST_SHIFT_RSTn = 0 | 21U,
  kWWDT_RST_SHIFT_RSTn = 0 | 22U,
  kRTC_RST_SHIFT_RSTn = 0 | 23U,
  kMAILBOX_RST_SHIFT_RSTn = 0 | 26U,
  kADC0_RST_SHIFT_RSTn = 0 | 27U,
  kMRT_RST_SHIFT_RSTn = 65536 | 0U,
  kOSTIMER0_RST_SHIFT_RSTn = 65536 | 1U,
  kSCT0_RST_SHIFT_RSTn = 65536 | 2U,
  kMCAN_RST_SHIFT_RSTn = 65536 | 7U,
  kUTICK_RST_SHIFT_RSTn = 65536 | 10U,
  kFC0_RST_SHIFT_RSTn = 65536 | 11U,
  kFC1_RST_SHIFT_RSTn = 65536 | 12U,
  kFC2_RST_SHIFT_RSTn = 65536 | 13U,
  kFC3_RST_SHIFT_RSTn = 65536 | 14U,
  kFC4_RST_SHIFT_RSTn = 65536 | 15U,
  kFC5_RST_SHIFT_RSTn = 65536 | 16U,
  kFC6_RST_SHIFT_RSTn = 65536 | 17U,
  kFC7_RST_SHIFT_RSTn = 65536 | 18U,
  kCTIMER2_RST_SHIFT_RSTn = 65536 | 22U,
  kUSB0D_RST_SHIFT_RSTn = 65536 | 25U,
  kCTIMER0_RST_SHIFT_RSTn = 65536 | 26U,
  kCTIMER1_RST_SHIFT_RSTn = 65536 | 27U,
  kEZHA_RST_SHIFT_RSTn = 65536 | 30U,
  kEZHB_RST_SHIFT_RSTn = 65536 | 31U,
  kDMA1_RST_SHIFT_RSTn = 131072 | 1U,
  kCMP_RST_SHIFT_RSTn = 131072 | 2U,
  kUSB1H_RST_SHIFT_RSTn = 131072 | 4U,
  kUSB1D_RST_SHIFT_RSTn = 131072 | 5U,
  kUSB1RAM_RST_SHIFT_RSTn = 131072 | 6U,
  kUSB1_RST_SHIFT_RSTn = 131072 | 7U,
  kFREQME_RST_SHIFT_RSTn = 131072 | 8U,
  kCDOG_RST_SHIFT_RSTn = 131072 | 11U,
  kRNG_RST_SHIFT_RSTn = 131072 | 13U,
  kSYSCTL_RST_SHIFT_RSTn = 131072 | 15U,
  kUSB0HMR_RST_SHIFT_RSTn = 131072 | 16U,

kGPIOSECINT_RST_SHIFT_RSTn = 131072 | 30U }
Enumeration for peripheral reset control bits.

Functions

- void RESET_SetPeripheralReset (reset_ip_name_t peripheral)
  Assert reset to peripheral.
- void RESET_ClearPeripheralReset (reset_ip_name_t peripheral)
  Clear reset to peripheral.
- void RESET_PeripheralReset (reset_ip_name_t peripheral)
  Reset peripheral module.

Driver version

- #define FSL_RESET_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  reset driver version 2.0.0.

6.2 Macro Definition Documentation

6.2.1 #define FSL_RESET_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

6.2.2 #define ADC_RSTS

Value:

```
{                       \
        kADC0_RST_SHIFT_RSTn \
    } /* Reset bits for ADC peripheral */
```

Array initializers with peripheral reset bits

6.3 Enumeration Type Documentation

6.3.1 enum SYSCON_RSTn_t

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

Enumerator

kROM_RST_SHIFT_RSTn ROM reset control
kSRAM1_RST_SHIFT_RSTn SRAM1 reset control
kSRAM2_RST_SHIFT_RSTn SRAM2 reset control
kFLASH_RST_SHIFT_RSTn Flash controller reset control
kFMC_RST_SHIFT_RSTn Flash accelerator reset control
kMUX0_RST_SHIFT_RSTn Input mux0 reset control
kIOCON_RST_SHIFT_RSTn IOCON reset control

kGPIO0_RST_SHIFT_RSTn GPIO0 reset control
kGPIO1_RST_SHIFT_RSTn GPIO1 reset control
kPINT_RST_SHIFT_RSTn Pin interrupt (PINT) reset control
kGINT_RST_SHIFT_RSTn Grouped interrupt (PINT) reset control.
kDMA0_RST_SHIFT_RSTn DMA reset control
kCRC_RST_SHIFT_RSTn CRC reset control
kWWDT_RST_SHIFT_RSTn Watchdog timer reset control
kRTC_RST_SHIFT_RSTn RTC reset control
kMAILBOX_RST_SHIFT_RSTn Mailbox reset control
kADC0_RST_SHIFT_RSTn ADC0 reset control
kMRT_RST_SHIFT_RSTn Multi-rate timer (MRT) reset control
kOSTIMER0_RST_SHIFT_RSTn OSTimer0 reset control
kSCT0_RST_SHIFT_RSTn SCTimer/PWM 0 (SCT0) reset control
kMCAN_RST_SHIFT_RSTn MCAN reset control
kUTICK_RST_SHIFT_RSTn Micro-tick timer reset control
kFC0_RST_SHIFT_RSTn Flexcomm Interface 0 reset control
kFC1_RST_SHIFT_RSTn Flexcomm Interface 1 reset control
kFC2_RST_SHIFT_RSTn Flexcomm Interface 2 reset control
kFC3_RST_SHIFT_RSTn Flexcomm Interface 3 reset control
kFC4_RST_SHIFT_RSTn Flexcomm Interface 4 reset control
kFC5_RST_SHIFT_RSTn Flexcomm Interface 5 reset control
kFC6_RST_SHIFT_RSTn Flexcomm Interface 6 reset control
kFC7_RST_SHIFT_RSTn Flexcomm Interface 7 reset control
kCTIMER2_RST_SHIFT_RSTn CTimer 2 reset control
kUSB0D_RST_SHIFT_RSTn USB0 Device reset control
kCTIMER0_RST_SHIFT_RSTn CTimer 0 reset control
kCTIMER1_RST_SHIFT_RSTn CTimer 1 reset control
kEZHA_RST_SHIFT_RSTn EZHA reset control
kEZHB_RST_SHIFT_RSTn EZHB reset control
kDMA1_RST_SHIFT_RSTn DMA1 reset control
kCMP_RST_SHIFT_RSTn CMP reset control
kUSB1H_RST_SHIFT_RSTn USBHS Host reset control
kUSB1D_RST_SHIFT_RSTn USBHS Device reset control
kUSB1RAM_RST_SHIFT_RSTn USB RAM reset control
kUSB1_RST_SHIFT_RSTn USBHS reset control
kFREQME_RST_SHIFT_RSTn FREQME reset control
kCDOG_RST_SHIFT_RSTn Code Watchdog reset control
kRNG_RST_SHIFT_RSTn RNG reset control
kSYSCTL_RST_SHIFT_RSTn SYSCTL reset control
kUSB0HMR_RST_SHIFT_RSTn USB0HMR reset control
kUSB0HSL_RST_SHIFT_RSTn USB0HSL reset control
kHASHCRYPT_RST_SHIFT_RSTn HASHCRYPT reset control
kPLULUT_RST_SHIFT_RSTn PLU LUT reset control
kCTIMER3_RST_SHIFT_RSTn CTimer 3 reset control
kCTIMER4_RST_SHIFT_RSTn CTimer 4 reset control

kPUF_RST_SHIFT_RSTn PUF reset control
kCASPER_RST_SHIFT_RSTn CASPER reset control
kANALOGCTL_RST_SHIFT_RSTn ANALOG_CTL reset control
kHSLSPI_RST_SHIFT_RSTn HS LSPI reset control
kGPIOSEC_RST_SHIFT_RSTn GPIO Secure reset control
kGPIOSECINT_RST_SHIFT_RSTn GPIO Secure int reset control

6.4 Function Documentation

6.4.1 void RESET_SetPeripheralReset (reset_ip_name_t *peripheral*)

Asserts reset signal to specified peripheral module.

Parameters

peripheral	Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

6.4.2 void RESET_ClearPeripheralReset (reset_ip_name_t *peripheral*)

Clears reset signal to specified peripheral module, allows it to operate.

Parameters

peripheral	Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

6.4.3 void RESET_PeripheralReset (reset_ip_name_t *peripheral*)

Reset peripheral module.

Parameters

peripheral	Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.

Chapter 7
ANACTRL: Analog Control Driver

7.1 ANACTRL function groups

7.2 Overview

7.3 Function groups

The ANACTRL driver supports initialization/configuration/operation for optimization/customization purpose.

7.3.1 Initialization and deinitialization

This function group is to enable/disable the clock for the ANACTRL module.

7.3.2 Set oscillators

The function ANACTRL_SetFro192M sets the on-chip high-speed Free Running Oscillator. The function ANACTRL_GetDefaultFro192MConfig() gets the default configuration.

The function ANACTRL_SetXo32M sets the 32 MHz Crystal oscillator. The function ANACTRL_GetDefaultXo32MConfig() gets the default configuration.

7.3.3 Measure Frequency

This function measures the target frequency according to the reference frequency.

7.3.4 Interrupt

Provides functions to enable/disable/clear ANACTRL interrupts.

7.3.5 Status

Provides functions to get the ANACTRL status.

Data Structures

- struct anactrl_fro192M_config_t

Configuration for FRO192M. More...

- struct anactrl_xo32M_config_t
  Configuration for XO32M. More...

Macros

- #define FSL_ANACTRL_DRIVER_VERSION (MAKE_VERSION(2, 3, 0)) /∗!< Version 2.3.0. ∗/‘
  ANACTRL driver version.

Enumerations

- enum _anactrl_interrupt_flags {
  kANACTRL_BodVbatFlag = ANACTRL_BOD_DCDC_INT_STATUS_BODVBAT_STATUS_MASK,
  kANACTRL_BodVbatInterruptFlag = ANACTRL_BOD_DCDC_INT_STATUS_BODVBAT_INT_STATUS_MASK,
  kANACTRL_BodVbatPowerFlag = ANACTRL_BOD_DCDC_INT_STATUS_BODVBAT_VAL_MASK,
  kANACTRL_BodCoreFlag = ANACTRL_BOD_DCDC_INT_STATUS_BODCORE_STATUS_MASK,
  kANACTRL_BodCoreInterruptFlag = ANACTRL_BOD_DCDC_INT_STATUS_BODCORE_INT_STATUS_MASK,
  kANACTRL_BodCorePowerFlag = ANACTRL_BOD_DCDC_INT_STATUS_BODCORE_VAL_MASK,
  kANACTRL_DcdcFlag = ANACTRL_BOD_DCDC_INT_STATUS_DCDC_STATUS_MASK,
  kANACTRL_DcdcInterruptFlag = ANACTRL_BOD_DCDC_INT_STATUS_DCDC_INT_STATUS_MASK,
  kANACTRL_DcdcPowerFlag = ANACTRL_BOD_DCDC_INT_STATUS_DCDC_VAL_MASK }
  ANACTRL interrupt flags.
- enum _anactrl_interrupt {
  kANACTRL_BodVbatInterruptEnable = ANACTRL_BOD_DCDC_INT_CTRL_BODVBAT_INT_ENABLE_MASK,
  kANACTRL_BodCoreInterruptEnable = ANACTRL_BOD_DCDC_INT_CTRL_BODCORE_INT_ENABLE_MASK,
  kANACTRL_DcdcInterruptEnable = ANACTRL_BOD_DCDC_INT_CTRL_DCDC_INT_ENABLE_MASK }
  ANACTRL interrupt control.
- enum _anactrl_flags {
  kANACTRL_FlashPowerDownFlag = ANACTRL_ANALOG_CTRL_STATUS_FLASH_PWRDWN_MASK,
  kANACTRL_FlashInitErrorFlag = ANACTRL_ANALOG_CTRL_STATUS_FLASH_INIT_ERROR_MASK }
  ANACTRL status flags.
- enum _anactrl_osc_flags {

kANACTRL_OutputClkValidFlag = ANACTRL_FRO192M_STATUS_CLK_VALID_MASK,
kANACTRL_CCOThresholdVoltageFlag = ANACTRL_FRO192M_STATUS_ATB_VCTRL_MASK,
kANACTRL_XO32MOutputReadyFlag = ANACTRL_XO32M_STATUS_XO_READY_MASK << 16U }
ANACTRL FRO192M and XO32M status flags.

Initialization and deinitialization

- void ANACTRL_Init (ANACTRL_Type *base)
  Initializes the ANACTRL mode, the module's clock will be enabled by invoking this function.
- void ANACTRL_Deinit (ANACTRL_Type *base)
  De-initializes ANACTRL module, the module's clock will be disabled by invoking this function.

Set oscillators

- void ANACTRL_SetFro192M (ANACTRL_Type *base, const anactrl_fro192M_config_t *config)
  Configs the on-chip high-speed Free Running Oscillator(FRO192M), such as enabling/disabling 12 MHZ clock output and enable/disable 96MHZ clock output.
- void ANACTRL_GetDefaultFro192MConfig (anactrl_fro192M_config_t *config)
  Gets the default configuration of FRO192M.
- void ANACTRL_SetXo32M (ANACTRL_Type *base, const anactrl_xo32M_config_t *config)
  Configs the 32 MHz Crystal oscillator(High-speed crystal oscillator), such as enable/disable output to CPU system, and so on.
- void ANACTRL_GetDefaultXo32MConfig (anactrl_xo32M_config_t *config)
  Gets the default configuration of XO32M.

Measure Frequency

- uint32_t ANACTRL_MeasureFrequency (ANACTRL_Type *base, uint8_t scale, uint32_t refClkFreq)
  Measures the frequency of the target clock source.

Interrupt Interface

- static void ANACTRL_EnableInterrupts (ANACTRL_Type *base, uint32_t mask)
  Enables the ANACTRL interrupts.
- static void ANACTRL_DisableInterrupts (ANACTRL_Type *base, uint32_t mask)
  Disables the ANACTRL interrupts.
- static void ANACTRL_ClearInterrupts (ANACTRL_Type *base, uint32_t mask)
  Clears the ANACTRL interrupts.

Status Interface

- static uint32_t ANACTRL_GetStatusFlags (ANACTRL_Type *base)
  Gets ANACTRL status flags.
- static uint32_t ANACTRL_GetOscStatusFlags (ANACTRL_Type *base)
  Gets ANACTRL oscillators status flags.
- static uint32_t ANACTRL_GetInterruptStatusFlags (ANACTRL_Type *base)

Gets ANACTRL interrupt status flags.

- static void ANACTRL_EnableVref1V (ANACTRL_Type *base, bool enable)
  Aux_Bias Control Interfaces.

7.4 Data Structure Documentation

7.4.1 struct anactrl_fro192M_config_t

This structure holds the configuration settings for the on-chip high-speed Free Running Oscillator. To initialize this structure to reasonable defaults, call the ANACTRL_GetDefaultFro192MConfig() function and pass a pointer to your config structure instance.

Data Fields

- bool enable12MHzClk
  Enable 12MHz clock.
- bool enable96MHzClk
  Enable 96MHz clock.

Field Documentation

(1) bool anactrl_fro192M_config_t::enable12MHzClk

(2) bool anactrl_fro192M_config_t::enable96MHzClk

7.4.2 struct anactrl_xo32M_config_t

This structure holds the configuration settings for the 32 MHz crystal oscillator. To initialize this structure to reasonable defaults, call the ANACTRL_GetDefaultXo32MConfig() function and pass a pointer to your config structure instance.

Data Fields

- bool enableACBufferBypass
  Enable XO AC buffer bypass in pll and top level.
- bool enableSysCLkOutput
  Enable XO 32 MHz output to CPU system, SCT, and CLKOUT.
- bool enableADCOutput
  Enable High speed crystal oscillator output to ADC.

Field Documentation

(1) bool anactrl_xo32M_config_t::enableACBufferBypass

(2) bool anactrl_xo32M_config_t::enableADCOutput

7.5 Macro Definition Documentation

7.5.1 #define FSL_ANACTRL_DRIVER_VERSION (MAKE_VERSION(2, 3, 0)) /*!< Version 2.3.0. */‘

7.6 Enumeration Type Documentation

7.6.1 enum _anactrl_interrupt_flags

Enumerator

kANACTRL_BodVbatFlag BOD VBAT Interrupt status before Interrupt Enable.
kANACTRL_BodVbatInterruptFlag BOD VBAT Interrupt status after Interrupt Enable.
kANACTRL_BodVbatPowerFlag Current value of BOD VBAT power status output.
kANACTRL_BodCoreFlag BOD CORE Interrupt status before Interrupt Enable.
kANACTRL_BodCoreInterruptFlag BOD CORE Interrupt status after Interrupt Enable.
kANACTRL_BodCorePowerFlag Current value of BOD CORE power status output.
kANACTRL_DcdcFlag DCDC Interrupt status before Interrupt Enable.
kANACTRL_DcdcInterruptFlag DCDC Interrupt status after Interrupt Enable.
kANACTRL_DcdcPowerFlag Current value of DCDC power status output.

7.6.2 enum _anactrl_interrupt

Enumerator

kANACTRL_BodVbatInterruptEnable BOD VBAT interrupt control.
kANACTRL_BodCoreInterruptEnable BOD CORE interrupt control.
kANACTRL_DcdcInterruptEnable DCDC interrupt control.

7.6.3 enum _anactrl_flags

Enumerator

kANACTRL_FlashPowerDownFlag Flash power-down status.
kANACTRL_FlashInitErrorFlag Flash initialization error status.

7.6.4 enum _anactrl_osc_flags

Enumerator

kANACTRL_OutputClkValidFlag Output clock valid signal.
kANACTRL_CCOThresholdVoltageFlag CCO threshold voltage detector output (signal vcco_ok).

kANACTRL_XO32MOutputReadyFlag Indicates XO out frequency statibilty.

7.7 Function Documentation

7.7.1 void ANACTRL_Init (ANACTRL_Type * *base*)

Parameters

base	ANACTRL peripheral base address.

7.7.2 void ANACTRL_Deinit (ANACTRL_Type * *base*)

Parameters

base	ANACTRL peripheral base address.

7.7.3 void ANACTRL_SetFro192M (ANACTRL_Type * *base*, const anactrl_fro192M_config_t * *config*)

Parameters

base	ANACTRL peripheral base address.
config	Pointer to FRO192M configuration structure. Refer to anactrl_fro192M_config_t structure.

7.7.4 void ANACTRL_GetDefaultFro192MConfig (anactrl_fro192M_config_t * *config*)

The default values are:

```
config->enable12MHzClk = true;
config->enable96MHzClk = false;
```

Parameters

config	Pointer to FRO192M configuration structure. Refer to anactrl_fro192M_config_t structure.

7.7.5 void ANACTRL_SetXo32M (ANACTRL_Type * *base*, const anactrl_xo32M_config_t * *config*)

Parameters

base	ANACTRL peripheral base address.
config	Pointer to XO32M configuration structure. Refer to anactrl_xo32M_config_t structure.

7.7.6 void ANACTRL_GetDefaultXo32MConfig (anactrl_xo32M_config_t * *config*)

The default values are:

```
config->enableSysCLkOutput = false;
config->enableACBufferBypass = false;
```

Parameters

config	Pointer to XO32M configuration structure. Refer to anactrl_xo32M_config_t structure.

7.7.7 uint32_t ANACTRL_MeasureFrequency (ANACTRL_Type * *base,* uint8_t *scale,* uint32_t *refClkFreq*)

This function measures target frequency according to a accurate reference frequency.The formula is: Ftarget = (CAPVAL * Freference) / ((1<<SCALE)-1)

Note

Both tartget and reference clocks are selectable by programming the target clock select FREQMEAS_TARGET register in INPUTMUX and reference clock select FREQMEAS_REF register in INPUTMUX.

Parameters

base	ANACTRL peripheral base address.
scale	Define the power of 2 count that ref counter counts to during measurement, ranges from 2 to 31.

refClkFreq	frequency of the reference clock.

Returns

frequency of the target clock.

7.7.8 static void ANACTRL_EnableInterrupts (ANACTRL_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	ANACTRL peripheral base address.
mask	The interrupt mask. Refer to "_anactrl_interrupt" enumeration.

7.7.9 static void ANACTRL_DisableInterrupts (ANACTRL_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	ANACTRL peripheral base address.
mask	The interrupt mask. Refer to "_anactrl_interrupt" enumeration.

7.7.10 static void ANACTRL_ClearInterrupts (ANACTRL_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	ANACTRL peripheral base address.
mask	The interrupt mask. Refer to "_anactrl_interrupt" enumeration.

7.7.11 static uint32_t ANACTRL_GetStatusFlags (ANACTRL_Type * *base*) [inline], [static]

This function gets Analog control status flags. The flags are returned as the logical OR value of the enumerators _anactrl_flags. To check for a specific status, compare the return value with enumerators in the _anactrl_flags. For example, to check whether the flash is in power down mode:

```
*     if (kANACTRL_FlashPowerDownFlag & ANACTRL_ANACTRL_GetStatusFlags(ANACTRL))
*     {
*         ...
*     }
*
```

Parameters

base	ANACTRL peripheral base address.

Returns

ANACTRL status flags which are given in the enumerators in the _anactrl_flags.

7.7.12 static uint32_t ANACTRL_GetOscStatusFlags (ANACTRL_Type * *base*) [inline], [static]

This function gets Anactrl oscillators status flags. The flags are returned as the logical OR value of the enumerators _anactrl_osc_flags. To check for a specific status, compare the return value with enumerators in the _anactrl_osc_flags. For example, to check whether the FRO192M clock output is valid:

```
*     if (kANACTRL_OutputClkValidFlag & ANACTRL_ANACTRL_GetOscStatusFlags(
      ANACTRL))
*     {
*         ...
*     }
*
```

Parameters

base	ANACTRL peripheral base address.

Returns

ANACTRL oscillators status flags which are given in the enumerators in the _anactrl_osc_flags.

7.7.13 static uint32_t ANACTRL_GetInterruptStatusFlags (ANACTRL_Type * *base*) [inline], [static]

This function gets Anactrl interrupt status flags. The flags are returned as the logical OR value of the enumerators _anactrl_interrupt_flags. To check for a specific status, compare the return value with enumerators in the _anactrl_interrupt_flags. For example, to check whether the VBAT voltage level is above the threshold:

```
*     if (kANACTRL_BodVbatPowerFlag & ANACTRL_ANACTRL_GetInterruptStatusFlags(
      ANACTRL))
*     {
*         ...
*     }
*
```

Parameters

base	ANACTRL peripheral base address.

Returns

ANACTRL oscillators status flags which are given in the enumerators in the _anactrl_osc_flags.

7.7.14 static void ANACTRL_EnableVref1V (ANACTRL_Type * *base*, bool *enable*) [inline], [static]

Enables/disabless 1V reference voltage buffer.

Parameters

base	ANACTRL peripheral base address.
enable	Used to enable or disable 1V reference voltage buffer.

Chapter 8
CASPER: The Cryptographic Accelerator and Signal Processing Engine with RAM sharing

8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Cryptographic Accelerator and Signal Processing Engine with RAM sharing (CASPER) module of MCUXpresso SDK devices. The CASPER peripheral provides acceleration of asymmetric cryptographic algorithms as well as optionally of certain signal processing algorithms. The cryptographic acceleration is normally used in conjunction with pure-hardware blocks for hashing and symmetric cryptography, thereby providing performance and energy efficiency for a range of cryptographic uses.

Blocking synchronous APIs are provided for selected cryptographic algorithms using CASPER hardware. The driver interface intends to be easily integrated with generic software crypto libraries such as mbedTLS or wolfSSL. The CASPER operations are complete (and results are made availabe for further usage) when a function returns. When called, these functions do not return until an CASPER operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status and also for plaintext or ciphertext data movements. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

8.2 CASPER Driver Initialization and deinitialization

CASPER Driver is initialized by calling the CASPER_Init() function, it resets the CASPER module and enables it's clock. CASPER Driver is deinitialized by calling the CASPER_Deinit() function, it disables CASPER module clock.

8.3 Comments about API usage in RTOS

CASPER operations provided by this driver are not re-entrant. Thus, application software shall ensure the CASPER module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

8.4 Comments about API usage in interrupt handler

All APIs shall not be used from interrupt handler as global variables are used.

8.5 CASPER Driver Examples

8.5.1 Simple examples

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

Modules

- casper_driver
- casper_driver_pkha

8.6 casper_driver

8.6.1 Overview

Enumerations

- enum casper_operation_t { ,
  kCASPER_OpMul6464Sum,
  kCASPER_OpMul6464FullSum,
  kCASPER_OpMul6464Reduce,
  kCASPER_OpAdd64 = 0x08,
  kCASPER_OpSub64 = 0x09,
  kCASPER_OpDouble64 = 0x0A,
  kCASPER_OpXor64 = 0x0B,
  kCASPER_OpRSub64 = 0x0C,
  kCASPER_OpShiftLeft32,
  kCASPER_OpShiftRight32 = 0x11,
  kCASPER_OpCopy = 0x14,
  kCASPER_OpRemask = 0x15,
  kCASPER_OpFill = 0x16,
  kCASPER_OpZero = 0x17,
  kCASPER_OpCompare = 0x18,
  kCASPER_OpCompareFast = 0x19 }
  CASPER operation.
- enum casper_algo_t {
  kCASPER_ECC_P256 = 0x01,
  kCASPER_ECC_P384 = 0x02,
  kCASPER_ECC_P521 = 0x03 }
  Algorithm used for CASPER operation.

Functions

- void CASPER_Init (CASPER_Type *base)
  Enables clock and disables reset for CASPER peripheral.
- void CASPER_Deinit (CASPER_Type *base)
  Disables clock for CASPER peripheral.

Driver version

- #define FSL_CASPER_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))
  CASPER driver version.

8.6.2 Macro Definition Documentation

8.6.2.1 #define FSL_CASPER_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))

Version 2.2.3.

Current version: 2.2.3

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Bug fix KPSDK-24531 double_scalar_multiplication() result may be all zeroes for some specific input
- Version 2.0.2
  - Bug fix KPSDK-25015 CASPER_MEMCPY hard-fault on LPC55xx when both source and destination buffers are outside of CASPER_RAM
- Version 2.0.3
  - Bug fix KPSDK-28107 RSUB, FILL and ZERO operations not implemented in enum _casper_operation.
- Version 2.0.4
  - For GCC compiler, enforce O1 optimize level, specifically to remove strict-aliasing option. This driver is very specific and requires -fno-strict-aliasing.
- Version 2.0.5
  - Fix sign-compare warning.
- Version 2.0.6
  - Fix IAR Pa082 warning.
- Version 2.0.7
  - Fix MISRA-C 2012 issue.
- Version 2.0.8
  - Add feature macro for CASPER_RAM_OFFSET.
- Version 2.0.9
  - Remove unused function Jac_oncurve().
  - Fix ECC384 build.
- Version 2.0.10
  - Fix MISRA-C 2012 issue.
- Version 2.1.0
  - Add ECC NIST P-521 elliptic curve.
- Version 2.2.0
  - Rework driver to support multiple curves at once.
- Version 2.2.1
  - Fix MISRA-C 2012 issue.
- Version 2.2.2
  - Enable hardware interleaving to RAMX0 and RAMX1 for CASPER by feature macro FSL_FEATURE_CASPER_RAM_HW_INTERLEAVE
- Version 2.2.3
  - Fix MISRA-C 2012 issues.

8.6.3 Enumeration Type Documentation

8.6.3.1 enum casper_operation_t

Enumerator

kCASPER_OpMul6464Sum Walking 1 or more of J loop, doing r=a∗b using 64x64=128.
kCASPER_OpMul6464FullSum Walking 1 or more of J loop, doing c,r=r+a∗b using 64x64=128, but assume inner j loop.
kCASPER_OpMul6464Reduce Walking 1 or more of J loop, doing c,r=r+a∗b using 64x64=128, but sum all of w.
kCASPER_OpAdd64 Walking 1 or more of J loop, doing c,r[-1]=r+a∗b using 64x64=128, but skip 1st write.
kCASPER_OpSub64 Walking add with off_AB, and in/out off_RES doing c,r=r+a+c using 64+64=65.
kCASPER_OpDouble64 Walking subtract with off_AB, and in/out off_RES doing r=r-a using 64-64=64, with last borrow implicit if any.
kCASPER_OpXor64 Walking add to self with off_RES doing c,r=r+r+c using 64+64=65.
kCASPER_OpRSub64 Walking XOR with off_AB, and in/out off_RES doing r=r^a using 64^64=64.
kCASPER_OpShiftLeft32 Walking subtract with off_AB, and in/out off_RES using r=a-r.
kCASPER_OpShiftRight32 Walking shift left doing r1,r=(b∗D)|r1, where D is 2^amt and is loaded by app (off_CD not used)
kCASPER_OpCopy Walking shift right doing r,r1=(b∗D)|r1, where D is 2^(32-amt) and is loaded by app (off_CD not used) and off_RES starts at MSW.
kCASPER_OpRemask Copy from ABoff to resoff, 64b at a time.
kCASPER_OpFill Copy and mask from ABoff to resoff, 64b at a time.
kCASPER_OpZero Fill RESOFF using 64 bits at a time with value in A and B.
kCASPER_OpCompare Fill RESOFF using 64 bits at a time of 0s.
kCASPER_OpCompareFast Compare two arrays, running all the way to the end.

8.6.3.2 enum casper_algo_t

Enumerator

kCASPER_ECC_P256 ECC_P256.
kCASPER_ECC_P384 ECC_P384.
kCASPER_ECC_P521 ECC_P521.

8.6.4 Function Documentation

8.6.4.1 void CASPER_Init (CASPER_Type ∗ *base*)

Enable clock and disable reset for CASPER.

Parameters

base	CASPER base address

8.6.4.2 void CASPER_Deinit (CASPER_Type * *base*)

Disable clock and enable reset.

Parameters

base	CASPER base address

8.7 casper_driver_pkha

8.7.1 Overview

Functions

- void CASPER_ModExp (CASPER_Type *base, const uint8_t *signature, const uint8_t *pubN, size_t wordLen, uint32_t pubE, uint8_t *plaintext)
  Performs modular exponentiation - (A^E) mod N.
- void CASPER_ecc_init (casper_algo_t curve)
  Initialize prime modulus mod in Casper memory .
- void CASPER_ECC_SECP256R1_Mul (CASPER_Type *base, uint32_t resX[8], uint32_t resY[8], uint32_t X[8], uint32_t Y[8], uint32_t scalar[8])
  Performs ECC secp256r1 point single scalar multiplication.
- void CASPER_ECC_SECP256R1_MulAdd (CASPER_Type *base, uint32_t resX[8], uint32_t resY[8], uint32_t X1[8], uint32_t Y1[8], uint32_t scalar1[8], uint32_t X2[8], uint32_t Y2[8], uint32_t scalar2[8])
  Performs ECC secp256r1 point double scalar multiplication.
- void CASPER_ECC_SECP384R1_Mul (CASPER_Type *base, uint32_t resX[12], uint32_t resY[12], uint32_t X[12], uint32_t Y[12], uint32_t scalar[12])
  Performs ECC secp384r1 point single scalar multiplication.
- void CASPER_ECC_SECP384R1_MulAdd (CASPER_Type *base, uint32_t resX[12], uint32_t resY[12], uint32_t X1[12], uint32_t Y1[12], uint32_t scalar1[12], uint32_t X2[12], uint32_t Y2[12], uint32_t scalar2[12])
  Performs ECC secp384r1 point double scalar multiplication.
- void CASPER_ECC_SECP521R1_Mul (CASPER_Type *base, uint32_t resX[18], uint32_t resY[18], uint32_t X[18], uint32_t Y[18], uint32_t scalar[18])
  Performs ECC secp521r1 point single scalar multiplication.
- void CASPER_ECC_SECP521R1_MulAdd (CASPER_Type *base, uint32_t resX[18], uint32_t resY[18], uint32_t X1[18], uint32_t Y1[18], uint32_t scalar1[18], uint32_t X2[18], uint32_t Y2[18], uint32_t scalar2[18])
  Performs ECC secp521r1 point double scalar multiplication.

8.7.2 Function Documentation

8.7.2.1 void CASPER_ModExp (CASPER_Type * *base*, const uint8_t * *signature*, const uint8_t * *pubN*, size_t *wordLen*, uint32_t *pubE*, uint8_t * *plaintext*)

This function performs modular exponentiation.

Parameters

	base	CASPER base address
	signature	first addend (in little endian format)
	pubN	modulus (in little endian format)
	wordLen	Size of pubN in bytes
	pubE	exponent
out	*plaintext*	Output array to store result of operation (in little endian format)

8.7.2.2 void CASPER_ecc_init (casper_algo_t *curve*)

Set the prime modulus mod in Casper memory and set N_wordlen according to selected algorithm.

Parameters

curve	elliptic curve algoritm

8.7.2.3 void CASPER_ECC_SECP256R1_Mul (CASPER_Type * *base,* uint32_t *resX[8],* uint32_t *resY[8],* uint32_t *X[8],* uint32_t *Y[8],* uint32_t *scalar[8]*)

This function performs ECC secp256r1 point single scalar multiplication [resX; resY] = scalar * [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate in normal form, little endian.
out	*resY*	Output Y affine coordinate in normal form, little endian.
	X	Input X affine coordinate in normal form, little endian.
	Y	Input Y affine coordinate in normal form, little endian.
	scalar	Input scalar integer, in normal form, little endian.

8.7.2.4 void CASPER_ECC_SECP256R1_MulAdd (CASPER_Type * *base,* uint32_t *resX[8],* uint32_t *resY[8],* uint32_t *X1[8],* uint32_t *Y1[8],* uint32_t *scalar1[8],* uint32_t *X2[8],* uint32_t *Y2[8],* uint32_t *scalar2[8]*)

This function performs ECC secp256r1 point double scalar multiplication [resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All

arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate.
out	*resY*	Output Y affine coordinate.
	X1	Input X1 affine coordinate.
	Y1	Input Y1 affine coordinate.
	scalar1	Input scalar1 integer.
	X2	Input X2 affine coordinate.
	Y2	Input Y2 affine coordinate.
	scalar2	Input scalar2 integer.

8.7.2.5 void CASPER_ECC_SECP384R1_Mul (CASPER_Type * *base*, uint32_t *resX[12]*, uint32_t *resY[12]*, uint32_t *X[12]*, uint32_t *Y[12]*, uint32_t *scalar[12]*)

This function performs ECC secp384r1 point single scalar multiplication [resX; resY] = scalar * [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate in normal form, little endian.
out	*resY*	Output Y affine coordinate in normal form, little endian.
	X	Input X affine coordinate in normal form, little endian.
	Y	Input Y affine coordinate in normal form, little endian.
	scalar	Input scalar integer, in normal form, little endian.

8.7.2.6 void CASPER_ECC_SECP384R1_MulAdd (CASPER_Type * *base*, uint32_t *resX[12]*, uint32_t *resY[12]*, uint32_t *X1[12]*, uint32_t *Y1[12]*, uint32_t *scalar1[12]*, uint32_t *X2[12]*, uint32_t *Y2[12]*, uint32_t *scalar2[12]*)

This function performs ECC secp384r1 point double scalar multiplication [resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate.
out	*resY*	Output Y affine coordinate.
	X1	Input X1 affine coordinate.
	Y1	Input Y1 affine coordinate.
	scalar1	Input scalar1 integer.
	X2	Input X2 affine coordinate.
	Y2	Input Y2 affine coordinate.
	scalar2	Input scalar2 integer.

8.7.2.7 void CASPER_ECC_SECP521R1_Mul (CASPER_Type * *base*, uint32_t *resX[18]*, uint32_t *resY[18]*, uint32_t *X[18]*, uint32_t *Y[18]*, uint32_t *scalar[18]*)

This function performs ECC secp521r1 point single scalar multiplication [resX; resY] = scalar * [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate in normal form, little endian.
out	*resY*	Output Y affine coordinate in normal form, little endian.
	X	Input X affine coordinate in normal form, little endian.
	Y	Input Y affine coordinate in normal form, little endian.
	scalar	Input scalar integer, in normal form, little endian.

8.7.2.8 void CASPER_ECC_SECP521R1_MulAdd (CASPER_Type * *base*, uint32_t *resX[18]*, uint32_t *resY[18]*, uint32_t *X1[18]*, uint32_t *Y1[18]*, uint32_t *scalar1[18]*, uint32_t *X2[18]*, uint32_t *Y2[18]*, uint32_t *scalar2[18]*)

This function performs ECC secp521r1 point double scalar multiplication [resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate.
out	*resY*	Output Y affine coordinate.
	X1	Input X1 affine coordinate.
	Y1	Input Y1 affine coordinate.
	scalar1	Input scalar1 integer.
	X2	Input X2 affine coordinate.
	Y2	Input Y2 affine coordinate.
	scalar2	Input scalar2 integer.

Chapter 9
CMP: Analog Comparator Driver

9.1 Overview

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

9.2 Function groups

The driver provides a set of functions to set two input sources of the on-chip comparator and compare the voltage of them.

9.2.1 Initialization and deinitialization

The function CMP_Init() initializes the CMP with specified configurations. The function CMP_GetDefaultConfig() gets the default configurations.

The function CMP_Deinit() disables the module clock.

9.2.2 Compare

The function CMP_SetInputChannels() configures the P-side and N-side input sources.

The function CMP_SetVREF() sets the reference voltage which can be dedicated to input 0 of both P and N sides.

The function CMP_GetOutput() gets the compare result of the two sides.

9.2.3 Interrupt

Provides functions to enable/disable/clear CMP interrupts.

The function CMP_EnableFilteredInterruptSource() allows users to select which analog comparator output (filtered or un-filtered) is used for interrupt detection.

9.2.4 Status

Provides functions to get the CMP status.

9.3 Typical use case

9.3.1 Polling Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/cmp_1

9.3.2 Interrupt Configuration

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/cmp_1

Data Structures

- struct cmp_config_t
  CMP configuration structure. *More...*

Enumerations

- enum _cmp_input_mux {
  kCMP_InputVREF = 0U,
  kCMP_Input1 = 1U,
  kCMP_Input2 = 2U,
  kCMP_Input3 = 3U,
  kCMP_Input4 = 4U,
  kCMP_Input5 = 5U }
  CMP input mux for positive and negative sides.
- enum _cmp_interrupt_type {
  kCMP_EdgeDisable = 0U,
  kCMP_EdgeRising = 2U,
  kCMP_EdgeFalling = 4U,
  kCMP_EdgeRisingFalling = 6U,
  kCMP_LevelDisable = 1U,
  kCMP_LevelHigh = 3U,
  kCMP_LevelLow = 5U }
  CMP interrupt type.
- enum cmp_vref_source_t {
  KCMP_VREFSourceVDDA = 1U,
  KCMP_VREFSourceInternalVREF = 0U }
  CMP Voltage Reference source.
- enum cmp_filtercgf_samplemode_t {
  kCMP_FilterSampleMode0 = 0U,
  kCMP_FilterSampleMode1 = 1U,
  kCMP_FilterSampleMode2 = 2U,
  kCMP_FilterSampleMode3 = 3U }
  CMP Filter sample mode.

- enum cmp_filtercgf_clkdiv_t {
  kCMP_FilterClockDivide1 = 0U,
  kCMP_FilterClockDivide2 = 1U,
  kCMP_FilterClockDivide4 = 2U,
  kCMP_FilterClockDivide8 = 3U,
  kCMP_FilterClockDivide16 = 4U,
  kCMP_FilterClockDivide32 = 5U,
  kCMP_FilterClockDivide64 = 6U }
  CMP Filter clock divider.

Driver version

- #define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2U, 2U, 1U))
  Driver version 2.2.1.

Initialization and deinitialization

- void CMP_Init (const cmp_config_t *config)
  CMP initialization.
- void CMP_Deinit (void)
  CMP deinitialization.
- void CMP_GetDefaultConfig (cmp_config_t *config)
  Initializes the CMP user configuration structure.

Compare Interface

- static void **CMP_SetInputChannels** (uint8_t positiveChannel, uint8_t negativeChannel)
- void CMP_SetVREF (const cmp_vref_config_t *config)
  Configures the VREFINPUT.
- static bool CMP_GetOutput (void)
  Get CMP compare output.

Interrupt Interface

- static void CMP_EnableInterrupt (uint32_t type)
  CMP enable interrupt.
- static void CMP_DisableInterrupt (void)
  CMP disable interrupt.
- static void CMP_ClearInterrupt (void)
  CMP clear interrupt.
- static void CMP_EnableFilteredInterruptSource (bool enable)
  Select which Analog comparator output (filtered or un-filtered) is used for interrupt detection.

Status Interface

- static bool CMP_GetPreviousInterruptStatus (void)
  Get CMP interrupt status before interupt enable.
- static bool CMP_GetInterruptStatus (void)
  Get CMP interrupt status after interupt enable.

Filter Interface

- static void CMP_FilterSampleConfig (cmp_filtercgf_samplemode_t filterSampleMode, cmp_filtercgf_clkdiv_t filterClockDivider)
  CMP Filter Sample Config.

9.4 Data Structure Documentation

9.4.1 struct cmp_config_t

Data Fields

- bool enableHysteresis
  Enable hysteresis.
- bool enableLowPower
  Enable low power mode.

Field Documentation

(1) bool cmp_config_t::enableHysteresis

(2) bool cmp_config_t::enableLowPower

9.5 Macro Definition Documentation

9.5.1 #define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2U, 2U, 1U))

9.6 Enumeration Type Documentation

9.6.1 enum _cmp_input_mux

Enumerator

kCMP_InputVREF Cmp input from VREF.
kCMP_Input1 Cmp input source 1.
kCMP_Input2 Cmp input source 2.
kCMP_Input3 Cmp input source 3.
kCMP_Input4 Cmp input source 4.
kCMP_Input5 Cmp input source 5.

9.6.2 enum _cmp_interrupt_type

Enumerator

kCMP_EdgeDisable Disable edge interupt.
kCMP_EdgeRising Interrupt on falling edge.
kCMP_EdgeFalling Interrupt on rising edge.

kCMP_EdgeRisingFalling Interrupt on both rising and falling edges.
kCMP_LevelDisable Disable level interupt.
kCMP_LevelHigh Interrupt on high level.
kCMP_LevelLow Interrupt on low level.

9.6.3 enum cmp_vref_source_t

Enumerator

KCMP_VREFSourceVDDA Select VDDA as VREF.
KCMP_VREFSourceInternalVREF Select internal VREF as VREF.

9.6.4 enum cmp_filtercgf_samplemode_t

Enumerator

kCMP_FilterSampleMode0 Bypass mode. Filtering is disabled.
kCMP_FilterSampleMode1 Filter 1 clock period.
kCMP_FilterSampleMode2 Filter 2 clock period.
kCMP_FilterSampleMode3 Filter 3 clock period.

9.6.5 enum cmp_filtercgf_clkdiv_t

Enumerator

kCMP_FilterClockDivide1 Filter clock period duration equals 1 analog comparator clock period.
kCMP_FilterClockDivide2 Filter clock period duration equals 2 analog comparator clock period.
kCMP_FilterClockDivide4 Filter clock period duration equals 4 analog comparator clock period.
kCMP_FilterClockDivide8 Filter clock period duration equals 8 analog comparator clock period.
kCMP_FilterClockDivide16 Filter clock period duration equals 16 analog comparator clock period.

kCMP_FilterClockDivide32 Filter clock period duration equals 32 analog comparator clock period.

kCMP_FilterClockDivide64 Filter clock period duration equals 64 analog comparator clock period.

9.7 Function Documentation

9.7.1 void CMP_Init (const cmp_config_t * *config*)

This function enables the CMP module and do necessary settings.

Parameters

config	Pointer to the configuration structure.

9.7.2 void CMP_Deinit (void)

This function gates the clock for CMP module.

9.7.3 void CMP_GetDefaultConfig (cmp_config_t * *config*)

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

```
*   config->enableHysteresis    = true;
*   config->enableLowPower      = true;
*   config->filterClockDivider  = kCMP_FilterClockDivide1;
*   config->filterSampleMode    = kCMP_FilterSampleMode0;
*
```

Parameters

config	Pointer to the configuration structure.

9.7.4 void CMP_SetVREF (const cmp_vref_config_t * *config*)

Parameters

config	Pointer to the configuration structure.

9.7.5 static bool CMP_GetOutput (void) [inline], [static]

Returns

The output result. true: voltage on positive side is greater than negative side. false: voltage on positive side is lower than negative side.

9.7.6 static void CMP_EnableInterrupt (uint32_t *type*) [inline], [static]

Parameters

type	CMP interrupt type. See "_cmp_interrupt_type".

9.7.7 static void CMP_EnableFilteredInterruptSource (bool *enable*) [inline], [static]

Parameters

enable	false: Select Analog Comparator raw output (unfiltered) as input for interrupt detection. true: Select Analog Comparator filtered output as input for interrupt detection.

Note

: When CMP is configured as the wakeup source in power down mode, this function must use the raw output as the interupt source, that is, call this function and set parameter enable to false.

9.7.8 static bool CMP_GetPreviousInterruptStatus (void) [inline], [static]

Returns

Interrupt status. true: interrupt pending, false: no interrupt pending.

9.7.9 static bool CMP_GetInterruptStatus (void) [inline], [static]

Returns

Interrupt status. true: interrupt pending, false: no interrupt pending.

9.7.10 static void CMP_FilterSampleConfig (cmp_filtercgf_samplemode_t *filterSampleMode,* cmp_filtercgf_clkdiv_t *filterClockDivider*) [inline], [static]

This function allows the users to configure the sampling mode and clock divider of the CMP Filter.

Parameters

filterSample-Mode	CMP Select filter sample mode
filterClock-Divider	CMP Set fileter clock divider

Chapter 10
Common Driver

10.1 Overview

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

Macros

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

Typedefs

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

Enumerations

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

kStatusGroup_POWER_MANAGER = 159 }
Status group numbers.
- enum {
kStatus_Success = MAKE_STATUS(kStatusGroup_Generic, 0),
kStatus_Fail = MAKE_STATUS(kStatusGroup_Generic, 1),
kStatus_ReadOnly = MAKE_STATUS(kStatusGroup_Generic, 2),
kStatus_OutOfRange = MAKE_STATUS(kStatusGroup_Generic, 3),
kStatus_InvalidArgument = MAKE_STATUS(kStatusGroup_Generic, 4),
kStatus_Timeout = MAKE_STATUS(kStatusGroup_Generic, 5),
kStatus_NoTransferInProgress,
kStatus_Busy = MAKE_STATUS(kStatusGroup_Generic, 7),
kStatus_NoData }
Generic status return codes.

Functions

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

Driver version

- #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))
common driver version.

Min/max macros

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

UINT16_MAX/UINT32_MAX value

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

Suppress fallthrough warning macro

- #define **SUPPRESS_FALL_THROUGH_WARNING**()

10.2 Macro Definition Documentation

10.2.1 #define FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ 1

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

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

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

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

10.2.4 #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))

10.2.5 #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U

10.2.6 #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U

10.2.7 #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U

10.2.8 #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U

10.2.9 #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U

10.2.10 #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U

10.2.11 #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U

10.2.12 #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U

10.2.13 #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U

10.2.14 #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U

10.2.15 #define DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U

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

10.3 Typedef Documentation

10.3.1 typedef int32_t status_t

10.4 Enumeration Type Documentation

10.4.1 enum _status_groups

Enumerator

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

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

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

10.4.2 anonymous enum

Enumerator

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

10.5 Function Documentation

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

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

Parameters

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

Return values

The	allocated memory.

10.5.2 void SDK_Free (void * *ptr*)

Parameters

ptr	The memory to be release.

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

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

Parameters

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

Chapter 11
CTIMER: Standard counter/timers

11.1 Overview

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

11.2 Function groups

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

11.2.1 Initialization and deinitialization

The function CTIMER_Init() initializes the cTimer with specified configurations. The function CTIMER_GetDefaultConfig() gets the default configurations. The initialization function configures the counter/timer mode and input selection when running in counter mode.

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

11.2.2 PWM Operations

The function CTIMER_SetupPwm() sets up channels for PWM output. Each channel has its own duty cycle, however the same PWM period is applied to all channels requesting the PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100 0=inactive signal(0% duty cycle) and 100=always active signal (100% duty cycle).

The function CTIMER_UpdatePwmDutycycle() updates the PWM signal duty cycle of a particular channel.

11.2.3 Match Operation

The function CTIMER_SetupMatch() sets up channels for match operation. Each channel is configured with a match value: if the counter should stop on match, if counter should reset on match, and output pin action. The output signal can be cleared, set, or toggled on match.

11.2.4 Input capture operations

The function CTIMER_SetupCapture() sets up an channel for input capture. The user can specify the capture edge and if a interrupt should be generated when processing the input signal.

11.3 Typical use case

11.3.1 Match example

Set up a match channel to toggle output when a match occurs. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/ctimer

11.3.2 PWM output example

Set up a channel for PWM output. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/ctimer

Files

- file fsl_ctimer.h

Data Structures

- struct ctimer_match_config_t
  Match configuration. *More...*
- struct ctimer_config_t
  Timer configuration structure. *More...*

Enumerations

- enum ctimer_capture_channel_t {
  kCTIMER_Capture_0 = 0U,
  kCTIMER_Capture_1,
  kCTIMER_Capture_2 }
  List of Timer capture channels.
- enum ctimer_capture_edge_t {
  kCTIMER_Capture_RiseEdge = 1U,
  kCTIMER_Capture_FallEdge = 2U,
  kCTIMER_Capture_BothEdge = 3U }
  List of capture edge options.
- enum ctimer_match_t {
  kCTIMER_Match_0 = 0U,
  kCTIMER_Match_1,
  kCTIMER_Match_2,
  kCTIMER_Match_3 }
  List of Timer match registers.
- enum ctimer_external_match_t {
  kCTIMER_External_Match_0 = (1U << 0),
  kCTIMER_External_Match_1 = (1U << 1),
  kCTIMER_External_Match_2 = (1U << 2),
  kCTIMER_External_Match_3 = (1U << 3) }

List of external match.

- enum ctimer_match_output_control_t {
  kCTIMER_Output_NoAction = 0U,
  kCTIMER_Output_Clear,
  kCTIMER_Output_Set,
  kCTIMER_Output_Toggle }
  List of output control options.
- enum ctimer_timer_mode_t
  List of Timer modes.
- enum ctimer_interrupt_enable_t {
  kCTIMER_Match0InterruptEnable = CTIMER_MCR_MR0I_MASK,
  kCTIMER_Match1InterruptEnable = CTIMER_MCR_MR1I_MASK,
  kCTIMER_Match2InterruptEnable = CTIMER_MCR_MR2I_MASK,
  kCTIMER_Match3InterruptEnable = CTIMER_MCR_MR3I_MASK,
  kCTIMER_Capture0InterruptEnable = CTIMER_CCR_CAP0I_MASK,
  kCTIMER_Capture1InterruptEnable = CTIMER_CCR_CAP1I_MASK,
  kCTIMER_Capture2InterruptEnable = CTIMER_CCR_CAP2I_MASK }
  List of Timer interrupts.
- enum ctimer_status_flags_t {
  kCTIMER_Match0Flag = CTIMER_IR_MR0INT_MASK,
  kCTIMER_Match1Flag = CTIMER_IR_MR1INT_MASK,
  kCTIMER_Match2Flag = CTIMER_IR_MR2INT_MASK,
  kCTIMER_Match3Flag = CTIMER_IR_MR3INT_MASK,
  kCTIMER_Capture0Flag = CTIMER_IR_CR0INT_MASK,
  kCTIMER_Capture1Flag = CTIMER_IR_CR1INT_MASK,
  kCTIMER_Capture2Flag = CTIMER_IR_CR2INT_MASK }
  List of Timer flags.
- enum ctimer_callback_type_t {
  kCTIMER_SingleCallback,
  kCTIMER_MultipleCallback }
  Callback type when registering for a callback.

Functions

- void CTIMER_SetupMatch (CTIMER_Type *base, ctimer_match_t matchChannel, const ctimer_match_config_t *config)
  Setup the match register.
- uint32_t CTIMER_GetOutputMatchStatus (CTIMER_Type *base, uint32_t matchChannel)
  Get the status of output match.
- void CTIMER_SetupCapture (CTIMER_Type *base, ctimer_capture_channel_t capture, ctimer_capture_edge_t edge, bool enableInt)
  Setup the capture.
- static uint32_t CTIMER_GetTimerCountValue (CTIMER_Type *base)
  Get the timer count value from TC register.
- void CTIMER_RegisterCallBack (CTIMER_Type *base, ctimer_callback_t *cb_func, ctimer_callback_type_t cb_type)
  Register callback.
- static void CTIMER_Reset (CTIMER_Type *base)

Reset the counter.

Driver version

- #define FSL_CTIMER_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
  Version 2.2.1.

Initialization and deinitialization

- void CTIMER_Init (CTIMER_Type *base, const ctimer_config_t *config)
  Ungates the clock and configures the peripheral for basic operation.
- void CTIMER_Deinit (CTIMER_Type *base)
  Gates the timer clock.
- void CTIMER_GetDefaultConfig (ctimer_config_t *config)
  Fills in the timers configuration structure with the default settings.

PWM setup operations

- status_t CTIMER_SetupPwmPeriod (CTIMER_Type *base, const ctimer_match_t pwmPeriodChannel, ctimer_match_t matchChannel, uint32_t pwmPeriod, uint32_t pulsePeriod, bool enableInt)
  Configures the PWM signal parameters.
- status_t CTIMER_SetupPwm (CTIMER_Type *base, const ctimer_match_t pwmPeriodChannel, ctimer_match_t matchChannel, uint8_t dutyCyclePercent, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, bool enableInt)
  Configures the PWM signal parameters.
- static void CTIMER_UpdatePwmPulsePeriod (CTIMER_Type *base, ctimer_match_t matchChannel, uint32_t pulsePeriod)
  Updates the pulse period of an active PWM signal.
- void CTIMER_UpdatePwmDutycycle (CTIMER_Type *base, const ctimer_match_t pwmPeriodChannel, ctimer_match_t matchChannel, uint8_t dutyCyclePercent)
  Updates the duty cycle of an active PWM signal.

Interrupt Interface

- static void CTIMER_EnableInterrupts (CTIMER_Type *base, uint32_t mask)
  Enables the selected Timer interrupts.
- static void CTIMER_DisableInterrupts (CTIMER_Type *base, uint32_t mask)
  Disables the selected Timer interrupts.
- static uint32_t CTIMER_GetEnabledInterrupts (CTIMER_Type *base)
  Gets the enabled Timer interrupts.

Status Interface

- static uint32_t CTIMER_GetStatusFlags (CTIMER_Type *base)
  Gets the Timer status flags.
- static void CTIMER_ClearStatusFlags (CTIMER_Type *base, uint32_t mask)
  Clears the Timer status flags.

Counter Start and Stop

- static void CTIMER_StartTimer (CTIMER_Type *base)
  Starts the Timer counter.
- static void CTIMER_StopTimer (CTIMER_Type *base)
  Stops the Timer counter.

11.4 Data Structure Documentation

11.4.1 struct ctimer_match_config_t

This structure holds the configuration settings for each match register.

Data Fields

- uint32_t matchValue
  This is stored in the match register.
- bool enableCounterReset
  true: Match will reset the counter false: Match will not reser the counter
- bool enableCounterStop
  true: Match will stop the counter false: Match will not stop the counter
- ctimer_match_output_control_t outControl
  Action to be taken on a match on the EM bit/output.
- bool outPinInitState
  Initial value of the EM bit/output.
- bool enableInterrupt
  true: Generate interrupt upon match false: Do not generate interrupt on match

11.4.2 struct ctimer_config_t

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

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

Data Fields

- ctimer_timer_mode_t mode
  Timer mode.
- ctimer_capture_channel_t input
  Input channel to increment the timer, used only in timer modes that rely on this input signal to increment TC.
- uint32_t prescale
  Prescale value.

11.5 Enumeration Type Documentation

11.5.1 enum ctimer_capture_channel_t

Enumerator

kCTIMER_Capture_0 Timer capture channel 0.
kCTIMER_Capture_1 Timer capture channel 1.
kCTIMER_Capture_2 Timer capture channel 2.

11.5.2 enum ctimer_capture_edge_t

Enumerator

kCTIMER_Capture_RiseEdge Capture on rising edge.
kCTIMER_Capture_FallEdge Capture on falling edge.
kCTIMER_Capture_BothEdge Capture on rising and falling edge.

11.5.3 enum ctimer_match_t

Enumerator

kCTIMER_Match_0 Timer match register 0.
kCTIMER_Match_1 Timer match register 1.
kCTIMER_Match_2 Timer match register 2.
kCTIMER_Match_3 Timer match register 3.

11.5.4 enum ctimer_external_match_t

Enumerator

kCTIMER_External_Match_0 External match 0.
kCTIMER_External_Match_1 External match 1.
kCTIMER_External_Match_2 External match 2.
kCTIMER_External_Match_3 External match 3.

11.5.5 enum ctimer_match_output_control_t

Enumerator

kCTIMER_Output_NoAction No action is taken.

kCTIMER_Output_Clear Clear the EM bit/output to 0.
kCTIMER_Output_Set Set the EM bit/output to 1.
kCTIMER_Output_Toggle Toggle the EM bit/output.

11.5.6 enum ctimer_interrupt_enable_t

Enumerator

kCTIMER_Match0InterruptEnable Match 0 interrupt.
kCTIMER_Match1InterruptEnable Match 1 interrupt.
kCTIMER_Match2InterruptEnable Match 2 interrupt.
kCTIMER_Match3InterruptEnable Match 3 interrupt.
kCTIMER_Capture0InterruptEnable Capture 0 interrupt.
kCTIMER_Capture1InterruptEnable Capture 1 interrupt.
kCTIMER_Capture2InterruptEnable Capture 2 interrupt.

11.5.7 enum ctimer_status_flags_t

Enumerator

kCTIMER_Match0Flag Match 0 interrupt flag.
kCTIMER_Match1Flag Match 1 interrupt flag.
kCTIMER_Match2Flag Match 2 interrupt flag.
kCTIMER_Match3Flag Match 3 interrupt flag.
kCTIMER_Capture0Flag Capture 0 interrupt flag.
kCTIMER_Capture1Flag Capture 1 interrupt flag.
kCTIMER_Capture2Flag Capture 2 interrupt flag.

11.5.8 enum ctimer_callback_type_t

When registering a callback an array of function pointers is passed the size could be 1 or 8, the callback type will tell that.

Enumerator

kCTIMER_SingleCallback Single Callback type where there is only one callback for the timer. based on the status flags different channels needs to be handled differently
kCTIMER_MultipleCallback Multiple Callback type where there can be 8 valid callbacks, one per channel. for both match/capture

11.6 Function Documentation

11.6.1 void CTIMER_Init (CTIMER_Type * *base*, const ctimer_config_t * *config*)

Note

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

Parameters

base	Ctimer peripheral base address
config	Pointer to the user configuration structure.

11.6.2 void CTIMER_Deinit (CTIMER_Type * *base*)

Parameters

base	Ctimer peripheral base address

11.6.3 void CTIMER_GetDefaultConfig (ctimer_config_t * *config*)

The default values are:

```
*   config->mode = kCTIMER_TimerMode;
*   config->input = kCTIMER_Capture_0;
*   config->prescale = 0;
*
```

Parameters

config	Pointer to the user configuration structure.

11.6.4 status_t CTIMER_SetupPwmPeriod (CTIMER_Type * *base*, const ctimer_match_t *pwmPeriodChannel*, ctimer_match_t *matchChannel*, uint32_t *pwmPeriod*, uint32_t *pulsePeriod*, bool *enableInt*)

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

Note

When setting PWM output from multiple output pins, all should use the same PWM period

Parameters

base	Ctimer peripheral base address
pwmPeriod-Channel	Specify the channel to control the PWM period
matchChannel	Match pin to be used to output the PWM signal
pwmPeriod	PWM period match value
pulsePeriod	Pulse width match value
enableInt	Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

11.6.5 status_t CTIMER_SetupPwm (CTIMER_Type * *base*, const ctimer_match_t *pwmPeriodChannel*, ctimer_match_t *matchChannel*, uint8_t *dutyCyclePercent*, uint32_t *pwmFreq_Hz*, uint32_t *srcClock_Hz*, bool *enableInt*)

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

Note

When setting PWM output from multiple output pins, all should use the same PWM frequency. Please use CTIMER_SetupPwmPeriod to set up the PWM with high resolution.

Parameters

base	Ctimer peripheral base address
pwmPeriod-Channel	Specify the channel to control the PWM period

matchChannel	Match pin to be used to output the PWM signal
dutyCycle-Percent	PWM pulse width; the value should be between 0 to 100
pwmFreq_Hz	PWM signal frequency in Hz
srcClock_Hz	Timer counter clock in Hz
enableInt	Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

11.6.6 static void CTIMER_UpdatePwmPulsePeriod (CTIMER_Type * *base*, ctimer_match_t *matchChannel*, uint32_t *pulsePeriod*) [inline], [static]

Parameters

base	Ctimer peripheral base address
matchChannel	Match pin to be used to output the PWM signal
pulsePeriod	New PWM pulse width match value

11.6.7 void CTIMER_UpdatePwmDutycycle (CTIMER_Type * *base*, const ctimer_match_t *pwmPeriodChannel*, ctimer_match_t *matchChannel*, uint8_t *dutyCyclePercent*)

Note

Please use CTIMER_SetupPwmPeriod to update the PWM with high resolution. This function can manually assign the specified channel to set the PWM cycle.

Parameters

base	Ctimer peripheral base address
pwmPeriod-Channel	Specify the channel to control the PWM period

matchChannel	Match pin to be used to output the PWM signal
dutyCycle-Percent	New PWM pulse width; the value should be between 0 to 100

11.6.8 void CTIMER_SetupMatch (CTIMER_Type * *base*, ctimer_match_t *matchChannel*, const ctimer_match_config_t * *config*)

User configuration is used to setup the match value and action to be taken when a match occurs.

Parameters

base	Ctimer peripheral base address
matchChannel	Match register to configure
config	Pointer to the match configuration structure

11.6.9 uint32_t CTIMER_GetOutputMatchStatus (CTIMER_Type * *base*, uint32_t *matchChannel*)

This function gets the status of output MAT, whether or not this output is connected to a pin. This status is driven to the MAT pins if the match function is selected via IOCON. 0 = LOW. 1 = HIGH.

Parameters

base	Ctimer peripheral base address
matchChannel	External match channel, user can obtain the status of multiple match channels at the same time by using the logic of "|" enumeration ctimer_external_match_t

Returns

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

11.6.10 void CTIMER_SetupCapture (CTIMER_Type * *base*, ctimer_capture-_channel_t *capture*, ctimer_capture_edge_t *edge*, bool *enableInt*)

Parameters

base	Ctimer peripheral base address
capture	Capture channel to configure
edge	Edge on the channel that will trigger a capture
enableInt	Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

11.6.11 static uint32_t CTIMER_GetTimerCountValue (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address.

Returns

return the timer count value.

11.6.12 void CTIMER_RegisterCallBack (CTIMER_Type * *base,* ctimer_callback_t * *cb_func,* ctimer_callback_type_t *cb_type*)

Parameters

base	Ctimer peripheral base address
cb_func	callback function
cb_type	callback function type, singular or multiple

11.6.13 static void CTIMER_EnableInterrupts (CTIMER_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

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

11.6.14 static void CTIMER_DisableInterrupts (CTIMER_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

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

11.6.15 static uint32_t CTIMER_GetEnabledInterrupts (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

Returns

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

11.6.16 static uint32_t CTIMER_GetStatusFlags (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

Returns

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

11.6.17 static void CTIMER_ClearStatusFlags (CTIMER_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	Ctimer peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration ctimer_status_flags_t

11.6.18 static void CTIMER_StartTimer (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

11.6.19 static void CTIMER_StopTimer (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

11.6.20 static void CTIMER_Reset (CTIMER_Type * *base*) [inline], [static]

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

Parameters

base	Ctimer peripheral base address

Chapter 12
FLEXCOMM: FLEXCOMM Driver

12.1 Overview

The MCUXpresso SDK provides a generic driver and multiple protocol-specific FLEXCOMM drivers for the FLEXCOMM module of MCUXpresso SDK devices.

Modules

- FLEXCOMM Driver

12.2 FLEXCOMM Driver

12.2.1 Overview

Typedefs

- typedef void(∗ flexcomm_irq_handler_t)(void ∗base, void ∗handle)
  Typedef for interrupt handler.

Enumerations

- enum FLEXCOMM_PERIPH_T {
  FLEXCOMM_PERIPH_NONE,
  FLEXCOMM_PERIPH_USART,
  FLEXCOMM_PERIPH_SPI,
  FLEXCOMM_PERIPH_I2C,
  FLEXCOMM_PERIPH_I2S_TX,
  FLEXCOMM_PERIPH_I2S_RX }
  FLEXCOMM peripheral modes.

Functions

- uint32_t FLEXCOMM_GetInstance (void ∗base)
  Returns instance number for FLEXCOMM module with given base address.
- status_t FLEXCOMM_Init (void ∗base, FLEXCOMM_PERIPH_T periph)
  Initializes FLEXCOMM and selects peripheral mode according to the second parameter.
- void FLEXCOMM_SetIRQHandler (void ∗base, flexcomm_irq_handler_t handler, void ∗flexcomm-Handle)
  Sets IRQ handler for given FLEXCOMM module.

Variables

- IRQn_Type const kFlexcommIrqs [ ]
  Array with IRQ number for each FLEXCOMM module.

Driver version

- #define FSL_FLEXCOMM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  FlexCOMM driver version 2.0.2.

12.2.2 Macro Definition Documentation

12.2.2.1 #define FSL_FLEXCOMM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

12.2.3 Typedef Documentation

12.2.3.1 typedef void(* flexcomm_irq_handler_t)(void *base, void *handle)

12.2.4 Enumeration Type Documentation

12.2.4.1 enum FLEXCOMM_PERIPH_T

Enumerator

FLEXCOMM_PERIPH_NONE No peripheral.
FLEXCOMM_PERIPH_USART USART peripheral.
FLEXCOMM_PERIPH_SPI SPI Peripheral.
FLEXCOMM_PERIPH_I2C I2C Peripheral.
FLEXCOMM_PERIPH_I2S_TX I2S TX Peripheral.
FLEXCOMM_PERIPH_I2S_RX I2S RX Peripheral.

12.2.5 Function Documentation

12.2.5.1 uint32_t FLEXCOMM_GetInstance (void * *base*)

12.2.5.2 status_t FLEXCOMM_Init (void * *base,* FLEXCOMM_PERIPH_T *periph*)

12.2.5.3 void FLEXCOMM_SetIRQHandler (void * *base,* flexcomm_irq_handler_t *handler,* void * *flexcommHandle*)

It is used by drivers register IRQ handler according to FLEXCOMM mode

12.2.6 Variable Documentation

12.2.6.1 IRQn_Type const kFlexcommIrqs[]

Chapter 13
I2C: Inter-Integrated Circuit Driver

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

13.2 Typical use case

13.2.1 Master Operation in functional method

```
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

/* Send start and slave address. */
I2C_MasterStart(EXAMPLE_I2C_MASTER_BASEADDR, 7-bit slave address,
      kI2C_Write/kI2C_Read);

/* Wait address sent out. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR)) & kI2C_IntPendingFlag))
{

}

if(status & kI2C_ReceiveNakFlag)
{
    return kStatus_I2C_Nak;
```

```
}

result = I2C_MasterWriteBlocking(EXAMPLE_I2C_MASTER_BASEADDR, txBuff, BUFFER_SIZE,
      kI2C_TransferDefaultFlag);

if(result)
{
    /* If error occours, send STOP. */
    I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
    return result;
}

while(!(I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR) & kI2C_IntPendingFlag))
{

}

/* Wait all data sent out, send STOP. */
I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
```

13.2.2 Master Operation in interrupt transactional method

```
i2c_master_handle_t g_m_handle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_handle_t *handle,
      status_t status, void *userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

I2C_MasterTransferCreateHandle(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle,
      i2c_master_callback, NULL);
I2C_MasterTransferNonBlocking(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle, &
      masterXfer);

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

13.2.3 Master Operation in DMA transactional method

```
i2c_master_dma_handle_t g_m_dma_handle;
dma_handle_t dmaHandle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_dma_handle_t *handle,
      status_t status, void *userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

DMA_EnableChannel(EXAMPLE_DMA, EXAMPLE_I2C_MASTER_CHANNEL);
DMA_CreateHandle(&dmaHandle, EXAMPLE_DMA, EXAMPLE_I2C_MASTER_CHANNEL);

I2C_MasterTransferCreateHandleDMA(EXAMPLE_I2C_MASTER_BASEADDR, &
      g_m_dma_handle, i2c_master_callback, NULL, &dmaHandle);
I2C_MasterTransferDMA(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_dma_handle, &masterXfer);

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

13.2.4 Slave Operation in functional method

```
i2c_slave_config_t slaveConfig;
uint8_t status;
status_t result = kStatus_Success;

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
       mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/kI2C_RangeMatch;
I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

/* Wait address match. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_SLAVE_BASEADDR)) & kI2C_AddressMatchFlag))
{

}
```

```
/* Slave transmit, master reading from slave. */
if (status & kI2C_TransferDirectionFlag)
{
    result = I2C_SlaveWriteBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}
else
{
    I2C_SlaveReadBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}

return result;
```

13.2.5 Slave Operation in interrupt transactional method

```
i2c_slave_config_t slaveConfig;
i2c_slave_handle_t g_s_handle;
volatile bool g_SlaveCompletionFlag = false;

static void i2c_slave_callback(I2C_Type *base, i2c_slave_transfer_t *xfer, void *
      userData)
{
    switch (xfer->event)
    {
        /*  Transmit request */
        case kI2C_SlaveTransmitEvent:
            /*  Update information for transmit process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /*  Receive request */
        case kI2C_SlaveReceiveEvent:
            /*  Update information for received process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /*  Transfer done */
        case kI2C_SlaveCompletionEvent:
            g_SlaveCompletionFlag = true;
            break;

        default:
            g_SlaveCompletionFlag = true;
            break;
    }
}

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
       mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/kI2C_RangeMatch;

I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

I2C_SlaveTransferCreateHandle(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
      i2c_slave_callback, NULL);

I2C_SlaveTransferNonBlocking(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
      kI2C_SlaveCompletionEvent);

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

```
g_SlaveCompletionFlag = false;
```

Modules

- I2C CMSIS Driver
- I2C DMA Driver
- I2C Driver
- I2C FreeRTOS Driver
- I2C Master Driver
- I2C Slave Driver

13.3 I2C Driver

13.3.1 Overview

Files

- file fsl_i2c.h

Macros

- #define I2C_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Retry times for waiting flag.
- #define I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK 1U /∗ Define to one means master ignores the last byte's nack and considers the transfer successful. ∗/
  Whether to ignore the nack signal of the last byte during master transmit.
- #define I2C_STAT_MSTCODE_IDLE (0U)
  Master Idle State Code.
- #define I2C_STAT_MSTCODE_RXREADY (1U)
  Master Receive Ready State Code.
- #define I2C_STAT_MSTCODE_TXREADY (2U)
  Master Transmit Ready State Code.
- #define I2C_STAT_MSTCODE_NACKADR (3U)
  Master NACK by slave on address State Code.
- #define I2C_STAT_MSTCODE_NACKDAT (4U)
  Master NACK by slave on data State Code.

Enumerations

- enum {
  kStatus_I2C_Busy = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 0),
  kStatus_I2C_Idle = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 1),
  kStatus_I2C_Nak,
  kStatus_I2C_InvalidParameter,
  kStatus_I2C_BitError = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 4),
  kStatus_I2C_ArbitrationLost = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 5),
  kStatus_I2C_NoTransferInProgress,
  kStatus_I2C_DmaRequestFail = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 7),
  kStatus_I2C_StartStopError = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 8),
  kStatus_I2C_UnexpectedState = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 9),
  kStatus_I2C_Timeout,
  kStatus_I2C_Addr_Nak = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 11),
  kStatus_I2C_EventTimeout = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 12),
  kStatus_I2C_SclLowTimeout = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 13) }
  I2C status return codes.

- enum _i2c_status_flags {
  kI2C_MasterPendingFlag = I2C_STAT_MSTPENDING_MASK,
  kI2C_MasterArbitrationLostFlag,
  kI2C_MasterStartStopErrorFlag,
  kI2C_MasterIdleFlag = 1UL << 5U,
  kI2C_MasterRxReadyFlag = 1UL << I2C_STAT_MSTSTATE_SHIFT,
  kI2C_MasterTxReadyFlag = 1UL << (I2C_STAT_MSTSTATE_SHIFT + 1U),
  kI2C_MasterAddrNackFlag = 1UL << 7U,
  kI2C_MasterDataNackFlag = 1UL << (I2C_STAT_MSTSTATE_SHIFT + 2U),
  kI2C_SlavePendingFlag = I2C_STAT_SLVPENDING_MASK,
  kI2C_SlaveNotStretching = I2C_STAT_SLVNOTSTR_MASK,
  kI2C_SlaveSelected,
  kI2C_SaveDeselected = I2C_STAT_SLVDESEL_MASK,
  kI2C_SlaveAddressedFlag = 1UL << 22U,
  kI2C_SlaveReceiveFlag = 1UL << I2C_STAT_SLVSTATE_SHIFT,
  kI2C_SlaveTransmitFlag = 1UL << (I2C_STAT_SLVSTATE_SHIFT + 1U),
  kI2C_SlaveAddress0MatchFlag = 1UL << 20U,
  kI2C_SlaveAddress1MatchFlag = 1UL << I2C_STAT_SLVIDX_SHIFT,
  kI2C_SlaveAddress2MatchFlag = 1UL << (I2C_STAT_SLVIDX_SHIFT + 1U),
  kI2C_SlaveAddress3MatchFlag = 1UL << 21U,
  kI2C_MonitorReadyFlag = I2C_STAT_MONRDY_MASK,
  kI2C_MonitorOverflowFlag = I2C_STAT_MONOV_MASK,
  kI2C_MonitorActiveFlag = I2C_STAT_MONACTIVE_MASK,
  kI2C_MonitorIdleFlag = I2C_STAT_MONIDLE_MASK,
  kI2C_EventTimeoutFlag = I2C_STAT_EVENTTIMEOUT_MASK,
  kI2C_SclTimeoutFlag = I2C_STAT_SCLTIMEOUT_MASK }
  I2C status flags.
- enum _i2c_interrupt_enable {
  kI2C_MasterPendingInterruptEnable,
  kI2C_MasterArbitrationLostInterruptEnable,
  kI2C_MasterStartStopErrorInterruptEnable,
  kI2C_SlavePendingInterruptEnable = I2C_STAT_SLVPENDING_MASK,
  kI2C_SlaveNotStretchingInterruptEnable,
  kI2C_SlaveDeselectedInterruptEnable = I2C_STAT_SLVDESEL_MASK,
  kI2C_MonitorReadyInterruptEnable = I2C_STAT_MONRDY_MASK,
  kI2C_MonitorOverflowInterruptEnable = I2C_STAT_MONOV_MASK,
  kI2C_MonitorIdleInterruptEnable = I2C_STAT_MONIDLE_MASK,
  kI2C_EventTimeoutInterruptEnable = I2C_STAT_EVENTTIMEOUT_MASK,
  kI2C_SclTimeoutInterruptEnable = I2C_STAT_SCLTIMEOUT_MASK }
  I2C interrupt enable.

Driver version

- #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

I2C driver version.

13.3.2 Macro Definition Documentation

13.3.2.1 #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

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

13.3.2.3 #define I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK 1U /* Define to one means master ignores the last byte's nack and considers the transfer successful. */

13.3.3 Enumeration Type Documentation

13.3.3.1 anonymous enum

Enumerator

kStatus_I2C_Busy The master is already performing a transfer.
kStatus_I2C_Idle The slave driver is idle.
kStatus_I2C_Nak The slave device sent a NAK in response to a byte.
kStatus_I2C_InvalidParameter Unable to proceed due to invalid parameter.
kStatus_I2C_BitError Transferred bit was not seen on the bus.
kStatus_I2C_ArbitrationLost Arbitration lost error.
kStatus_I2C_NoTransferInProgress Attempt to abort a transfer when one is not in progress.
kStatus_I2C_DmaRequestFail DMA request failed.
kStatus_I2C_StartStopError Start and stop error.
kStatus_I2C_UnexpectedState Unexpected state.
kStatus_I2C_Timeout Timeout when waiting for I2C master/slave pending status to set to continue transfer.
kStatus_I2C_Addr_Nak NAK received for Address.
kStatus_I2C_EventTimeout Timeout waiting for bus event.
kStatus_I2C_SclLowTimeout Timeout SCL signal remains low.

13.3.3.2 enum _i2c_status_flags

Note

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

Enumerator

kI2C_MasterPendingFlag The I2C module is waiting for software interaction. bit 0

kI2C_MasterArbitrationLostFlag The arbitration of the bus was lost. There was collision on the bus. bit 4
kI2C_MasterStartStopErrorFlag There was an error during start or stop phase of the transaction. bit 6
kI2C_MasterIdleFlag The I2C master idle status. bit 5
kI2C_MasterRxReadyFlag The I2C master rx ready status. bit 1
kI2C_MasterTxReadyFlag The I2C master tx ready status. bit 2
kI2C_MasterAddrNackFlag The I2C master address nack status. bit 7
kI2C_MasterDataNackFlag The I2C master data nack status. bit 3
kI2C_SlavePendingFlag The I2C module is waiting for software interaction. bit 8
kI2C_SlaveNotStretching Indicates whether the slave is currently stretching clock (0 = yes, 1 = no). bit 11
kI2C_SlaveSelected Indicates whether the slave is selected by an address match. bit 14
kI2C_SaveDeselected Indicates that slave was previously deselected (deselect event took place, w1c). bit 15
kI2C_SlaveAddressedFlag One of the I2C slave's 4 addresses is matched. bit 22
kI2C_SlaveReceiveFlag Slave receive data available. bit 9
kI2C_SlaveTransmitFlag Slave data can be transmitted. bit 10
kI2C_SlaveAddress0MatchFlag Slave address0 match. bit 20
kI2C_SlaveAddress1MatchFlag Slave address1 match. bit 12
kI2C_SlaveAddress2MatchFlag Slave address2 match. bit 13
kI2C_SlaveAddress3MatchFlag Slave address3 match. bit 21
kI2C_MonitorReadyFlag The I2C monitor ready interrupt. bit 16
kI2C_MonitorOverflowFlag The monitor data overrun interrupt. bit 17
kI2C_MonitorActiveFlag The monitor is active. bit 18
kI2C_MonitorIdleFlag The monitor idle interrupt. bit 19
kI2C_EventTimeoutFlag The bus event timeout interrupt. bit 24
kI2C_SclTimeoutFlag The SCL timeout interrupt. bit 25

13.3.3.3 enum _i2c_interrupt_enable

Note

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

Enumerator

kI2C_MasterPendingInterruptEnable The I2C master communication pending interrupt.
kI2C_MasterArbitrationLostInterruptEnable The I2C master arbitration lost interrupt.
kI2C_MasterStartStopErrorInterruptEnable The I2C master start/stop timing error interrupt.
kI2C_SlavePendingInterruptEnable The I2C slave communication pending interrupt.
kI2C_SlaveNotStretchingInterruptEnable The I2C slave not streching interrupt, deep-sleep mode can be entered only when this interrupt occurs.
kI2C_SlaveDeselectedInterruptEnable The I2C slave deselection interrupt.
kI2C_MonitorReadyInterruptEnable The I2C monitor ready interrupt.

kI2C_MonitorOverflowInterruptEnable The monitor data overrun interrupt.
kI2C_MonitorIdleInterruptEnable The monitor idle interrupt.
kI2C_EventTimeoutInterruptEnable The bus event timeout interrupt.
kI2C_SclTimeoutInterruptEnable The SCL timeout interrupt.

13.4 I2C Master Driver

13.4.1 Overview

Data Structures

- struct i2c_master_config_t
  Structure with settings to initialize the I2C master module. More...
- struct i2c_master_transfer_t
  Non-blocking transfer descriptor structure. More...
- struct i2c_master_handle_t
  Driver handle for master non-blocking APIs. More...

Typedefs

- typedef void(∗ i2c_master_transfer_callback_t)(I2C_Type ∗base, i2c_master_handle_t ∗handle, status_t completionStatus, void ∗userData)
  Master completion callback function pointer type.

Enumerations

- enum i2c_direction_t {
  kI2C_Write = 0U,
  kI2C_Read = 1U }
  Direction of master and slave transfers.
- enum _i2c_master_transfer_flags {
  kI2C_TransferDefaultFlag = 0x00U,
  kI2C_TransferNoStartFlag = 0x01U,
  kI2C_TransferRepeatedStartFlag = 0x02U,
  kI2C_TransferNoStopFlag = 0x04U }
  Transfer option flags.
- enum _i2c_transfer_states
  States for the state machine used by transactional APIs.

Initialization and deinitialization

- void I2C_MasterGetDefaultConfig (i2c_master_config_t ∗masterConfig)
  Provides a default configuration for the I2C master peripheral.
- void I2C_MasterInit (I2C_Type ∗base, const i2c_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes the I2C master peripheral.
- void I2C_MasterDeinit (I2C_Type ∗base)
  Deinitializes the I2C master peripheral.
- uint32_t I2C_GetInstance (I2C_Type ∗base)
  Returns an instance number given a base address.

- static void I2C_MasterReset (I2C_Type *base)
  Performs a software reset.
- static void I2C_MasterEnable (I2C_Type *base, bool enable)
  Enables or disables the I2C module as master.

Status

- uint32_t I2C_GetStatusFlags (I2C_Type *base)
  Gets the I2C status flags.
- static void I2C_ClearStatusFlags (I2C_Type *base, uint32_t statusMask)
  Clears the I2C status flag state.
- static void I2C_MasterClearStatusFlags (I2C_Type *base, uint32_t statusMask)
  Clears the I2C master status flag state.

Interrupts

- static void I2C_EnableInterrupts (I2C_Type *base, uint32_t interruptMask)
  Enables the I2C interrupt requests.
- static void I2C_DisableInterrupts (I2C_Type *base, uint32_t interruptMask)
  Disables the I2C interrupt requests.
- static uint32_t I2C_GetEnabledInterrupts (I2C_Type *base)
  Returns the set of currently enabled I2C interrupt requests.

Bus operations

- void I2C_MasterSetBaudRate (I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the I2C bus frequency for master transactions.
- void I2C_MasterSetTimeoutValue (I2C_Type *base, uint8_t timeout_Ms, uint32_t srcClock_Hz)
  Sets the I2C bus timeout value.
- static bool I2C_MasterGetBusIdleState (I2C_Type *base)
  Returns whether the bus is idle.
- 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.
- static 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 void *txBuff, size_t txSize, uint32_t flags)
  Performs a polling send transfer on the I2C bus.
- status_t I2C_MasterReadBlocking (I2C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)
  Performs a polling receive transfer 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.

Non-blocking

- void I2C_MasterTransferCreateHandle (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_callback_t callback, void *userData)
  Creates a new handle for the I2C master non-blocking APIs.
- status_t I2C_MasterTransferNonBlocking (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_t *xfer)
  Performs a non-blocking transaction on the I2C bus.
- status_t I2C_MasterTransferGetCount (I2C_Type *base, i2c_master_handle_t *handle, size_t *count)
  Returns number of bytes transferred so far.
- status_t I2C_MasterTransferAbort (I2C_Type *base, i2c_master_handle_t *handle)
  Terminates a non-blocking I2C master transmission early.

IRQ handler

- void I2C_MasterTransferHandleIRQ (I2C_Type *base, i2c_master_handle_t *handle)
  Reusable routine to handle master interrupts.

13.4.2 Data Structure Documentation

13.4.2.1 struct i2c_master_config_t

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the I2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

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

Data Fields

- bool enableMaster
  Whether to enable master mode.
- uint32_t baudRate_Bps
  Desired baud rate in bits per second.
- bool enableTimeout
  Enable internal timeout function.
- uint8_t timeout_Ms
  Event timeout and SCL low timeout value.

Field Documentation

(1) bool i2c_master_config_t::enableMaster

(2) uint32_t i2c_master_config_t::baudRate_Bps

(3) bool i2c_master_config_t::enableTimeout

(4) uint8_t i2c_master_config_t::timeout_Ms

13.4.2.2 struct _i2c_master_transfer

I2C master transfer typedef.

This structure is used to pass transaction parameters to the I2C_MasterTransferNonBlocking() API.

Data Fields

- uint32_t flags
  Bit mask of options for the transfer.
- uint8_t slaveAddress
  The 7-bit slave address.
- i2c_direction_t direction
  Either kI2C_Read or kI2C_Write.
- uint32_t subaddress
  Sub address.
- size_t subaddressSize
  Length of sub address to send in bytes.
- void * data
  Pointer to data to transfer.
- size_t dataSize
  Number of bytes to transfer.

Field Documentation

(1) uint32_t i2c_master_transfer_t::flags

See enumeration _i2c_master_transfer_flags for available options. Set to 0 or kI2C_TransferDefaultFlag for normal transfers.

(2) uint8_t i2c_master_transfer_t::slaveAddress

(3) i2c_direction_t i2c_master_transfer_t::direction

(4) uint32_t i2c_master_transfer_t::subaddress

Transferred MSB first.

(5) size_t i2c_master_transfer_t::subaddressSize

Maximum size is 4 bytes.

(6) void* i2c_master_transfer_t::data

(7) size_t i2c_master_transfer_t::dataSize

13.4.2.3 struct _i2c_master_handle

I2C master handle typedef.

Note

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

Data Fields

- uint8_t state
  Transfer state machine current state.
- uint32_t transferCount
  Indicates progress of the transfer.
- uint32_t remainingBytes
  Remaining byte count in current state.
- uint8_t * buf
  Buffer pointer for current state.
- bool checkAddrNack
  Whether to check the nack signal is detected during addressing.
- i2c_master_transfer_t transfer
  Copy of the current transfer info.
- i2c_master_transfer_callback_t completionCallback
  Callback function pointer.
- void * userData
  Application data passed to callback.

Field Documentation

(1) uint8_t i2c_master_handle_t::state

(2) uint32_t i2c_master_handle_t::remainingBytes

(3) uint8_t* i2c_master_handle_t::buf

(4) bool i2c_master_handle_t::checkAddrNack

(5) i2c_master_transfer_t i2c_master_handle_t::transfer

(6) i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback

(7) void* i2c_master_handle_t::userData

13.4.3 Typedef Documentation

13.4.3.1 typedef void(* i2c_master_transfer_callback_t)(I2C_Type *base, i2c_master_handle_t *handle, status_t completionStatus, void *userData)

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to I2C_MasterTransferCreateHandle().

Parameters

base	The I2C peripheral base address.
completion-Status	Either kStatus_Success or an error code describing how the transfer completed.
userData	Arbitrary pointer-sized value passed from the application.

13.4.4 Enumeration Type Documentation

13.4.4.1 enum i2c_direction_t

Enumerator

kI2C_Write Master transmit.
kI2C_Read Master receive.

13.4.4.2 enum _i2c_master_transfer_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the _i2c_-master_transfer::flags field.

Enumerator

kI2C_TransferDefaultFlag Transfer starts with a start signal, stops with a stop signal.
kI2C_TransferNoStartFlag Don't send a start condition, address, and sub address.
kI2C_TransferRepeatedStartFlag Send a repeated start condition.
kI2C_TransferNoStopFlag Don't send a stop condition.

13.4.4.3 enum _i2c_transfer_states

13.4.5 Function Documentation

13.4.5.1 void I2C_MasterGetDefaultConfig (i2c_master_config_t * *masterConfig*)

This function provides the following default configuration for the I2C master peripheral:

```
*  masterConfig->enableMaster            = true;
*  masterConfig->baudRate_Bps            = 100000U;
*  masterConfig->enableTimeout           = false;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with I2C_MasterInit().

Parameters

out	*masterConfig*	User provided configuration structure for default values. Refer to i2c_master_config_t.

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

This function enables the peripheral clock and initializes the I2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

base	The I2C peripheral base address.
masterConfig	User provided peripheral configuration. Use I2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
srcClock_Hz	Frequency in Hertz of the I2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

13.4.5.3 void I2C_MasterDeinit (I2C_Type * *base*)

This function disables the I2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

base	The I2C peripheral base address.

13.4.5.4 uint32_t I2C_GetInstance (I2C_Type * *base*)

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

base	The I2C peripheral base address.

Returns

I2C instance number starting from 0.

13.4.5.5 static void I2C_MasterReset (I2C_Type * *base*) [inline], [static]

Restores the I2C master peripheral to reset conditions.

Parameters

base	The I2C peripheral base address.

13.4.5.6 static void I2C_MasterEnable (I2C_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	The I2C peripheral base address.
enable	Pass true to enable or false to disable the specified I2C as master.

13.4.5.7 uint32_t I2C_GetStatusFlags (I2C_Type * *base*)

A bit mask with the state of all I2C status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The I2C peripheral base address.

Returns

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

See Also

_i2c_status_flags.

13.4.5.8 static void I2C_ClearStatusFlags (I2C_Type * *base*, uint32_t *statusMask*) [inline], [static]

Refer to kI2C_CommonAllClearStatusFlags, kI2C_MasterAllClearStatusFlags and kI2C_SlaveAllClear-StatusFlags to see the clearable flags. Attempts to clear other flags has no effect.

Parameters

base	The I2C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of the members in kI2C_CommonAllClearStatusFlags, kI2C_MasterAllClearStatusFlags and kI2C_SlaveAllClearStatusFlags. You may pass the result of a previous call to I2C_GetStatusFlags().

See Also

_i2c_status_flags, _i2c_master_status_flags and _i2c_slave_status_flags.

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

Deprecated Do not use this function. It has been superceded by I2C_ClearStatusFlags The following status register flags can be cleared:

- kI2C_MasterArbitrationLostFlag
- kI2C_MasterStartStopErrorFlag

Attempts to clear other flags has no effect.

Parameters

base	The I2C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _i2c_status_flags enumerators OR'd together. You may pass the result of a previous call to I2C_GetStatusFlags().

See Also

_i2c_status_flags.

13.4.5.10 static void I2C_EnableInterrupts (I2C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Parameters

base	The I2C peripheral base address.
interruptMask	Bit mask of interrupts to enable. See _i2c_interrupt_enable for the set of constants that should be OR'd together to form the bit mask.

13.4.5.11 static void I2C_DisableInterrupts (I2C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Parameters

base	The I2C peripheral base address.
interruptMask	Bit mask of interrupts to disable. See _i2c_interrupt_enable for the set of constants that should be OR'd together to form the bit mask.

13.4.5.12 static uint32_t I2C_GetEnabledInterrupts (I2C_Type * *base*) [inline], [static]

Parameters

base	The I2C peripheral base address.

Returns

A bitmask composed of _i2c_interrupt_enable enumerators OR'd together to indicate the set of enabled interrupts.

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

The I2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

base	The I2C peripheral base address.
srcClock_Hz	I2C functional clock frequency in Hertz.
baudRate_Bps	Requested bus frequency in bits per second.

13.4.5.14 void I2C_MasterSetTimeoutValue (I2C_Type * *base*, uint8_t *timeout_Ms*, uint32_t *srcClock_Hz*)

If the SCL signal remains low or bus does not have event longer than the timeout value, kI2C_SclTimeout-Flag or kI2C_EventTimeoutFlag is set. This can indicete the bus is held by slave or any fault occurs to the I2C module.

Parameters

base	The I2C peripheral base address.
timeout_Ms	Timeout value in millisecond.
srcClock_Hz	I2C functional clock frequency in Hertz.

13.4.5.15 static bool I2C_MasterGetBusIdleState (I2C_Type * *base*) [inline], [static]

Requires the master mode to be enabled.

Parameters

base	The I2C peripheral base address.

Return values

true	Bus is busy.
false	Bus is idle.

13.4.5.16 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.

13.4.5.17 status_t I2C_MasterStop (I2C_Type * *base*)

Return values

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

13.4.5.18 static status_t I2C_MasterRepeatedStart (I2C_Type * *base*, uint8_t *address*, i2c_direction_t *direction*) [inline], [static]

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.

13.4.5.19 status_t I2C_MasterWriteBlocking (I2C_Type * *base*, const void * *txBuff*, size_t *txSize*, uint32_t *flags*)

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_I2C_Nak.

Parameters

base	The I2C peripheral base address.
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 control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

Return values

kStatus_Success	Data was sent successfully.
kStatus_I2C_Busy	Another master is currently utilizing the bus.
kStatus_I2C_Nak	The slave device sent a NAK in response to a byte.
kStatus_I2C_Arbitration-Lost	Arbitration lost error.

13.4.5.20 status_t I2C_MasterReadBlocking (I2C_Type * *base*, void * *rxBuff*, size_t *rxSize*, uint32_t *flags*)

Parameters

base	The I2C peripheral base address.
rxBuff	The pointer to the data to be transferred.
rxSize	The length in bytes of the data to be transferred.
flags	Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

Return values

kStatus_Success	Data was received successfully.
kStatus_I2C_Busy	Another master is currently utilizing the bus.
kStatus_I2C_Nak	The slave device sent a NAK in response to a byte.
kStatus_I2C_Arbitration-Lost	Arbitration lost error.

13.4.5.21 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.
kStataus_I2C_Addr_Nak	Transfer error, receive NAK during addressing.

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

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I2C_MasterTransferAbort() API shall be called.

Parameters

	base	The I2C peripheral base address.
`out`	*handle*	Pointer to the I2C master driver handle.
	callback	User provided pointer to the asynchronous callback function.
	userData	User provided pointer to the application callback data.

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

Parameters

base	The I2C peripheral base address.
handle	Pointer to the I2C master driver handle.
xfer	The pointer to the transfer descriptor.

Return values

kStatus_Success	The transaction was started successfully.
kStatus_I2C_Busy	Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

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

Parameters

	base	The I2C peripheral base address.
	handle	Pointer to the I2C master driver handle.
out	*count*	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_Success	
kStatus_I2C_Busy	

13.4.5.25 status_t I2C_MasterTransferAbort (I2C_Type * *base*, i2c_master_handle_t * *handle*)

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the I2C peripheral's IRQ priority.

Parameters

base	The I2C peripheral base address.
handle	Pointer to the I2C master driver handle.

Return values

kStatus_Success	A transaction was successfully aborted.
kStatus_I2C_Timeout	Timeout during polling for flags.

13.4.5.26 void I2C_MasterTransferHandleIRQ (I2C_Type * *base*, i2c_master_handle_t * *handle*)

Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

Parameters

base	The I2C peripheral base address.
handle	Pointer to the I2C master driver handle.

13.5 I2C Slave Driver

13.5.1 Overview

Data Structures

- struct i2c_slave_address_t
  Data structure with 7-bit Slave address and Slave address disable. More...
- struct i2c_slave_config_t
  Structure with settings to initialize the I2C slave module. More...
- struct i2c_slave_transfer_t
  I2C slave transfer structure. More...
- struct i2c_slave_handle_t
  I2C slave handle structure. More...

Typedefs

- typedef void(* i2c_slave_transfer_callback_t)(I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *userData)
  Slave event callback function pointer type.
- typedef void(* flexcomm_i2c_master_irq_handler_t)(I2C_Type *base, i2c_master_handle_t *handle)
  Typedef for master interrupt handler.
- typedef void(* flexcomm_i2c_slave_irq_handler_t)(I2C_Type *base, i2c_slave_handle_t *handle)
  Typedef for slave interrupt handler.

Enumerations

- enum i2c_slave_address_register_t {
  kI2C_SlaveAddressRegister0 = 0U,
  kI2C_SlaveAddressRegister1 = 1U,
  kI2C_SlaveAddressRegister2 = 2U,
  kI2C_SlaveAddressRegister3 = 3U }
  I2C slave address register.
- enum i2c_slave_address_qual_mode_t {
  kI2C_QualModeMask = 0U,
  kI2C_QualModeExtend }
  I2C slave address match options.
- enum i2c_slave_bus_speed_t
  I2C slave bus speed options.
- enum i2c_slave_transfer_event_t {
  kI2C_SlaveAddressMatchEvent = 0x01U,
  kI2C_SlaveTransmitEvent = 0x02U,
  kI2C_SlaveReceiveEvent = 0x04U,
  kI2C_SlaveCompletionEvent = 0x20U,
  kI2C_SlaveDeselectedEvent,

kI2C_SlaveAllEvents }
Set of events sent to the callback for non blocking slave transfers.
- enum i2c_slave_fsm_t
I2C slave software finite state machine states.

Slave initialization and deinitialization

- void I2C_SlaveGetDefaultConfig (i2c_slave_config_t *slaveConfig)
Provides a default configuration for the I2C slave peripheral.
- status_t I2C_SlaveInit (I2C_Type *base, const i2c_slave_config_t *slaveConfig, uint32_t srcClock_Hz)
Initializes the I2C slave peripheral.
- void I2C_SlaveSetAddress (I2C_Type *base, i2c_slave_address_register_t addressRegister, uint8_t address, bool addressDisable)
Configures Slave Address n register.
- void I2C_SlaveDeinit (I2C_Type *base)
Deinitializes the I2C slave peripheral.
- static void I2C_SlaveEnable (I2C_Type *base, bool enable)
Enables or disables the I2C module as slave.

Slave status

- static void I2C_SlaveClearStatusFlags (I2C_Type *base, uint32_t statusMask)
Clears the I2C status flag state.

Slave bus operations

- status_t I2C_SlaveWriteBlocking (I2C_Type *base, const uint8_t *txBuff, size_t txSize)
Performs a polling send transfer on the I2C bus.
- status_t I2C_SlaveReadBlocking (I2C_Type *base, uint8_t *rxBuff, size_t rxSize)
Performs a polling receive transfer on the I2C bus.

Slave non-blocking

- void I2C_SlaveTransferCreateHandle (I2C_Type *base, i2c_slave_handle_t *handle, i2c_slave_transfer_callback_t callback, void *userData)
Creates a new handle for the I2C slave non-blocking APIs.
- status_t I2C_SlaveTransferNonBlocking (I2C_Type *base, i2c_slave_handle_t *handle, uint32_t eventMask)
Starts accepting slave transfers.
- status_t I2C_SlaveSetSendBuffer (I2C_Type *base, volatile i2c_slave_transfer_t *transfer, const void *txData, size_t txSize, uint32_t eventMask)
Starts accepting master read from slave requests.
- status_t I2C_SlaveSetReceiveBuffer (I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *rxData, size_t rxSize, uint32_t eventMask)

Starts accepting master write to slave requests.

- static uint32_t I2C_SlaveGetReceivedAddress (I2C_Type *base, volatile i2c_slave_transfer_t *transfer)
  Returns the slave address sent by the I2C master.
- void I2C_SlaveTransferAbort (I2C_Type *base, i2c_slave_handle_t *handle)
  Aborts the slave non-blocking transfers.
- 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.

Slave IRQ handler

- void I2C_SlaveTransferHandleIRQ (I2C_Type *base, i2c_slave_handle_t *handle)
  Reusable routine to handle slave interrupts.

13.5.2 Data Structure Documentation

13.5.2.1 struct i2c_slave_address_t

Data Fields

- uint8_t address
  7-bit Slave address SLVADR.
- bool addressDisable
  Slave address disable SADISABLE.

Field Documentation

(1) uint8_t i2c_slave_address_t::address

(2) bool i2c_slave_address_t::addressDisable

13.5.2.2 struct i2c_slave_config_t

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the I2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

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

Data Fields

- i2c_slave_address_t address0
  Slave's 7-bit address and disable.
- i2c_slave_address_t address1
  Alternate slave 7-bit address and disable.
- i2c_slave_address_t address2
  Alternate slave 7-bit address and disable.

- i2c_slave_address_t address3
  Alternate slave 7-bit address and disable.
- i2c_slave_address_qual_mode_t qualMode
  Qualify mode for slave address 0.
- uint8_t qualAddress
  Slave address qualifier for address 0.
- i2c_slave_bus_speed_t busSpeed
  Slave bus speed mode.
- bool enableSlave
  Enable slave mode.

Field Documentation

(1) i2c_slave_address_t i2c_slave_config_t::address0

(2) i2c_slave_address_t i2c_slave_config_t::address1

(3) i2c_slave_address_t i2c_slave_config_t::address2

(4) i2c_slave_address_t i2c_slave_config_t::address3

(5) i2c_slave_address_qual_mode_t i2c_slave_config_t::qualMode

(6) uint8_t i2c_slave_config_t::qualAddress

(7) i2c_slave_bus_speed_t i2c_slave_config_t::busSpeed

If the slave function stretches SCL to allow for software response, it must provide sufficient data setup time to the master before releasing the stretched clock. This is accomplished by inserting one clock time of CLKDIV at that point. The busSpeed value is used to configure CLKDIV such that one clock time is greater than the tSU;DAT value noted in the I2C bus specification for the I2C mode that is being used. If the busSpeed mode is unknown at compile time, use the longest data setup time kI2C_SlaveStandardMode (250 ns)

(8) bool i2c_slave_config_t::enableSlave

13.5.2.3 struct i2c_slave_transfer_t

Data Fields

- i2c_slave_handle_t * handle
  Pointer to handle that contains this transfer.
- i2c_slave_transfer_event_t event
  Reason the callback is being invoked.
- uint8_t receivedAddress
  Matching address send by master.
- uint32_t eventMask
  Mask of enabled events.
- uint8_t * rxData
  Transfer buffer for receive data.

- const uint8_t * txData
  Transfer buffer for transmit data.
- size_t txSize
  Transfer size.
- size_t rxSize
  Transfer size.
- size_t transferredCount
  Number of bytes transferred during this transfer.
- status_t completionStatus
  Success or error code describing how the transfer completed.

Field Documentation

(1) i2c_slave_handle_t* i2c_slave_transfer_t::handle

(2) i2c_slave_transfer_event_t i2c_slave_transfer_t::event

(3) uint8_t i2c_slave_transfer_t::receivedAddress

7-bits plus R/nW bit0

(4) uint32_t i2c_slave_transfer_t::eventMask

(5) size_t i2c_slave_transfer_t::transferredCount

(6) status_t i2c_slave_transfer_t::completionStatus

Only applies for kI2C_SlaveCompletionEvent.

13.5.2.4 struct _i2c_slave_handle

I2C slave handle typedef.

Note

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

Data Fields

- volatile i2c_slave_transfer_t transfer
  I2C slave transfer.
- volatile bool isBusy
  Whether transfer is busy.
- volatile i2c_slave_fsm_t slaveFsm
  slave transfer state machine.
- i2c_slave_transfer_callback_t callback
  Callback function called at transfer event.
- void * userData
  Callback parameter passed to callback.

Field Documentation

(1) volatile i2c_slave_transfer_t i2c_slave_handle_t::transfer

(2) volatile bool i2c_slave_handle_t::isBusy

(3) volatile i2c_slave_fsm_t i2c_slave_handle_t::slaveFsm

(4) i2c_slave_transfer_callback_t i2c_slave_handle_t::callback

(5) void* i2c_slave_handle_t::userData

13.5.3 Typedef Documentation

13.5.3.1 typedef void(* i2c_slave_transfer_callback_t)(I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *userData)

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I2C_SlaveSetCallback() function after you have created a handle.

Parameters

base	Base address for the I2C instance on which the event occurred.
transfer	Pointer to transfer descriptor containing values passed to and/or from the callback.
userData	Arbitrary pointer-sized value passed from the application.

13.5.3.2 typedef void(* flexcomm_i2c_master_irq_handler_t)(I2C_Type *base, i2c_master_handle_t *handle)

13.5.3.3 typedef void(* flexcomm_i2c_slave_irq_handler_t)(I2C_Type *base, i2c_slave_handle_t *handle)

13.5.4 Enumeration Type Documentation

13.5.4.1 enum i2c_slave_address_register_t

Enumerator

kI2C_SlaveAddressRegister0	Slave Address 0 register.
kI2C_SlaveAddressRegister1	Slave Address 1 register.
kI2C_SlaveAddressRegister2	Slave Address 2 register.
kI2C_SlaveAddressRegister3	Slave Address 3 register.

13.5.4.2 enum i2c_slave_address_qual_mode_t

Enumerator

kI2C_QualModeMask The SLVQUAL0 field (qualAddress) is used as a logical mask for matching address0.

kI2C_QualModeExtend The SLVQUAL0 (qualAddress) field is used to extend address 0 matching in a range of addresses.

13.5.4.3 enum i2c_slave_bus_speed_t

13.5.4.4 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() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

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

Enumerator

kI2C_SlaveAddressMatchEvent Received the slave address after a start or repeated start.

kI2C_SlaveTransmitEvent Callback is requested to provide data to transmit (slave-transmitter role).

kI2C_SlaveReceiveEvent Callback is requested to provide a buffer in which to place received data (slave-receiver role).

kI2C_SlaveCompletionEvent All data in the active transfer have been consumed.

kI2C_SlaveDeselectedEvent The slave function has become deselected (SLVSEL flag changing from 1 to 0.

kI2C_SlaveAllEvents Bit mask of all available events.

13.5.5 Function Documentation

13.5.5.1 void I2C_SlaveGetDefaultConfig (i2c_slave_config_t * *slaveConfig*)

This function provides the following default configuration for the I2C slave peripheral:

```
*  slaveConfig->enableSlave = true;
*  slaveConfig->address0.disable = false;
*  slaveConfig->address0.address = 0u;
*  slaveConfig->address1.disable = true;
*  slaveConfig->address2.disable = true;
*  slaveConfig->address3.disable = true;
*  slaveConfig->busSpeed = kI2C_SlaveStandardMode;
*
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with I2C_SlaveInit(). Be sure to override at least the *address0.address* member of the configuration structure with the desired slave address.

Parameters

out	*slaveConfig*	User provided configuration structure that is set to default values. Refer to i2c_slave_config_t.

13.5.5.2 status_t I2C_SlaveInit (I2C_Type * *base*, const i2c_slave_config_t * *slaveConfig*, uint32_t *srcClock_Hz*)

This function enables the peripheral clock and initializes the I2C slave peripheral as described by the user provided configuration.

Parameters

base	The I2C peripheral base address.
slaveConfig	User provided peripheral configuration. Use I2C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
srcClock_Hz	Frequency in Hertz of the I2C functional clock. Used to calculate CLKDIV value to provide enough data setup time for master when slave stretches the clock.

13.5.5.3 void I2C_SlaveSetAddress (I2C_Type * *base*, i2c_slave_address_register_t *addressRegister*, uint8_t *address*, bool *addressDisable*)

This function writes new value to Slave Address register.

Parameters

base	The I2C peripheral base address.
addressRegister	The module supports multiple address registers. The parameter determines which one shall be changed.
address	The slave address to be stored to the address register for matching.
addressDisable	Disable matching of the specified address register.

13.5.5.4 void I2C_SlaveDeinit (I2C_Type * *base*)

This function disables the I2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

base	The I2C peripheral base address.

13.5.5.5 static void I2C_SlaveEnable (I2C_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	The I2C peripheral base address.
enable	True to enable or flase to disable.

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

The following status register flags can be cleared:

- slave deselected flag

Attempts to clear other flags has no effect.

Parameters

base	The I2C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _i2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to I2C_SlaveGetStatusFlags().

See Also

_i2c_slave_flags.

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

The function executes blocking address phase and blocking data phase.

Parameters

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

Returns

kStatus_Success Data has been sent.
kStatus_Fail Unexpected slave state (master data write while master read from slave is expected).

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

The function executes blocking address phase and blocking data phase.

Parameters

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

Returns

kStatus_Success Data has been received.
kStatus_Fail Unexpected slave state (master data read while master write to slave is expected).

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

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I2C_SlaveTransferAbort() API shall be called.

Parameters

	base	The I2C peripheral base address.
out	*handle*	Pointer to the I2C slave driver handle.
	callback	User provided pointer to the asynchronous callback function.
	userData	User provided pointer to the application callback data.

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

Call this API after calling I2C_SlaveInit() and I2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to I2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

If no slave Tx transfer is busy, a master read from slave request invokes kI2C_SlaveTransmitEvent callback. If no slave Rx transfer is busy, a master write to slave request invokes kI2C_SlaveReceiveEvent callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the 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.

13.5.5.11 status_t I2C_SlaveSetSendBuffer (I2C_Type * *base*, volatile i2c_slave_transfer_t * *transfer*, const void * *txData*, size_t *txSize*, uint32_t *eventMask*)

The function can be called in response to kI2C_SlaveTransmitEvent callback to start a new slave Tx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The I2C peripheral base address.
transfer	Pointer to i2c_slave_transfer_t structure.
txData	Pointer to data to send to master.
txSize	Size of txData in bytes.
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.

13.5.5.12 status_t I2C_SlaveSetReceiveBuffer (I2C_Type * *base*, volatile i2c_slave_transfer_t * *transfer*, void * *rxData*, size_t *rxSize*, uint32_t *eventMask*)

The function can be called in response to kI2C_SlaveReceiveEvent callback to start a new slave Rx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The I2C peripheral base address.
transfer	Pointer to i2c_slave_transfer_t structure.
rxData	Pointer to data to store data from master.
rxSize	Size of rxData in bytes.
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.

13.5.5.13 static uint32_t I2C_SlaveGetReceivedAddress (I2C_Type * *base*, volatile i2c_slave_transfer_t * *transfer*) [inline], [static]

This function should only be called from the address match event callback kI2C_SlaveAddressMatchEvent.

Parameters

base	The I2C peripheral base address.
transfer	The I2C slave transfer.

Returns

The 8-bit address matched by the I2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

13.5.5.14 void I2C_SlaveTransferAbort (I2C_Type * *base*, i2c_slave_handle_t * *handle*)

Note

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

Parameters

base	The I2C peripheral base address.
handle	Pointer to i2c_slave_handle_t structure which stores the transfer state.

Return values

kStatus_Success	
kStatus_I2C_Idle	

13.5.5.15 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.

13.5.5.16 void I2C_SlaveTransferHandleIRQ (I2C_Type * *base*, i2c_slave_handle_t * *handle*)

Note

This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

base	The I2C peripheral base address.

handle	Pointer to i2c_slave_handle_t structure which stores the transfer state.

13.6 I2C DMA Driver

13.6.1 Overview

Data Structures

- struct i2c_master_dma_handle_t
  I2C master dma transfer structure. *More...*

Macros

- #define I2C_MAX_DMA_TRANSFER_COUNT 1024
  Maximum lenght of single DMA transfer (determined by capability of the DMA engine)

Typedefs

- typedef void(∗ i2c_master_dma_transfer_callback_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, status_t status, void ∗userData)
  I2C master dma transfer callback typedef.
- typedef void(∗ flexcomm_i2c_dma_master_irq_handler_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle)
  Typedef for master dma handler.

Driver version

- #define FSL_I2C_DMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  I2C DMA driver version.

I2C Block DMA Transfer Operation

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

13.6.2 Data Structure Documentation

13.6.2.1 struct _i2c_master_dma_handle

I2C master dma handle typedef.

Data Fields

- uint8_t state
  Transfer state machine current state.
- uint32_t transferCount
  Indicates progress of the transfer.
- uint32_t remainingBytesDMA
  Remaining byte count to be transferred using DMA.
- uint8_t * buf
  Buffer pointer for current state.
- bool checkAddrNack
  Whether to check the nack signal is detected during addressing.
- dma_handle_t * dmaHandle
  The DMA handler used.
- i2c_master_transfer_t transfer
  Copy of the current transfer info.
- i2c_master_dma_transfer_callback_t completionCallback
  Callback function called after dma transfer finished.
- void * userData
  Callback parameter passed to callback function.

Field Documentation

(1) uint8_t i2c_master_dma_handle_t::state

(2) uint32_t i2c_master_dma_handle_t::remainingBytesDMA

(3) uint8_t* i2c_master_dma_handle_t::buf

(4) bool i2c_master_dma_handle_t::checkAddrNack

(5) dma_handle_t* i2c_master_dma_handle_t::dmaHandle

(6) i2c_master_transfer_t i2c_master_dma_handle_t::transfer

(7) i2c_master_dma_transfer_callback_t i2c_master_dma_handle_t::completionCallback

(8) void* i2c_master_dma_handle_t::userData

13.6.3 Macro Definition Documentation

13.6.3.1 #define FSL_I2C_DMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

13.6.4 Typedef Documentation

13.6.4.1 typedef void(∗ i2c_master_dma_transfer_callback_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, status_t status, void ∗userData)

13.6.4.2 typedef void(∗ flexcomm_i2c_dma_master_irq_handler_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle)

13.6.5 Function Documentation

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

Parameters

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

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

Parameters

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

Return values

kStatus_Success	Sucessully 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.

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

Parameters

base	I2C peripheral base address
handle	pointer to i2c_master_dma_handle_t structure
count	Number of bytes transferred so far by the non-blocking transaction.

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

Parameters

base	I2C peripheral base address
handle	pointer to i2c_master_dma_handle_t structure

13.7 I2C FreeRTOS Driver

13.7.1 Overview

Data Structures

- struct i2c_rtos_handle_t
  I2C FreeRTOS handle. More...

Driver version

- #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 8))
  I2C FreeRTOS driver version 2.0.8.

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 I2C transfer.

13.7.2 Data Structure Documentation

13.7.2.1 struct i2c_rtos_handle_t

Data Fields

- I2C_Type * base
  I2C base address.
- i2c_master_handle_t drv_handle
  A handle of the underlying driver, treated as opaque by the RTOS layer.
- status_t async_status
  Transactional state of the underlying driver.
- SemaphoreHandle_t mutex
  A mutex to lock the handle during a transfer.
- SemaphoreHandle_t semaphore
  A semaphore to notify and unblock task when the transfer ends.

13.7.3 Macro Definition Documentation

13.7.3.1 #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 8))

13.7.4 Function Documentation

13.7.4.1 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	Configuration structure to set-up I2C in master mode.
srcClock_Hz	Frequency of input clock of the I2C module.

Returns

status of the operation.

13.7.4.2 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.

13.7.4.3 status_t I2C_RTOS_Transfer (i2c_rtos_handle_t * *handle*, i2c_master_transfer_t * *transfer*)

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

Parameters

handle	The RTOS I2C handle.
transfer	Structure specifying the transfer parameters.

Returns

status of the operation.

13.8 I2C CMSIS Driver

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

The I2C CMSIS driver includes transactional APIs.

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

13.8.1 I2C CMSIS Driver

13.8.1.1 Master Operation in interrupt transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}
/*Init I2C MASTER*/
EXAMPLE_I2C_MASTER.Initialize(I2C_MasterSignalEvent_t);

EXAMPLE_I2C_MASTER.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
EXAMPLE_I2C_MASTER.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transmit*/
EXAMPLE_I2C_MASTER.MasterTransmit(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

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

13.8.1.2 Master Operation in DMA transactional method

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

/* Init DMA*/
DMA_Init(EXAMPLE_DMA);
```

```
/*Init I2C MASTER*/
EXAMPLE_I2C_MASTER.Initialize(I2C_MasterSignalEvent_t);

EXAMPLE_I2C_MASTER.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
EXAMPLE_I2C_MASTER.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transfer*/
EXAMPLE_I2C_MASTER.MasterReceive(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

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

13.8.1.3 Slave Operation in interrupt transactional method

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

/*Init I2C SLAVE*/
EXAMPLE_I2C_SLAVE.Initialize(I2C_SlaveSignalEvent_t);

EXAMPLE_I2C_SLAVE.PowerControl(ARM_POWER_FULL);

/*config slave addr*/
EXAMPLE_I2C_SLAVE.Control(ARM_I2C_OWN_ADDRESS, I2C_MASTER_SLAVE_ADDR);

/*start transfer*/
EXAMPLE_I2C_SLAVE.SlaveReceive(g_slave_buff, I2C_DATA_LENGTH);

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

Chapter 14
I2S: I2S Driver

14.1 Overview

The MCUXpresso SDK provides the peripheral driver for the I2S function of FLEXCOMM module of MCUXpresso SDK devices.

The I2S module is used to transmit or receive digital audio data. Only transmit or receive is enabled at one time in one module.

Driver currently supports one (primary) channel pair per one I2S enabled FLEXCOMM module only.

14.2 I2S Driver Initialization and Configuration

I2S_TxInit() and I2S_RxInit() functions ungate the clock for the FLEXCOMM module, assign I2S function to FLEXCOMM module and configure audio data format and other I2S operational settings. I2S_TxInit() is used when I2S should transmit data, I2S_RxInit() when it should receive data.

I2S_TxGetDefaultConfig() and I2S_RxGetDefaultConfig() functions can be used to set the module configuration structure with default values for transmit and receive function, respectively.

I2S_Deinit() function resets the FLEXCOMM module.

I2S_TxTransferCreateHandle() function creates transactional handle for transmit in interrupt mode.

I2S_RxTransferCreateHandle() function creates transactional handle for receive in interrupt mode.

I2S_TxTransferCreateHandleDMA() function creates transactional handle for transmit in DMA mode.

I2S_RxTransferCreateHandleDMA() function creates transactional handle for receive in DMA mode.

14.3 I2S Transmit Data

I2S_TxTransferNonBlocking() function is used to add data buffer to transmit in interrupt mode. It also begins transmission if not transmitting yet.

I2S_RxTransferNonBlocking() function is used to add data buffer to receive data into in interrupt mode. It also begins reception if not receiving yet.

I2S_TxTransferSendDMA() function is used to add data buffer to transmit in DMA mode. It also begins transmission if not transmitting yet.

I2S_RxTransferReceiveDMA() function is used to add data buffer to receive data into in DMA mode. It also begins reception if not receiving yet.

The transfer of data will be stopped automatically when all data buffers queued using the above functions will be processed and no new data buffer is enqueued meanwhile. If the above functions are not called frequently enough, I2S stop followed by restart may keep occurring resulting in drops audio stream.

14.4 I2S Interrupt related functions

I2S_EnableInterrupts() function is used to enable interrupts in FIFO interrupt register. Regular use cases do not require this function to be called from application code.

I2S_DisableInterrupts() function is used to disable interrupts in FIFO interrupt register. Regular use cases do not require this function to be called from application code.

I2S_GetEnabledInterrupts() function returns interrupts enabled in FIFO interrupt register. Regular use cases do not require this function to be called from application code.

I2S_TxHandleIRQ() and I2S_RxHandleIRQ() functions are called from ISR which is invoked when actual FIFO level decreases to configured watermark value.

I2S_DMACallback() function is called from ISR which is invoked when DMA transfer (actual descriptor) finishes.

14.5 I2S Other functions

I2S_Enable() function enables I2S function in FLEXCOMM module. Regular use cases do not require this function to be called from application code.

I2S_Disable() function disables I2S function in FLEXCOMM module. Regular use cases do not require this function to be called from application code.

I2S_TransferGetErrorCount() function returns the number of FIFO underruns or overruns in interrupt mode.

I2S_TransferGetCount() function returns the number of bytes transferred in interrupt mode.

I2S_TxTransferAbort() function aborts trasmit operation in interrupt mode.

I2S_RxTransferAbort() function aborts receive operation in interrupt mode.

I2S_TransferAbortDMA() function aborts transmit or receive operation in DMA mode.

14.6 I2S Data formats

14.6.1 DMA mode

Length of buffer for transmit or receive has to be multiply of 4 bytes. Buffer address has to be aligned to 4-bytes. Data are put into or taken from FIFO unaltered in DMA mode so buffer has to be prepared according to following information. If oneChannel is enabled, then the buffer should contain valid data only, that is to say, audio data should be put continuously in the buffer Take 8 bit data as example:

```
LSB
L07
L06
L05
L04
L03
```

L02
L01
L00
L07
L06
L05
L04
L03
L02
L01
L00
L07
L06
L05
L04
L03
L02
L01
L00
L07
L06
L05
L04
L03
L02
L01
L00

If i2s_config_t.dataLength (channel bit width) is between 4 and 16, every word in buffer should contain data for left and right channels.

MSB																												**LSB**
R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

Rnn - right channel bit nn

Lnn - left channel bit nn

Note that for example if i2s_config_t.dataLength = 7, bits on positions R07-R15 and L07-L15 are ignored (buffer "wastes space").

If i2s_config_t.dataLength (channel bit width) is between 17 and 24 and i2s_config_t.pack48 = false:

Even words (counting from zero):

MSB																											LSB
							L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

Odd words (counting from zero):

MSB																											LSB
							R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00

If i2s_config_t.dataLength (channel bit width) is between 17 and 24 and i2s_config_t.pack48 = true:

Even words (counting from zero):

MSB																											LSB
R07	R06	R05	R04	R03	R02	R01	R00	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

Odd words (counting from zero):

MSB																											LSB
														R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08

If i2s_config_t.dataLength (channel bit width) is between 25 and 32:

Even words (counting from zero):

MSB																											LSB
L31	L30	L29	L28	L27	L26	L25	L24	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

Odd words (counting from zero):

MSB																											LSB
R31	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00

14.6.2 Interrupt mode

If i2s_config_t.dataLength (channel bit width) is 4:

Buffer does not need to be aligned (buffer is read / written by single bytes, each byte contain left and right channel):

MSB							**LSB**
R03	R02	R01	R00	L03	L02	L01	L00

If i2s_config_t.dataLength (channel bit width) is between 5 and 8:

Length of buffer for transmit or receive has to be multiply of 2 bytes. Buffer address has to be aligned to 2-bytes.

MSB															**LSB**
R07	R06	R05	R04	R03	R02	R01	R00	L07	L06	L05	L04	L03	L02	L01	L00

If i2s_config_t.dataLength (channel bit width) is between 9 and 16:

Length of buffer for transmit or receive has to be multiply of 4 bytes. Buffer address has to be aligned to 4-bytes.

MSB																															**LSB**
R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

If i2s_config_t.dataLength (channel bit width) is between 17 and 24 and i2s_config_t.pack48 = false(mono channel audio data is supported):

Length of buffer for transmit or receive has to be multiply of 6 bytes.

MSB																																									
R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07

If i2s_config_t.dataLength (channel bit width) is between 17 and 24 and i2s_config_t.pack48 = true:

Length of buffer for transmit or receive has to be multiply of 6 bytes. Buffer address has to be aligned to 4-bytes.

MSB																																									
R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07

If i2s_config_t.dataLength (channel bit width) is between 25 and 32 and i2s_config_t.oneChannel = false:

Buffer for transmit or receive has to be multiply of 8 bytes. Buffer address has to be aligned to 4-bytes.

Even words (counting from zero):

MSB																												LSB
L31	L30	L29	L28	L27	L26	L25	L24	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

Odd words (counting from zero):

MSB																												LSB
R31	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00

If i2s_config_t.dataLength (channel bit width) is between 25 and 32 and i2s_config_t.oneChannel = true:

Buffer for transmit or receive has to be multiply of 4 bytes. Buffer address has to be aligned to 4-bytes.

MSB																												LSB
L31	L30	L29	L28	L27	L26	L25	L24	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

14.7 I2S Driver Examples

14.7.1 Interrupt mode examples

Transmit example

```
void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_handle_t handle;

    I2S_TxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_TxInit(I2S0, &config);

    I2S_TxTransferCreateHandle(I2S0, &handle, TxCallback, NULL);

    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_TxTransferNonBlocking(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
       finishes */
    I2S_TxTransferNonBlocking(I2S0, &handle, someTransfer);
}

void TxCallback(I2S_Type *base, i2s_handle_t *handle, status_t completionStatus, void *userData)
{
    i2s_tranfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
```

```
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_TxTransferNonBlocking(base, handle, transfer);
    }
}
```

Receive example

```
void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_handle_t handle;

    I2S_RxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_RxInit(I2S0, &config);

    I2S_RxTransferCreateHandle(I2S0, &handle, RxCallback, NULL);

    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_RxTransferNonBlocking(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
       finishes */
    I2S_RxTransferNonBlocking(I2S0, &handle, someTransfer);
}

void RxCallback(I2S_Type *base, i2s_handle_t *handle, status_t completionStatus, void *userData)
{
    i2s_tranfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_RxTransferNonBlocking(base, handle, transfer);
    }
}
```

14.7.2 DMA mode examples

Transmit example

```
void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_dma_handle_t handle;

    I2S_TxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_TxInit(I2S0, &config);

    I2S_TxTransferCreateHandleDMA(I2S0, &handle, TxCallback, NULL);
```

```
    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_TxTransferNonBlockingDMA(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
       finishes */
    I2S_TxTransferNonBlockingDMA(I2S0, &handle, someTransfer);
}

void TxCallback(I2S_Type *base, i2s_dma_handle_t *handle, status_t completionStatus, void *userData
     )
{
    i2s_tranfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_TxTransferNonBlockingDMA(base, handle, transfer);
    }
}
```

Receive example

```
void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_dma_handle_t handle;

    I2S_RxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_RxInit(I2S0, &config);

    I2S_RxTransferCreateHandleDMA(I2S0, &handle, RxCallback, NULL);

    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_RxTransferNonBlockingDMA(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
       finishes */
    I2S_RxTransferNonBlockingDMA(I2S0, &handle, someTransfer);
}

void RxCallback(I2S_Type *base, i2s_dma_handle_t *handle, status_t completionStatus, void *userData
     )
{
    i2s_tranfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_RxTransferNonBlockingDMA(base, handle, transfer);
    }
}
```

Modules

- I2S Driver

14.8 I2S Driver

14.8.1 Overview

Files

- file fsl_i2s.h

Data Structures

- struct i2s_config_t
  I2S configuration structure. More...
- struct i2s_transfer_t
  Buffer to transfer from or receive audio data into. More...
- struct i2s_handle_t
  Members not to be accessed / modified outside of the driver. More...

Macros

- #define I2S_NUM_BUFFERS (4U)
  Number of buffers .

Typedefs

- typedef void(∗ i2s_transfer_callback_t)(I2S_Type ∗base, i2s_handle_t ∗handle, status_t completionStatus, void ∗userData)
  Callback function invoked from transactional API on completion of a single buffer transfer.

Enumerations

- enum {
  kStatus_I2S_BufferComplete,
  kStatus_I2S_Done = MAKE_STATUS(kStatusGroup_I2S, 1),
  kStatus_I2S_Busy }
  _i2s_status I2S status codes.
- enum i2s_flags_t {
  kI2S_TxErrorFlag = I2S_FIFOINTENSET_TXERR_MASK,
  kI2S_TxLevelFlag = I2S_FIFOINTENSET_TXLVL_MASK,
  kI2S_RxErrorFlag = I2S_FIFOINTENSET_RXERR_MASK,
  kI2S_RxLevelFlag = I2S_FIFOINTENSET_RXLVL_MASK }
  I2S flags.

- enum i2s_master_slave_t {
  kI2S_MasterSlaveNormalSlave = 0x0,
  kI2S_MasterSlaveWsSyncMaster = 0x1,
  kI2S_MasterSlaveExtSckMaster = 0x2,
  kI2S_MasterSlaveNormalMaster = 0x3 }
  Master / slave mode.
- enum i2s_mode_t {
  kI2S_ModeI2sClassic = 0x0,
  kI2S_ModeDspWs50 = 0x1,
  kI2S_ModeDspWsShort = 0x2,
  kI2S_ModeDspWsLong = 0x3 }
  I2S mode.
- enum {
  kI2S_SecondaryChannel1 = 0U,
  kI2S_SecondaryChannel2 = 1U,
  kI2S_SecondaryChannel3 = 2U }
  _i2s_secondary_channel I2S secondary channel.

Driver version

- #define FSL_I2S_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))
  I2S driver version 2.2.2.

Initialization and deinitialization

- void I2S_TxInit (I2S_Type *base, const i2s_config_t *config)
  Initializes the FLEXCOMM peripheral for I2S transmit functionality.
- void I2S_RxInit (I2S_Type *base, const i2s_config_t *config)
  Initializes the FLEXCOMM peripheral for I2S receive functionality.
- void I2S_TxGetDefaultConfig (i2s_config_t *config)
  Sets the I2S Tx configuration structure to default values.
- void I2S_RxGetDefaultConfig (i2s_config_t *config)
  Sets the I2S Rx configuration structure to default values.
- void I2S_Deinit (I2S_Type *base)
  De-initializes the I2S peripheral.
- void I2S_SetBitClockRate (I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)
  Transmitter/Receiver bit clock rate configurations.

Non-blocking API

- void I2S_TxTransferCreateHandle (I2S_Type *base, i2s_handle_t *handle, i2s_transfer_callback_t callback, void *userData)
  Initializes handle for transfer of audio data.

- status_t I2S_TxTransferNonBlocking (I2S_Type *base, i2s_handle_t *handle, i2s_transfer_t transfer)
  Begins or queue sending of the given data.
- void I2S_TxTransferAbort (I2S_Type *base, i2s_handle_t *handle)
  Aborts sending of data.
- void I2S_RxTransferCreateHandle (I2S_Type *base, i2s_handle_t *handle, i2s_transfer_callback_t callback, void *userData)
  Initializes handle for reception of audio data.
- status_t I2S_RxTransferNonBlocking (I2S_Type *base, i2s_handle_t *handle, i2s_transfer_t transfer)
  Begins or queue reception of data into given buffer.
- void I2S_RxTransferAbort (I2S_Type *base, i2s_handle_t *handle)
  Aborts receiving of data.
- status_t I2S_TransferGetCount (I2S_Type *base, i2s_handle_t *handle, size_t *count)
  Returns number of bytes transferred so far.
- status_t I2S_TransferGetErrorCount (I2S_Type *base, i2s_handle_t *handle, size_t *count)
  Returns number of buffer underruns or overruns.

Enable / disable

- static void I2S_Enable (I2S_Type *base)
  Enables I2S operation.
- void I2S_EnableSecondaryChannel (I2S_Type *base, uint32_t channel, bool oneChannel, uint32_t position)
  Enables I2S secondary channel.
- static void I2S_DisableSecondaryChannel (I2S_Type *base, uint32_t channel)
  Disables I2S secondary channel.
- static void I2S_Disable (I2S_Type *base)
  Disables I2S operation.

Interrupts

- static void I2S_EnableInterrupts (I2S_Type *base, uint32_t interruptMask)
  Enables I2S FIFO interrupts.
- static void I2S_DisableInterrupts (I2S_Type *base, uint32_t interruptMask)
  Disables I2S FIFO interrupts.
- static uint32_t I2S_GetEnabledInterrupts (I2S_Type *base)
  Returns the set of currently enabled I2S FIFO interrupts.
- status_t I2S_EmptyTxFifo (I2S_Type *base)
  Flush the valid data in TX fifo.
- void I2S_TxHandleIRQ (I2S_Type *base, i2s_handle_t *handle)
  Invoked from interrupt handler when transmit FIFO level decreases.
- void I2S_RxHandleIRQ (I2S_Type *base, i2s_handle_t *handle)
  Invoked from interrupt handler when receive FIFO level decreases.

14.8.2 Data Structure Documentation

14.8.2.1 struct i2s_config_t

Data Fields

- i2s_master_slave_t masterSlave
  Master / slave configuration.
- i2s_mode_t mode
  I2S mode.
- bool rightLow
  Right channel data in low portion of FIFO.
- bool leftJust
  Left justify data in FIFO.
- bool sckPol
  SCK polarity.
- bool wsPol
  WS polarity.
- uint16_t divider
  Flexcomm function clock divider (1 - 4096)
- bool oneChannel
  true mono, false stereo
- uint8_t dataLength
  Data length (4 - 32)
- uint16_t frameLength
  Frame width (4 - 512)
- uint16_t position
  Data position in the frame.
- uint8_t watermark
  FIFO trigger level.
- bool txEmptyZero
  Transmit zero when buffer becomes empty or last item.
- bool pack48
  Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)

14.8.2.2 struct i2s_transfer_t

Data Fields

- uint8_t * data
  Pointer to data buffer.
- size_t dataSize
  Buffer size in bytes.

Field Documentation

(1) uint8_t* i2s_transfer_t::data

(2) size_t i2s_transfer_t::dataSize

14.8.2.3 struct _i2s_handle

Transactional state of the intialized transfer or receive I2S operation.

Data Fields

- volatile uint32_t state
  State of transfer.
- i2s_transfer_callback_t completionCallback
  Callback function pointer.
- void * userData
  Application data passed to callback.
- bool oneChannel
  true mono, false stereo
- uint8_t dataLength
  Data length (4 - 32)
- bool pack48
  Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)
- uint8_t watermark
  FIFO trigger level.
- bool useFifo48H
  When dataLength 17-24: true use FIFOWR48H, false use FIFOWR.
- volatile i2s_transfer_t i2sQueue [I2S_NUM_BUFFERS]
  Transfer queue storing transfer buffers.
- volatile uint8_t queueUser
  Queue index where user's next transfer will be stored.
- volatile uint8_t queueDriver
  Queue index of buffer actually used by the driver.
- volatile uint32_t errorCount
  Number of buffer underruns/overruns.
- volatile uint32_t transferCount
  Number of bytes transferred.

14.8.3 Macro Definition Documentation

14.8.3.1 #define FSL_I2S_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))

14.8.3.2 #define I2S_NUM_BUFFERS (4U)

14.8.4 Typedef Documentation

14.8.4.1 typedef void(* i2s_transfer_callback_t)(I2S_Type *base, i2s_handle_t *handle, status_t completionStatus, void *userData)

Parameters

base	I2S base pointer.
handle	pointer to I2S transaction.
completion-Status	status of the transaction.
userData	optional pointer to user arguments data.

14.8.5 Enumeration Type Documentation

14.8.5.1 anonymous enum

Enumerator

kStatus_I2S_BufferComplete Transfer from/into a single buffer has completed.
kStatus_I2S_Done All buffers transfers have completed.
kStatus_I2S_Busy Already performing a transfer and cannot queue another buffer.

14.8.5.2 enum i2s_flags_t

Note

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

Enumerator

kI2S_TxErrorFlag TX error interrupt.
kI2S_TxLevelFlag TX level interrupt.
kI2S_RxErrorFlag RX error interrupt.
kI2S_RxLevelFlag RX level interrupt.

14.8.5.3 enum i2s_master_slave_t

Enumerator

kI2S_MasterSlaveNormalSlave Normal slave.
kI2S_MasterSlaveWsSyncMaster WS synchronized master.
kI2S_MasterSlaveExtSckMaster Master using existing SCK.
kI2S_MasterSlaveNormalMaster Normal master.

14.8.5.4 enum i2s_mode_t

Enumerator

kI2S_ModeI2sClassic I2S classic mode.
kI2S_ModeDspWs50 DSP mode, WS having 50% duty cycle.
kI2S_ModeDspWsShort DSP mode, WS having one clock long pulse.
kI2S_ModeDspWsLong DSP mode, WS having one data slot long pulse.

14.8.5.5 anonymous enum

Enumerator

kI2S_SecondaryChannel1 secondary channel 1
kI2S_SecondaryChannel2 secondary channel 2
kI2S_SecondaryChannel3 secondary channel 3

14.8.6 Function Documentation

14.8.6.1 void I2S_TxInit (I2S_Type * *base*, const i2s_config_t * *config*)

Ungates the FLEXCOMM clock and configures the module for I2S transmission using a configuration structure. The configuration structure can be custom filled or set with default values by I2S_TxGetDefaultConfig().

Note

This API should be called at the beginning of the application to use the I2S driver.

Parameters

base	I2S base pointer.
config	pointer to I2S configuration structure.

14.8.6.2 void I2S_RxInit (I2S_Type * *base*, const i2s_config_t * *config*)

Ungates the FLEXCOMM clock and configures the module for I2S receive using a configuration structure. The configuration structure can be custom filled or set with default values by I2S_RxGetDefaultConfig().

Note

This API should be called at the beginning of the application to use the I2S driver.

Parameters

base	I2S base pointer.
config	pointer to I2S configuration structure.

14.8.6.3 void I2S_TxGetDefaultConfig (i2s_config_t * *config*)

This API initializes the configuration structure for use in I2S_TxInit(). The initialized structure can remain unchanged in I2S_TxInit(), or it can be modified before calling I2S_TxInit(). Example:

```
i2s_config_t config;
I2S_TxGetDefaultConfig(&config);
```

Default values:

```
*   config->masterSlave = kI2S_MasterSlaveNormalMaster;
*   config->mode = kI2S_ModeI2sClassic;
*   config->rightLow = false;
*   config->leftJust = false;
*   config->pdmData = false;
*   config->sckPol = false;
*   config->wsPol = false;
*   config->divider = 1;
*   config->oneChannel = false;
*   config->dataLength = 16;
*   config->frameLength = 32;
*   config->position = 0;
*   config->watermark = 4;
*   config->txEmptyZero = true;
*   config->pack48 = false;
*
```

Parameters

config	pointer to I2S configuration structure.

14.8.6.4 void I2S_RxGetDefaultConfig (i2s_config_t * *config*)

This API initializes the configuration structure for use in I2S_RxInit(). The initialized structure can remain unchanged in I2S_RxInit(), or it can be modified before calling I2S_RxInit(). Example:

```
i2s_config_t config;
I2S_RxGetDefaultConfig(&config);
```

Default values:

```
*   config->masterSlave = kI2S_MasterSlaveNormalSlave;
*   config->mode = kI2S_ModeI2sClassic;
*   config->rightLow = false;
*   config->leftJust = false;
*   config->pdmData = false;
*   config->sckPol = false;
*   config->wsPol = false;
*   config->divider = 1;
*   config->oneChannel = false;
*   config->dataLength = 16;
*   config->frameLength = 32;
*   config->position = 0;
*   config->watermark = 4;
*   config->txEmptyZero = false;
*   config->pack48 = false;
*
```

Parameters

config	pointer to I2S configuration structure.

14.8.6.5 void I2S_Deinit (I2S_Type * *base*)

This API gates the FLEXCOMM clock. The I2S module can't operate unless I2S_TxInit or I2S_RxInit is called to enable the clock.

Parameters

base	I2S base pointer.

14.8.6.6 void I2S_SetBitClockRate (I2S_Type * *base*, uint32_t *sourceClockHz*, uint32_t *sampleRate*, uint32_t *bitWidth*, uint32_t *channelNumbers*)

Parameters

base	SAI base pointer.
sourceClockHz	bit clock source frequency.
sampleRate	audio data sample rate.
bitWidth	audio data bitWidth.
channel-Numbers	audio channel numbers.

14.8.6.7 void I2S_TxTransferCreateHandle (I2S_Type * *base*, i2s_handle_t * *handle*, i2s_transfer_callback_t *callback*, void * *userData*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
callback	function to be called back when transfer is done or fails.
userData	pointer to data passed to callback.

14.8.6.8 status_t I2S_TxTransferNonBlocking (I2S_Type * *base*, i2s_handle_t * *handle*, i2s_transfer_t *transfer*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
transfer	data buffer.

Return values

kStatus_Success	
kStatus_I2S_Busy	if all queue slots are occupied with unsent buffers.

14.8.6.9 void I2S_TxTransferAbort (I2S_Type * *base*, i2s_handle_t * *handle*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.

14.8.6.10 void I2S_RxTransferCreateHandle (I2S_Type * *base*, i2s_handle_t * *handle*, i2s_transfer_callback_t *callback*, void * *userData*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
callback	function to be called back when transfer is done or fails.
userData	pointer to data passed to callback.

14.8.6.11 status_t I2S_RxTransferNonBlocking (I2S_Type * *base*, i2s_handle_t * *handle*, i2s_transfer_t *transfer*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
transfer	data buffer.

Return values

kStatus_Success	
kStatus_I2S_Busy	if all queue slots are occupied with buffers which are not full.

14.8.6.12 void I2S_RxTransferAbort (I2S_Type * *base*, i2s_handle_t * *handle*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.

14.8.6.13 status_t I2S_TransferGetCount (I2S_Type * *base*, i2s_handle_t * *handle*, size_t * *count*)

Parameters

	base	I2S base pointer.
	handle	pointer to handle structure.
out	*count*	number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_Success	
kStatus_NoTransferInProgress	there is no non-blocking transaction currently in progress.

14.8.6.14 status_t I2S_TransferGetErrorCount (I2S_Type * *base*, i2s_handle_t * *handle*, size_t * *count*)

Parameters

	base	I2S base pointer.
	handle	pointer to handle structure.
out	*count*	number of transmit errors encountered so far by the non-blocking transaction.

Return values

kStatus_Success	
kStatus_NoTransferInProgress	there is no non-blocking transaction currently in progress.

14.8.6.15 static void I2S_Enable (I2S_Type * *base*) [inline], [static]

Parameters

base	I2S base pointer.

14.8.6.16 void I2S_EnableSecondaryChannel (I2S_Type * *base*, uint32_t *channel*, bool *oneChannel*, uint32_t *position*)

Parameters

base	I2S base pointer.
channel	seondary channel channel number, reference _i2s_secondary_channel.
oneChannel	true is treated as single channel, functionality left channel for this pair.
position	define the location within the frame of the data, should not bigger than 0x1FFU.

14.8.6.17 static void I2S_DisableSecondaryChannel (I2S_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	I2S base pointer.
channel	seondary channel channel number, reference _i2s_secondary_channel.

14.8.6.18 static void I2S_Disable (I2S_Type * *base*) [inline], [static]

Parameters

base	I2S base pointer.

14.8.6.19 static void I2S_EnableInterrupts (I2S_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Parameters

base	I2S base pointer.
interruptMask	bit mask of interrupts to enable. See i2s_flags_t for the set of constants that should be OR'd together to form the bit mask.

14.8.6.20 static void I2S_DisableInterrupts (I2S_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Parameters

base	I2S base pointer.
interruptMask	bit mask of interrupts to enable. See i2s_flags_t for the set of constants that should be OR'd together to form the bit mask.

14.8.6.21 static uint32_t I2S_GetEnabledInterrupts (I2S_Type * *base*) [inline], [static]

Parameters

base	I2S base pointer.

Returns

A bitmask composed of i2s_flags_t enumerators OR'd together to indicate the set of enabled interrupts.

14.8.6.22 status_t I2S_EmptyTxFifo (I2S_Type * *base*)

Parameters

base	I2S base pointer.

Returns

kStatus_Fail empty TX fifo failed, kStatus_Success empty tx fifo success.

14.8.6.23 void I2S_TxHandleIRQ (I2S_Type * *base*, i2s_handle_t * *handle*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.

14.8.6.24 void I2S_RxHandleIRQ (I2S_Type * *base*, i2s_handle_t * *handle*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.

Chapter 15
SPI: Serial Peripheral Interface Driver

15.1 Overview

SPI driver includes functional APIs and transactional APIs.

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

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional 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.

15.2 Typical use case

15.2.1 SPI master transfer using an interrupt method

```
#define BUFFER_LEN (64)
spi_master_handle_t spiHandle;
spi_master_config_t masterConfig;
spi_transfer_t xfer;
volatile bool isFinished = false;

const uint8_t sendData[BUFFER_LEN] = [......];
uint8_t receiveBuff[BUFFER_LEN];

void SPI_UserCallback(SPI_Type *base, spi_master_handle_t *handle, status_t status, void *userData)
{
    isFinished = true;
}

void main(void)
{
    //...

    SPI_MasterGetDefaultConfig(&masterConfig);

    SPI_MasterInit(SPI0, &masterConfig, srcClock_Hz);
    SPI_MasterTransferCreateHandle(SPI0, &spiHandle, SPI_UserCallback, NULL);

    // Prepare to send.
```

```
    xfer.txData = sendData;
    xfer.rxData = receiveBuff;
    xfer.dataSize = sizeof(sendData);

    // Send out.
    SPI_MasterTransferNonBlocking(SPI0, &spiHandle, &xfer);

    // Wait send finished.
    while (!isFinished)
    {
    }

    // ...
}
```

15.2.2 SPI Send/receive using a DMA method

```
#define BUFFER_LEN (64)
spi_dma_handle_t spiHandle;
dma_handle_t g_spiTxDmaHandle;
dma_handle_t g_spiRxDmaHandle;
spi_config_t masterConfig;
spi_transfer_t xfer;
volatile bool isFinished;

uint8_t sendData[BUFFER_LEN] = ...;
uint8_t receiveBuff[BUFFER_LEN];

void SPI_UserCallback(SPI_Type *base, spi_dma_handle_t *handle, status_t status, void *userData)
{
    isFinished = true;
}

void main(void)
{
    //...

    // Initialize DMA peripheral
    DMA_Init(DMA0);

    // Initialize SPI peripheral
    SPI_MasterGetDefaultConfig(&masterConfig);
    masterConfig.sselNum = SPI_SSEL;
    SPI_MasterInit(SPI0, &masterConfig, srcClock_Hz);

    // Enable DMA channels connected to SPI0 Tx/SPI0 Rx request lines
    DMA_EnableChannel(SPI0, SPI_MASTER_TX_CHANNEL);
    DMA_EnableChannel(SPI0, SPI_MASTER_RX_CHANNEL);

    // Set DMA channels priority
    DMA_SetChannelPriority(SPI0, SPI_MASTER_TX_CHANNEL,
      kDMA_ChannelPriority3);
    DMA_SetChannelPriority(SPI0, SPI_MASTER_RX_CHANNEL,
      kDMA_ChannelPriority2);

    // Creates the DMA handle.
    DMA_CreateHandle(&masterTxHandle, SPI0, SPI_MASTER_TX_CHANNEL);
    DMA_CreateHandle(&masterRxHandle, SPI0, SPI_MASTER_RX_CHANNEL);

    // Create SPI DMA handle
    SPI_MasterTransferCreateHandleDMA(SPI0, spiHandle, SPI_UserCallback,
      NULL, &g_spiTxDmaHandle, &g_spiRxDmaHandle);

    // Prepares to send.
    xfer.txData = sendData;
```

```
    xfer.rxData = receiveBuff;
    xfer.dataSize = sizeof(sendData);

    // Sends out.
    SPI_MasterTransferDMA(SPI0, &spiHandle, &xfer);

    // Waits for send to complete.
    while (!isFinished)
    {
    }

    // ...
}
```

Modules

- SPI CMSIS driver
- SPI DMA Driver
- SPI Driver
- SPI FreeRTOS driver

15.3 SPI Driver

15.3.1 Overview

This section describes the programming interface of the SPI DMA driver.

Files

- file fsl_spi.h

Data Structures

- struct spi_delay_config_t
  SPI delay time configure structure. More...
- struct spi_master_config_t
  SPI master user configure structure. More...
- struct spi_slave_config_t
  SPI slave user configure structure. More...
- struct spi_transfer_t
  SPI transfer structure. More...
- struct spi_half_duplex_transfer_t
  SPI half-duplex(master only) transfer structure. More...
- struct spi_config_t
  Internal configuration structure used in 'spi' and 'spi_dma' driver. More...
- struct spi_master_handle_t
  SPI transfer handle structure. More...

Macros

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

Typedefs

- typedef spi_master_handle_t spi_slave_handle_t
  Slave handle type.
- typedef void(∗ spi_master_callback_t)(SPI_Type ∗base, spi_master_handle_t ∗handle, status_t status, void ∗userData)
  SPI master callback for finished transmit.
- typedef void(∗ spi_slave_callback_t)(SPI_Type ∗base, spi_slave_handle_t ∗handle, status_t status, void ∗userData)
  SPI slave callback for finished transmit.

- typedef void(∗ flexcomm_spi_master_irq_handler_t)(SPI_Type ∗base, spi_master_handle_t ∗handle)
  Typedef for master interrupt handler.
- typedef void(∗ flexcomm_spi_slave_irq_handler_t)(SPI_Type ∗base, spi_slave_handle_t ∗handle)
  Typedef for slave interrupt handler.

Enumerations

- enum spi_xfer_option_t {
  kSPI_FrameDelay = (SPI_FIFOWR_EOF_MASK),
  kSPI_FrameAssert = (SPI_FIFOWR_EOT_MASK) }
  SPI transfer option.
- enum spi_shift_direction_t {
  kSPI_MsbFirst = 0U,
  kSPI_LsbFirst = 1U }
  SPI data shifter direction options.
- enum spi_clock_polarity_t {
  kSPI_ClockPolarityActiveHigh = 0x0U,
  kSPI_ClockPolarityActiveLow }
  SPI clock polarity configuration.
- enum spi_clock_phase_t {
  kSPI_ClockPhaseFirstEdge = 0x0U,
  kSPI_ClockPhaseSecondEdge }
  SPI clock phase configuration.
- enum spi_txfifo_watermark_t {
  kSPI_TxFifo0 = 0,
  kSPI_TxFifo1 = 1,
  kSPI_TxFifo2 = 2,
  kSPI_TxFifo3 = 3,
  kSPI_TxFifo4 = 4,
  kSPI_TxFifo5 = 5,
  kSPI_TxFifo6 = 6,
  kSPI_TxFifo7 = 7 }
  txFIFO watermark values
- enum spi_rxfifo_watermark_t {
  kSPI_RxFifo1 = 0,
  kSPI_RxFifo2 = 1,
  kSPI_RxFifo3 = 2,
  kSPI_RxFifo4 = 3,
  kSPI_RxFifo5 = 4,
  kSPI_RxFifo6 = 5,
  kSPI_RxFifo7 = 6,
  kSPI_RxFifo8 = 7 }
  rxFIFO watermark values
- enum spi_data_width_t {

kSPI_Data4Bits = 3,
kSPI_Data5Bits = 4,
kSPI_Data6Bits = 5,
kSPI_Data7Bits = 6,
kSPI_Data8Bits = 7,
kSPI_Data9Bits = 8,
kSPI_Data10Bits = 9,
kSPI_Data11Bits = 10,
kSPI_Data12Bits = 11,
kSPI_Data13Bits = 12,
kSPI_Data14Bits = 13,
kSPI_Data15Bits = 14,
kSPI_Data16Bits = 15 }
Transfer data width.

- enum spi_ssel_t {
  kSPI_Ssel0 = 0,
  kSPI_Ssel1 = 1,
  kSPI_Ssel2 = 2,
  kSPI_Ssel3 = 3 }
  Slave select.
- enum spi_spol_t
  ssel polarity
- enum {
  kStatus_SPI_Busy = MAKE_STATUS(kStatusGroup_LPC_SPI, 0),
  kStatus_SPI_Idle = MAKE_STATUS(kStatusGroup_LPC_SPI, 1),
  kStatus_SPI_Error = MAKE_STATUS(kStatusGroup_LPC_SPI, 2),
  kStatus_SPI_BaudrateNotSupport,
  kStatus_SPI_Timeout = MAKE_STATUS(kStatusGroup_LPC_SPI, 4) }
  SPI transfer status.
- enum _spi_interrupt_enable {
  kSPI_RxLvlIrq = SPI_FIFOINTENSET_RXLVL_MASK,
  kSPI_TxLvlIrq = SPI_FIFOINTENSET_TXLVL_MASK }
  SPI interrupt sources.
- enum _spi_statusflags {
  kSPI_TxEmptyFlag = SPI_FIFOSTAT_TXEMPTY_MASK,
  kSPI_TxNotFullFlag = SPI_FIFOSTAT_TXNOTFULL_MASK,
  kSPI_RxNotEmptyFlag = SPI_FIFOSTAT_RXNOTEMPTY_MASK,
  kSPI_RxFullFlag = SPI_FIFOSTAT_RXFULL_MASK }
  SPI status flags.

Variables

- volatile uint8_t s_dummyData [ ]
  Global variable for dummy data value setting.

Driver version

- #define FSL_SPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
  SPI driver version.

15.3.2 Data Structure Documentation

15.3.2.1 struct spi_delay_config_t

Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maxinun value of these delay time is 15.

Data Fields

- uint8_t preDelay
  Delay between SSEL assertion and the beginning of transfer.
- uint8_t postDelay
  Delay between the end of transfer and SSEL deassertion.
- uint8_t frameDelay
  Delay between frame to frame.
- uint8_t transferDelay
  Delay between transfer to transfer.

Field Documentation

(1) uint8_t spi_delay_config_t::preDelay

(2) uint8_t spi_delay_config_t::postDelay

(3) uint8_t spi_delay_config_t::frameDelay

(4) uint8_t spi_delay_config_t::transferDelay

15.3.2.2 struct spi_master_config_t

Data Fields

- bool enableLoopback
  Enable loopback for test purpose.
- bool enableMaster
  Enable SPI at initialization time.
- spi_clock_polarity_t polarity
  Clock polarity.
- spi_clock_phase_t phase
  Clock phase.
- spi_shift_direction_t direction
  MSB or LSB.
- uint32_t baudRate_Bps

Baud Rate for SPI in Hz.
- spi_data_width_t dataWidth
  Width of the data.
- spi_ssel_t sselNum
  Slave select number.
- spi_spol_t sselPol
  Configure active CS polarity.
- uint8_t txWatermark
  txFIFO watermark
- uint8_t rxWatermark
  rxFIFO watermark
- spi_delay_config_t delayConfig
  Delay configuration.

Field Documentation

(1) spi_delay_config_t spi_master_config_t::delayConfig

15.3.2.3 struct spi_slave_config_t

Data Fields

- bool enableSlave
  Enable SPI at initialization time.
- spi_clock_polarity_t polarity
  Clock polarity.
- spi_clock_phase_t phase
  Clock phase.
- spi_shift_direction_t direction
  MSB or LSB.
- spi_data_width_t dataWidth
  Width of the data.
- spi_spol_t sselPol
  Configure active CS polarity.
- uint8_t txWatermark
  txFIFO watermark
- uint8_t rxWatermark
  rxFIFO watermark

15.3.2.4 struct spi_transfer_t

Data Fields

- uint8_t * txData
  Send buffer.
- uint8_t * rxData
  Receive buffer.
- uint32_t configFlags
  Additional option to control transfer, spi_xfer_option_t.
- size_t dataSize

Transfer bytes.

Field Documentation

(1) uint32_t spi_transfer_t::configFlags

15.3.2.5 struct spi_half_duplex_transfer_t

Data Fields

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

Field Documentation

(1) uint32_t spi_half_duplex_transfer_t::configFlags

(2) bool spi_half_duplex_transfer_t::isPcsAssertInTransfer

true for assert and false for deassert.

(3) bool spi_half_duplex_transfer_t::isTransmitFirst

15.3.2.6 struct spi_config_t

15.3.2.7 struct _spi_master_handle

Master handle type.

Data Fields

- uint8_t *volatile txData
  Transfer buffer.
- uint8_t *volatile rxData
  Receive buffer.
- volatile size_t txRemainingBytes
  Number of data to be transmitted [in bytes].
- volatile size_t rxRemainingBytes

Number of data to be received [in bytes].
- volatile int8_t toReceiveCount
  The number of data expected to receive in data width.
- size_t totalByteCount
  A number of transfer bytes.
- volatile uint32_t state
  SPI internal state.
- spi_master_callback_t callback
  SPI callback.
- void * userData
  Callback parameter.
- uint8_t dataWidth
  Width of the data [Valid values: 1 to 16].
- uint8_t sselNum
  Slave select number to be asserted when transferring data [Valid values: 0 to 3].
- uint32_t configFlags
  Additional option to control transfer.
- uint8_t txWatermark
  txFIFO watermark
- uint8_t rxWatermark
  rxFIFO watermark

Field Documentation

(1) volatile int8_t spi_master_handle_t::toReceiveCount

Since the received count and sent count should be the same to complete the transfer, if the sent count is x and the received count is y, toReceiveCount is x-y.

15.3.3 Macro Definition Documentation

15.3.3.1 #define FSL_SPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

15.3.3.2 #define SPI_DUMMYDATA (0xFFU)

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

15.3.4 Typedef Documentation

15.3.4.1 typedef void(* flexcomm_spi_master_irq_handler_t)(SPI_Type *base, spi_master_handle_t *handle)

15.3.4.2 typedef void(* flexcomm_spi_slave_irq_handler_t)(SPI_Type *base, spi_slave_handle_t *handle)

15.3.5 Enumeration Type Documentation

15.3.5.1 enum spi_xfer_option_t

Enumerator

kSPI_FrameDelay A delay may be inserted, defined in the DLY register.
kSPI_FrameAssert SSEL will be deasserted at the end of a transfer.

15.3.5.2 enum spi_shift_direction_t

Enumerator

kSPI_MsbFirst Data transfers start with most significant bit.
kSPI_LsbFirst Data transfers start with least significant bit.

15.3.5.3 enum spi_clock_polarity_t

Enumerator

kSPI_ClockPolarityActiveHigh Active-high SPI clock (idles low).
kSPI_ClockPolarityActiveLow Active-low SPI clock (idles high).

15.3.5.4 enum spi_clock_phase_t

Enumerator

kSPI_ClockPhaseFirstEdge First edge on SCK occurs at the middle of the first cycle of a data transfer.
kSPI_ClockPhaseSecondEdge First edge on SCK occurs at the start of the first cycle of a data transfer.

15.3.5.5 enum spi_txfifo_watermark_t

Enumerator

kSPI_TxFifo0 SPI tx watermark is empty.
kSPI_TxFifo1 SPI tx watermark at 1 item.
kSPI_TxFifo2 SPI tx watermark at 2 items.
kSPI_TxFifo3 SPI tx watermark at 3 items.
kSPI_TxFifo4 SPI tx watermark at 4 items.
kSPI_TxFifo5 SPI tx watermark at 5 items.
kSPI_TxFifo6 SPI tx watermark at 6 items.
kSPI_TxFifo7 SPI tx watermark at 7 items.

15.3.5.6 enum spi_rxfifo_watermark_t

Enumerator

kSPI_RxFifo1 SPI rx watermark at 1 item.
kSPI_RxFifo2 SPI rx watermark at 2 items.
kSPI_RxFifo3 SPI rx watermark at 3 items.
kSPI_RxFifo4 SPI rx watermark at 4 items.
kSPI_RxFifo5 SPI rx watermark at 5 items.
kSPI_RxFifo6 SPI rx watermark at 6 items.
kSPI_RxFifo7 SPI rx watermark at 7 items.
kSPI_RxFifo8 SPI rx watermark at 8 items.

15.3.5.7 enum spi_data_width_t

Enumerator

kSPI_Data4Bits 4 bits data width
kSPI_Data5Bits 5 bits data width
kSPI_Data6Bits 6 bits data width
kSPI_Data7Bits 7 bits data width
kSPI_Data8Bits 8 bits data width
kSPI_Data9Bits 9 bits data width
kSPI_Data10Bits 10 bits data width
kSPI_Data11Bits 11 bits data width
kSPI_Data12Bits 12 bits data width
kSPI_Data13Bits 13 bits data width
kSPI_Data14Bits 14 bits data width
kSPI_Data15Bits 15 bits data width
kSPI_Data16Bits 16 bits data width

15.3.5.8 enum spi_ssel_t

Enumerator

kSPI_Ssel0 Slave select 0.
kSPI_Ssel1 Slave select 1.
kSPI_Ssel2 Slave select 2.
kSPI_Ssel3 Slave select 3.

15.3.5.9 anonymous enum

Enumerator

kStatus_SPI_Busy SPI bus is busy.

kStatus_SPI_Idle SPI is idle.
kStatus_SPI_Error SPI error.
kStatus_SPI_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_SPI_Timeout SPI timeout polling status flags.

15.3.5.10 enum _spi_interrupt_enable

Enumerator

kSPI_RxLvlIrq Rx level interrupt.
kSPI_TxLvlIrq Tx level interrupt.

15.3.5.11 enum _spi_statusflags

Enumerator

kSPI_TxEmptyFlag txFifo is empty
kSPI_TxNotFullFlag txFifo is not full
kSPI_RxNotEmptyFlag rxFIFO is not empty
kSPI_RxFullFlag rxFIFO is full

15.3.6 Variable Documentation

15.3.6.1 volatile uint8_t s_dummyData[]

15.4 SPI DMA Driver

15.4.1 Overview

This section describes the programming interface of the SPI DMA driver.

Files

- file fsl_spi_dma.h

Data Structures

- struct spi_dma_handle_t
  SPI DMA transfer handle, users should not touch the content of the handle. More...

Typedefs

- typedef void(∗ spi_dma_callback_t)(SPI_Type ∗base, spi_dma_handle_t ∗handle, status_t status, void ∗userData)
  SPI DMA callback called at the end of transfer.

Driver version

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

DMA Transactional

- status_t SPI_MasterTransferCreateHandleDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_dma_callback_t callback, void ∗userData, dma_handle_t ∗txHandle, dma_handle_t ∗rxHandle)
  Initialize the SPI master DMA handle.
- status_t SPI_MasterTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_transfer_t ∗xfer)
  Perform a non-blocking SPI transfer using DMA.
- status_t SPI_MasterHalfDuplexTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_half_duplex_transfer_t ∗xfer)
  Transfers a block of data using a DMA method.
- static status_t SPI_SlaveTransferCreateHandleDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_dma_callback_t callback, void ∗userData, dma_handle_t ∗txHandle, dma_handle_t ∗rxHandle)
  Initialize the SPI slave DMA handle.
- static status_t SPI_SlaveTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_transfer_t ∗xfer)
  Perform a non-blocking SPI transfer using DMA.

- void SPI_MasterTransferAbortDMA (SPI_Type *base, spi_dma_handle_t *handle)
  Abort a SPI transfer using DMA.
- status_t SPI_MasterTransferGetCountDMA (SPI_Type *base, spi_dma_handle_t *handle, size_t *count)
  Gets the master DMA transfer remaining bytes.
- static void SPI_SlaveTransferAbortDMA (SPI_Type *base, spi_dma_handle_t *handle)
  Abort a SPI transfer using DMA.
- static status_t SPI_SlaveTransferGetCountDMA (SPI_Type *base, spi_dma_handle_t *handle, size_t *count)
  Gets the slave DMA transfer remaining bytes.

15.4.2 Data Structure Documentation

15.4.2.1 struct _spi_dma_handle

Data Fields

- volatile bool txInProgress
  Send transfer finished.
- volatile bool rxInProgress
  Receive transfer finished.
- dma_handle_t * txHandle
  DMA handler for SPI send.
- dma_handle_t * rxHandle
  DMA handler for SPI receive.
- uint8_t bytesPerFrame
  Bytes in a frame for SPI transfer.
- spi_dma_callback_t callback
  Callback for SPI DMA transfer.
- void * userData
  User Data for SPI DMA callback.
- uint32_t state
  Internal state of SPI DMA transfer.
- size_t transferSize
  Bytes need to be transfer.

15.4.3 Macro Definition Documentation

15.4.3.1 #define FSL_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

15.4.4 Typedef Documentation

15.4.4.1 typedef void(* spi_dma_callback_t)(SPI_Type *base, spi_dma_handle_t *handle, status_t status, void *userData)

15.4.5 Function Documentation

15.4.5.1 status_t SPI_MasterTransferCreateHandleDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_dma_callback_t *callback*, void * *userData*, dma_handle_t * *txHandle*, dma_handle_t * *rxHandle*)

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

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	User callback function called at the end of a transfer.
userData	User data for callback.
txHandle	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
rxHandle	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

15.4.5.2 status_t SPI_MasterTransferDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_transfer_t * *xfer*)

Note

This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.
xfer	Pointer to dma transfer structure.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.

kStatus_SPI_Busy	SPI is not idle, is running another transfer.

15.4.5.3 status_t SPI_MasterHalfDuplexTransferDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_half_duplex_transfer_t * *xfer*)

This function using polling way to do the first half transimission and using DMA way to do the srcond half transimission, the transfer mechanism is half-duplex. When do the second half transimission, code will return right away. When all data is transferred, the callback function is called.

Parameters

base	SPI base pointer
handle	A pointer to the spi_master_dma_handle_t structure which stores the transfer state.
xfer	A pointer to the spi_half_duplex_transfer_t structure.

Returns

status of status_t.

15.4.5.4 static status_t SPI_SlaveTransferCreateHandleDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_dma_callback_t *callback*, void * *userData*, dma_handle_t * *txHandle*, dma_handle_t * *rxHandle*) [inline], [static]

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

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	User callback function called at the end of a transfer.
userData	User data for callback.
txHandle	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
rxHandle	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

15.4.5.5 static status_t SPI_SlaveTransferDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_transfer_t * *xfer*) [inline], [static]

Note

This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.
xfer	Pointer to dma transfer structure.

Return values

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

15.4.5.6 void SPI_MasterTransferAbortDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*)

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.

15.4.5.7 status_t SPI_MasterTransferGetCountDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, size_t * *count*)

This function gets the master DMA transfer remaining bytes.

Parameters

base	SPI peripheral base address.
handle	A pointer to the spi_dma_handle_t structure which stores the transfer state.
count	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

15.4.5.8 static void SPI_SlaveTransferAbortDMA (SPI_Type * *base,* spi_dma_handle_t * *handle*) [inline], [static]

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.

15.4.5.9 static status_t SPI_SlaveTransferGetCountDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, size_t * *count*) [inline], [static]

This function gets the slave DMA transfer remaining bytes.

Parameters

base	SPI peripheral base address.
handle	A pointer to the spi_dma_handle_t structure which stores the transfer state.
count	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

15.5 SPI FreeRTOS driver

15.5.1 Overview

This section describes the programming interface of the SPI FreeRTOS driver.

Files

- file fsl_spi_freertos.h

Data Structures

- struct spi_rtos_handle_t
  SPI FreeRTOS handle. More...

Driver version

- #define FSL_SPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  SPI FreeRTOS driver version 2.1.0.

SPI RTOS Operation

- status_t SPI_RTOS_Init (spi_rtos_handle_t *handle, SPI_Type *base, const spi_master_config_t *masterConfig, uint32_t srcClock_Hz)
  Initializes SPI.
- status_t SPI_RTOS_Deinit (spi_rtos_handle_t *handle)
  Deinitializes the SPI.
- status_t SPI_RTOS_Transfer (spi_rtos_handle_t *handle, spi_transfer_t *transfer)
  Performs SPI transfer.

15.5.2 Data Structure Documentation

15.5.2.1 struct spi_rtos_handle_t

Data Fields

- SPI_Type * base
  SPI base address.
- spi_master_handle_t drv_handle
  Handle of the underlying driver, treated as opaque by the RTOS layer.
- SemaphoreHandle_t mutex
  Mutex to lock the handle during a trasfer.
- SemaphoreHandle_t event

Semaphore to notify and unblock task when transfer ends.

15.5.3 Macro Definition Documentation

15.5.3.1 #define FSL_SPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

15.5.4 Function Documentation

15.5.4.1 status_t SPI_RTOS_Init (spi_rtos_handle_t * *handle*, SPI_Type * *base*, const spi_master_config_t * *masterConfig*, uint32_t *srcClock_Hz*)

This function initializes the SPI module and related RTOS context.

Parameters

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

Returns

status of the operation.

15.5.4.2 status_t SPI_RTOS_Deinit (spi_rtos_handle_t * *handle*)

This function deinitializes the SPI module and related RTOS context.

Parameters

handle	The RTOS SPI handle.

15.5.4.3 status_t SPI_RTOS_Transfer (spi_rtos_handle_t * *handle*, spi_transfer_t * *transfer*)

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

Parameters

handle	The RTOS SPI handle.
transfer	Structure specifying the transfer parameters.

Returns

status of the operation.

15.6 SPI CMSIS driver

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

15.6.1 Function groups

15.6.1.1 SPI CMSIS GetVersion Operation

This function group will return the SPI CMSIS Driver version to user.

15.6.1.2 SPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

15.6.1.3 SPI CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize the instance in master mode or slave mode. And this API must be called before you configure an instance or after you Deinit an instance.The right steps to start an instance is that you must initialize the instance which been slected firstly,then you can power on the instance.After these all have been done,you can configure the instance by using control operation.If you want to Uninitialize the instance, you must power off the instance first.

15.6.1.4 SPI CMSIS Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

15.6.1.5 SPI CMSIS Status Operation

This function group gets the SPI transfer status.

15.6.1.6 SPI CMSIS Control Operation

This function can configure instance as master mode or slave mode, set baudrate for master mode transfer, get current baudrate of master mode transfer,set transfer data bits and other control command.

15.6.2 Typical use case

15.6.2.1 Master Operation

```
/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*SPI master init*/
DRIVER_MASTER_SPI.Initialize(SPI_MasterSignalEvent_t);
DRIVER_MASTER_SPI.PowerControl(ARM_POWER_FULL);
DRIVER_MASTER_SPI.Control(ARM_SPI_MODE_MASTER, TRANSFER_BAUDRATE);

/* Start master transfer */
DRIVER_MASTER_SPI.Transfer(masterTxData, masterRxData, TRANSFER_SIZE);

/* Master power off */
DRIVER_MASTER_SPI.PowerControl(ARM_POWER_OFF);

/* Master uninitialize */
DRIVER_MASTER_SPI.Uninitialize();
```

15.6.2.2 Slave Operation

```
/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*SPI slave init*/
DRIVER_SLAVE_SPI.Initialize(SPI_SlaveSignalEvent_t);
DRIVER_SLAVE_SPI.PowerControl(ARM_POWER_FULL);
DRIVER_SLAVE_SPI.Control(ARM_SPI_MODE_SLAVE, false);

/* Start slave transfer */
DRIVER_SLAVE_SPI.Transfer(slaveTxData, slaveRxData, TRANSFER_SIZE);

/* slave power off */
DRIVER_SLAVE_SPI.PowerControl(ARM_POWER_OFF);

/* slave uninitialize */
DRIVER_SLAVE_SPI.Uninitialize();
```

Chapter 16
USART: Universal Synchronous/Asynchronous Receiver/-Transmitter Driver

16.1 Overview

The MCUXpresso SDK provides a peripheral UART driver for the Universal Synchronous Receiver/-Transmitter (USART) module of MCUXpresso SDK devices. Driver does not support synchronous mode.

The USART driver includes two parts: functional APIs and transactional APIs.

Functional APIs are used for USART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the USART peripheral and know how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. USART functional operation groups provide the functional APIs 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 usart_handle_t as the second parameter. Initialize the handle by calling the USART_TransferCreateHandle() API.

Transactional APIs support asynchronous transfer, which means that the functions USART_TransferSendNonBlocking() and USART_TransferReceiveNonBlocking() set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_USART_TxIdle and kStatus_USART_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 USART_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 USART_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_USART_RxIdle.

If the receive ring buffer is full, the upper layer is informed through a callback with the kStatus_USART_RxRingBufferOverrun. In the callback function, the upper layer reads data out from the ring buffer. If not, the oldest 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:

```
USART_TransferCreateHandle(USART0, &handle, USART_UserCallback, NULL);
```

In this example, the buffer size is 32, but only 31 bytes are used for saving data.

16.2 Typical use case

16.2.1 USART Send/receive using a polling method

```
uint8_t ch;
USART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableTx = true;
user_config.enableRx = true;

USART_Init(USART1,&user_config,120000000U);

while(1)
{
    USART_ReadBlocking(USART1, &ch, 1);
    USART_WriteBlocking(USART1, &ch, 1);
}
```

16.2.2 USART Send/receive using an interrupt method

```
usart_handle_t g_usartHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = ’H’, ’e’, ’l’, ’l’, ’o’];
uint8_t receiveData[32];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);
    USART_TransferCreateHandle(USART1, &g_usartHandle, USART_UserCallback, NULL);

    // Prepare to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData);
    txFinished = false;

    // Send out.
    USART_TransferSendNonBlocking(USART1, &g_usartHandle, &sendXfer);
```

```
    // Wait send finished.
    while (!txFinished)
    {
    }

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData);
    rxFinished = false;

    // Receive.
    USART_TransferReceiveNonBlocking(USART1, &g_usartHandle, &receiveXfer,
      NULL);

    // Wait receive finished.
    while (!rxFinished)
    {
    }

    // ...
}
```

16.2.3 USART Receive using the ringbuffer feature

```
#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE     32

usart_handle_t g_usartHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    size_t bytesRead;
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);
    USART_TransferCreateHandle(USART1, &g_usartHandle, USART_UserCallback, NULL);
    USART_TransferStartRingBuffer(USART1, &g_usartHandle, ringBuffer,
      RING_BUFFER_SIZE);
    // Now the RX is working in background, receive in to ring buffer.

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData);
```

```
    rxFinished = false;

    // Receive.
    USART_TransferReceiveNonBlocking(USART1, &g_usartHandle, &receiveXfer);

    if (bytesRead = RX_DATA_SIZE) /* Have read enough data. */
    {
        ;
    }
    else
    {
        if (bytesRead) /* Received some data, process first. */
        {
            ;
        }

        // Wait receive finished.
        while (!rxFinished)
        {
        }
    }

    // ...
}
```

16.2.4 USART Send/Receive using the DMA method

```
usart_handle_t g_usartHandle;
dma_handle_t g_usartTxDmaHandle;
dma_handle_t g_usartRxDmaHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = ['H', 'e', 'l', 'l', 'o'];
uint8_t receiveData[32];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);

    // Set up the DMA
```

```
    DMA_Init(DMA0);
    DMA_EnableChannel(DMA0, USART_TX_DMA_CHANNEL);
    DMA_EnableChannel(DMA0, USART_RX_DMA_CHANNEL);

    DMA_CreateHandle(&g_usartTxDmaHandle, DMA0, USART_TX_DMA_CHANNEL);
    DMA_CreateHandle(&g_usartRxDmaHandle, DMA0, USART_RX_DMA_CHANNEL);

    USART_TransferCreateHandleDMA(USART1, &g_usartHandle, USART_UserCallback,
      NULL, &g_usartTxDmaHandle, &g_usartRxDmaHandle);

    // Prepare to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData);
    txFinished = false;

    // Send out.
    USART_TransferSendDMA(USART1, &g_usartHandle, &sendXfer);

    // Wait send finished.
    while (!txFinished)
    {
    }

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData);
    rxFinished = false;

    // Receive.
    USART_TransferReceiveDMA(USART1, &g_usartHandle, &receiveXfer);

    // Wait receive finished.
    while (!rxFinished)
    {
    }

    // ...
}
```

Modules

- USART CMSIS Driver
- USART DMA Driver
- USART Driver
- USART FreeRTOS Driver

16.3 USART Driver

16.3.1 Overview

Data Structures

- struct usart_config_t
  USART configuration structure. More...
- struct usart_transfer_t
  USART transfer structure. More...
- struct usart_handle_t
  USART handle structure. More...

Macros

- #define UART_RETRY_TIMES 0U /∗ Defining to zero means to keep waiting for the flag until it is assert/deassert. ∗/
  Retry times for waiting flag.

Typedefs

- typedef void(∗ usart_transfer_callback_t)(USART_Type ∗base, usart_handle_t ∗handle, status_t status, void ∗userData)
  USART transfer callback function.
- typedef void(∗ flexcomm_usart_irq_handler_t)(USART_Type ∗base, usart_handle_t ∗handle)
  Typedef for usart interrupt handler.

Enumerations

- enum {
  kStatus_USART_TxBusy = MAKE_STATUS(kStatusGroup_LPC_USART, 0),
  kStatus_USART_RxBusy = MAKE_STATUS(kStatusGroup_LPC_USART, 1),
  kStatus_USART_TxIdle = MAKE_STATUS(kStatusGroup_LPC_USART, 2),
  kStatus_USART_RxIdle = MAKE_STATUS(kStatusGroup_LPC_USART, 3),
  kStatus_USART_TxError = MAKE_STATUS(kStatusGroup_LPC_USART, 7),
  kStatus_USART_RxError = MAKE_STATUS(kStatusGroup_LPC_USART, 9),
  kStatus_USART_RxRingBufferOverrun = MAKE_STATUS(kStatusGroup_LPC_USART, 8),
  kStatus_USART_NoiseError = MAKE_STATUS(kStatusGroup_LPC_USART, 10),
  kStatus_USART_FramingError = MAKE_STATUS(kStatusGroup_LPC_USART, 11),
  kStatus_USART_ParityError = MAKE_STATUS(kStatusGroup_LPC_USART, 12),
  kStatus_USART_BaudrateNotSupport,
  kStatus_USART_Timeout = MAKE_STATUS(kStatusGroup_LPC_USART, 14) }
  Error codes for the USART driver.

- enum usart_sync_mode_t {
  kUSART_SyncModeDisabled = 0x0U,
  kUSART_SyncModeSlave = 0x2U,
  kUSART_SyncModeMaster = 0x3U }
  USART synchronous mode.
- enum usart_parity_mode_t {
  kUSART_ParityDisabled = 0x0U,
  kUSART_ParityEven = 0x2U,
  kUSART_ParityOdd = 0x3U }
  USART parity mode.
- enum usart_stop_bit_count_t {
  kUSART_OneStopBit = 0U,
  kUSART_TwoStopBit = 1U }
  USART stop bit count.
- enum usart_data_len_t {
  kUSART_7BitsPerChar = 0U,
  kUSART_8BitsPerChar = 1U }
  USART data size.
- enum usart_clock_polarity_t {
  kUSART_RxSampleOnFallingEdge = 0x0U,
  kUSART_RxSampleOnRisingEdge = 0x1U }
  USART clock polarity configuration, used in sync mode.
- enum usart_txfifo_watermark_t {
  kUSART_TxFifo0 = 0,
  kUSART_TxFifo1 = 1,
  kUSART_TxFifo2 = 2,
  kUSART_TxFifo3 = 3,
  kUSART_TxFifo4 = 4,
  kUSART_TxFifo5 = 5,
  kUSART_TxFifo6 = 6,
  kUSART_TxFifo7 = 7 }
  txFIFO watermark values
- enum usart_rxfifo_watermark_t {
  kUSART_RxFifo1 = 0,
  kUSART_RxFifo2 = 1,
  kUSART_RxFifo3 = 2,
  kUSART_RxFifo4 = 3,
  kUSART_RxFifo5 = 4,
  kUSART_RxFifo6 = 5,
  kUSART_RxFifo7 = 6,
  kUSART_RxFifo8 = 7 }
  rxFIFO watermark values
- enum _usart_interrupt_enable
  USART interrupt configuration structure, default settings all disabled.
- enum _usart_flags {

kUSART_TxError = (USART_FIFOSTAT_TXERR_MASK),
kUSART_RxError = (USART_FIFOSTAT_RXERR_MASK),
kUSART_TxFifoEmptyFlag = (USART_FIFOSTAT_TXEMPTY_MASK),
kUSART_TxFifoNotFullFlag = (USART_FIFOSTAT_TXNOTFULL_MASK),
kUSART_RxFifoNotEmptyFlag = (USART_FIFOSTAT_RXNOTEMPTY_MASK),
kUSART_RxFifoFullFlag = (USART_FIFOSTAT_RXFULL_MASK) }
USART status flags.

Functions

- uint32_t USART_GetInstance (USART_Type *base)
  Returns instance number for USART peripheral base address.

Driver version

- #define FSL_USART_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  USART driver version.

Initialization and deinitialization

- status_t USART_Init (USART_Type *base, const usart_config_t *config, uint32_t srcClock_Hz)
  Initializes a USART instance with user configuration structure and peripheral clock.
- void USART_Deinit (USART_Type *base)
  Deinitializes a USART instance.
- void USART_GetDefaultConfig (usart_config_t *config)
  Gets the default configuration structure.
- status_t USART_SetBaudRate (USART_Type *base, uint32_t baudrate_Bps, uint32_t srcClock_Hz)
  Sets the USART instance baud rate.
- status_t USART_Enable32kMode (USART_Type *base, uint32_t baudRate_Bps, bool enable32kMode, uint32_t srcClock_Hz)
  Enable 32 kHz mode which USART uses clock from the RTC oscillator as the clock source.
- void USART_Enable9bitMode (USART_Type *base, bool enable)
  Enable 9-bit data mode for USART.
- static void USART_SetMatchAddress (USART_Type *base, uint8_t address)
  Set the USART slave address.
- static void USART_EnableMatchAddress (USART_Type *base, bool match)
  Enable the USART match address feature.

Status

- static uint32_t USART_GetStatusFlags (USART_Type *base)
  Get USART status flags.
- static void USART_ClearStatusFlags (USART_Type *base, uint32_t mask)
  Clear USART status flags.

Interrupts

- static void USART_EnableInterrupts (USART_Type *base, uint32_t mask)
  Enables USART interrupts according to the provided mask.
- static void USART_DisableInterrupts (USART_Type *base, uint32_t mask)
  Disables USART interrupts according to a provided mask.
- static uint32_t USART_GetEnabledInterrupts (USART_Type *base)
  Returns enabled USART interrupts.
- static void USART_EnableTxDMA (USART_Type *base, bool enable)
  Enable DMA for Tx.
- static void USART_EnableRxDMA (USART_Type *base, bool enable)
  Enable DMA for Rx.
- static void USART_EnableCTS (USART_Type *base, bool enable)
  Enable CTS.
- static void USART_EnableContinuousSCLK (USART_Type *base, bool enable)
  Continuous Clock generation.
- static void USART_EnableAutoClearSCLK (USART_Type *base, bool enable)
  Enable Continuous Clock generation bit auto clear.
- static void USART_SetRxFifoWatermark (USART_Type *base, uint8_t water)
  Sets the rx FIFO watermark.
- static void USART_SetTxFifoWatermark (USART_Type *base, uint8_t water)
  Sets the tx FIFO watermark.

Bus Operations

- static void USART_WriteByte (USART_Type *base, uint8_t data)
  Writes to the FIFOWR register.
- static uint8_t USART_ReadByte (USART_Type *base)
  Reads the FIFORD register directly.
- static uint8_t USART_GetRxFifoCount (USART_Type *base)
  Gets the rx FIFO data count.
- static uint8_t USART_GetTxFifoCount (USART_Type *base)
  Gets the tx FIFO data count.
- void USART_SendAddress (USART_Type *base, uint8_t address)
  Transmit an address frame in 9-bit data mode.
- status_t USART_WriteBlocking (USART_Type *base, const uint8_t *data, size_t length)
  Writes to the TX register using a blocking method.
- status_t USART_ReadBlocking (USART_Type *base, uint8_t *data, size_t length)
  Read RX data register using a blocking method.

Transactional

- status_t USART_TransferCreateHandle (USART_Type *base, usart_handle_t *handle, usart_transfer_callback_t callback, void *userData)
  Initializes the USART handle.
- status_t USART_TransferSendNonBlocking (USART_Type *base, usart_handle_t *handle, usart_transfer_t *xfer)
  Transmits a buffer of data using the interrupt method.

- void USART_TransferStartRingBuffer (USART_Type *base, usart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)
  Sets up the RX ring buffer.
- void USART_TransferStopRingBuffer (USART_Type *base, usart_handle_t *handle)
  Aborts the background transfer and uninstalls the ring buffer.
- size_t USART_TransferGetRxRingBufferLength (usart_handle_t *handle)
  Get the length of received data in RX ring buffer.
- void USART_TransferAbortSend (USART_Type *base, usart_handle_t *handle)
  Aborts the interrupt-driven data transmit.
- status_t USART_TransferGetSendCount (USART_Type *base, usart_handle_t *handle, uint32_t *count)
  Get the number of bytes that have been sent out to bus.
- status_t USART_TransferReceiveNonBlocking (USART_Type *base, usart_handle_t *handle, usart_transfer_t *xfer, size_t *receivedBytes)
  Receives a buffer of data using an interrupt method.
- void USART_TransferAbortReceive (USART_Type *base, usart_handle_t *handle)
  Aborts the interrupt-driven data receiving.
- status_t USART_TransferGetReceiveCount (USART_Type *base, usart_handle_t *handle, uint32_t *count)
  Get the number of bytes that have been received.
- void USART_TransferHandleIRQ (USART_Type *base, usart_handle_t *handle)
  USART IRQ handle function.

16.3.2 Data Structure Documentation

16.3.2.1 struct usart_config_t

Data Fields

- uint32_t baudRate_Bps
  USART baud rate.
- usart_parity_mode_t parityMode
  Parity mode, disabled (default), even, odd.
- usart_stop_bit_count_t stopBitCount
  Number of stop bits, 1 stop bit (default) or 2 stop bits.
- usart_data_len_t bitCountPerChar
  Data length - 7 bit, 8 bit.
- bool loopback
  Enable peripheral loopback.
- bool enableRx
  Enable RX.
- bool enableTx
  Enable TX.
- bool enableContinuousSCLK
  USART continuous Clock generation enable in synchronous master mode.
- bool enableMode32k
  USART uses 32 kHz clock from the RTC oscillator as the clock source.
- bool enableHardwareFlowControl
  Enable hardware control RTS/CTS.

- usart_txfifo_watermark_t txWatermark
  txFIFO watermark
- usart_rxfifo_watermark_t rxWatermark
  rxFIFO watermark
- usart_sync_mode_t syncMode
  Transfer mode select - asynchronous, synchronous master, synchronous slave.
- usart_clock_polarity_t clockPolarity
  Selects the clock polarity and sampling edge in synchronous mode.

Field Documentation

(1) bool usart_config_t::enableContinuousSCLK

(2) bool usart_config_t::enableMode32k

(3) usart_sync_mode_t usart_config_t::syncMode

(4) usart_clock_polarity_t usart_config_t::clockPolarity

16.3.2.2 struct usart_transfer_t

Data Fields

- size_t dataSize
  The byte count to be transfer.
- uint8_t * data
  The buffer of data to be transfer.
- uint8_t * rxData
  The buffer to receive data.
- const uint8_t * txData
  The buffer of data to be sent.

Field Documentation

(1) uint8_t* usart_transfer_t::data

(2) uint8_t* usart_transfer_t::rxData

(3) const uint8_t* usart_transfer_t::txData

(4) size_t usart_transfer_t::dataSize

16.3.2.3 struct _usart_handle

Data Fields

- const uint8_t *volatile txData
  Address of remaining data to send.
- volatile size_t txDataSize
  Size of the remaining data to send.
- size_t txDataSizeAll

Size of the data to send out.
- uint8_t *volatile rxData
  Address of remaining data to receive.
- volatile size_t rxDataSize
  Size of the remaining data to receive.
- size_t rxDataSizeAll
  Size of the data to receive.
- uint8_t * rxRingBuffer
  Start address of the receiver ring buffer.
- size_t rxRingBufferSize
  Size of the ring buffer.
- volatile uint16_t rxRingBufferHead
  Index for the driver to store received data into ring buffer.
- volatile uint16_t rxRingBufferTail
  Index for the user to get data from the ring buffer.
- usart_transfer_callback_t callback
  Callback function.
- void * userData
  USART callback function parameter.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.
- uint8_t txWatermark
  txFIFO watermark
- uint8_t rxWatermark
  rxFIFO watermark

Field Documentation

(1) const uint8_t* volatile usart_handle_t::txData

(2) volatile size_t usart_handle_t::txDataSize

(3) size_t usart_handle_t::txDataSizeAll

(4) uint8_t* volatile usart_handle_t::rxData

(5) volatile size_t usart_handle_t::rxDataSize

(6) size_t usart_handle_t::rxDataSizeAll

(7) uint8_t* usart_handle_t::rxRingBuffer

(8) size_t usart_handle_t::rxRingBufferSize

(9) volatile uint16_t usart_handle_t::rxRingBufferHead

(10) volatile uint16_t usart_handle_t::rxRingBufferTail

(11) usart_transfer_callback_t usart_handle_t::callback

(12) void∗ usart_handle_t::userData

(13) volatile uint8_t usart_handle_t::txState

16.3.3 Macro Definition Documentation

16.3.3.1 #define FSL_USART_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

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

16.3.4 Typedef Documentation

16.3.4.1 typedef void(∗ usart_transfer_callback_t)(USART_Type ∗base, usart_handle_t ∗handle, status_t status, void ∗userData)

16.3.4.2 typedef void(∗ flexcomm_usart_irq_handler_t)(USART_Type ∗base, usart_handle_t ∗handle)

16.3.5 Enumeration Type Documentation

16.3.5.1 anonymous enum

Enumerator

kStatus_USART_TxBusy Transmitter is busy.
kStatus_USART_RxBusy Receiver is busy.
kStatus_USART_TxIdle USART transmitter is idle.
kStatus_USART_RxIdle USART receiver is idle.
kStatus_USART_TxError Error happens on txFIFO.
kStatus_USART_RxError Error happens on rxFIFO.
kStatus_USART_RxRingBufferOverrun Error happens on rx ring buffer.
kStatus_USART_NoiseError USART noise error.
kStatus_USART_FramingError USART framing error.
kStatus_USART_ParityError USART parity error.
kStatus_USART_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_USART_Timeout USART time out.

16.3.5.2 enum usart_sync_mode_t

Enumerator

kUSART_SyncModeDisabled Asynchronous mode.
kUSART_SyncModeSlave Synchronous slave mode.

kUSART_SyncModeMaster Synchronous master mode.

16.3.5.3 enum usart_parity_mode_t

Enumerator

kUSART_ParityDisabled Parity disabled.
kUSART_ParityEven Parity enabled, type even, bit setting: PE|PT = 10.
kUSART_ParityOdd Parity enabled, type odd, bit setting: PE|PT = 11.

16.3.5.4 enum usart_stop_bit_count_t

Enumerator

kUSART_OneStopBit One stop bit.
kUSART_TwoStopBit Two stop bits.

16.3.5.5 enum usart_data_len_t

Enumerator

kUSART_7BitsPerChar Seven bit mode.
kUSART_8BitsPerChar Eight bit mode.

16.3.5.6 enum usart_clock_polarity_t

Enumerator

kUSART_RxSampleOnFallingEdge Un_RXD is sampled on the falling edge of SCLK.
kUSART_RxSampleOnRisingEdge Un_RXD is sampled on the rising edge of SCLK.

16.3.5.7 enum usart_txfifo_watermark_t

Enumerator

kUSART_TxFifo0 USART tx watermark is empty.
kUSART_TxFifo1 USART tx watermark at 1 item.
kUSART_TxFifo2 USART tx watermark at 2 items.
kUSART_TxFifo3 USART tx watermark at 3 items.
kUSART_TxFifo4 USART tx watermark at 4 items.
kUSART_TxFifo5 USART tx watermark at 5 items.
kUSART_TxFifo6 USART tx watermark at 6 items.
kUSART_TxFifo7 USART tx watermark at 7 items.

16.3.5.8 enum usart_rxfifo_watermark_t

Enumerator

kUSART_RxFifo1 USART rx watermark at 1 item.
kUSART_RxFifo2 USART rx watermark at 2 items.
kUSART_RxFifo3 USART rx watermark at 3 items.
kUSART_RxFifo4 USART rx watermark at 4 items.
kUSART_RxFifo5 USART rx watermark at 5 items.
kUSART_RxFifo6 USART rx watermark at 6 items.
kUSART_RxFifo7 USART rx watermark at 7 items.
kUSART_RxFifo8 USART rx watermark at 8 items.

16.3.5.9 enum _usart_flags

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

Enumerator

kUSART_TxError TEERR bit, sets if TX buffer is error.
kUSART_RxError RXERR bit, sets if RX buffer is error.
kUSART_TxFifoEmptyFlag TXEMPTY bit, sets if TX buffer is empty.
kUSART_TxFifoNotFullFlag TXNOTFULL bit, sets if TX buffer is not full.
kUSART_RxFifoNotEmptyFlag RXNOEMPTY bit, sets if RX buffer is not empty.
kUSART_RxFifoFullFlag RXFULL bit, sets if RX buffer is full.

16.3.6 Function Documentation

16.3.6.1 uint32_t USART_GetInstance (USART_Type * *base*)

16.3.6.2 status_t USART_Init (USART_Type * *base*, const usart_config_t * *config*, uint32_t *srcClock_Hz*)

This function configures the USART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the USART_GetDefaultConfig() function. Example below shows how to use this API to configure USART.

```
*  usart_config_t usartConfig;
*  usartConfig.baudRate_Bps = 115200U;
*  usartConfig.parityMode = kUSART_ParityDisabled;
*  usartConfig.stopBitCount = kUSART_OneStopBit;
*  USART_Init(USART1, &usartConfig, 20000000U);
*
```

Parameters

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

Return values

kStatus_USART_BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_InvalidArgument	USART base address is not valid
kStatus_Success	Status USART initialize succeed

16.3.6.3 void USART_Deinit (USART_Type * *base*)

This function waits for TX complete, disables TX and RX, and disables the USART clock.

Parameters

base	USART peripheral base address.

16.3.6.4 void USART_GetDefaultConfig (usart_config_t * *config*)

This function initializes the USART configuration structure to a default value. The default values are: usartConfig->baudRate_Bps = 115200U; usartConfig->parityMode = kUSART_ParityDisabled; usartConfig->stopBitCount = kUSART_OneStopBit; usartConfig->bitCountPerChar = kUSART_8BitsPerChar; usartConfig->loopback = false; usartConfig->enableTx = false; usartConfig->enableRx = false;

Parameters

config	Pointer to configuration structure.

16.3.6.5 status_t USART_SetBaudRate (USART_Type * *base*, uint32_t *baudrate_Bps*, uint32_t *srcClock_Hz*)

This function configures the USART module baud rate. This function is used to update the USART module baud rate after the USART module is initialized by the USART_Init.

```
*  USART_SetBaudRate(USART1, 115200U, 20000000U);
*
```

Parameters

base	USART peripheral base address.
baudrate_Bps	USART baudrate to be set.
srcClock_Hz	USART clock source frequency in HZ.

Return values

kStatus_USART_BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_Success	Set baudrate succeed.
kStatus_InvalidArgument	One or more arguments are invalid.

16.3.6.6 status_t USART_Enable32kMode (USART_Type * *base*, uint32_t *baudRate_Bps*, bool *enableMode32k*, uint32_t *srcClock_Hz*)

Please note that in order to use a 32 kHz clock to operate USART properly, the RTC oscillator and its 32 kHz output must be manully enabled by user, by calling RTC_Init and setting SYSCON_RTCOSCCTRL_EN bit to 1. And in 32kHz clocking mode the USART can only work at 9600 baudrate or at the baudrate that 9600 can evenly divide, eg: 4800, 3200.

Parameters

base	USART peripheral base address.
baudRate_Bps	USART baudrate to be set..
enableMode32k	true is 32k mode, false is normal mode.
srcClock_Hz	USART clock source frequency in HZ.

Return values

kStatus_USART_BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_Success	Set baudrate succeed.
kStatus_InvalidArgument	One or more arguments are invalid.

16.3.6.7 void USART_Enable9bitMode (USART_Type * *base*, bool *enable*)

This function set the 9-bit mode for USART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

base	USART peripheral base address.
enable	true to enable, false to disable.

16.3.6.8 static void USART_SetMatchAddress (USART_Type * *base*, uint8_t *address*) [inline], [static]

This function configures the address for USART module that works as slave in 9-bit data mode. When the address detection is enabled, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note

Any USART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

base	USART peripheral base address.
address	USART slave address.

16.3.6.9 static void USART_EnableMatchAddress (USART_Type * *base*, bool *match*) [inline], [static]

Parameters

base	USART peripheral base address.
match	true to enable match address, false to disable.

16.3.6.10 static uint32_t USART_GetStatusFlags (USART_Type * *base*) [inline], [static]

This function get all USART status flags, the flags are returned as the logical OR value of the enumerators _usart_flags. To check a specific status, compare the return value with enumerators in _usart_flags. For example, to check whether the TX is empty:

```
*     if (kUSART_TxFifoNotFullFlag &
```

```
*      USART_GetStatusFlags(USART1))
*      {
*          ...
*      }
*
```

Parameters

base	USART peripheral base address.

Returns

USART status flags which are ORed by the enumerators in the _usart_flags.

16.3.6.11 static void USART_ClearStatusFlags (USART_Type * *base*, uint32_t *mask*) [inline], [static]

This function clear supported USART status flags Flags that can be cleared or set are: kUSART_TxError kUSART_RxError For example:

```
*      USART_ClearStatusFlags(USART1, kUSART_TxError |
       kUSART_RxError)
*
```

Parameters

base	USART peripheral base address.
mask	status flags to be cleared.

16.3.6.12 static void USART_EnableInterrupts (USART_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the USART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _usart_interrupt_enable. For example, to enable TX empty interrupt and RX full interrupt:

```
*      USART_EnableInterrupts(USART1, kUSART_TxLevelInterruptEnable |
       kUSART_RxLevelInterruptEnable);
*
```

Parameters

base	USART peripheral base address.
mask	The interrupts to enable. Logical OR of _usart_interrupt_enable.

16.3.6.13 static void USART_DisableInterrupts (USART_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the USART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _usart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
*     USART_DisableInterrupts(USART1, kUSART_TxLevelInterruptEnable |
      kUSART_RxLevelInterruptEnable);
*
```

Parameters

base	USART peripheral base address.
mask	The interrupts to disable. Logical OR of _usart_interrupt_enable.

16.3.6.14 static uint32_t USART_GetEnabledInterrupts (USART_Type * *base*) [inline], [static]

This function returns the enabled USART interrupts.

Parameters

base	USART peripheral base address.

16.3.6.15 static void USART_EnableCTS (USART_Type * *base*, bool *enable*) [inline], [static]

This function will determine whether CTS is used for flow control.

Parameters

base	USART peripheral base address.
enable	Enable CTS or not, true for enable and false for disable.

16.3.6.16 static void USART_EnableContinuousSCLK (USART_Type * *base,* bool *enable*) [inline], [static]

By default, SCLK is only output while data is being transmitted in synchronous mode. Enable this funciton, SCLK will run continuously in synchronous mode, allowing characters to be received on Un_RxD independently from transmission on Un_TXD).

Parameters

base	USART peripheral base address.
enable	Enable Continuous Clock generation mode or not, true for enable and false for disable.

16.3.6.17 static void USART_EnableAutoClearSCLK (USART_Type * *base,* bool *enable*) [inline], [static]

While enable this cuntion, the Continuous Clock bit is automatically cleared when a complete character has been received. This bit is cleared at the same time.

Parameters

base	USART peripheral base address.
enable	Enable auto clear or not, true for enable and false for disable.

16.3.6.18 static void USART_SetRxFifoWatermark (USART_Type * *base,* uint8_t *water*) [inline], [static]

Parameters

base	USART peripheral base address.
water	Rx FIFO watermark.

16.3.6.19 static void USART_SetTxFifoWatermark (USART_Type * *base,* uint8_t *water*) [inline], [static]

Parameters

base	USART peripheral base address.
water	Tx FIFO watermark.

16.3.6.20 static void USART_WriteByte (USART_Type * *base,* uint8_t *data*) [inline], [static]

This function writes data to the txFIFO directly. The upper layer must ensure that txFIFO has space for data to write before calling this function.

Parameters

base	USART peripheral base address.
data	The byte to write.

16.3.6.21 static uint8_t USART_ReadByte (USART_Type * *base*) [inline], [static]

This function reads data from the rxFIFO directly. The upper layer must ensure that the rxFIFO is not empty before calling this function.

Parameters

base	USART peripheral base address.

Returns

The byte read from USART data register.

16.3.6.22 static uint8_t USART_GetRxFifoCount (USART_Type * *base*) [inline], [static]

Parameters

base	USART peripheral base address.

Returns

rx FIFO data count.

16.3.6.23 static uint8_t USART_GetTxFifoCount (USART_Type * *base*) [inline], [static]

Parameters

base	USART peripheral base address.

Returns

tx FIFO data count.

16.3.6.24 void USART_SendAddress (USART_Type * *base*, uint8_t *address*)

Parameters

base	USART peripheral base address.
address	USART slave address.

16.3.6.25 status_t USART_WriteBlocking (USART_Type * *base*, const uint8_t * *data*, size_t *length*)

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Parameters

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

Return values

kStatus_USART_Timeout	Transmission timed out and was aborted.
kStatus_InvalidArgument	Invalid argument.
kStatus_Success	Successfully wrote all data.

16.3.6.26 status_t USART_ReadBlocking (USART_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 read data from the TX register.

Parameters

base	USART peripheral base address.
data	Start address of the buffer to store the received data.
length	Size of the buffer.

Return values

kStatus_USART_FramingError	Receiver overrun happened while receiving data.
kStatus_USART_ParityError	Noise error happened while receiving data.
kStatus_USART_NoiseError	Framing error happened while receiving data.
kStatus_USART_RxError	Overflow or underflow rxFIFO happened.
kStatus_USART_Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully received all data.

16.3.6.27 status_t USART_TransferCreateHandle (USART_Type * *base*, usart_handle_t * *handle*, usart_transfer_callback_t *callback*, void * *userData*)

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

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
callback	The callback function.
userData	The parameter of the callback function.

16.3.6.28 status_t USART_TransferSendNonBlocking (USART_Type * *base*, usart_handle_t * *handle*, usart_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 IRQ handler, the USART driver calls the callback function and passes the kStatus_USART_TxIdle as status parameter.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
xfer	USART transfer structure. See usart_transfer_t.

Return values

kStatus_Success	Successfully start the data transmission.
kStatus_USART_TxBusy	Previous transmission still not finished, data not all written to TX register yet.
kStatus_InvalidArgument	Invalid argument.

16.3.6.29 void USART_TransferStartRingBuffer (USART_Type * *base*, usart_handle_t * *handle*, uint8_t * *ringBuffer*, size_t *ringBufferSize*)

This function sets up the RX ring buffer to a specific USART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the USART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

Parameters

base	USART peripheral base address.
handle	USART 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.

16.3.6.30 void USART_TransferStopRingBuffer (USART_Type * *base*, usart_handle_t * *handle*)

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

16.3.6.31 size_t USART_TransferGetRxRingBufferLength (usart_handle_t * *handle*)

Parameters

handle	USART handle pointer.

Returns

Length of received data in RX ring buffer.

16.3.6.32 void USART_TransferAbortSend (USART_Type * *base,* usart_handle_t * *handle*)

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are still not sent out.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

16.3.6.33 status_t USART_TransferGetSendCount (USART_Type * *base,* usart_handle_t * *handle,* uint32_t * *count*)

This function gets the number of bytes that have been sent out to bus by interrupt method.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferInProgress	No send in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

16.3.6.34 status_t USART_TransferReceiveNonBlocking (USART_Type * *base*, usart_handle_t * *handle*, usart_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 USART driver. When the new data arrives, the receive request is serviced first. When all data is received, the USART driver notifies the upper layer through a callback function and passes the status parameter kStatus_USART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the xfer->data and this function returns with the parameter `receivedBytes` set to 5. For the left 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the USART 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	USART peripheral base address.
handle	USART handle pointer.
xfer	USART transfer structure, see usart_transfer_t.
receivedBytes	Bytes received from the ring buffer directly.

Return values

kStatus_Success	Successfully queue the transfer into transmit queue.
kStatus_USART_RxBusy	Previous receive request is not finished.
kStatus_InvalidArgument	Invalid argument.

16.3.6.35 void USART_TransferAbortReceive (USART_Type * *base,* usart_handle_t * *handle*)

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

16.3.6.36 status_t USART_TransferGetReceiveCount (USART_Type * *base*, usart_handle_t * *handle*, uint32_t * *count*)

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

Parameters

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

Return values

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

16.3.6.37 void USART_TransferHandleIRQ (USART_Type * *base*, usart_handle_t * *handle*)

This function handles the USART transmit and receive IRQ request.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

16.4 USART DMA Driver

16.4.1 Overview

Files

- file fsl_usart_dma.h

Data Structures

- struct usart_dma_handle_t
  UART DMA handle. More...

Typedefs

- typedef void(∗ usart_dma_transfer_callback_t)(USART_Type ∗base, usart_dma_handle_t ∗handle, status_t status, void ∗userData)
  UART transfer callback function.

Driver version

- #define FSL_USART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  USART dma driver version.

DMA transactional

- status_t USART_TransferCreateHandleDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, usart_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  Initializes the USART handle which is used in transactional functions.
- status_t USART_TransferSendDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, usart_transfer_t ∗xfer)
  Sends data using DMA.
- status_t USART_TransferReceiveDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, usart_transfer_t ∗xfer)
  Receives data using DMA.
- void USART_TransferAbortSendDMA (USART_Type ∗base, usart_dma_handle_t ∗handle)
  Aborts the sent data using DMA.
- void USART_TransferAbortReceiveDMA (USART_Type ∗base, usart_dma_handle_t ∗handle)
  Aborts the received data using DMA.
- status_t USART_TransferGetReceiveCountDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, uint32_t ∗count)
  Get the number of bytes that have been received.

- status_t USART_TransferGetSendCountDMA (USART_Type *base, usart_dma_handle_t *handle, uint32_t *count)
  Get the number of bytes that have been sent.

16.4.2 Data Structure Documentation

16.4.2.1 struct _usart_dma_handle

Data Fields

- USART_Type * base
  UART peripheral base address.
- usart_dma_transfer_callback_t callback
  Callback function.
- void * userData
  UART callback function parameter.
- size_t rxDataSizeAll
  Size of the data to receive.
- size_t txDataSizeAll
  Size of the data to send out.
- dma_handle_t * txDmaHandle
  The DMA TX channel used.
- dma_handle_t * rxDmaHandle
  The DMA RX channel used.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.

Field Documentation

(1) USART_Type* usart_dma_handle_t::base

(2) usart_dma_transfer_callback_t usart_dma_handle_t::callback

(3) void* usart_dma_handle_t::userData

(4) size_t usart_dma_handle_t::rxDataSizeAll

(5) size_t usart_dma_handle_t::txDataSizeAll

(6) dma_handle_t* usart_dma_handle_t::txDmaHandle

(7) dma_handle_t* usart_dma_handle_t::rxDmaHandle

(8) volatile uint8_t usart_dma_handle_t::txState

16.4.3 Macro Definition Documentation

16.4.3.1 #define FSL_USART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

16.4.4 Typedef Documentation

16.4.4.1 typedef void(* usart_dma_transfer_callback_t)(USART_Type *base, usart_dma_handle_t *handle, status_t status, void *userData)

16.4.5 Function Documentation

16.4.5.1 status_t USART_TransferCreateHandleDMA (USART_Type * *base*, usart_dma_handle_t * *handle*, usart_dma_transfer_callback_t *callback*, void * *userData*, dma_handle_t * *txDmaHandle*, dma_handle_t * *rxDmaHandle*)

Parameters

base	USART peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
callback	Callback function.
userData	User data.
txDmaHandle	User-requested DMA handle for TX DMA transfer.
rxDmaHandle	User-requested DMA handle for RX DMA transfer.

16.4.5.2 status_t USART_TransferSendDMA (USART_Type * *base*, usart_dma_handle_t * *handle*, usart_transfer_t * *xfer*)

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

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
xfer	USART DMA transfer structure. See usart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_USART_TxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

16.4.5.3 status_t USART_TransferReceiveDMA (USART_Type * *base*, usart_dma_handle_t * *handle*, usart_transfer_t * *xfer*)

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

Parameters

base	USART peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
xfer	USART DMA transfer structure. See usart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_USART_RxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

16.4.5.4 void USART_TransferAbortSendDMA (USART_Type * *base*, usart_dma_handle_t * *handle*)

This function aborts send data using DMA.

Parameters

base	USART peripheral base address
handle	Pointer to usart_dma_handle_t structure

16.4.5.5 void USART_TransferAbortReceiveDMA (USART_Type * *base*, usart_dma_handle_t * *handle*)

This function aborts the received data using DMA.

Parameters

base	USART peripheral base address
handle	Pointer to usart_dma_handle_t structure

16.4.5.6 status_t USART_TransferGetReceiveCountDMA (USART_Type * *base*, usart_dma_handle_t * *handle*, uint32_t * *count*)

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

Parameters

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

Return values

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

16.4.5.7 status_t USART_TransferGetSendCountDMA (USART_Type * *base*, usart_dma_handle_t * *handle*, uint32_t * *count*)

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

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
count	Sent bytes count.

Return values

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

16.5 USART FreeRTOS Driver

16.5.1 Overview

Files

- file fsl_usart_freertos.h

Data Structures

- struct rtos_usart_config
  FLEX USART configuration structure. *More...*
- struct usart_rtos_handle_t
  FLEX USART FreeRTOS handle. *More...*

Driver version

- #define FSL_USART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  USART FreeRTOS driver version.

USART RTOS Operation

- int USART_RTOS_Init (usart_rtos_handle_t *handle, usart_handle_t *t_handle, const struct rtos_usart_config *cfg)
  Initializes a USART instance for operation in RTOS.
- int USART_RTOS_Deinit (usart_rtos_handle_t *handle)
  Deinitializes a USART instance for operation.

USART transactional Operation

- int USART_RTOS_Send (usart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length)
  Sends data in the background.
- int USART_RTOS_Receive (usart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length, size_t *received)
  Receives data.

16.5.2 Data Structure Documentation

16.5.2.1 struct rtos_usart_config

Data Fields

- USART_Type * base

USART base address.
- uint32_t srcclk
  USART source clock in Hz.
- uint32_t baudrate
  Desired communication speed.
- usart_parity_mode_t parity
  Parity setting.
- usart_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.

16.5.2.2 struct usart_rtos_handle_t

Data Fields

- USART_Type * base
  USART base address.
- usart_transfer_t txTransfer
  TX transfer structure.
- usart_transfer_t rxTransfer
  RX transfer structure.
- SemaphoreHandle_t rxSemaphore
  RX semaphore for resource sharing.
- SemaphoreHandle_t txSemaphore
  TX semaphore for resource sharing.
- EventGroupHandle_t rxEvent
  RX completion event.
- EventGroupHandle_t txEvent
  TX completion event.
- void * t_state
  Transactional state of the underlying driver.

16.5.3 Macro Definition Documentation

16.5.3.1 #define FSL_USART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

16.5.4 Function Documentation

16.5.4.1 int USART_RTOS_Init (usart_rtos_handle_t * *handle*, usart_handle_t * *t_handle*, const struct rtos_usart_config * *cfg*)

Parameters

handle	The RTOS USART handle, the pointer to allocated space for RTOS context.
t_handle	The pointer to allocated space where to store transactional layer internal state.
cfg	The pointer to the parameters required to configure the USART after initialization.

Returns

0 succeed, others fail.

16.5.4.2 int USART_RTOS_Deinit (usart_rtos_handle_t * *handle*)

This function deinitializes the USART module, sets all register values to reset value, and releases the resources.

Parameters

handle	The RTOS USART handle.

16.5.4.3 int USART_RTOS_Send (usart_rtos_handle_t * *handle,* uint8_t * *buffer,* uint32_t *length*)

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

Parameters

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

16.5.4.4 int USART_RTOS_Receive (usart_rtos_handle_t * *handle,* uint8_t * *buffer,* uint32_t *length,* size_t * *received*)

This function receives data from USART. It is a synchronous API. If data is immediately available, it is returned immediately and the number of bytes received.

Parameters

handle	The RTOS USART handle.
buffer	The pointer to buffer where to write received data.
length	The number of bytes to receive.
received	The pointer to a variable of size_t where the number of received data is filled.

16.6 USART CMSIS Driver

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

The USART driver includes transactional APIs.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements please write custom code.

16.6.1 USART Send Methods

16.6.1.1 USART Send/receive using an interrupt method

```
/* USART  callback */
void USART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txOnGoing = false;
    }
}
Driver_USART0.Initialize(USART_Callback);
Driver_USART0.PowerControl(ARM_POWER_FULL);
/* Send g_tipString out. */
txOnGoing = true;
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);

/* Wait send finished */
while (txOnGoing)
{
}
```

16.6.1.2 USART Send/Receive using the DMA method

```
/* USART  callback */
void USART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txOnGoing = false;
    }

}

Driver_USART0.Initialize(USART_Callback);
DMA_Init(DMA0);
Driver_USART0.PowerControl(ARM_POWER_FULL);

/* Send g_tipString out. */
txOnGoing = true;
```

```
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);

/* Wait send finished */
while (txOnGoing)
{
}
```

Chapter 17
GINT: Group GPIO Input Interrupt Driver

17.1 Overview

The MCUXpresso SDK provides a driver for the Group GPIO Input Interrupt (GINT).

It can configure one or more pins to generate a group interrupt when the pin conditions are met. The pins do not have to be configured as GPIO pins.

17.2 Group GPIO Input Interrupt Driver operation

GINT_SetCtrl() and GINT_ConfigPins() functions configure the pins.

GINT_EnableCallback() function enables the callback functionality. Callback function is called when the pin conditions are met.

17.3 Typical use case

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

Files

- file fsl_gint.h

Typedefs

- typedef void(∗ gint_cb_t)(void)
  GINT Callback function.

Enumerations

- enum gint_comb_t {
  kGINT_CombineOr = 0U,
  kGINT_CombineAnd = 1U }
  GINT combine inputs type.
- enum gint_trig_t {
  kGINT_TrigEdge = 0U,
  kGINT_TrigLevel = 1U }
  GINT trigger type.

Functions

- void GINT_Init (GINT_Type ∗base)
  Initialize GINT peripheral.
- void GINT_SetCtrl (GINT_Type ∗base, gint_comb_t comb, gint_trig_t trig, gint_cb_t callback)

Setup GINT peripheral control parameters.
- void GINT_GetCtrl (GINT_Type *base, gint_comb_t *comb, gint_trig_t *trig, gint_cb_t *callback)
  Get GINT peripheral control parameters.
- void GINT_ConfigPins (GINT_Type *base, gint_port_t port, uint32_t polarityMask, uint32_t enableMask)
  Configure GINT peripheral pins.
- void GINT_GetConfigPins (GINT_Type *base, gint_port_t port, uint32_t *polarityMask, uint32_t *enableMask)
  Get GINT peripheral pin configuration.
- void GINT_EnableCallback (GINT_Type *base)
  Enable callback.
- void GINT_DisableCallback (GINT_Type *base)
  Disable callback.
- static void GINT_ClrStatus (GINT_Type *base)
  Clear GINT status.
- static uint32_t GINT_GetStatus (GINT_Type *base)
  Get GINT status.
- void GINT_Deinit (GINT_Type *base)
  Deinitialize GINT peripheral.

Driver version

- #define FSL_GINT_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  Version 2.0.3.

17.4 Macro Definition Documentation

17.4.1 #define FSL_GINT_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

17.5 Typedef Documentation

17.5.1 typedef void(* gint_cb_t)(void)

17.6 Enumeration Type Documentation

17.6.1 enum gint_comb_t

Enumerator

kGINT_CombineOr A grouped interrupt is generated when any one of the enabled inputs is active.
kGINT_CombineAnd A grouped interrupt is generated when all enabled inputs are active.

17.6.2 enum gint_trig_t

Enumerator

kGINT_TrigEdge Edge triggered based on polarity.
kGINT_TrigLevel Level triggered based on polarity.

17.7 Function Documentation

17.7.1 void GINT_Init (GINT_Type * *base*)

This function initializes the GINT peripheral and enables the clock.

Parameters

base	Base address of the GINT peripheral.

Return values

None.	

17.7.2 void GINT_SetCtrl (GINT_Type * *base*, gint_comb_t *comb*, gint_trig_t *trig*, gint_cb_t *callback*)

This function sets the control parameters of GINT peripheral.

Parameters

base	Base address of the GINT peripheral.
comb	Controls if the enabled inputs are logically ORed or ANDed for interrupt generation.
trig	Controls if the enabled inputs are level or edge sensitive based on polarity.
callback	This function is called when configured group interrupt is generated.

Return values

None.	

17.7.3 void GINT_GetCtrl (GINT_Type * *base*, gint_comb_t * *comb*, gint_trig_t * *trig*, gint_cb_t * *callback*)

This function returns the control parameters of GINT peripheral.

Parameters

base	Base address of the GINT peripheral.
comb	Pointer to store combine input value.
trig	Pointer to store trigger value.
callback	Pointer to store callback function.

Return values

None.	

17.7.4 void GINT_ConfigPins (GINT_Type * *base*, gint_port_t *port*, uint32_t *polarityMask*, uint32_t *enableMask*)

This function enables and controls the polarity of enabled pin(s) of a given port.

Parameters

base	Base address of the GINT peripheral.
port	Port number.
polarityMask	Each bit position selects the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH.
enableMask	Each bit position selects if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.

Return values

None.	

17.7.5 void GINT_GetConfigPins (GINT_Type * *base*, gint_port_t *port*, uint32_t * *polarityMask*, uint32_t * *enableMask*)

This function returns the pin configuration of a given port.

Parameters

base	Base address of the GINT peripheral.

port	Port number.
polarityMask	Pointer to store the polarity mask Each bit position indicates the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH.
enableMask	Pointer to store the enable mask. Each bit position indicates if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.

Return values

None.	

17.7.6 void GINT_EnableCallback (GINT_Type * *base*)

This function enables the interrupt for the selected GINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

base	Base address of the GINT peripheral.

Return values

None.	

17.7.7 void GINT_DisableCallback (GINT_Type * *base*)

This function disables the interrupt for the selected GINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

base	Base address of the peripheral.

Return values

None.	

17.7.8 static void GINT_ClrStatus (GINT_Type * *base*) [inline], [static]

This function clears the GINT status bit.

Parameters

base	Base address of the GINT peripheral.

Return values

None.	

17.7.9 static uint32_t GINT_GetStatus (GINT_Type * *base*) [inline], [static]

This function returns the GINT status.

Parameters

base	Base address of the GINT peripheral.

Return values

status	= 0 No group interrupt request. = 1 Group interrupt request active.

17.7.10 void GINT_Deinit (GINT_Type * *base*)

This function disables the GINT clock.

Parameters

base	Base address of the GINT peripheral.

Return values

None.	

Chapter 18
Hashcrypt: The Cryptographic Accelerator

18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Hashcrypt peripheral. The Hashcrypt peripheral provides one or more engines to perform specific symmetric crypto algorithms, including hashing and en/decryption. The cryptographic acceleration is normally used in conjunction with pure-hardware blocks for hashing and symmetric cryptography, thereby providing performance and energy efficiency for a range of cryptographic uses.

Blocking synchronous APIs are provided for selected cryptographic algorithms using Hashcrypt hardware. The driver interface intends to be easily integrated with generic software crypto libraries such as mbed-TLS. The Hashcrypt operations are complete (and results are made availabe for further usage) when a function returns. When called, these functions do not return until an Hashcrypt operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status and also for plaintext or ciphertext data movements. These functions provide typical interface to upper layer or application software. There is one non-blocking function provided for the purpose of background hashing. HASHCRYPT_SHA_UpdateNonBlocking() starts hashing of an input message while the CPU can continue executing.

18.2 Hashcrypt Driver Initialization and deinitialization

Hashcrypt Driver is initialized by calling the HASHCRYPT_Init() function, it enables clock and disables reset for Hashcrypt peripheral. Hashcrypt Driver is deinitialized by calling the HASHCRYPT_Deinit() function, it sisables clock and enables reset.

18.3 Comments about API usage in RTOS

Hashcrypt operations provided by this driver are not re-entrant. Thus, application software shall ensure the Hashcrypt module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

18.4 Comments about API usage in interrupt handler

APIs can be used from interrupt handler although execution time shall be considered (interrupt latency increases considerably).

18.5 Hashcrypt Driver Examples

18.5.1 Simple examples

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

Modules

- Hashcrypt AES
- Hashcrypt Background HASH
- Hashcrypt HASH

18.6 Hashcrypt AES

18.6.1 Overview

Data Structures

- struct hashcrypt_handle_t
  Specify HASHCRYPT's key resource. More...

Macros

- #define HASHCRYPT_AES_BLOCK_SIZE 16U
  AES block size in bytes.

Enumerations

- enum hashcrypt_aes_mode_t {
  kHASHCRYPT_AesEcb = 0U,
  kHASHCRYPT_AesCbc = 1U,
  kHASHCRYPT_AesCtr = 2U }
  AES mode.
- enum hashcrypt_aes_keysize_t {
  kHASHCRYPT_Aes128 = 0U,
  kHASHCRYPT_Aes192 = 1U,
  kHASHCRYPT_Aes256 = 2U,
  kHASHCRYPT_InvalidKey = 3U }
  Size of AES key.
- enum hashcrypt_key_t {
  kHASHCRYPT_UserKey = 0xc3c3U,
  kHASHCRYPT_SecretKey = 0x3c3cU }
  HASHCRYPT key source selection.

Functions

- status_t HASHCRYPT_AES_SetKey (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *key, size_t keySize)
  Set AES key to hashcrypt_handle_t struct and optionally to HASHCRYPT.
- status_t HASHCRYPT_AES_EncryptEcb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size)
  Encrypts AES on one or multiple 128-bit block(s).
- status_t HASHCRYPT_AES_DecryptEcb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size)
  Decrypts AES on one or multiple 128-bit block(s).
- status_t HASHCRYPT_AES_EncryptCbc (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16])

Encrypts AES using CBC block mode.

- status_t HASHCRYPT_AES_DecryptCbc (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16])
  Decrypts AES using CBC block mode.
- status_t HASHCRYPT_AES_CryptCtr (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *input, uint8_t *output, size_t size, uint8_t counter[HASHCRYPT_AES_BLOCK_SIZE], uint8_t counterlast[HASHCRYPT_AES_BLOCK_SIZE], size_t *szLeft)
  Encrypts or decrypts AES using CTR block mode.
- status_t HASHCRYPT_AES_CryptOfb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *input, uint8_t *output, size_t size, const uint8_t iv[HASHCRYPT_AES_BLOCK_SIZE])
  Encrypts or decrypts AES using OFB block mode.
- status_t HASHCRYPT_AES_EncryptCfb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16])
  Encrypts AES using CFB block mode.
- status_t HASHCRYPT_AES_DecryptCfb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16])
  Decrypts AES using CFB block mode.

18.6.2 Data Structure Documentation

18.6.2.1 struct _hashcrypt_handle

Data Fields

- uint32_t keyWord [8]
  Copy of user key (set by HASHCRYPT_AES_SetKey().
- hashcrypt_key_t keyType
  For operations with key (such as AES encryption/decryption), specify key type.

Field Documentation

(1) uint32_t hashcrypt_handle_t::keyWord[8]

(2) hashcrypt_key_t hashcrypt_handle_t::keyType

18.6.3 Enumeration Type Documentation

18.6.3.1 enum hashcrypt_aes_mode_t

Enumerator

kHASHCRYPT_AesEcb AES ECB mode.
kHASHCRYPT_AesCbc AES CBC mode.
kHASHCRYPT_AesCtr AES CTR mode.

18.6.3.2 enum hashcrypt_aes_keysize_t

Enumerator

kHASHCRYPT_Aes128 AES 128 bit key.
kHASHCRYPT_Aes192 AES 192 bit key.
kHASHCRYPT_Aes256 AES 256 bit key.
kHASHCRYPT_InvalidKey AES invalid key.

18.6.3.3 enum hashcrypt_key_t

Enumerator

kHASHCRYPT_UserKey HASHCRYPT user key.
kHASHCRYPT_SecretKey HASHCRYPT secret key (dedicated hw bus from PUF)

18.6.4 Function Documentation

18.6.4.1 status_t HASHCRYPT_AES_SetKey (HASHCRYPT_Type * *base*, hashcrypt_handle_t * *handle*, const uint8_t * *key*, size_t *keySize*)

Sets the AES key for encryption/decryption with the hashcrypt_handle_t structure. The hashcrypt_handle_t input argument specifies key source.

Parameters

base	HASHCRYPT peripheral base address.
handle	Handle used for the request.
key	0-mod-4 aligned pointer to AES key.
keySize	AES key size in bytes. Shall equal 16, 24 or 32.

Returns

status from set key operation

18.6.4.2 status_t HASHCRYPT_AES_EncryptEcb (HASHCRYPT_Type * *base*, hashcrypt_handle_t * *handle*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, size_t *size*)

Encrypts AES. The source plaintext and destination ciphertext can overlap in system memory.

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	plaintext	Input plain text to encrypt
out	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.

Returns

Status from encrypt operation

18.6.4.3 status_t HASHCRYPT_AES_DecryptEcb (HASHCRYPT_Type * *base*, hashcrypt_handle_t * *handle*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, size_t *size*)

Decrypts AES. The source ciphertext and destination plaintext can overlap in system memory.

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	ciphertext	Input plain text to encrypt
out	*plaintext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.

Returns

Status from decrypt operation

18.6.4.4 status_t HASHCRYPT_AES_EncryptCbc (HASHCRYPT_Type * *base*, hashcrypt_handle_t * *handle*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, size_t *size*, const uint8_t *iv[16]*)

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	plaintext	Input plain text to encrypt
out	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

18.6.4.5 status_t HASHCRYPT_AES_DecryptCbc (HASHCRYPT_Type * *base*, hashcrypt_handle_t * *handle*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, size_t *size*, const uint8_t *iv[16]*)

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	ciphertext	Input cipher text to decrypt
out	*plaintext*	Output plain text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.

Returns

Status from decrypt operation

18.6.4.6 status_t HASHCRYPT_AES_CryptCtr (HASHCRYPT_Type * *base*, hashcrypt_handle_t * *handle*, const uint8_t * *input*, uint8_t * *output*, size_t *size*, uint8_t *counter[HASHCRYPT_AES_BLOCK_SIZE]*, uint8_t *counterlast[HASHCRYPT_AES_BLOCK_SIZE]*, size_t * *szLeft*)

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	input	Input data for CTR block mode
out	*output*	Output data for CTR block mode
	size	Size of input and output data in bytes
in,out	*counter*	Input counter (updates on return)
out	*counterlast*	Output cipher of last counter, for chained CTR calls (statefull encryption). NULL can be passed if chained calls are not used.
out	*szLeft*	Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from encrypt operation

18.6.4.7 status_t HASHCRYPT_AES_CryptOfb (HASHCRYPT_Type * *base*, hashcrypt_handle_t * *handle*, const uint8_t * *input*, uint8_t * *output*, size_t *size*, const uint8_t *iv[HASHCRYPT_AES_BLOCK_SIZE]*)

Encrypts or decrypts AES using OFB block mode. AES OFB mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	input	Input data for OFB block mode
out	*output*	Output data for OFB block mode
	size	Size of input and output data in bytes
	iv	Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

18.6.4.8 status_t HASHCRYPT_AES_EncryptCfb (HASHCRYPT_Type * *base,* hashcrypt_handle_t * *handle,* const uint8_t * *plaintext,* uint8_t * *ciphertext,* size_t *size,* const uint8_t *iv[16]*)

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	plaintext	Input plain text to encrypt
out	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

18.6.4.9 status_t HASHCRYPT_AES_DecryptCfb (HASHCRYPT_Type * *base*, hashcrypt_handle_t * *handle*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, size_t *size*, const uint8_t *iv[16]*)

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	ciphertext	Input cipher text to decrypt
out	*plaintext*	Output plaintext text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

18.7 Hashcrypt HASH

18.7.1 Overview

Data Structures

- struct hashcrypt_hash_ctx_t
  Storage type used to save hash context. More...

Macros

- #define HASHCRYPT_HASH_CTX_SIZE 30
  HASHCRYPT HASH Context size.

Typedefs

- typedef void(∗ hashcrypt_callback_t)(HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, status_t status, void ∗userData)
  HASHCRYPT background hash callback function.

Functions

- status_t HASHCRYPT_SHA (HASHCRYPT_Type ∗base, hashcrypt_algo_t algo, const uint8_t ∗input, size_t inputSize, uint8_t ∗output, size_t ∗outputSize)
  Create HASH on given data.
- status_t HASHCRYPT_SHA_Init (HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, hashcrypt_algo_t algo)
  Initialize HASH context.
- status_t HASHCRYPT_SHA_Update (HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, const uint8_t ∗input, size_t inputSize)
  Add data to current HASH.
- status_t HASHCRYPT_SHA_Finish (HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, uint8_t ∗output, size_t ∗outputSize)
  Finalize hashing.

18.7.2 Data Structure Documentation

18.7.2.1 struct hashcrypt_hash_ctx_t

Data Fields

- uint32_t x [HASHCRYPT_HASH_CTX_SIZE]
  storage

18.7.3 Macro Definition Documentation

18.7.3.1 #define HASHCRYPT_HASH_CTX_SIZE 30

18.7.4 Typedef Documentation

18.7.4.1 typedef void(∗ hashcrypt_callback_t)(HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, status_t status, void ∗userData)

18.7.5 Function Documentation

18.7.5.1 status_t HASHCRYPT_SHA (HASHCRYPT_Type ∗ *base,* hashcrypt_algo_t *algo,* const uint8_t ∗ *input,* size_t *inputSize,* uint8_t ∗ *output,* size_t ∗ *outputSize*)

Perform the full SHA in one function call. The function is blocking.

Parameters

	base	HASHCRYPT peripheral base address
	algo	Underlaying algorithm to use for hash computation.
	input	Input data
	inputSize	Size of input data in bytes
out	*output*	Output hash data
out	*outputSize*	Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

18.7.5.2 status_t HASHCRYPT_SHA_Init (HASHCRYPT_Type ∗ *base,* hashcrypt_hash_ctx_t ∗ *ctx,* hashcrypt_algo_t *algo*)

This function initializes the HASH.

Parameters

	base	HASHCRYPT peripheral base address
out	*ctx*	Output hash context
	algo	Underlaying algorithm to use for hash computation.

Returns

Status of initialization

18.7.5.3 status_t HASHCRYPT_SHA_Update (HASHCRYPT_Type * *base*, hashcrypt_hash_ctx_t * *ctx*, const uint8_t * *input*, size_t *inputSize*)

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed. The functions blocks. If it returns kStatus_Success, the running hash has been updated (HASHCRYPT has processed the input data), so the memory at input pointer can be released back to system. The HASHCRYPT context buffer is updated with the running hash and with all necessary information to support possible context switch.

Parameters

	base	HASHCRYPT peripheral base address
in,out	*ctx*	HASH context
	input	Input data
	inputSize	Size of input data in bytes

Returns

Status of the hash update operation

18.7.5.4 status_t HASHCRYPT_SHA_Finish (HASHCRYPT_Type * *base*, hashcrypt_hash_ctx_t * *ctx*, uint8_t * *output*, size_t * *outputSize*)

Outputs the final hash (computed by HASHCRYPT_HASH_Update()) and erases the context.

Parameters

	base	HASHCRYPT peripheral base address
in,out	*ctx*	Input hash context
out	*output*	Output hash data
in,out	*outputSize*	Optional parameter (can be passed as NULL). On function entry, it specifies the size of output[] buffer. On function return, it stores the number of updated output bytes.

Returns

Status of the hash finish operation

18.8 Hashcrypt Background HASH

18.8.1 Overview

Functions

- void HASHCRYPT_SHA_SetCallback (HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx, hashcrypt_callback_t callback, void *userData)
  Initializes the HASHCRYPT handle for background hashing.
- status_t HASHCRYPT_SHA_UpdateNonBlocking (HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx, const uint8_t *input, size_t inputSize)
  Create running hash on given data.

18.8.2 Function Documentation

18.8.2.1 void HASHCRYPT_SHA_SetCallback (HASHCRYPT_Type * *base*, hashcrypt_hash_ctx_t * *ctx*, hashcrypt_callback_t *callback*, void * *userData*)

This function initializes the hash context for background hashing (Non-blocking) APIs. This is less typical interface to hash function, but can be used for parallel processing, when main CPU has something else to do. Example is digital signature RSASSA-PKCS1-V1_5-VERIFY((n,e),M,S) algorithm, where background hashing of M can be started, then CPU can compute S^e mod n (in parallel with background hashing) and once the digest becomes available, CPU can proceed to comparison of EM with EM'.

Parameters

	base	HASHCRYPT peripheral base address.
out	*ctx*	Hash context.
	callback	Callback function.
	userData	User data (to be passed as an argument to callback function, once callback is invoked from isr).

18.8.2.2 status_t HASHCRYPT_SHA_UpdateNonBlocking (HASHCRYPT_Type * *base*, hashcrypt_hash_ctx_t * *ctx*, const uint8_t * *input*, size_t *inputSize*)

Configures the HASHCRYPT to compute new running hash as AHB master and returns immediately. HASHCRYPT AHB Master mode supports only aligned `input` address and can be called only once per continuous block of data. Every call to this function must be preceded with HASHCRYPT_SHA_Init() and finished with HASHCRYPT_SHA_Finish(). Once callback function is invoked by HASHCRYPT isr, it should set a flag for the main application to finalize the hashing (padding) and to read out the final digest by calling HASHCRYPT_SHA_Finish().

Parameters

base	HASHCRYPT peripheral base address
ctx	Specifies callback. Last incomplete 512-bit block of the input is copied into clear buffer for padding.
input	32-bit word aligned pointer to Input data.
inputSize	Size of input data in bytes (must be word aligned)

Returns

Status of the hash update operation.

Chapter 19
IAP: In Application Programming Driver

19.1 Overview

The MCUXpresso SDK provides a driver for the In Application Programming (IAP).

It provides a set of functions to call the on-chip in application programming interface. User code executing from on-chip RAM can call these function to read information like part id, read and write flash, read and write ffr.

19.2 In Application Programming operation

FLASH_Init() Initializes the global flash properties structure members

FLASH_Erase() Erases the flash sectors encompassed by parameters passed into function

FLASH_Program() Programs flash with data at locations passed in through parameters

FLASH_VerifyErase() Verifies an erasure of the desired flash area hase been erased

FLASH_VerifyProgram() Verifies programming of the desired flash area hase been programed

FLASH_GetProperty() Returns the desired flash property.

FFR_Init() Generic APIs for FFR

FFR_Deinit() Generic APIs for FFR

FFR_CustomerPagesInit() APIs to access CFPA pages

FFR_InfieldPageWrite() APIs to access CFPA pages

FFR_GetCustomerInfieldData() APIs to access CMPA pages

FFR_GetCustomerData() Read data stored in 'Customer Factory CFG Page'

FFR_KeystoreWrite() Read data stored in 'Customer Factory CFG Page'

FFR_KeystoreGetAC() Read data stored in 'Customer Factory CFG Page'

FFR_KeystoreGetKC() Read data stored in 'Customer Factory CFG Page'

FFR_GetUUID() Read data stored in 'NXP Manufacuring Programmed CFG Page'

FFR_GetManufactureData() Read data stored in 'NXP Manufacuring Programmed CFG Page'

kb_init() Initialize ROM API for a given operation

kb_deinit() Cleans up the ROM API context

kb_execute() Perform the operation configured during init

skboot_authenticate() Authenticate entry function with ARENA allocator init

HASH_IRQHandler() Interface for image authentication API

kb_init() Initialize ROM API for a given operation

kb_deinit() Cleans up the ROM API context

kb_execute() Perform the operation configured during init

skboot_authenticate() Authenticate entry function with ARENA allocator init

HASH_IRQHandler() Interface for image authentication API

19.3 Typical use case

19.3.1 IAP Basic Operations

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/iap1

Modules

- IAP_FFR Driver
- IAP_KBP Driver

Files

- file fsl_iap.h

Data Structures

- struct flash_ecc_log_t
  Flash ECC log info. More...
- struct flash_mode_config_t
  Flash controller paramter config. More...
- struct flash_ffr_config_t
  Flash controller paramter config. More...
- struct flash_config_t
  Flash driver state information. More...

Enumerations

- enum flash_property_tag_t {
  kFLASH_PropertyPflashSectorSize = 0x00U,
  kFLASH_PropertyPflashTotalSize = 0x01U,
  kFLASH_PropertyPflashBlockSize = 0x02U,
  kFLASH_PropertyPflashBlockCount = 0x03U,
  kFLASH_PropertyPflashBlockBaseAddr = 0x04U,
  kFLASH_PropertyPflashPageSize = 0x30U,
  kFLASH_PropertyPflashSystemFreq = 0x31U,
  kFLASH_PropertyFfrSectorSize = 0x40U,
  kFLASH_PropertyFfrTotalSize = 0x41U,
  kFLASH_PropertyFfrBlockBaseAddr = 0x42U,
  kFLASH_PropertyFfrPageSize = 0x43U }

Enumeration for various flash properties.

- enum _flash_max_erase_page_value { kFLASH_MaxPagesToErase = 100U }
  Enumeration for flash max pages to erase.
- enum _flash_alignment_property {
  kFLASH_AlignementUnitVerifyErase = 4,
  kFLASH_AlignementUnitProgram = 512,
  kFLASH_AlignementUnitSingleWordRead = 16 }
  Enumeration for flash alignment property.
- enum _flash_read_ecc_option { , kFLASH_ReadWithEccOff = 1 }
  Enumeration for flash read ecc option.
- enum _flash_read_margin_option {
  kFLASH_ReadMarginNormal = 0,
  kFLASH_ReadMarginVsProgram = 1,
  kFLASH_ReadMarginVsErase = 2,
  kFLASH_ReadMarginIllegalBitCombination = 3 }
  Enumeration for flash read margin option.
- enum _flash_read_dmacc_option {
  kFLASH_ReadDmaccDisabled = 0,
  kFLASH_ReadDmaccEnabled = 1 }
  Enumeration for flash read dmacc option.
- enum _flash_ramp_control_option {
  kFLASH_RampControlDivisionFactorReserved = 0,
  kFLASH_RampControlDivisionFactor256 = 1,
  kFLASH_RampControlDivisionFactor128 = 2,
  kFLASH_RampControlDivisionFactor64 = 3 }
  Enumeration for flash ramp control option.

Functions

- status_t FLASH_Read (flash_config_t *config, uint32_t start, uint8_t *dest, uint32_t lengthInBytes)
  Reads flash at locations passed in through parameters.

Flash version

- enum _flash_driver_version_constants {
  kFLASH_DriverVersionName = 'F',
  kFLASH_DriverVersionMajor = 2,
  kFLASH_DriverVersionMinor = 1,
  kFLASH_DriverVersionBugfix = 3 }
  Flash driver version for ROM.
- #define MAKE_VERSION(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))
  Constructs the version number for drivers.
- #define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))
  Flash driver version for SDK.

Flash configuration

- #define FSL_FEATURE_FLASH_IP_IS_C040HD_ATFC (1)

Flash IP Type.
- #define **FSL_FEATURE_FLASH_IP_IS_C040HD_FC** (0)

Flash status

- enum _flash_status {
  kStatus_FLASH_Success = MAKE_STATUS(kStatusGroupGeneric, 0),
  kStatus_FLASH_InvalidArgument = MAKE_STATUS(kStatusGroupGeneric, 4),
  kStatus_FLASH_SizeError = MAKE_STATUS(kStatusGroupFlashDriver, 0),
  kStatus_FLASH_AlignmentError,
  kStatus_FLASH_AddressError = MAKE_STATUS(kStatusGroupFlashDriver, 2),
  kStatus_FLASH_AccessError,
  kStatus_FLASH_ProtectionViolation,
  kStatus_FLASH_CommandFailure,
  kStatus_FLASH_UnknownProperty = MAKE_STATUS(kStatusGroupFlashDriver, 6),
  kStatus_FLASH_EraseKeyError = MAKE_STATUS(kStatusGroupFlashDriver, 7),
  kStatus_FLASH_RegionExecuteOnly,
  kStatus_FLASH_ExecuteInRamFunctionNotReady,
  kStatus_FLASH_CommandNotSupported = MAKE_STATUS(kStatusGroupFlashDriver, 11),
  kStatus_FLASH_ReadOnlyProperty = MAKE_STATUS(kStatusGroupFlashDriver, 12),
  kStatus_FLASH_InvalidPropertyValue,
  kStatus_FLASH_InvalidSpeculationOption,
  kStatus_FLASH_EccError,
  kStatus_FLASH_CompareError,
  kStatus_FLASH_RegulationLoss = MAKE_STATUS(kStatusGroupFlashDriver, 0x12),
  kStatus_FLASH_InvalidWaitStateCycles,
  kStatus_FLASH_OutOfDateCfpaPage,
  kStatus_FLASH_BlankIfrPageData = MAKE_STATUS(kStatusGroupFlashDriver, 0x21),
  kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce,
  kStatus_FLASH_ProgramVerificationNotAllowed,
  kStatus_FLASH_HashCheckError,
  kStatus_FLASH_SealedFfrRegion = MAKE_STATUS(kStatusGroupFlashDriver, 0x25),
  kStatus_FLASH_FfrRegionWriteBroken,
  kStatus_FLASH_NmpaAccessNotAllowed,
  kStatus_FLASH_CmpaCfgDirectEraseNotAllowed,
  kStatus_FLASH_FfrBankIsLocked = MAKE_STATUS(kStatusGroupFlashDriver, 0x29) }
  Flash driver status codes.
- #define kStatusGroupGeneric 0
  Flash driver status group.
- #define **kStatusGroupFlashDriver** 1
- #define MAKE_STATUS(group, code) ((((group)*100) + (code)))
  Constructs a status code value from a group and a code number.

Flash API key

- enum _flash_driver_api_keys { kFLASH_ApiEraseKey = FOUR_CHAR_CODE('l', 'f', 'e', 'k') }
  Enumeration for Flash driver API keys.

- #define FOUR_CHAR_CODE(a, b, c, d) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))
  Constructs the four character code for the Flash driver API key.

Initialization

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

Erasing

- status_t FLASH_Erase (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)
  Erases the flash sectors encompassed by parameters passed into function.

Programming

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

Verification

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

Properties

- status_t FLASH_GetProperty (flash_config_t *config, flash_property_tag_t whichProperty, uint32_t *value)
  Returns the desired flash property.

19.4 Data Structure Documentation

19.4.1 struct flash_ecc_log_t

19.4.2 struct flash_mode_config_t

19.4.3 struct flash_ffr_config_t

19.4.4 struct flash_config_t

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

Data Fields

- uint32_t PFlashBlockBase
  A base address of the first PFlash block.
- uint32_t PFlashTotalSize
  The size of the combined PFlash block.
- uint32_t PFlashBlockCount
  A number of PFlash blocks.
- uint32_t PFlashPageSize
  The size in bytes of a page of PFlash.
- uint32_t PFlashSectorSize
  The size in bytes of a sector of PFlash.

Field Documentation

(1) uint32_t flash_config_t::PFlashTotalSize

(2) uint32_t flash_config_t::PFlashBlockCount

(3) uint32_t flash_config_t::PFlashPageSize

(4) uint32_t flash_config_t::PFlashSectorSize

19.5 Macro Definition Documentation

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

19.5.2 #define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))

Version 2.1.4.

19.5.3 #define FSL_FEATURE_FLASH_IP_IS_C040HD_ATFC (1)

19.5.4 #define kStatusGroupGeneric 0

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

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

19.6 Enumeration Type Documentation

19.6.1 enum _flash_driver_version_constants

Enumerator

kFLASH_DriverVersionName Flash driver version name.
kFLASH_DriverVersionMajor Major flash driver version.
kFLASH_DriverVersionMinor Minor flash driver version.
kFLASH_DriverVersionBugfix Bugfix for flash driver version.

19.6.2 enum _flash_status

Enumerator

kStatus_FLASH_Success API is executed successfully.
kStatus_FLASH_InvalidArgument Invalid argument.
kStatus_FLASH_SizeError Error size.
kStatus_FLASH_AlignmentError Parameter is not aligned with the specified baseline.
kStatus_FLASH_AddressError Address is out of range.
kStatus_FLASH_AccessError Invalid instruction codes and out-of bound addresses.
kStatus_FLASH_ProtectionViolation The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_CommandFailure Run-time error during command execution.
kStatus_FLASH_UnknownProperty Unknown property.
kStatus_FLASH_EraseKeyError API erase key is invalid.
kStatus_FLASH_RegionExecuteOnly The current region is execute-only.
kStatus_FLASH_ExecuteInRamFunctionNotReady Execute-in-RAM function is not available.
kStatus_FLASH_CommandNotSupported Flash API is not supported.
kStatus_FLASH_ReadOnlyProperty The flash property is read-only.
kStatus_FLASH_InvalidPropertyValue The flash property value is out of range.
kStatus_FLASH_InvalidSpeculationOption The option of flash prefetch speculation is invalid.
kStatus_FLASH_EccError A correctable or uncorrectable error during command execution.
kStatus_FLASH_CompareError Destination and source memory contents do not match.
kStatus_FLASH_RegulationLoss A loss of regulation during read.
kStatus_FLASH_InvalidWaitStateCycles The wait state cycle set to r/w mode is invalid.
kStatus_FLASH_OutOfDateCfpaPage CFPA page version is out of date.
kStatus_FLASH_BlankIfrPageData Blank page cannnot be read.
kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce Encrypted flash subregions are not erased at once.
kStatus_FLASH_ProgramVerificationNotAllowed Program verification is not allowed when the encryption is enabled.
kStatus_FLASH_HashCheckError Hash check of page data is failed.
kStatus_FLASH_SealedFfrRegion The FFR region is sealed.
kStatus_FLASH_FfrRegionWriteBroken The FFR Spec region is not allowed to be written discontinuously.

kStatus_FLASH_NmpaAccessNotAllowed The NMPA region is not allowed to be read/written/erased.

kStatus_FLASH_CmpaCfgDirectEraseNotAllowed The CMPA Cfg region is not allowed to be erased directly.

kStatus_FLASH_FfrBankIsLocked The FFR bank region is locked.

19.6.3 enum _flash_driver_api_keys

Note

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

Enumerator

kFLASH_ApiEraseKey Key value used to validate all flash erase APIs.

19.6.4 enum flash_property_tag_t

Enumerator

kFLASH_PropertyPflashSectorSize Pflash sector size property.
kFLASH_PropertyPflashTotalSize Pflash total size property.
kFLASH_PropertyPflashBlockSize Pflash block size property.
kFLASH_PropertyPflashBlockCount Pflash block count property.
kFLASH_PropertyPflashBlockBaseAddr Pflash block base address property.
kFLASH_PropertyPflashPageSize Pflash page size property.
kFLASH_PropertyPflashSystemFreq System Frequency System Frequency.
kFLASH_PropertyFfrSectorSize FFR sector size property.
kFLASH_PropertyFfrTotalSize FFR total size property.
kFLASH_PropertyFfrBlockBaseAddr FFR block base address property.
kFLASH_PropertyFfrPageSize FFR page size property.

19.6.5 enum _flash_max_erase_page_value

Enumerator

kFLASH_MaxPagesToErase The max value in pages to erase.

19.6.6 enum _flash_alignment_property

Enumerator

kFLASH_AlignementUnitVerifyErase The alignment unit in bytes used for verify erase operation.
kFLASH_AlignementUnitProgram The alignment unit in bytes used for program operation.
kFLASH_AlignementUnitSingleWordRead The alignment unit in bytes used for verify program operation. The alignment unit in bytes used for SingleWordRead command.

19.6.7 enum _flash_read_ecc_option

Enumerator

kFLASH_ReadWithEccOff ECC is on.

19.6.8 enum _flash_read_margin_option

Enumerator

kFLASH_ReadMarginNormal Normal read.
kFLASH_ReadMarginVsProgram Margin vs. program
kFLASH_ReadMarginVsErase Margin vs. erase
kFLASH_ReadMarginIllegalBitCombination Illegal bit combination.

19.6.9 enum _flash_read_dmacc_option

Enumerator

kFLASH_ReadDmaccDisabled Memory word.
kFLASH_ReadDmaccEnabled DMACC word.

19.6.10 enum _flash_ramp_control_option

Enumerator

kFLASH_RampControlDivisionFactorReserved Reserved.
kFLASH_RampControlDivisionFactor256 clk48mhz / 256 = 187.5KHz
kFLASH_RampControlDivisionFactor128 clk48mhz / 128 = 375KHz
kFLASH_RampControlDivisionFactor64 clk48mhz / 64 = 750KHz

19.7 Function Documentation

19.7.1 status_t FLASH_Init (flash_config_t * *config*)

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

Parameters

config	Pointer to the storage for the driver runtime state.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.
kStatus_FLASH_-CommandNotSupported	Flash API is not supported.
kStatus_FLASH_Ecc-Error	A correctable or uncorrectable error during command execution.

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

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

Parameters

config	The pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be erased. The start address need to be 512bytes-aligned.
lengthInBytes	The length, given in bytes (not words or long-words) to be erased. Must be 512bytes-aligned.
key	The value used to validate all flash erase APIs.

Return values

kStatus_FLASH_Success	API was executed successfully; the appropriate number of flash sectors based on the desired start address and length were erased successfully.

kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-AlignmentError	The parameter is not aligned with the specified baseline.
kStatus_FLASH_Address-Error	The address is out of range.
kStatus_FLASH_Erase-KeyError	The API erase key is invalid.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.
kStatus_FLASH_-CommandNotSupported	Flash API is not supported.
kStatus_FLASH_Ecc-Error	A correctable or uncorrectable error during command execution.

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

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

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be programmed. Must be 512bytes-aligned.
src	A pointer to the source buffer of data that is to be programmed into the flash.
lengthInBytes	The length, given in bytes (not words or long-words), to be programmed. Must be 512bytes-aligned.

Return values

kStatus_FLASH_Success	API was executed successfully; the desired data were programed successfully into flash based on desired start address and length.

kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FLASH_Address-Error	Address is out of range.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.
kStatus_FLASH_-CommandNotSupported	Flash API is not supported.
kStatus_FLASH_Ecc-Error	A correctable or uncorrectable error during command execution.

19.7.4 status_t FLASH_Read (flash_config_t * *config*, uint32_t *start*, uint8_t * *dest*, uint32_t *lengthInBytes*)

This function read the flash memory from a given flash area as determined by the start address and the length.

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be read.
dest	A pointer to the dest buffer of data that is to be read from the flash.
lengthInBytes	The length, given in bytes (not words or long-words), to be read.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

kStatus_FLASH_AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FLASH_AddressError	Address is out of range.
kStatus_FLASH_AccessError	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_CommandFailure	Run-time error during the command execution.
kStatus_FLASH_CommandFailure	Run-time error during the command execution.
kStatus_FLASH_CommandNotSupported	Flash API is not supported.
kStatus_FLASH_EccError	A correctable or uncorrectable error during command execution.

19.7.5 status_t FLASH_VerifyErase (flash_config_t * *config*, uint32_t *start*, uint32_t *lengthInBytes*)

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

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be verified. The start address need to be 512bytes-aligned.
lengthInBytes	The length, given in bytes (not words or long-words), to be verified. Must be 512bytes-aligned.

Return values

kStatus_FLASH_Success	API was executed successfully; the specified FLASH region has been erased.
kStatus_FLASH_InvalidArgument	An invalid argument is provided.

kStatus_FLASH_AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FLASH_AddressError	Address is out of range.
kStatus_FLASH_AccessError	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_CommandFailure	Run-time error during the command execution.
kStatus_FLASH_CommandFailure	Run-time error during the command execution.
kStatus_FLASH_CommandNotSupported	Flash API is not supported.
kStatus_FLASH_EccError	A correctable or uncorrectable error during command execution.

19.7.6 status_t FLASH_VerifyProgram (flash_config_t * *config*, uint32_t *start*, uint32_t *lengthInBytes*, const uint8_t * *expectedData*, uint32_t * *failedAddress*, uint32_t * *failedData*)

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

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be verified. need be 512bytes-aligned.
lengthInBytes	The length, given in bytes (not words or long-words), to be verified. need be 512bytes-aligned.
expectedData	A pointer to the expected data that is to be verified against.
failedAddress	A pointer to the returned failing address.
failedData	A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

kStatus_FLASH_Success	API was executed successfully; the desired data have been successfully programed into specified FLASH region.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FLASH_Address-Error	Address is out of range.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.
kStatus_FLASH_-CommandNotSupported	Flash API is not supported.
kStatus_FLASH_Ecc-Error	A correctable or uncorrectable error during command execution.

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

Parameters

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

Return values

kStatus_FLASH_Success	API was executed successfully; the flash property was stored to value.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

kStatus_FLASH_-UnknownProperty	An unknown property tag.

19.8 IAP_FFR Driver

19.8.1 Overview

Files

- file fsl_iap_ffr.h

Macros

- #define ALIGN_DOWN(x, a) ((x) & (uint32_t)(-((int32_t)(a))))
  Alignment(down) utility.
- #define ALIGN_UP(x, a) (-((int32_t)((uint32_t)(-((int32_t)(x))) & (uint32_t)(-((int32_t)(a))))))
  Alignment(up) utility.

Enumerations

- enum _flash_ffr_page_offset {
  kFfrPageOffset_CFPA = 0,
  kFfrPageOffset_CFPA_Scratch = 0,
  kFfrPageOffset_CFPA_Cfg = 1,
  kFfrPageOffset_CFPA_CfgPong = 2,
  kFfrPageOffset_CMPA = 3,
  kFfrPageOffset_CMPA_Cfg = 3,
  kFfrPageOffset_CMPA_Key = 4,
  kFfrPageOffset_NMPA = 7,
  kFfrPageOffset_NMPA_Romcp = 7,
  kFfrPageOffset_NMPA_Repair = 9,
  kFfrPageOffset_NMPA_Cfg = 15,
  kFfrPageOffset_NMPA_End = 16 }
  flash ffr page offset.
- enum _flash_ffr_page_num {
  kFfrPageNum_CFPA = 3,
  kFfrPageNum_CMPA = 4,
  kFfrPageNum_NMPA = 10 }
  flash ffr page number.

Flash IFR version

- #define FSL_FLASH_IFR_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  Flash IFR driver version for SDK.

FFR APIs

- status_t FFR_Init (flash_config_t *config)
  Initializes the global FFR properties structure members.
- status_t FFR_Lock_All (flash_config_t *config)
  Enable firewall for all flash banks.
- status_t FFR_InfieldPageWrite (flash_config_t *config, uint8_t *page_data, uint32_t valid_len)
  APIs to access CFPA pages.
- status_t FFR_GetCustomerInfieldData (flash_config_t *config, uint8_t *pData, uint32_t offset, uint32_t len)
  APIs to access CFPA pages.
- status_t FFR_CustFactoryPageWrite (flash_config_t *config, uint8_t *page_data, bool seal_part)
  APIs to access CMPA pages.
- status_t FFR_GetCustomerData (flash_config_t *config, uint8_t *pData, uint32_t offset, uint32_t len)
  APIs to access CMPA page.
- status_t FFR_GetUUID (flash_config_t *config, uint8_t *uuid)
  APIs to access CMPA page.
- status_t FFR_KeystoreWrite (flash_config_t *config, ffr_key_store_t *pKeyStore)
  This routine writes the 3 pages allocated for Key store data,.
- status_t FFR_KeystoreGetAC (flash_config_t *config, uint8_t *pActivationCode)
  Get/Read Key store code routines.
- status_t FFR_KeystoreGetKC (flash_config_t *config, uint8_t *pKeyCode, ffr_key_type_t keyIndex)
  Get/Read Key store code routines.

19.8.2 Macro Definition Documentation

19.8.2.1 #define FSL_FLASH_IFR_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

Version 2.1.0.

19.8.2.2 #define ALIGN_DOWN(*x, a*) ((x) & (uint32_t)(-((int32_t)(a))))

19.8.2.3 #define ALIGN_UP(*x, a*) (-((int32_t)((uint32_t)(-((int32_t)(x))) & (uint32_t)(-((int32_t)(a))))))

19.8.3 Enumeration Type Documentation

19.8.3.1 enum _flash_ffr_page_offset

Enumerator

kFfrPageOffset_CFPA Customer In-Field programmed area.
kFfrPageOffset_CFPA_Scratch CFPA Scratch page.
kFfrPageOffset_CFPA_Cfg CFPA Configuration area (Ping page)

kFfrPageOffset_CFPA_CfgPong Same as CFPA page (Pong page)
kFfrPageOffset_CMPA Customer Manufacturing programmed area.
kFfrPageOffset_CMPA_Cfg CMPA Configuration area (Part of CMPA)
kFfrPageOffset_CMPA_Key Key Store area (Part of CMPA)
kFfrPageOffset_NMPA NXP Manufacturing programmed area.
kFfrPageOffset_NMPA_Romcp ROM patch area (Part of NMPA)
kFfrPageOffset_NMPA_Repair Repair area (Part of NMPA)
kFfrPageOffset_NMPA_Cfg NMPA configuration area (Part of NMPA)
kFfrPageOffset_NMPA_End Reserved (Part of NMPA)

19.8.3.2 enum _flash_ffr_page_num

Enumerator

kFfrPageNum_CFPA Customer In-Field programmed area.
kFfrPageNum_CMPA Customer Manufacturing programmed area.
kFfrPageNum_NMPA NXP Manufacturing programmed area.

19.8.4 Function Documentation

19.8.4.1 status_t FFR_Init (flash_config_t * *config*)

Parameters

config	A pointer to the storage for the driver runtime state.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

19.8.4.2 status_t FFR_Lock_All (flash_config_t * *config*)

CFPA, CMPA, and NMPA flash areas region will be locked, After this function executed; Unless the board is reset again.

Parameters

config	A pointer to the storage for the driver runtime state.

Return values

kStatus_FLASH_Success	An invalid argument is provided.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

19.8.4.3 status_t FFR_InfieldPageWrite (flash_config_t * *config*, uint8_t * *page_data*, uint32_t *valid_len*)

This routine will erase CFPA and program the CFPA page with passed data.

Parameters

config	A pointer to the storage for the driver runtime state.
page_data	A pointer to the source buffer of data that is to be programmed into the CFPA.
valid_len	The length, given in bytes, to be programmed.

Return values

kStatus_FLASH_Success	The desire page-data were programed successfully into CFPA.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FLASH_Ffr-BankIsLocked	The CFPA was locked.
kStatus_FLASH_OutOf-DateCfpaPage	It is not newest CFPA page.

19.8.4.4 status_t FFR_GetCustomerInfieldData (flash_config_t * *config*, uint8_t * *pData*, uint32_t *offset*, uint32_t *len*)

Generic read function, used by customer to read data stored in 'Customer In-field Page'.

Parameters

config	A pointer to the storage for the driver runtime state.
pData	A pointer to the dest buffer of data that is to be read from 'Customer In-field Page'.
offset	An offset from the 'Customer In-field Page' start address.
len	The length, given in bytes, to be read.

Return values

kStatus_FLASH_Success	Get data from 'Customer In-field Page'.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FLASH_-CommandFailure	access error.

19.8.4.5 status_t FFR_CustFactoryPageWrite (flash_config_t * *config*, uint8_t * *page_data*, bool *seal_part*)

This routine will erase "customer factory page" and program the page with passed data. If 'seal_part' parameter is TRUE then the routine will compute SHA256 hash of the page contents and then programs the pages. 1.During development customer code uses this API with 'seal_part' set to FALSE. 2.During manufacturing this parameter should be set to TRUE to seal the part from further modifications 3.This routine checks if the page is sealed or not. A page is said to be sealed if the SHA256 value in the page has non-zero value. On boot ROM locks the firewall for the region if hash is programmed anyways. So, write/erase commands will fail eventually.

Parameters

config	A pointer to the storage for the driver runtime state.
page_data	A pointer to the source buffer of data that is to be programmed into the "customer factory page".
seal_part	Set fasle for During development customer code.

Return values

kStatus_FLASH_Success	The desire page-data were programed successfully into CMPA.
kStatus_FLASH_Invalid-Argument	Parameter is not aligned with the specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FLASH_-CommandFailure	access error.

19.8.4.6 status_t FFR_GetCustomerData (flash_config_t * *config*, uint8_t * *pData*, uint32_t *offset*, uint32_t *len*)

Read data stored in 'Customer Factory CFG Page'.

Parameters

config	A pointer to the storage for the driver runtime state.
pData	A pointer to the dest buffer of data that is to be read from the Customer Factory CFG Page.
offset	Address offset relative to the CMPA area.
len	The length, given in bytes to be read.

Return values

kStatus_FLASH_Success	Get data from 'Customer Factory CFG Page'.
kStatus_FLASH_Invalid-Argument	Parameter is not aligned with the specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FLASH_-CommandFailure	access error.

19.8.4.7 status_t FFR_GetUUID (flash_config_t * *config*, uint8_t * *uuid*)

1.SW should use this API routine to get the UUID of the chip. 2.Calling routine should pass a pointer to buffer which can hold 128-bit value.

19.8.4.8 status_t FFR_KeystoreWrite (flash_config_t * *config*, ffr_key_store_t * *pKeyStore*)

1.Used during manufacturing. Should write pages when 'customer factory page' is not in sealed state. 2.Optional routines to set individual data members (activation code, key codes etc) to construct the key store structure in RAM before committing it to IFR/FFR.

Parameters

config	A pointer to the storage for the driver runtime state.
pKeyStore	A Pointer to the 3 pages allocated for Key store data. that will be written to 'customer factory page'.

Return values

kStatus_FLASH_Success	The key were programed successfully into FFR.
kStatus_FLASH_Invalid-Argument	Parameter is not aligned with the specified baseline.
kStatus_FTFx_Address-Error	Address is out of range.
kStatus_FLASH_-CommandFailure	access error.

19.8.4.9 status_t FFR_KeystoreGetAC (flash_config_t * *config*, uint8_t * *pActivationCode*)

1. Calling code should pass buffer pointer which can hold activation code 1192 bytes.
2. Check if flash aperture is small or regular and read the data appropriately.

19.8.4.10 status_t FFR_KeystoreGetKC (flash_config_t * *config*, uint8_t * *pKeyCode*, ffr_key_type_t *keyIndex*)

1. Calling code should pass buffer pointer which can hold key code 52 bytes.
2. Check if flash aperture is small or regular and read the data appropriately.
3. keyIndex specifies which key code is read.

19.9 IAP_KBP Driver

19.9.1 Overview

Data Structures

- struct kb_region_t
  Memory region definition. More...
- struct kb_load_sb_t
  User-provided options passed into kb_init(). More...
- struct memory_region_interface_t
  Interface to memory operations for one region of memory. More...
- struct memory_map_entry_t
  Structure of a memory map entry. More...

Macros

- #define kStatusGroup_RomApi (108U)
  ROM API status group number.

Enumerations

- enum {
  kStatus_RomApiExecuteCompleted = kStatus_Success,
  kStatus_RomApiNeedMoreData,
  kStatus_RomApiBufferSizeNotEnough,
  kStatus_RomApiInvalidBuffer }
  ROM API status codes.
- enum kb_operation_t {
  kRomAuthenticateImage = 1,
  kRomLoadImage = 2 }
  Details of the operation to be performed by the ROM.
- enum _kb_security_profile
  Security constraint flags, Security profile flags.

Functions

- status_t kb_init (kb_session_ref_t **session, const kb_options_t *options)
  Initialize ROM API for a given operation.
- status_t kb_deinit (kb_session_ref_t *session)
  Cleans up the ROM API context.
- status_t kb_execute (kb_session_ref_t *session, const uint8_t *data, uint32_t dataLength)
  Perform the operation configured during init.

19.9.2 Data Structure Documentation

19.9.2.1 struct kb_region_t

19.9.2.2 struct kb_load_sb_t

The buffer field is a pointer to memory provided by the caller for use by Kboot during execution of the operation. Minimum size is the size of each certificate in the chain plus 432 bytes additional per certificate.

The profile field is a mask that specifies which features are required in the SB file or image being processed. This includes the minimum AES and RSA key sizes. See the _kb_security_profile enum for profile mask constants. The image being loaded or authenticated must match the profile or an error will be returned.

minBuildNumber is an optional field that can be used to prevent version rollback. The API will check the build number of the image, and if it is less than minBuildNumber will fail with an error.

maxImageLength is used to verify the offsetToCertificateBlockHeaderInBytes value at the beginning of a signed image. It should be set to the length of the SB file. If verifying an image in flash, it can be set to the internal flash size or a large number like 0x10000000.

userRHK can optionally be used by the user to override the RHK in IFR. If userRHK is not NULL, it points to a 32-byte array containing the SHA-256 of the root certificate's RSA public key.

The regions field points to an array of memory regions that the SB file being loaded is allowed to access. If regions is NULL, then all memory is accessible by the SB file. This feature is required to prevent a malicious image from erasing good code or RAM contents while it is being loaded, only for us to find that the image is inauthentic when we hit the end of the section.

overrideSBBootSectionID lets the caller override the default section of the SB file that is processed during a kKbootLoadSB operation. By default, the section specified in the firstBootableSectionID field of the SB header is loaded. If overrideSBBootSectionID is non-zero, then the section with the given ID will be loaded instead.

The userSBKEK field lets a user provide their own AES-256 key for unwrapping keys in an SB file during the kKbootLoadSB operation. userSBKEK should point to a 32-byte AES-256 key. If userSBKEK is NULL then the IFR SBKEK will be used. After kb_init() returns, the caller should zero out the data pointed to by userSBKEK, as the API will have installed the key in the CAU3.

19.9.2.3 struct memory_region_interface_t

19.9.2.4 struct memory_map_entry_t

19.9.3 Enumeration Type Documentation

19.9.3.1 anonymous enum

Enumerator

kStatus_RomApiExecuteCompleted ROM successfully process the whole sb file/boot image.
kStatus_RomApiNeedMoreData ROM needs more data to continue processing the boot image.
kStatus_RomApiBufferSizeNotEnough The user buffer is not enough for use by Kboot during execution of the operation.
kStatus_RomApiInvalidBuffer The user buffer is not ok for sbloader or authentication.

19.9.3.2 enum kb_operation_t

The kRomAuthenticateImage operation requires the entire signed image to be available to the application.

Enumerator

kRomAuthenticateImage Authenticate a signed image.
kRomLoadImage Load SB file.

19.9.4 Function Documentation

19.9.4.1 status_t kb_init (kb_session_ref_t ** *session*, const kb_options_t * *options*)

Inits the ROM API based on the options provided by the application in the second argument. Every call to rom_init() should be paired with a call to rom_deinit().

Return values

kStatus_Success	API was executed successfully.
kStatus_InvalidArgument	An invalid argument is provided.
kStatus_RomApiBuffer-SizeNotEnough	The user buffer is not enough for use by Kboot during execution of the operation.

kStatus_RomApiInvalidBuffer	The user buffer is not ok for sbloader or authentication.
kStatus_SKBOOT_Fail	Return the failed status of secure boot.
kStatus_SKBOOT_KeyStoreMarkerInvalid	The key code for the particular PRINCE region is not present in the keystore
kStatus_SKBOOT_Success	Return the successful status of secure boot.

19.9.4.2 status_t kb_deinit (kb_session_ref_t * *session*)

After this call, the context parameter can be reused for another operation by calling rom_init() again.

Return values

kStatus_Success	API was executed successfully

19.9.4.3 status_t kb_execute (kb_session_ref_t * *session*, const uint8_t * *data*, uint32_t *dataLength*)

This application must call this API repeatedly, passing in sequential chunks of data from the boot image (SB file) that is to be processed. The ROM will perform the selected operation on this data and return. The application may call this function with as much or as little data as it wishes, which can be used to select the granularity of time given to the application in between executing the operation.

Parameters

session	Current ROM context pointer.
data	Buffer of boot image data provided to the ROM by the application.
dataLength	Length in bytes of the data in the buffer provided to the ROM.

Return values

kStatus_Success	ROM successfully process the part of sb file/boot image.
kStatus_RomApiExecuteCompleted	ROM successfully process the whole sb file/boot image.

kStatus_Fail	An error occurred while executing the operation.
kStatus_RomApiNeed-MoreData	No error occurred, but the ROM needs more data to continue processing the boot image.
kStatus_RomApiBuffer-SizeNotEnough	user buffer is not enough for use by Kboot during execution of the operation.

Chapter 20
INPUTMUX: Input Multiplexing Driver

20.1 Overview

The MCUXpresso SDK provides a driver for the Input multiplexing (INPUTMUX).

It configures the inputs to the pin interrupt block, DMA trigger, and frequency measure function. Once configured, the clock is not needed for the inputmux.

20.2 Input Multiplexing Driver operation

INPUTMUX_AttachSignal function configures the specified input

20.3 Typical use case

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

Files

- file fsl_inputmux.h
- file fsl_inputmux_connections.h

Functions

- void INPUTMUX_Init (INPUTMUX_Type ∗base)
  Initialize INPUTMUX peripheral.
- void INPUTMUX_AttachSignal (INPUTMUX_Type ∗base, uint32_t index, inputmux_connection_t connection)
  Attaches a signal.
- void INPUTMUX_Deinit (INPUTMUX_Type ∗base)
  Deinitialize INPUTMUX peripheral.

Input multiplexing connections

- enum inputmux_connection_t {
  kINPUTMUX_SctGpi0ToSct0 = 0U + (SCT0_INMUX0 << PMUX_SHIFT) ,
  kINPUTMUX_DebugHaltedToSct0 = 23U + (SCT0_INMUX0 << PMUX_SHIFT) ,
  kINPUTMUX_I2sSharedWs1ToTimer0Captsel = 24U + (TIMER0CAPTSEL0 << PMUX_SHIFT) ,
  kINPUTMUX_I2sSharedWs1ToTimer1Captsel = 24U + (TIMER1CAPTSEL0 << PMUX_SHIFT) ,
  kINPUTMUX_I2sSharedWs1ToTimer2Captsel = 24U + (TIMER2CAPTSEL0 << PMUX_SHIF-

T) ,
kINPUTMUX_GpioPort1Pin31ToPintsel = 63U + (PINTSEL0 << PMUX_SHIFT) ,
kINPUTMUX_HashDmaRxToDma0 = 21U + (DMA0_ITRIG_INMUX0 << PMUX_SHIFT) ,
kINPUTMUX_Dma0Adc0Ch1TrigoutToTriginChannels = 22U + (DMA0_OTRIG_INMUX0 << PMUX_SHIFT) ,
kINPUTMUX_FreqmeGpioClk_bRef = 7u + (FREQMEAS_REF_REG << PMUX_SHIFT) ,
kINPUTMUX_FreqmeGpioClk_bTarget = 7u + (FREQMEAS_TARGET_REG << PMUX_SHIFT) ,
kINPUTMUX_I2sSharedWs1ToTimer3Captsel = 24U + (TIMER3CAPTSEL0 << PMUX_SHIFT) ,
kINPUTMUX_GpioPort0Pin31ToPintSecsel = 31U + (PINTSECSEL0 << PMUX_SHIFT) ,
kINPUTMUX_HashDmaRxToDma1 = 14U + (DMA1_ITRIG_INMUX0 << PMUX_SHIFT) }
INPUTMUX connections type.

- enum inputmux_signal_t {
kINPUTMUX_HashCryptToDmac0Ch0RequestEna = 0U + (DMA0_REQ_ENA_ID << ENA_SHIFT) ,
kINPUTMUX_Adc0FIFO1ToDmac0Ch22RequestEna = 22U + (DMA0_REQ_ENA_ID << ENA_SHIFT) ,
kINPUTMUX_Flexcomm3TxToDmac1Ch9RequestEna = 9U + (DMA1_REQ_ENA_ID << ENA_SHIFT) ,
kINPUTMUX_Dmac0InputTriggerHashOutEna = 21U + (DMA0_ITRIG_ENA_ID << ENA_SHIFT) }
INPUTMUX signal enable/disable type.
- #define SCT0_INMUX0 0x00U
Periphinmux IDs.
- #define **TIMER0CAPTSEL0** 0x20U
- #define **TIMER1CAPTSEL0** 0x40U
- #define **TIMER2CAPTSEL0** 0x60U
- #define **PINTSEL_PMUX_ID** 0xC0U
- #define **PINTSEL0** 0xC0U
- #define **DMA0_ITRIG_INMUX0** 0xE0U
- #define **DMA0_OTRIG_INMUX0** 0x160U
- #define **FREQMEAS_REF_REG** 0x180U
- #define **FREQMEAS_TARGET_REG** 0x184U
- #define **TIMER3CAPTSEL0** 0x1A0U
- #define **TIMER4CAPTSEL0** 0x1C0U
- #define **PINTSECSEL0** 0x1E0U
- #define **DMA1_ITRIG_INMUX0** 0x200U
- #define **DMA1_OTRIG_INMUX0** 0x240U
- #define **DMA0_REQ_ENA_ID** 0x740U
- #define **DMA1_REQ_ENA_ID** 0x760U
- #define **DMA0_ITRIG_ENA_ID** 0x780U
- #define **DMA1_ITRIG_ENA_ID** 0x7A0U
- #define **ENA_SHIFT** 8U
- #define **PMUX_SHIFT** 20U

Driver version

- #define FSL_INPUTMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
Group interrupt driver version for SDK.

20.4 Enumeration Type Documentation

20.4.1 enum inputmux_connection_t

Enumerator

kINPUTMUX_SctGpi0ToSct0 SCT0 INMUX.
kINPUTMUX_DebugHaltedToSct0 TIMER0 CAPTSEL.
kINPUTMUX_I2sSharedWs1ToTimer0Captsel TIMER1 CAPTSEL.
kINPUTMUX_I2sSharedWs1ToTimer1Captsel TIMER2 CAPTSEL.
kINPUTMUX_I2sSharedWs1ToTimer2Captsel Pin interrupt select.
kINPUTMUX_GpioPort1Pin31ToPintsel DMA0 Input trigger.
kINPUTMUX_HashDmaRxToDma0 DMA0 output trigger.
kINPUTMUX_Dma0Adc0Ch1TrigoutToTriginChannels Selection for frequency measurement reference clock.
kINPUTMUX_FreqmeGpioClk_bRef Selection for frequency measurement target clock.
kINPUTMUX_FreqmeGpioClk_bTarget TIMER3 CAPTSEL.
kINPUTMUX_I2sSharedWs1ToTimer3Captsel Timer4 CAPTSEL.
kINPUTMUX_GpioPort0Pin31ToPintSecsel DMA1 Input trigger.
kINPUTMUX_HashDmaRxToDma1 DMA1 output trigger.

20.4.2 enum inputmux_signal_t

Enumerator

kINPUTMUX_HashCryptToDmac0Ch0RequestEna DMA0 REQ signal.
kINPUTMUX_Adc0FIFO1ToDmac0Ch22RequestEna DMA1 REQ signal.
kINPUTMUX_Flexcomm3TxToDmac1Ch9RequestEna DMA0 input trigger source enable.
kINPUTMUX_Dmac0InputTriggerHashOutEna DMA1 input trigger source enable.

20.5 Function Documentation

20.5.1 void INPUTMUX_Init (INPUTMUX_Type * *base*)

This function enables the INPUTMUX clock.

Parameters

base	Base address of the INPUTMUX peripheral.

Return values

None.	

20.5.2 void INPUTMUX_AttachSignal (INPUTMUX_Type * *base*, uint32_t *index*, inputmux_connection_t *connection*)

This function gates the INPUTPMUX clock.

Parameters

base	Base address of the INPUTMUX peripheral.
index	Destination peripheral to attach the signal to.
connection	Selects connection.

Return values

None.	

20.5.3 void INPUTMUX_Deinit (INPUTMUX_Type * *base*)

This function disables the INPUTMUX clock.

Parameters

base	Base address of the INPUTMUX peripheral.

Return values

None.	

Chapter 21
LPADC: 12-bit SAR Analog-to-Digital Converter Driver

21.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit SAR Analog-to-Digital Converter (LPADC) module of MCUXpresso SDK devices.

21.2 Typical use case

21.2.1 Polling Configuration

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

21.2.2 Interrupt Configuration

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

Files

- file fsl_lpadc.h

Data Structures

- struct lpadc_config_t
  LPADC global configuration. *More...*
- struct lpadc_conv_command_config_t
  Define structure to keep the configuration for conversion command. *More...*
- struct lpadc_conv_trigger_config_t
  Define structure to keep the configuration for conversion trigger. *More...*
- struct lpadc_conv_result_t
  Define the structure to keep the conversion result. *More...*

Macros

- #define LPADC_GET_ACTIVE_COMMAND_STATUS(statusVal) ((statusVal & ADC_STAT_CMDACT_MASK) >> ADC_STAT_CMDACT_SHIFT)
  Define the MACRO function to get command status from status value.
- #define LPADC_GET_ACTIVE_TRIGGER_STATUE(statusVal) ((statusVal & ADC_STAT_TRGACT_MASK) >> ADC_STAT_TRGACT_SHIFT)
  Define the MACRO function to get trigger status from status value.

Enumerations

- enum _lpadc_status_flags {
  kLPADC_ResultFIFO0OverflowFlag = ADC_STAT_FOF0_MASK,
  kLPADC_ResultFIFO0ReadyFlag = ADC_STAT_RDY0_MASK,
  kLPADC_ResultFIFO1OverflowFlag = ADC_STAT_FOF1_MASK,
  kLPADC_ResultFIFO1ReadyFlag = ADC_STAT_RDY1_MASK }
  Define hardware flags of the module.
- enum _lpadc_interrupt_enable {
  kLPADC_ResultFIFO0OverflowInterruptEnable = ADC_IE_FOFIE0_MASK,
  kLPADC_FIFO0WatermarkInterruptEnable = ADC_IE_FWMIE0_MASK,
  kLPADC_ResultFIFO1OverflowInterruptEnable = ADC_IE_FOFIE1_MASK,
  kLPADC_FIFO1WatermarkInterruptEnable = ADC_IE_FWMIE1_MASK,
  kLPADC_TriggerExceptionInterruptEnable = ADC_IE_TEXC_IE_MASK,
  kLPADC_Trigger0CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 0UL),
  kLPADC_Trigger1CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 1UL),
  kLPADC_Trigger2CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 2UL),
  kLPADC_Trigger3CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 3UL),
  kLPADC_Trigger4CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 4UL),
  kLPADC_Trigger5CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 5UL),
  kLPADC_Trigger6CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 6UL),
  kLPADC_Trigger7CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 7UL),
  kLPADC_Trigger8CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 8UL),
  kLPADC_Trigger9CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 9UL),
  kLPADC_Trigger10CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 10UL),
  kLPADC_Trigger11CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 11UL),
  kLPADC_Trigger12CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 12UL),
  kLPADC_Trigger13CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 13UL),
  kLPADC_Trigger14CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 14UL),
  kLPADC_Trigger15CompletionInterruptEnable = ADC_IE_TCOMP_IE(1UL << 15UL) }
  Define interrupt switchers of the module.
- enum _lpadc_trigger_status_flags {

kLPADC_Trigger0InterruptedFlag = 1UL << 0UL,
kLPADC_Trigger1InterruptedFlag = 1UL << 1UL,
kLPADC_Trigger2InterruptedFlag = 1UL << 2UL,
kLPADC_Trigger3InterruptedFlag = 1UL << 3UL,
kLPADC_Trigger4InterruptedFlag = 1UL << 4UL,
kLPADC_Trigger5InterruptedFlag = 1UL << 5UL,
kLPADC_Trigger6InterruptedFlag = 1UL << 6UL,
kLPADC_Trigger7InterruptedFlag = 1UL << 7UL,
kLPADC_Trigger8InterruptedFlag = 1UL << 8UL,
kLPADC_Trigger9InterruptedFlag = 1UL << 9UL,
kLPADC_Trigger10InterruptedFlag = 1UL << 10UL,
kLPADC_Trigger11InterruptedFlag = 1UL << 11UL,
kLPADC_Trigger12InterruptedFlag = 1UL << 12UL,
kLPADC_Trigger13InterruptedFlag = 1UL << 13UL,
kLPADC_Trigger14InterruptedFlag = 1UL << 14UL,
kLPADC_Trigger15InterruptedFlag = 1UL << 15UL,
kLPADC_Trigger0CompletedFlag = 1UL << 16UL,
kLPADC_Trigger1CompletedFlag = 1UL << 17UL,
kLPADC_Trigger2CompletedFlag = 1UL << 18UL,
kLPADC_Trigger3CompletedFlag = 1UL << 19UL,
kLPADC_Trigger4CompletedFlag = 1UL << 20UL,
kLPADC_Trigger5CompletedFlag = 1UL << 21UL,
kLPADC_Trigger6CompletedFlag = 1UL << 22UL,
kLPADC_Trigger7CompletedFlag = 1UL << 23UL,
kLPADC_Trigger8CompletedFlag = 1UL << 24UL,
kLPADC_Trigger9CompletedFlag = 1UL << 25UL,
kLPADC_Trigger10CompletedFlag = 1UL << 26UL,
kLPADC_Trigger11CompletedFlag = 1UL << 27UL,
kLPADC_Trigger12CompletedFlag = 1UL << 28UL,
kLPADC_Trigger13CompletedFlag = 1UL << 29UL,
kLPADC_Trigger14CompletedFlag = 1UL << 30UL,
kLPADC_Trigger15CompletedFlag = 1UL << 31UL }

The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.

- enum lpadc_sample_scale_mode_t {
  kLPADC_SamplePartScale = 0U,
  kLPADC_SampleFullScale = 1U }

  Define enumeration of sample scale mode.
- enum lpadc_sample_channel_mode_t {
  kLPADC_SampleChannelSingleEndSideA = 0U,
  kLPADC_SampleChannelSingleEndSideB = 1U,
  kLPADC_SampleChannelDiffBothSide = 2U,
  kLPADC_SampleChannelDualSingleEndBothSide }

  Define enumeration of channel sample mode.
- enum lpadc_hardware_average_mode_t {

kLPADC_HardwareAverageCount1 = 0U,
kLPADC_HardwareAverageCount2 = 1U,
kLPADC_HardwareAverageCount4 = 2U,
kLPADC_HardwareAverageCount8 = 3U,
kLPADC_HardwareAverageCount16 = 4U,
kLPADC_HardwareAverageCount32 = 5U,
kLPADC_HardwareAverageCount64 = 6U,
kLPADC_HardwareAverageCount128 = 7U }
Define enumeration of hardware average selection.

- enum lpadc_sample_time_mode_t {
  kLPADC_SampleTimeADCK3 = 0U,
  kLPADC_SampleTimeADCK5 = 1U,
  kLPADC_SampleTimeADCK7 = 2U,
  kLPADC_SampleTimeADCK11 = 3U,
  kLPADC_SampleTimeADCK19 = 4U,
  kLPADC_SampleTimeADCK35 = 5U,
  kLPADC_SampleTimeADCK67 = 6U,
  kLPADC_SampleTimeADCK131 = 7U }
  Define enumeration of sample time selection.
- enum lpadc_hardware_compare_mode_t {
  kLPADC_HardwareCompareDisabled = 0U,
  kLPADC_HardwareCompareStoreOnTrue = 2U,
  kLPADC_HardwareCompareRepeatUntilTrue = 3U }
  Define enumeration of hardware compare mode.
- enum lpadc_conversion_resolution_mode_t {
  kLPADC_ConversionResolutionStandard = 0U,
  kLPADC_ConversionResolutionHigh = 1U }
  Define enumeration of conversion resolution mode.
- enum lpadc_conversion_average_mode_t {
  kLPADC_ConversionAverage1 = 0U,
  kLPADC_ConversionAverage2 = 1U,
  kLPADC_ConversionAverage4 = 2U,
  kLPADC_ConversionAverage8 = 3U,
  kLPADC_ConversionAverage16 = 4U,
  kLPADC_ConversionAverage32 = 5U,
  kLPADC_ConversionAverage64 = 6U,
  kLPADC_ConversionAverage128 = 7U }
  Define enumeration of conversion averages mode.
- enum lpadc_reference_voltage_source_t {
  kLPADC_ReferenceVoltageAlt1 = 0U,
  kLPADC_ReferenceVoltageAlt2 = 1U,
  kLPADC_ReferenceVoltageAlt3 = 2U }
  Define enumeration of reference voltage source.
- enum lpadc_power_level_mode_t {

kLPADC_PowerLevelAlt1 = 0U,
kLPADC_PowerLevelAlt2 = 1U,
kLPADC_PowerLevelAlt3 = 2U,
kLPADC_PowerLevelAlt4 = 3U }
Define enumeration of power configuration.

- enum lpadc_trigger_priority_policy_t {
  kLPADC_TriggerPriorityPreemptImmediately = 0U,
  kLPADC_TriggerPriorityPreemptSoftly = 1U,
  kLPADC_TriggerPriorityPreemptSubsequently = 2U }
  Define enumeration of trigger priority policy.

Driver version

- #define FSL_LPADC_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))
  LPADC driver version 2.5.1.

Initialization & de-initialization.

- void LPADC_Init (ADC_Type *base, const lpadc_config_t *config)
  Initializes the LPADC module.
- void LPADC_GetDefaultConfig (lpadc_config_t *config)
  Gets an available pre-defined settings for initial configuration.
- void LPADC_Deinit (ADC_Type *base)
  De-initializes the LPADC module.
- static void LPADC_Enable (ADC_Type *base, bool enable)
  Switch on/off the LPADC module.
- static void LPADC_DoResetFIFO0 (ADC_Type *base)
  Do reset the conversion FIFO0.
- static void LPADC_DoResetFIFO1 (ADC_Type *base)
  Do reset the conversion FIFO1.
- static void LPADC_DoResetConfig (ADC_Type *base)
  Do reset the module's configuration.

Status

- static uint32_t LPADC_GetStatusFlags (ADC_Type *base)
  Get status flags.
- static void LPADC_ClearStatusFlags (ADC_Type *base, uint32_t mask)
  Clear status flags.
- static uint32_t LPADC_GetTriggerStatusFlags (ADC_Type *base)
  Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.
- static void LPADC_ClearTriggerStatusFlags (ADC_Type *base, uint32_t mask)
  Clear trigger status flags.

Interrupts

- static void LPADC_EnableInterrupts (ADC_Type *base, uint32_t mask)
  Enable interrupts.

- static void LPADC_DisableInterrupts (ADC_Type *base, uint32_t mask)
  Disable interrupts.

DMA Control

- static void LPADC_EnableFIFO0WatermarkDMA (ADC_Type *base, bool enable)
  Switch on/off the DMA trigger for FIFO0 watermark event.
- static void LPADC_EnableFIFO1WatermarkDMA (ADC_Type *base, bool enable)
  Switch on/off the DMA trigger for FIFO1 watermark event.

Trigger and conversion with FIFO.

- static uint32_t LPADC_GetConvResultCount (ADC_Type *base, uint8_t index)
  Get the count of result kept in conversion FIFOn.
- bool LPADC_GetConvResult (ADC_Type *base, lpadc_conv_result_t *result, uint8_t index)
  brief Get the result in conversion FIFOn.
- void LPADC_SetConvTriggerConfig (ADC_Type *base, uint32_t triggerId, const lpadc_conv_trigger_config_t *config)
  Configure the conversion trigger source.
- void LPADC_GetDefaultConvTriggerConfig (lpadc_conv_trigger_config_t *config)
  Gets an available pre-defined settings for trigger's configuration.
- static void LPADC_DoSoftwareTrigger (ADC_Type *base, uint32_t triggerIdMask)
  Do software trigger to conversion command.
- void LPADC_SetConvCommandConfig (ADC_Type *base, uint32_t commandId, const lpadc_conv_command_config_t *config)
  Configure conversion command.
- void LPADC_GetDefaultConvCommandConfig (lpadc_conv_command_config_t *config)
  Gets an available pre-defined settings for conversion command's configuration.
- static void LPADC_SetOffsetValue (ADC_Type *base, uint32_t valueA, uint32_t valueB)
  Set proper offset value to trim ADC.
- static void LPADC_EnableOffsetCalibration (ADC_Type *base, bool enable)
  Enable the offset calibration function.
- void LPADC_DoOffsetCalibration (ADC_Type *base)
  Do offset calibration.
- void LPADC_DoAutoCalibration (ADC_Type *base)
  brief Do auto calibration.

21.3 Data Structure Documentation

21.3.1 struct lpadc_config_t

This structure would used to keep the settings for initialization.

Data Fields

- bool enableInDozeMode
  Control system transition to Stop and Wait power modes while ADC is converting.
- lpadc_conversion_average_mode_t conversionAverageMode

Auto-Calibration Averages.

- bool enableAnalogPreliminary
  ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).
- uint32_t powerUpDelay
  When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize.
- lpadc_reference_voltage_source_t referenceVoltageSource
  Selects the voltage reference high used for conversions.
- lpadc_power_level_mode_t powerLevelMode
  Power Configuration Selection.
- lpadc_trigger_priority_policy_t triggerPriorityPolicy
  Control how higher priority triggers are handled, see to lpadc_trigger_priority_policy_t.
- bool enableConvPause
  Enables the ADC pausing function.
- uint32_t convPauseDelay
  Controls the duration of pausing during command execution sequencing.
- uint32_t FIFO0Watermark
  FIFO0Watermark is a programmable threshold setting.
- uint32_t FIFO1Watermark
  FIFO1Watermark is a programmable threshold setting.

Field Documentation

(1) bool lpadc_config_t::enableInDozeMode

When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

(2) lpadc_conversion_average_mode_t lpadc_config_t::conversionAverageMode

(3) bool lpadc_config_t::enableAnalogPreliminary

(4) uint32_t lpadc_config_t::powerUpDelay

The startup delay count of (powerUpDelay $*$ 4) ADCK cycles must result in a longer delay than the analog startup time.

(5) lpadc_reference_voltage_source_t lpadc_config_t::referenceVoltageSource

(6) lpadc_power_level_mode_t lpadc_config_t::powerLevelMode

(7) lpadc_trigger_priority_policy_t lpadc_config_t::triggerPriorityPolicy

(8) bool lpadc_config_t::enableConvPause

When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in

"Compare Until True" configuration.

(9) uint32_t lpadc_config_t::convPauseDelay

The pause delay is a count of (convPauseDelay∗4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

(10) uint32_t lpadc_config_t::FIFO0Watermark

When the number of datawords stored in the ADC Result FIFO0 is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

(11) uint32_t lpadc_config_t::FIFO1Watermark

When the number of datawords stored in the ADC Result FIFO1 is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

21.3.2 struct lpadc_conv_command_config_t

Data Fields

- lpadc_sample_channel_mode_t sampleChannelMode
  Channel sample mode.
- uint32_t channelNumber
  Channel number, select the channel or channel pair.
- uint32_t chainedNextCommandNumber
  Selects the next command to be executed after this command completes.
- bool enableAutoChannelIncrement
  Loop with increment: when disabled, the "loopCount" field selects the number of times the selected channel is converted consecutively; when enabled, the "loopCount" field defines how many consecutive channels are converted as part of the command execution.
- uint32_t loopCount
  Selects how many times this command executes before finish and transition to the next command or Idle state.
- lpadc_hardware_average_mode_t hardwareAverageMode
  Hardware average selection.
- lpadc_sample_time_mode_t sampleTimeMode
  Sample time selection.
- lpadc_hardware_compare_mode_t hardwareCompareMode
  Hardware compare selection.
- uint32_t hardwareCompareValueHigh
  Compare Value High.
- uint32_t hardwareCompareValueLow
  Compare Value Low.
- lpadc_conversion_resolution_mode_t conversionResolutionMode
  Conversion resolution mode.
- bool enableWaitTrigger
  Wait for trigger assertion before execution: when disabled, this command will be automatically executed;

when enabled, the active trigger must be asserted again before executing this command.

Field Documentation

(1) lpadc_sample_channel_mode_t lpadc_conv_command_config_t::sampleChannelMode

(2) uint32_t lpadc_conv_command_config_t::channelNumber

(3) uint32_t lpadc_conv_command_config_t::chainedNextCommandNumber

1-15 is available, 0 is to terminate the chain after this command.

(4) bool lpadc_conv_command_config_t::enableAutoChannelIncrement

(5) uint32_t lpadc_conv_command_config_t::loopCount

Command executes LOOP+1 times. 0-15 is available.

(6) lpadc_hardware_average_mode_t lpadc_conv_command_config_t::hardwareAverageMode

(7) lpadc_sample_time_mode_t lpadc_conv_command_config_t::sampleTimeMode

(8) lpadc_hardware_compare_mode_t lpadc_conv_command_config_t::hardwareCompareMode

(9) uint32_t lpadc_conv_command_config_t::hardwareCompareValueHigh

The available value range is in 16-bit.

(10) uint32_t lpadc_conv_command_config_t::hardwareCompareValueLow

The available value range is in 16-bit.

(11) lpadc_conversion_resolution_mode_t lpadc_conv_command_config_t::conversionResolutionMode

(12) bool lpadc_conv_command_config_t::enableWaitTrigger

21.3.3 struct lpadc_conv_trigger_config_t

Data Fields

- uint32_t targetCommandId
  Select the command from command buffer to execute upon detect of the associated trigger event.
- uint32_t delayPower
  Select the trigger delay duration to wait at the start of servicing a trigger event.
- uint32_t priority
  Sets the priority of the associated trigger source.
- bool enableHardwareTrigger
  Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not.

Field Documentation

(1) uint32_t lpadc_conv_trigger_config_t::targetCommandId

(2) uint32_t lpadc_conv_trigger_config_t::delayPower

When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is 2^delayPower ADCK cycles. The available value range is 4-bit.

(3) uint32_t lpadc_conv_trigger_config_t::priority

If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

(4) bool lpadc_conv_trigger_config_t::enableHardwareTrigger

THe software trigger is always available.

21.3.4 struct lpadc_conv_result_t

Data Fields

- uint32_t commandIdSource
  Indicate the command buffer being executed that generated this result.
- uint32_t loopCountIndex
  Indicate the loop count value during command execution that generated this result.
- uint32_t triggerIdSource
  Indicate the trigger source that initiated a conversion and generated this result.
- uint16_t convValue
  Data result.

Field Documentation

(1) uint32_t lpadc_conv_result_t::commandIdSource

(2) uint32_t lpadc_conv_result_t::loopCountIndex

(3) uint32_t lpadc_conv_result_t::triggerIdSource

(4) uint16_t lpadc_conv_result_t::convValue

21.4 Macro Definition Documentation

21.4.1 #define FSL_LPADC_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))

21.4.2 #define LPADC_GET_ACTIVE_COMMAND_STATUS(*statusVal*) ((statusVal & ADC_STAT_CMDACT_MASK) >> ADC_STAT_CMDACT_SHIFT)

The statusVal is the return value from LPADC_GetStatusFlags().

21.4.3 #define LPADC_GET_ACTIVE_TRIGGER_STATUE(*statusVal*) ((statusVal & ADC_STAT_TRGACT_MASK) >> ADC_STAT_TRGACT_SHIFT)

The statusVal is the return value from LPADC_GetStatusFlags().

21.5 Enumeration Type Documentation

21.5.1 enum _lpadc_status_flags

Enumerator

kLPADC_ResultFIFO0OverflowFlag Indicates that more data has been written to the Result FIFO 0 than it can hold.

kLPADC_ResultFIFO0ReadyFlag Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

kLPADC_ResultFIFO1OverflowFlag Indicates that more data has been written to the Result FIFO 1 than it can hold.

kLPADC_ResultFIFO1ReadyFlag Indicates when the number of valid datawords in the result FIFO 1 is greater than the setting watermark level.

21.5.2 enum _lpadc_interrupt_enable

Enumerator

kLPADC_ResultFIFO0OverflowInterruptEnable Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

kLPADC_FIFO0WatermarkInterruptEnable Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

kLPADC_ResultFIFO1OverflowInterruptEnable Configures ADC to generate overflow interrupt requests when FOF1 flag is asserted.

kLPADC_FIFO1WatermarkInterruptEnable Configures ADC to generate watermark interrupt requests when RDY1 flag is asserted.

kLPADC_TriggerExceptionInterruptEnable Configures ADC to generate trigger exception interrupt.

kLPADC_Trigger0CompletionInterruptEnable Configures ADC to generate interrupt when trigger 0 completion.

kLPADC_Trigger1CompletionInterruptEnable Configures ADC to generate interrupt when trigger 1 completion.

kLPADC_Trigger2CompletionInterruptEnable Configures ADC to generate interrupt when trigger 2 completion.
kLPADC_Trigger3CompletionInterruptEnable Configures ADC to generate interrupt when trigger 3 completion.
kLPADC_Trigger4CompletionInterruptEnable Configures ADC to generate interrupt when trigger 4 completion.
kLPADC_Trigger5CompletionInterruptEnable Configures ADC to generate interrupt when trigger 5 completion.
kLPADC_Trigger6CompletionInterruptEnable Configures ADC to generate interrupt when trigger 6 completion.
kLPADC_Trigger7CompletionInterruptEnable Configures ADC to generate interrupt when trigger 7 completion.
kLPADC_Trigger8CompletionInterruptEnable Configures ADC to generate interrupt when trigger 8 completion.
kLPADC_Trigger9CompletionInterruptEnable Configures ADC to generate interrupt when trigger 9 completion.
kLPADC_Trigger10CompletionInterruptEnable Configures ADC to generate interrupt when trigger 10 completion.
kLPADC_Trigger11CompletionInterruptEnable Configures ADC to generate interrupt when trigger 11 completion.
kLPADC_Trigger12CompletionInterruptEnable Configures ADC to generate interrupt when trigger 12 completion.
kLPADC_Trigger13CompletionInterruptEnable Configures ADC to generate interrupt when trigger 13 completion.
kLPADC_Trigger14CompletionInterruptEnable Configures ADC to generate interrupt when trigger 14 completion.
kLPADC_Trigger15CompletionInterruptEnable Configures ADC to generate interrupt when trigger 15 completion.

21.5.3 enum _lpadc_trigger_status_flags

Enumerator

kLPADC_Trigger0InterruptedFlag Trigger 0 is interrupted by a high priority exception.
kLPADC_Trigger1InterruptedFlag Trigger 1 is interrupted by a high priority exception.
kLPADC_Trigger2InterruptedFlag Trigger 2 is interrupted by a high priority exception.
kLPADC_Trigger3InterruptedFlag Trigger 3 is interrupted by a high priority exception.
kLPADC_Trigger4InterruptedFlag Trigger 4 is interrupted by a high priority exception.
kLPADC_Trigger5InterruptedFlag Trigger 5 is interrupted by a high priority exception.
kLPADC_Trigger6InterruptedFlag Trigger 6 is interrupted by a high priority exception.
kLPADC_Trigger7InterruptedFlag Trigger 7 is interrupted by a high priority exception.
kLPADC_Trigger8InterruptedFlag Trigger 8 is interrupted by a high priority exception.
kLPADC_Trigger9InterruptedFlag Trigger 9 is interrupted by a high priority exception.
kLPADC_Trigger10InterruptedFlag Trigger 10 is interrupted by a high priority exception.

kLPADC_Trigger11InterruptedFlag Trigger 11 is interrupted by a high priority exception.
kLPADC_Trigger12InterruptedFlag Trigger 12 is interrupted by a high priority exception.
kLPADC_Trigger13InterruptedFlag Trigger 13 is interrupted by a high priority exception.
kLPADC_Trigger14InterruptedFlag Trigger 14 is interrupted by a high priority exception.
kLPADC_Trigger15InterruptedFlag Trigger 15 is interrupted by a high priority exception.
kLPADC_Trigger0CompletedFlag Trigger 0 is completed and trigger 0 has enabled completion interrupts.
kLPADC_Trigger1CompletedFlag Trigger 1 is completed and trigger 1 has enabled completion interrupts.
kLPADC_Trigger2CompletedFlag Trigger 2 is completed and trigger 2 has enabled completion interrupts.
kLPADC_Trigger3CompletedFlag Trigger 3 is completed and trigger 3 has enabled completion interrupts.
kLPADC_Trigger4CompletedFlag Trigger 4 is completed and trigger 4 has enabled completion interrupts.
kLPADC_Trigger5CompletedFlag Trigger 5 is completed and trigger 5 has enabled completion interrupts.
kLPADC_Trigger6CompletedFlag Trigger 6 is completed and trigger 6 has enabled completion interrupts.
kLPADC_Trigger7CompletedFlag Trigger 7 is completed and trigger 7 has enabled completion interrupts.
kLPADC_Trigger8CompletedFlag Trigger 8 is completed and trigger 8 has enabled completion interrupts.
kLPADC_Trigger9CompletedFlag Trigger 9 is completed and trigger 9 has enabled completion interrupts.
kLPADC_Trigger10CompletedFlag Trigger 10 is completed and trigger 10 has enabled completion interrupts.
kLPADC_Trigger11CompletedFlag Trigger 11 is completed and trigger 11 has enabled completion interrupts.
kLPADC_Trigger12CompletedFlag Trigger 12 is completed and trigger 12 has enabled completion interrupts.
kLPADC_Trigger13CompletedFlag Trigger 13 is completed and trigger 13 has enabled completion interrupts.
kLPADC_Trigger14CompletedFlag Trigger 14 is completed and trigger 14 has enabled completion interrupts.
kLPADC_Trigger15CompletedFlag Trigger 15 is completed and trigger 15 has enabled completion interrupts.

21.5.4 enum lpadc_sample_scale_mode_t

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This

reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Enumerator

kLPADC_SamplePartScale Use divided input voltage signal. (Factor of 30/64).
kLPADC_SampleFullScale Full scale (Factor of 1).

21.5.5 enum lpadc_sample_channel_mode_t

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Enumerator

kLPADC_SampleChannelSingleEndSideA Single end mode, using side A.
kLPADC_SampleChannelSingleEndSideB Single end mode, using side B.
kLPADC_SampleChannelDiffBothSide Differential mode, using A and B.
kLPADC_SampleChannelDualSingleEndBothSide Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

21.5.6 enum lpadc_hardware_average_mode_t

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Enumerator

kLPADC_HardwareAverageCount1 Single conversion.
kLPADC_HardwareAverageCount2 2 conversions averaged.
kLPADC_HardwareAverageCount4 4 conversions averaged.
kLPADC_HardwareAverageCount8 8 conversions averaged.
kLPADC_HardwareAverageCount16 16 conversions averaged.
kLPADC_HardwareAverageCount32 32 conversions averaged.
kLPADC_HardwareAverageCount64 64 conversions averaged.
kLPADC_HardwareAverageCount128 128 conversions averaged.

21.5.7 enum lpadc_sample_time_mode_t

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Enumerator

kLPADC_SampleTimeADCK3 3 ADCK cycles total sample time.
kLPADC_SampleTimeADCK5 5 ADCK cycles total sample time.
kLPADC_SampleTimeADCK7 7 ADCK cycles total sample time.
kLPADC_SampleTimeADCK11 11 ADCK cycles total sample time.
kLPADC_SampleTimeADCK19 19 ADCK cycles total sample time.
kLPADC_SampleTimeADCK35 35 ADCK cycles total sample time.
kLPADC_SampleTimeADCK67 69 ADCK cycles total sample time.
kLPADC_SampleTimeADCK131 131 ADCK cycles total sample time.

21.5.8 enum lpadc_hardware_compare_mode_t

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Enumerator

kLPADC_HardwareCompareDisabled Compare disabled.
kLPADC_HardwareCompareStoreOnTrue Compare enabled. Store on true.
kLPADC_HardwareCompareRepeatUntilTrue Compare enabled. Repeat channel acquisition until true.

21.5.9 enum lpadc_conversion_resolution_mode_t

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to lpadc_sample_channel_mode_t

Enumerator

kLPADC_ConversionResolutionStandard Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.
kLPADC_ConversionResolutionHigh High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

21.5.10 enum lpadc_conversion_average_mode_t

Configure the converion average number for auto-calibration.

Enumerator

kLPADC_ConversionAverage1 Single conversion.

kLPADC_ConversionAverage2 2 conversions averaged.
kLPADC_ConversionAverage4 4 conversions averaged.
kLPADC_ConversionAverage8 8 conversions averaged.
kLPADC_ConversionAverage16 16 conversions averaged.
kLPADC_ConversionAverage32 32 conversions averaged.
kLPADC_ConversionAverage64 64 conversions averaged.
kLPADC_ConversionAverage128 128 conversions averaged.

21.5.11 enum lpadc_reference_voltage_source_t

For detail information, need to check the SoC's specification.

Enumerator

kLPADC_ReferenceVoltageAlt1 Option 1 setting.
kLPADC_ReferenceVoltageAlt2 Option 2 setting.
kLPADC_ReferenceVoltageAlt3 Option 3 setting.

21.5.12 enum lpadc_power_level_mode_t

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Enumerator

kLPADC_PowerLevelAlt1 Lowest power setting.
kLPADC_PowerLevelAlt2 Next lowest power setting.
kLPADC_PowerLevelAlt3 ...
kLPADC_PowerLevelAlt4 Highest power setting.

21.5.13 enum lpadc_trigger_priority_policy_t

This selection controls how higher priority triggers are handled.

Enumerator

kLPADC_TriggerPriorityPreemptImmediately If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started.

kLPADC_TriggerPriorityPreemptSoftly If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated.

kLPADC_TriggerPriorityPreemptSubsequently If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger.

21.6 Function Documentation

21.6.1 void LPADC_Init (ADC_Type * *base,* const lpadc_config_t * *config*)

Parameters

base	LPADC peripheral base address.
config	Pointer to configuration structure. See "lpadc_config_t".

21.6.2 void LPADC_GetDefaultConfig (lpadc_config_t * *config*)

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

```
*   config->enableInDozeMode        = true;
*   config->enableAnalogPreliminary = false;
*   config->powerUpDelay            = 0x80;
*   config->referenceVoltageSource  = kLPADC_ReferenceVoltageAlt1;
*   config->powerLevelMode          = kLPADC_PowerLevelAlt1;
*   config->triggerPriorityPolicy   = kLPADC_TriggerPriorityPreemptImmediately
      ;
*   config->enableConvPause         = false;
*   config->convPauseDelay          = 0U;
*   config->FIFOWatermark           = 0U;
*
```

Parameters

config	Pointer to configuration structure.

21.6.3 void LPADC_Deinit (ADC_Type * *base*)

Parameters

base	LPADC peripheral base address.

21.6.4 static void LPADC_Enable (ADC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	LPADC peripheral base address.
enable	switcher to the module.

21.6.5 static void LPADC_DoResetFIFO0 (ADC_Type * *base*) [inline], [static]

Parameters

base	LPADC peripheral base address.

21.6.6 static void LPADC_DoResetFIFO1 (ADC_Type * *base*) [inline], [static]

Parameters

base	LPADC peripheral base address.

21.6.7 static void LPADC_DoResetConfig (ADC_Type * *base*) [inline], [static]

Reset all ADC internal logic and registers, except the Control Register (ADCx_CTRL).

Parameters

base	LPADC peripheral base address.

21.6.8 static uint32_t LPADC_GetStatusFlags (ADC_Type * *base*) [inline], [static]

Parameters

base	LPADC peripheral base address.

Returns

status flags' mask. See to _lpadc_status_flags.

21.6.9 static void LPADC_ClearStatusFlags (ADC_Type * *base,* uint32_t *mask*) [inline], [static]

Only the flags can be cleared by writing ADCx_STATUS register would be cleared by this API.

Parameters

base	LPADC peripheral base address.
mask	Mask value for flags to be cleared. See to _lpadc_status_flags.

21.6.10 static uint32_t LPADC_GetTriggerStatusFlags (ADC_Type * *base*) [inline], [static]

Parameters

base	LPADC peripheral base address.

Returns

The OR'ed value of _lpadc_trigger_status_flags.

21.6.11 static void LPADC_ClearTriggerStatusFlags (ADC_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	LPADC peripheral base address.
mask	The mask of trigger status flags to be cleared, should be the OR'ed value of _lpadc_-trigger_status_flags.

21.6.12 static void LPADC_EnableInterrupts (ADC_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	LPADC peripheral base address.
mask	Mask value for interrupt events. See to _lpadc_interrupt_enable.

21.6.13 static void LPADC_DisableInterrupts (ADC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	LPADC peripheral base address.
mask	Mask value for interrupt events. See to _lpadc_interrupt_enable.

21.6.14 static void LPADC_EnableFIFO0WatermarkDMA (ADC_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	LPADC peripheral base address.
enable	Switcher to the event.

21.6.15 static void LPADC_EnableFIFO1WatermarkDMA (ADC_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	LPADC peripheral base address.
enable	Switcher to the event.

21.6.16 static uint32_t LPADC_GetConvResultCount (ADC_Type * *base*, uint8_t *index*) [inline], [static]

Parameters

base	LPADC peripheral base address.
index	Result FIFO index.

Returns

The count of result kept in conversion FIFOn.

21.6.17 bool LPADC_GetConvResult (ADC_Type * *base*, lpadc_conv_result_t * *result*, uint8_t *index*)

param base LPADC peripheral base address. param result Pointer to structure variable that keeps the conversion result in conversion FIFOn. param index Result FIFO index.

return Status whether FIFOn entry is valid.

21.6.18 void LPADC_SetConvTriggerConfig (ADC_Type * *base*, uint32_t *triggerId*, const lpadc_conv_trigger_config_t * *config*)

Each programmable trigger can launch the conversion command in command buffer.

Parameters

base	LPADC peripheral base address.
triggerId	ID for each trigger. Typically, the available value range is from 0.
config	Pointer to configuration structure. See to lpadc_conv_trigger_config_t.

21.6.19 void LPADC_GetDefaultConvTriggerConfig (lpadc_conv_trigger_config_t * *config*)

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

```
*   config->commandIdSource       = 0U;
*   config->loopCountIndex        = 0U;
*   config->triggerIdSource       = 0U;
*   config->enableHardwareTrigger = false;
*
```

Parameters

config	Pointer to configuration structure.

21.6.20 static void LPADC_DoSoftwareTrigger (ADC_Type * *base*, uint32_t *triggerIdMask*) [inline], [static]

Parameters

base	LPADC peripheral base address.
triggerIdMask	Mask value for software trigger indexes, which count from zero.

21.6.21 void LPADC_SetConvCommandConfig (ADC_Type * *base*, uint32_t *commandId*, const lpadc_conv_command_config_t * *config*)

Parameters

base	LPADC peripheral base address.
commandId	ID for command in command buffer. Typically, the available value range is 1 - 15.
config	Pointer to configuration structure. See to lpadc_conv_command_config_t.

21.6.22 void LPADC_GetDefaultConvCommandConfig (lpadc_conv_command_config_t * *config*)

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
*   config->sampleScaleMode            = kLPADC_SampleFullScale;
*   config->channelSampleMode          = kLPADC_SampleChannelSingleEndSideA
      ;
*   config->channelNumber              = 0U;
*   config->chainedNextCmdNumber       = 0U;
*   config->enableAutoChannelIncrement = false;
*   config->loopCount                  = 0U;
*   config->hardwareAverageMode        = kLPADC_HardwareAverageCount1;
*   config->sampleTimeMode             = kLPADC_SampleTimeADCK3;
*   config->hardwareCompareMode        = kLPADC_HardwareCompareDisabled;
*   config->hardwareCompareValueHigh   = 0U;
*   config->hardwareCompareValueLow    = 0U;
*   config->conversionResolutionMode  = kLPADC_ConversionResolutionStandard
      ;
*   config->enableWaitTrigger          = false;
*
```

Parameters

config	Pointer to configuration structure.

21.6.23 static void LPADC_SetOffsetValue (ADC_Type * *base,* uint32_t *valueA,* uint32_t *valueB*) [inline], [static]

Set the offset trim value for offset calibration manually.

Parameters

base	LPADC peripheral base address.
valueA	Setting offset value A.
valueB	Setting offset value B.

Note

In normal adc sequence, the values are automatically calculated by LPADC_EnableOffset-Calibration.

21.6.24 static void LPADC_EnableOffsetCalibration (ADC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	LPADC peripheral base address.
enable	switcher to the calibration function.

21.6.25 void LPADC_DoOffsetCalibration (ADC_Type * *base*)

Parameters

base	LPADC peripheral base address.

21.6.26 void LPADC_DoAutoCalibration (ADC_Type * *base*)

param base LPADC peripheral base address.

Chapter 22
CRC: Cyclic Redundancy Check Driver

22.1 Overview

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

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

22.2 CRC Driver Initialization and Configuration

CRC_Init() function enables the clock for the CRC module in the LPC SYSCON block and fully (re-)configures the CRC module according to configuration structure. It also starts checksum computation by writing the seed.

The seed member of the configuration structure is the initial checksum for which new data can be added to. When starting new checksum computation, the seed should be set to the initial checksum per the CRC protocol specification. For continued checksum operation, the seed should be set to the intermediate checksum value as obtained from previous calls to CRC_GetConfig() function. After CRC_Init(), one or multiple CRC_WriteData() calls follow to update checksum with data, then CRC_Get16bitResult() or CRC_Get32bitResult() follows to read the result. CRC_Init() can be called as many times as required, which allows for runtime changes of the CRC protocol.

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

CRC_Deinit() function disables clock to the CRC module.

CRC_Reset() performs hardware reset of the CRC module.

22.3 CRC Write Data

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

CRC_WriteSeed() Write seed (initial checksum) to CRC module.

22.4 CRC Get Checksum

The CRC_Get16bitResult() or CRC_Get32bitResult() function is used to read the CRC module checksum register. The bit reverse and 1's complement operations are already applied to the result if previously

configured. Use CRC_GetConfig() function to get the actual checksum without bit reverse and 1's complement applied so it can be used as seed when resuming calculation later.

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

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

CRC_Init() / CRC_WriteData() / CRC_GetConfig() to get intermediate checksum to be used as seed value in future.

CRC_Init() / CRC_WriteData() / ... / CRC_WriteData() / CRC_GetConfig() to get intermediate checksum.

22.5 Comments about API usage in RTOS

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

The triplets

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

Should be protected by RTOS mutex to protect CRC module against concurrent accesses from different tasks. For example: Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crc

Files

- file fsl_crc.h

Data Structures

- struct crc_config_t
  CRC protocol configuration. *More...*

Macros

- #define CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT 1
  Default configuration structure filled by CRC_GetDefaultConfig().

Enumerations

- enum crc_polynomial_t {
  kCRC_Polynomial_CRC_CCITT = 0U,
  kCRC_Polynomial_CRC_16 = 1U,
  kCRC_Polynomial_CRC_32 = 2U }
  CRC polynomials to use.

Functions

- void CRC_Init (CRC_Type *base, const crc_config_t *config)
  Enables and configures the CRC peripheral module.
- static void CRC_Deinit (CRC_Type *base)
  Disables the CRC peripheral module.
- void CRC_Reset (CRC_Type *base)
  resets CRC peripheral module.
- void CRC_WriteSeed (CRC_Type *base, uint32_t seed)
  Write seed to CRC peripheral module.
- void CRC_GetDefaultConfig (crc_config_t *config)
  Loads default values to CRC protocol configuration structure.
- void CRC_GetConfig (CRC_Type *base, crc_config_t *config)
  Loads actual values configured in CRC peripheral to CRC protocol configuration structure.
- void CRC_WriteData (CRC_Type *base, const uint8_t *data, size_t dataSize)
  Writes data to the CRC module.
- static uint32_t CRC_Get32bitResult (CRC_Type *base)
  Reads 32-bit checksum from the CRC module.
- static uint16_t CRC_Get16bitResult (CRC_Type *base)
  Reads 16-bit checksum from the CRC module.

Driver version

- #define FSL_CRC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  CRC driver version.

22.6 Data Structure Documentation

22.6.1 struct crc_config_t

This structure holds the configuration for the CRC protocol.

Data Fields

- crc_polynomial_t polynomial
  CRC polynomial.
- bool reverseIn
  Reverse bits on input.
- bool complementIn
  Perform 1's complement on input.
- bool reverseOut
  Reverse bits on output.
- bool complementOut
  Perform 1's complement on output.
- uint32_t seed
  Starting checksum value.

Field Documentation

(1) crc_polynomial_t crc_config_t::polynomial

(2) bool crc_config_t::reverseIn

(3) bool crc_config_t::complementIn

(4) bool crc_config_t::reverseOut

(5) bool crc_config_t::complementOut

(6) uint32_t crc_config_t::seed

22.7 Macro Definition Documentation

22.7.1 #define FSL_CRC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

Version 2.1.1.

Current version: 2.1.1

Change log:

- Version 2.0.0
  - initial version
- Version 2.0.1
  - add explicit type cast when writing to WR_DATA
- Version 2.0.2
  - Fix MISRA issue
- Version 2.1.0
  - Add CRC_WriteSeed function
- Version 2.1.1
  - Fix MISRA issue

22.7.2 #define CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT 1

Uses CRC-16/CCITT-FALSE as default.

22.8 Enumeration Type Documentation

22.8.1 enum crc_polynomial_t

Enumerator

kCRC_Polynomial_CRC_CCITT x^16+x^12+x^5+1
kCRC_Polynomial_CRC_16 x^16+x^15+x^2+1
kCRC_Polynomial_CRC_32 x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x+

22.9 Function Documentation

22.9.1 void CRC_Init (CRC_Type * *base,* const crc_config_t * *config*)

This functions enables the CRC peripheral clock in the LPC SYSCON block. It also configures the CRC engine and starts checksum computation by writing the seed.

Parameters

base	CRC peripheral address.
config	CRC module configuration structure.

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

This functions disables the CRC peripheral clock in the LPC SYSCON block.

Parameters

base	CRC peripheral address.

22.9.3 void CRC_Reset (CRC_Type * *base*)

Parameters

base	CRC peripheral address.

22.9.4 void CRC_WriteSeed (CRC_Type * *base*, uint32_t *seed*)

Parameters

base	CRC peripheral address.
seed	CRC Seed value.

22.9.5 void CRC_GetDefaultConfig (crc_config_t * *config*)

Loads default values to CRC protocol configuration structure. The default values are:

```
*   config->polynomial = kCRC_Polynomial_CRC_CCITT;
*   config->reverseIn = false;
*   config->complementIn = false;
*   config->reverseOut = false;
*   config->complementOut = false;
*   config->seed = 0xFFFFU;
*
```

Parameters

config	CRC protocol configuration structure

22.9.6 void CRC_GetConfig (CRC_Type * *base*, crc_config_t * *config*)

The values, including seed, can be used to resume CRC calculation later.

Parameters

base	CRC peripheral address.
config	CRC protocol configuration structure

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

Writes input data buffer bytes to CRC data register.

Parameters

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

22.9.8 static uint32_t CRC_Get32bitResult (CRC_Type * *base*) [inline], [static]

Reads CRC data register.

Parameters

base	CRC peripheral address.

Returns

final 32-bit checksum, after configured bit reverse and complement operations.

22.9.9 static uint16_t CRC_Get16bitResult (CRC_Type * *base*) [inline], [static]

Reads CRC data register.

Parameters

base	CRC peripheral address.

Returns

final 16-bit checksum, after configured bit reverse and complement operations.

Chapter 23
DMA: Direct Memory Access Controller Driver

23.1 Overview

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

23.2 Typical use case

23.2.1 DMA Operation

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

Files

- file fsl_dma.h

Data Structures

- struct dma_descriptor_t
  DMA descriptor structure. *More...*
- struct dma_xfercfg_t
  DMA transfer configuration. *More...*
- struct dma_channel_trigger_t
  DMA channel trigger. *More...*
- struct dma_channel_config_t
  DMA channel trigger. *More...*
- struct dma_transfer_config_t
  DMA transfer configuration. *More...*
- struct dma_handle_t
  DMA transfer handle structure. *More...*

Macros

- #define DMA_MAX_TRANSFER_COUNT 0x400U
  DMA max transfer size.
- #define FSL_FEATURE_DMA_NUMBER_OF_CHANNELSn(x) FSL_FEATURE_DMA_NUMBER_OF_CHANNELS
  DMA channel numbers.
- #define FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE (16U)
  DMA head link descriptor table align size.
- #define DMA_ALLOCATE_HEAD_DESCRIPTORS(name, number) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)
  DMA head descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.

- #define DMA_ALLOCATE_HEAD_DESCRIPTORS_AT_NONCACHEABLE(name, number) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)
  DMA head descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.
- #define DMA_ALLOCATE_LINK_DESCRIPTORS(name, number) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)
  DMA link descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.
- #define DMA_ALLOCATE_LINK_DESCRIPTORS_AT_NONCACHEABLE(name, number) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)
  DMA link descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.
- #define DMA_ALLOCATE_DATA_TRANSFER_BUFFER(name, width) SDK_ALIGN(name, width)
  DMA transfer buffer address need to align with the transfer width.
- #define DMA_COMMON_REG_GET(base, channel, reg) (((volatile uint32_t *)(&((base)->COMMON[0].reg)))[DMA_CHANNEL_GROUP(channel)])
  DMA linked descriptor address algin size.
- #define DMA_DESCRIPTOR_END_ADDRESS(start, inc, bytes, width) ((uint32_t *)((uint32_t)(start) + (inc) * (bytes) - (inc) * (width)))
  DMA descriptor end address calculate.
- #define DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc, bytes)
  DMA channel transfer configurations macro.

Typedefs

- typedef void(* dma_callback)(struct _dma_handle *handle, void *userData, bool transferDone, uint32_t intmode)
  Define Callback function for DMA.

Enumerations

- enum { kStatus_DMA_Busy = MAKE_STATUS(kStatusGroup_DMA, 0) }
  _dma_transfer_status DMA transfer status
- enum {
  kDMA_AddressInterleave0xWidth = 0U,
  kDMA_AddressInterleave1xWidth = 1U,
  kDMA_AddressInterleave2xWidth = 2U,
  kDMA_AddressInterleave4xWidth = 4U }
  _dma_addr_interleave_size dma address interleave size
- enum {
  kDMA_Transfer8BitWidth = 1U,
  kDMA_Transfer16BitWidth = 2U,
  kDMA_Transfer32BitWidth = 4U }

_dma_transfer_width dma transfer width

- enum dma_priority_t {
  kDMA_ChannelPriority0 = 0,
  kDMA_ChannelPriority1,
  kDMA_ChannelPriority2,
  kDMA_ChannelPriority3,
  kDMA_ChannelPriority4,
  kDMA_ChannelPriority5,
  kDMA_ChannelPriority6,
  kDMA_ChannelPriority7 }
  DMA channel priority.
- enum dma_irq_t {
  kDMA_IntA,
  kDMA_IntB,
  kDMA_IntError }
  DMA interrupt flags.
- enum dma_trigger_type_t {
  kDMA_NoTrigger = 0,
  kDMA_LowLevelTrigger = DMA_CHANNEL_CFG_HWTRIGEN(1) | DMA_CHANNEL_CFG-_TRIGTYPE(1),
  kDMA_HighLevelTrigger,
  kDMA_FallingEdgeTrigger = DMA_CHANNEL_CFG_HWTRIGEN(1),
  kDMA_RisingEdgeTrigger }
  DMA trigger type.
- enum {
  kDMA_BurstSize1 = 0U,
  kDMA_BurstSize2 = 1U,
  kDMA_BurstSize4 = 2U,
  kDMA_BurstSize8 = 3U,
  kDMA_BurstSize16 = 4U,
  kDMA_BurstSize32 = 5U,
  kDMA_BurstSize64 = 6U,
  kDMA_BurstSize128 = 7U,
  kDMA_BurstSize256 = 8U,
  kDMA_BurstSize512 = 9U,
  kDMA_BurstSize1024 = 10U }
  _dma_burst_size DMA burst size
- enum dma_trigger_burst_t {

kDMA_SingleTransfer = 0,
kDMA_LevelBurstTransfer = DMA_CHANNEL_CFG_TRIGBURST(1),
kDMA_EdgeBurstTransfer1 = DMA_CHANNEL_CFG_TRIGBURST(1),
kDMA_EdgeBurstTransfer2,
kDMA_EdgeBurstTransfer4,
kDMA_EdgeBurstTransfer8,
kDMA_EdgeBurstTransfer16,
kDMA_EdgeBurstTransfer32,
kDMA_EdgeBurstTransfer64,
kDMA_EdgeBurstTransfer128,
kDMA_EdgeBurstTransfer256,
kDMA_EdgeBurstTransfer512,
kDMA_EdgeBurstTransfer1024 }
DMA trigger burst.

- enum dma_burst_wrap_t {
  kDMA_NoWrap = 0,
  kDMA_SrcWrap = DMA_CHANNEL_CFG_SRCBURSTWRAP(1),
  kDMA_DstWrap = DMA_CHANNEL_CFG_DSTBURSTWRAP(1),
  kDMA_SrcAndDstWrap }
  DMA burst wrapping.
- enum dma_transfer_type_t {
  kDMA_MemoryToMemory = 0x0U,
  kDMA_PeripheralToMemory,
  kDMA_MemoryToPeripheral,
  kDMA_StaticToStatic }
  DMA transfer type.

Driver version

- #define FSL_DMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 3))
  DMA driver version.

DMA initialization and De-initialization

- void DMA_Init (DMA_Type *base)
  Initializes DMA peripheral.
- void DMA_Deinit (DMA_Type *base)
  Deinitializes DMA peripheral.
- void DMA_InstallDescriptorMemory (DMA_Type *base, void *addr)
  Install DMA descriptor memory.

DMA Channel Operation

- static bool DMA_ChannelIsActive (DMA_Type *base, uint32_t channel)
  Return whether DMA channel is processing transfer.
- static bool DMA_ChannelIsBusy (DMA_Type *base, uint32_t channel)
  Return whether DMA channel is busy.
- static void DMA_EnableChannelInterrupts (DMA_Type *base, uint32_t channel)

Enables the interrupt source for the DMA transfer.
- static void DMA_DisableChannelInterrupts (DMA_Type *base, uint32_t channel)
  Disables the interrupt source for the DMA transfer.
- static void DMA_EnableChannel (DMA_Type *base, uint32_t channel)
  Enable DMA channel.
- static void DMA_DisableChannel (DMA_Type *base, uint32_t channel)
  Disable DMA channel.
- static void DMA_EnableChannelPeriphRq (DMA_Type *base, uint32_t channel)
  Set PERIPHREQEN of channel configuration register.
- static void DMA_DisableChannelPeriphRq (DMA_Type *base, uint32_t channel)
  Get PERIPHREQEN value of channel configuration register.
- void DMA_ConfigureChannelTrigger (DMA_Type *base, uint32_t channel, dma_channel_trigger_t *trigger)
  Set trigger settings of DMA channel.
- void DMA_SetChannelConfig (DMA_Type *base, uint32_t channel, dma_channel_trigger_t *trigger, bool isPeriph)
  set channel config.
- uint32_t DMA_GetRemainingBytes (DMA_Type *base, uint32_t channel)
  Gets the remaining bytes of the current DMA descriptor transfer.
- static void DMA_SetChannelPriority (DMA_Type *base, uint32_t channel, dma_priority_t priority)
  Set priority of channel configuration register.
- static dma_priority_t DMA_GetChannelPriority (DMA_Type *base, uint32_t channel)
  Get priority of channel configuration register.
- static void DMA_SetChannelConfigValid (DMA_Type *base, uint32_t channel)
  Set channel configuration valid.
- static void DMA_DoChannelSoftwareTrigger (DMA_Type *base, uint32_t channel)
  Do software trigger for the channel.
- static void DMA_LoadChannelTransferConfig (DMA_Type *base, uint32_t channel, uint32_t xfer)
  Load channel transfer configurations.
- void DMA_CreateDescriptor (dma_descriptor_t *desc, dma_xfercfg_t *xfercfg, void *srcAddr, void *dstAddr, void *nextDesc)
  Create application specific DMA descriptor to be used in a chain in transfer.
- void DMA_SetupDescriptor (dma_descriptor_t *desc, uint32_t xfercfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc)
  setup dma descriptor
- void DMA_SetupChannelDescriptor (dma_descriptor_t *desc, uint32_t xfercfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc, dma_burst_wrap_t wrapType, uint32_t burstSize)
  setup dma channel descriptor
- void DMA_LoadChannelDescriptor (DMA_Type *base, uint32_t channel, dma_descriptor_t *descriptor)
  load channel transfer decriptor.

DMA Transactional Operation

- void DMA_AbortTransfer (dma_handle_t *handle)
  Abort running transfer by handle.
- void DMA_CreateHandle (dma_handle_t *handle, DMA_Type *base, uint32_t channel)
  Creates the DMA handle.
- void DMA_SetCallback (dma_handle_t *handle, dma_callback callback, void *userData)
  Installs a callback function for the DMA transfer.

- void DMA_PrepareTransfer (dma_transfer_config_t *config, void *srcAddr, void *dstAddr, uint32_t byteWidth, uint32_t transferBytes, dma_transfer_type_t type, void *nextDesc)
  Prepares the DMA transfer structure.
- void DMA_PrepareChannelTransfer (dma_channel_config_t *config, void *srcStartAddr, void *dstStartAddr, uint32_t xferCfg, dma_transfer_type_t type, dma_channel_trigger_t *trigger, void *nextDesc)
  Prepare channel transfer configurations.
- status_t DMA_SubmitTransfer (dma_handle_t *handle, dma_transfer_config_t *config)
  Submits the DMA transfer request.
- void DMA_SubmitChannelTransferParameter (dma_handle_t *handle, uint32_t xferCfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc)
  Submit channel transfer paramter directly.
- void DMA_SubmitChannelDescriptor (dma_handle_t *handle, dma_descriptor_t *descriptor)
  Submit channel descriptor.
- status_t DMA_SubmitChannelTransfer (dma_handle_t *handle, dma_channel_config_t *config)
  Submits the DMA channel transfer request.
- void DMA_StartTransfer (dma_handle_t *handle)
  DMA start transfer.
- void DMA_IRQHandle (DMA_Type *base)
  DMA IRQ handler for descriptor transfer complete.

23.3 Data Structure Documentation

23.3.1 struct dma_descriptor_t

Data Fields

- volatile uint32_t xfercfg
  Transfer configuration.
- void * srcEndAddr
  Last source address of DMA transfer.
- void * dstEndAddr
  Last destination address of DMA transfer.
- void * linkToNextDesc
  Address of next DMA descriptor in chain.

23.3.2 struct dma_xfercfg_t

Data Fields

- bool valid
  Descriptor is ready to transfer.
- bool reload
  Reload channel configuration register after current descriptor is exhausted.
- bool swtrig
  Perform software trigger.
- bool clrtrig

Clear trigger.
- bool intA
  Raises IRQ when transfer is done and set IRQA status register flag.
- bool intB
  Raises IRQ when transfer is done and set IRQB status register flag.
- uint8_t byteWidth
  Byte width of data to transfer.
- uint8_t srcInc
  Increment source address by 'srcInc' x 'byteWidth'.
- uint8_t dstInc
  Increment destination address by 'dstInc' x 'byteWidth'.
- uint16_t transferCount
  Number of transfers.

Field Documentation

(1) bool dma_xfercfg_t::swtrig

Transfer if fired when 'valid' is set

23.3.3 struct dma_channel_trigger_t

Data Fields

- dma_trigger_type_t type
  Select hardware trigger as edge triggered or level triggered.
- dma_trigger_burst_t burst
  Select whether hardware triggers cause a single or burst transfer.
- dma_burst_wrap_t wrap
  Select wrap type, source wrap or dest wrap, or both.

Field Documentation

(1) dma_trigger_type_t dma_channel_trigger_t::type

(2) dma_trigger_burst_t dma_channel_trigger_t::burst

(3) dma_burst_wrap_t dma_channel_trigger_t::wrap

23.3.4 struct dma_channel_config_t

Data Fields

- void * srcStartAddr
  Source data address.
- void * dstStartAddr
  Destination data address.
- void * nextDesc

Chain custom descriptor.
- uint32_t xferCfg
  channel transfer configurations
- dma_channel_trigger_t * trigger
  DMA trigger type.
- bool isPeriph
  select the request type

23.3.5 struct dma_transfer_config_t

Data Fields

- uint8_t * srcAddr
  Source data address.
- uint8_t * dstAddr
  Destination data address.
- uint8_t * nextDesc
  Chain custom descriptor.
- dma_xfercfg_t xfercfg
  Transfer options.
- bool isPeriph
  DMA transfer is driven by peripheral.

23.3.6 struct dma_handle_t

Data Fields

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

Field Documentation

(1) dma_callback dma_handle_t::callback

Invoked when transfer of descriptor with interrupt flag finishes

23.4 Macro Definition Documentation

23.4.1 #define FSL_DMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 3))

Version 2.4.3.

23.4.2 #define DMA_ALLOCATE_HEAD_DESCRIPTORS(*name, number*) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)

Parameters

name	Allocate decriptor name.
number	Number of descriptor to be allocated.

23.4.3 #define DMA_ALLOCATE_HEAD_DESCRIPTORS_AT_NONCACHEABLE(*name, number*) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)

Parameters

name	Allocate decriptor name.
number	Number of descriptor to be allocated.

23.4.4 #define DMA_ALLOCATE_LINK_DESCRIPTORS(*name, number*) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)

Parameters

name	Allocate decriptor name.
number	Number of descriptor to be allocated.

23.4.5 #define DMA_ALLOCATE_LINK_DESCRIPTORS_AT_NONCACHEABLE(*name, number*) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)

Parameters

name	Allocate decriptor name.
number	Number of descriptor to be allocated.

23.4.6 #define DMA_DESCRIPTOR_END_ADDRESS(*start, inc, bytes, width*) ((uint32_t *)((uint32_t)(start) + (inc) * (bytes) - (inc) * (width)))

Parameters

start	start address
inc	address interleave size
bytes	transfer bytes
width	transfer width

23.4.7 #define DMA_CHANNEL_XFER(*reload, clrTrig, intA, intB, width, srcInc, dstInc, bytes*)

Value:

```
DMA_CHANNEL_XFERCFG_CFGVALID_MASK | DMA_CHANNEL_XFERCFG_RELOAD(reload) | DMA_CHANNEL_XFERCFG_CLRTRIG(
      clrTrig) | \
        DMA_CHANNEL_XFERCFG_SETINTA(intA) | DMA_CHANNEL_XFERCFG_SETINTB(intB) |
              \
        DMA_CHANNEL_XFERCFG_WIDTH(width == 4UL ? 2UL : (width - 1UL)) |
              \
        DMA_CHANNEL_XFERCFG_SRCINC(srcInc == (uint32_t)
      kDMA_AddressInterleave4xWidth ? (srcInc - 1UL) : srcInc) |   \
        DMA_CHANNEL_XFERCFG_DSTINC(dstInc == (uint32_t)
      kDMA_AddressInterleave4xWidth ? (dstInc - 1UL) : dstInc) |   \
        DMA_CHANNEL_XFERCFG_XFERCOUNT(bytes / width - 1UL)
```

Parameters

reload	true is reload link descriptor after current exhaust, false is not
clrTrig	true is clear trigger status, wait software trigger, false is not
intA	enable interruptA
intB	enable interruptB
width	transfer width
srcInc	source address interleave size
dstInc	destination address interleave size
bytes	transfer bytes

23.5 Typedef Documentation

23.5.1 typedef void(∗ dma_callback)(struct _dma_handle ∗handle, void ∗userData, bool transferDone, uint32_t intmode)

23.6 Enumeration Type Documentation

23.6.1 anonymous enum

Enumerator

kStatus_DMA_Busy Channel is busy and can't handle the transfer request.

23.6.2 anonymous enum

Enumerator

kDMA_AddressInterleave0xWidth dma source/destination address no interleave
kDMA_AddressInterleave1xWidth dma source/destination address interleave 1xwidth
kDMA_AddressInterleave2xWidth dma source/destination address interleave 2xwidth
kDMA_AddressInterleave4xWidth dma source/destination address interleave 3xwidth

23.6.3 anonymous enum

Enumerator

kDMA_Transfer8BitWidth dma channel transfer bit width is 8 bit
kDMA_Transfer16BitWidth dma channel transfer bit width is 16 bit
kDMA_Transfer32BitWidth dma channel transfer bit width is 32 bit

23.6.4 enum dma_priority_t

Enumerator

kDMA_ChannelPriority0 Highest channel priority - priority 0.
kDMA_ChannelPriority1 Channel priority 1.
kDMA_ChannelPriority2 Channel priority 2.
kDMA_ChannelPriority3 Channel priority 3.
kDMA_ChannelPriority4 Channel priority 4.
kDMA_ChannelPriority5 Channel priority 5.
kDMA_ChannelPriority6 Channel priority 6.
kDMA_ChannelPriority7 Lowest channel priority - priority 7.

23.6.5 enum dma_irq_t

Enumerator

kDMA_IntA DMA interrupt flag A.
kDMA_IntB DMA interrupt flag B.
kDMA_IntError DMA interrupt flag error.

23.6.6 enum dma_trigger_type_t

Enumerator

kDMA_NoTrigger Trigger is disabled.
kDMA_LowLevelTrigger Low level active trigger.
kDMA_HighLevelTrigger High level active trigger.
kDMA_FallingEdgeTrigger Falling edge active trigger.
kDMA_RisingEdgeTrigger Rising edge active trigger.

23.6.7 anonymous enum

Enumerator

kDMA_BurstSize1 burst size 1 transfer
kDMA_BurstSize2 burst size 2 transfer
kDMA_BurstSize4 burst size 4 transfer
kDMA_BurstSize8 burst size 8 transfer
kDMA_BurstSize16 burst size 16 transfer
kDMA_BurstSize32 burst size 32 transfer
kDMA_BurstSize64 burst size 64 transfer
kDMA_BurstSize128 burst size 128 transfer
kDMA_BurstSize256 burst size 256 transfer
kDMA_BurstSize512 burst size 512 transfer
kDMA_BurstSize1024 burst size 1024 transfer

23.6.8 enum dma_trigger_burst_t

Enumerator

kDMA_SingleTransfer Single transfer.
kDMA_LevelBurstTransfer Burst transfer driven by level trigger.
kDMA_EdgeBurstTransfer1 Perform 1 transfer by edge trigger.
kDMA_EdgeBurstTransfer2 Perform 2 transfers by edge trigger.
kDMA_EdgeBurstTransfer4 Perform 4 transfers by edge trigger.
kDMA_EdgeBurstTransfer8 Perform 8 transfers by edge trigger.
kDMA_EdgeBurstTransfer16 Perform 16 transfers by edge trigger.
kDMA_EdgeBurstTransfer32 Perform 32 transfers by edge trigger.
kDMA_EdgeBurstTransfer64 Perform 64 transfers by edge trigger.
kDMA_EdgeBurstTransfer128 Perform 128 transfers by edge trigger.
kDMA_EdgeBurstTransfer256 Perform 256 transfers by edge trigger.
kDMA_EdgeBurstTransfer512 Perform 512 transfers by edge trigger.
kDMA_EdgeBurstTransfer1024 Perform 1024 transfers by edge trigger.

23.6.9 enum dma_burst_wrap_t

Enumerator

kDMA_NoWrap Wrapping is disabled.
kDMA_SrcWrap Wrapping is enabled for source.
kDMA_DstWrap Wrapping is enabled for destination.
kDMA_SrcAndDstWrap Wrapping is enabled for source and destination.

23.6.10 enum dma_transfer_type_t

Enumerator

kDMA_MemoryToMemory Transfer from memory to memory (increment source and destination)
kDMA_PeripheralToMemory Transfer from peripheral to memory (increment only destination)
kDMA_MemoryToPeripheral Transfer from memory to peripheral (increment only source)
kDMA_StaticToStatic Peripheral to static memory (do not increment source or destination)

23.7 Function Documentation

23.7.1 void DMA_Init (DMA_Type * *base*)

This function enable the DMA clock, set descriptor table and enable DMA peripheral.

Parameters

base	DMA peripheral base address.

23.7.2 void DMA_Deinit (DMA_Type * *base*)

This function gates the DMA clock.

Parameters

base	DMA peripheral base address.

23.7.3 void DMA_InstallDescriptorMemory (DMA_Type * *base,* void * *addr*)

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, althrough current DMA driver has a default DMA descriptor buffer, but it support one DMA descriptor for one channel only.

Parameters

base	DMA base address.
addr	DMA descriptor address

23.7.4 static bool DMA_ChannelIsActive (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

True for active state, false otherwise.

23.7.5 static bool DMA_ChannelIsBusy (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

True for busy state, false otherwise.

23.7.6 static void DMA_EnableChannelInterrupts (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

23.7.7 static void DMA_DisableChannelInterrupts (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

23.7.8 static void DMA_EnableChannel (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

23.7.9 static void DMA_DisableChannel (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

23.7.10 static void DMA_EnableChannelPeriphRq (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

23.7.11 static void DMA_DisableChannelPeriphRq (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

True for enabled PeriphRq, false for disabled.

23.7.12 void DMA_ConfigureChannelTrigger (DMA_Type * *base*, uint32_t *channel*, dma_channel_trigger_t * *trigger*)

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

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
trigger	trigger configuration.

23.7.13 void DMA_SetChannelConfig (DMA_Type * *base*, uint32_t *channel*, dma_channel_trigger_t * *trigger*, bool *isPeriph*)

This function provide a interface to configure channel configuration reisters.

Parameters

base	DMA base address.
channel	DMA channel number.
trigger	channel configurations structure.
isPeriph	true is periph request, false is not.

23.7.14 uint32_t DMA_GetRemainingBytes (DMA_Type * *base,* uint32_t *channel*)

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

The number of bytes which have not been transferred yet.

23.7.15 static void DMA_SetChannelPriority (DMA_Type * *base,* uint32_t *channel,* dma_priority_t *priority*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
priority	Channel priority value.

23.7.16 static dma_priority_t DMA_GetChannelPriority (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

Channel priority value.

23.7.17 static void DMA_SetChannelConfigValid (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

23.7.18 static void DMA_DoChannelSoftwareTrigger (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

23.7.19 static void DMA_LoadChannelTransferConfig (DMA_Type * *base*, uint32_t *channel*, uint32_t *xfer*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
xfer	transfer configurations.

23.7.20 void DMA_CreateDescriptor (dma_descriptor_t * *desc*, dma_xfercfg_t * *xfercfg*, void * *srcAddr*, void * *dstAddr*, void * *nextDesc*)

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

Parameters

desc	DMA descriptor address.
xfercfg	Transfer configuration for DMA descriptor.
srcAddr	Address of last item to transmit
dstAddr	Address of last item to receive.
nextDesc	Address of next descriptor in chain.

23.7.21 void DMA_SetupDescriptor (dma_descriptor_t * *desc*, uint32_t *xfercfg*, void * *srcStartAddr*, void * *dstStartAddr*, void * *nextDesc*)

Note: This function do not support configure wrap descriptor.

Parameters

desc	DMA descriptor address.
xfercfg	Transfer configuration for DMA descriptor.
srcStartAddr	Start address of source address.
dstStartAddr	Start address of destination address.
nextDesc	Address of next descriptor in chain.

23.7.22 void DMA_SetupChannelDescriptor (dma_descriptor_t * *desc*, uint32_t *xfercfg*, void * *srcStartAddr*, void * *dstStartAddr*, void * *nextDesc*, dma_burst_wrap_t *wrapType*, uint32_t *burstSize*)

Note: This function support configure wrap descriptor.

Parameters

desc	DMA descriptor address.
xfercfg	Transfer configuration for DMA descriptor.
srcStartAddr	Start address of source address.
dstStartAddr	Start address of destination address.

nextDesc	Address of next descriptor in chain.
wrapType	burst wrap type.
burstSize	burst size, reference _dma_burst_size.

23.7.23 void DMA_LoadChannelDescriptor (DMA_Type * *base*, uint32_t *channel*, dma_descriptor_t * *descriptor*)

This function can be used to load desscriptor to driver internal channel descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

1. for the polling transfer, application can allocate a local descriptor memory table to prepare a descriptor firstly and then call this api to load the configured descriptor to driver descriptor table.

```
*   DMA_Init(DMA0);
*   DMA_EnableChannel(DMA0, DEMO_DMA_CHANNEL);
*   DMA_SetupDescriptor(desc, xferCfg, s_srcBuffer, &s_destBuffer[0], NULL);
*   DMA_LoadChannelDescriptor(DMA0, DEMO_DMA_CHANNEL, (
      dma_descriptor_t *)desc);
*   DMA_DoChannelSoftwareTrigger(DMA0, DEMO_DMA_CHANNEL);
*   while(DMA_ChannelIsBusy(DMA0, DEMO_DMA_CHANNEL))
*   {}
*
```

Parameters

base	DMA base address.
channel	DMA channel.
descriptor	configured DMA descriptor.

23.7.24 void DMA_AbortTransfer (dma_handle_t * *handle*)

This function aborts DMA transfer specified by handle.

Parameters

handle	DMA handle pointer.

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

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

Parameters

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

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

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

Parameters

handle	DMA handle pointer.
callback	DMA callback function pointer.
userData	Parameter for callback function.

23.7.27 void DMA_PrepareTransfer (dma_transfer_config_t * *config*, void * *srcAddr*, void * *dstAddr*, uint32_t *byteWidth*, uint32_t *transferBytes*, dma_transfer_type_t *type*, void * *nextDesc*)

Deprecated Do not use this function. It has been superceded by DMA_PrepareChannelTransfer. This function prepares the transfer configuration structure according to the user input.

Parameters

config	The user configuration structure of type dma_transfer_t.
srcAddr	DMA transfer source address.
dstAddr	DMA transfer destination address.
byteWidth	DMA transfer destination address width(bytes).
transferBytes	DMA transfer bytes to be transferred.
type	DMA transfer type.
nextDesc	Chain custom descriptor to transfer.

Note

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

23.7.28 void DMA_PrepareChannelTransfer (dma_channel_config_t * *config*, void * *srcStartAddr*, void * *dstStartAddr*, uint32_t *xferCfg*, dma_transfer_type_t *type*, dma_channel_trigger_t * *trigger*, void * *nextDesc*)

This function used to prepare channel transfer configurations.

Parameters

config	Pointer to DMA channel transfer configuration structure.
srcStartAddr	source start address.
dstStartAddr	destination start address.
xferCfg	xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
type	transfer type.
trigger	DMA channel trigger configurations.
nextDesc	address of next descriptor.

23.7.29 status_t DMA_SubmitTransfer (dma_handle_t * *handle*, dma_transfer_config_t * *config*)

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

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

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

Return values

kStatus_DMA_Success	It means submit transfer request succeed.
kStatus_DMA_QueueFull	It means TCD queue is full. Submit transfer request is not allowed.
kStatus_DMA_Busy	It means the given channel is busy, need to submit request later.

23.7.30 void DMA_SubmitChannelTransferParameter (dma_handle_t * *handle*, uint32_t *xferCfg*, void * *srcStartAddr*, void * *dstStartAddr*, void * *nextDesc*)

This function used to configue channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

1. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
   DMA_SetChannelConfig(base, channel, trigger, isPeriph);
   DMA_CreateHandle(handle, base, channel)
   DMA_SubmitChannelTransferParameter(handle,
      DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, NULL);
   DMA_StartTransfer(handle)
*
```

2. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
   define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[3]);

   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
   DMA_SetupDescriptor(nextDesc2,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
   DMA_SetChannelConfig(base, channel, trigger, isPeriph);
   DMA_CreateHandle(handle, base, channel)
   DMA_SubmitChannelTransferParameter(handle,
      DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, nextDesc0);
   DMA_StartTransfer(handle);
*
```

Parameters

handle	Pointer to DMA handle.
xferCfg	xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
srcStartAddr	source start address.
dstStartAddr	destination start address.
nextDesc	address of next descriptor.

23.7.31 void DMA_SubmitChannelDescriptor (dma_handle_t * *handle*, dma_descriptor_t * *descriptor*)

This function used to configue channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, this functiono is typical for the ping pong case:

1. for the ping pong case, application should responsible for the descriptor, for example, application should prepare two descriptor table with macro.

```
   define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[2]);

   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
   DMA_SetChannelConfig(base, channel, trigger, isPeriph);
   DMA_CreateHandle(handle, base, channel)
   DMA_SubmitChannelDescriptor(handle,  nextDesc0);
   DMA_StartTransfer(handle);
*
```

Parameters

handle	Pointer to DMA handle.
descriptor	descriptor to submit.

23.7.32 status_t DMA_SubmitChannelTransfer (dma_handle_t * *handle*, dma_channel_config_t * *config*)

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time. It is used for the case:

1. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
   DMA_CreateHandle(handle, base, channel)
   DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
      trigger,NULL);
   DMA_SubmitChannelTransfer(handle, config)
   DMA_StartTransfer(handle)
*
```

2. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
   define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);
   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
   DMA_SetupDescriptor(nextDesc2,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
   DMA_CreateHandle(handle, base, channel)
   DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
      trigger,nextDesc0);
   DMA_SubmitChannelTransfer(handle, config)
   DMA_StartTransfer(handle)
*
```

3. for the ping pong case, application should responsible for link descriptor, for example, application should prepare two descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
   define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);

   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
   DMA_CreateHandle(handle, base, channel)
   DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
      trigger,nextDesc0);
   DMA_SubmitChannelTransfer(handle, config)
   DMA_StartTransfer(handle)
*
```

Parameters

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

Return values

kStatus_DMA_Success	It means submit transfer request succeed.
kStatus_DMA_QueueFull	It means TCD queue is full. Submit transfer request is not allowed.
kStatus_DMA_Busy	It means the given channel is busy, need to submit request later.

23.7.33 void DMA_StartTransfer (dma_handle_t * *handle*)

This function enables the channel request. User can call this function after submitting the transfer request It will trigger transfer start with software trigger only when hardware trigger is not used.

Parameters

handle	DMA handle pointer.

23.7.34 void DMA_IRQHandle (DMA_Type * *base*)

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

Parameters

base	DMA base address.

Chapter 24
GPIO: General Purpose I/O

24.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Purpose I/O (GPIO) module of MCUXpresso SDK devices.

24.2 Function groups

24.2.1 Initialization and deinitialization

The function GPIO_PinInit() initializes the GPIO with specified configuration.

24.2.2 Pin manipulation

The function GPIO_PinWrite() set output state of selected GPIO pin. The function GPIO_PinRead() read input value of selected GPIO pin.

24.2.3 Port manipulation

The function GPIO_PortSet() sets the output level of selected GPIO pins to the logic 1. The function GPIO_PortClear() sets the output level of selected GPIO pins to the logic 0. The function GPIO_PortToggle() reverse the output level of selected GPIO pins. The function GPIO_PortRead() read input value of selected port.

24.2.4 Port masking

The function GPIO_PortMaskedSet() set port mask, only pins masked by 0 will be enabled in following functions. The function GPIO_PortMaskedWrite() sets the state of selected GPIO port, only pins masked by 0 will be affected. The function GPIO_PortMaskedRead() reads the state of selected GPIO port, only pins masked by 0 are enabled for read, pins masked by 1 are read as 0.

24.3 Typical use case

Example use of GPIO API. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/gpio

Files

- file fsl_gpio.h

Data Structures

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

Enumerations

- enum gpio_pin_direction_t {
  kGPIO_DigitalInput = 0U,
  kGPIO_DigitalOutput = 1U }
  LPC GPIO direction definition.

Functions

- static void GPIO_PortSet (GPIO_Type *base, uint32_t port, 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 port, 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 port, uint32_t mask)
  Reverses current output logic of the multiple GPIO pins.

Driver version

- #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))
  LPC GPIO driver version.

GPIO Configuration

- void GPIO_PortInit (GPIO_Type *base, uint32_t port)
  Initializes the GPIO peripheral.
- void GPIO_PinInit (GPIO_Type *base, uint32_t port, uint32_t pin, const gpio_pin_config_t *config)
  Initializes a GPIO pin used by the board.

GPIO Output Operations

- static void GPIO_PinWrite (GPIO_Type *base, uint32_t port, uint32_t pin, uint8_t output)
  Sets the output level of the one GPIO pin to the logic 1 or 0.

GPIO Input Operations

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

24.4 Data Structure Documentation

24.4.1 struct gpio_pin_config_t

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

Data Fields

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

24.5 Macro Definition Documentation

24.5.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))

24.6 Enumeration Type Documentation

24.6.1 enum gpio_pin_direction_t

Enumerator

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

24.7 Function Documentation

24.7.1 void GPIO_PortInit (GPIO_Type * *base*, uint32_t *port*)

This function ungates the GPIO clock.

Parameters

base	GPIO peripheral base pointer.
port	GPIO port number.

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

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

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

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

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
pin	GPIO pin number
config	GPIO pin configuration pointer

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

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
pin	GPIO pin number
output	GPIO pin output logic level. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

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

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
pin	GPIO pin number

Return values

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

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

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
mask	GPIO pin number macro

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

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
mask	GPIO pin number macro

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

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
mask	GPIO pin number macro

Chapter 25
IOCON: I/O pin configuration

25.1 Overview

The MCUXpresso SDK provides a peripheral driver for the I/O pin configuration (IOCON) module of MCUXpresso SDK devices.

25.2 Function groups

25.2.1 Pin mux set

The function IOCONPinMuxSet() set pinmux for single pin according to selected configuration.

25.2.2 Pin mux set

The function IOCON_SetPinMuxing() set pinmux for group of pins according to selected configuration.

25.3 Typical use case

Example use of IOCON API to selection of GPIO mode. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/iocon

Files

- file fsl_iocon.h

Data Structures

- struct iocon_group_t
  Array of IOCON pin definitions passed to IOCON_SetPinMuxing() must be in this format. More...

Macros

- #define IOCON_FUNC0 0x0
  IOCON function and mode selection definitions.
- #define IOCON_FUNC1 0x1
  Selects pin function 1.
- #define IOCON_FUNC2 0x2
  Selects pin function 2.
- #define IOCON_FUNC3 0x3
  Selects pin function 3.
- #define IOCON_FUNC4 0x4
  Selects pin function 4.

- #define IOCON_FUNC5 0x5
  Selects pin function 5.
- #define IOCON_FUNC6 0x6
  Selects pin function 6.
- #define IOCON_FUNC7 0x7
  Selects pin function 7.
- #define IOCON_FUNC8 0x8
  Selects pin function 8.
- #define IOCON_FUNC9 0x9
  Selects pin function 9.
- #define IOCON_FUNC10 0xA
  Selects pin function 10.
- #define IOCON_FUNC11 0xB
  Selects pin function 11.
- #define IOCON_FUNC12 0xC
  Selects pin function 12.
- #define IOCON_FUNC13 0xD
  Selects pin function 13.
- #define IOCON_FUNC14 0xE
  Selects pin function 14.
- #define IOCON_FUNC15 0xF
  Selects pin function 15.

Functions

- __STATIC_INLINE void IOCON_PinMuxSet (IOCON_Type *base, uint8_t port, uint8_t pin, uint32_t modefunc)
  Sets I/O Control pin mux.
- __STATIC_INLINE void IOCON_SetPinMuxing (IOCON_Type *base, const iocon_group_t *pinArray, uint32_t arrayLength)
  Set all I/O Control pin muxing.

Driver version

- #define FSL_IOCON_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  IOCON driver version.

25.4 Data Structure Documentation

25.4.1 struct iocon_group_t

25.5 Macro Definition Documentation

25.5.1 #define FSL_IOCON_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

25.5.2 #define IOCON_FUNC0 0x0

Note

See the User Manual for specific modes and functions supported by the various pins.Selects pin function 0

25.6 Function Documentation

25.6.1 __STATIC_INLINE void IOCON_PinMuxSet (IOCON_Type * *base,* uint8_t *port,* uint8_t *pin,* uint32_t *modefunc*)

Parameters

base	: The base of IOCON peripheral on the chip
port	: GPIO port to mux
pin	: GPIO pin to mux
modefunc	: OR'ed values of type IOCON_*

Returns

Nothing

25.6.2 __STATIC_INLINE void IOCON_SetPinMuxing (IOCON_Type * *base,* const iocon_group_t * *pinArray,* uint32_t *arrayLength*)

Parameters

base	: The base of IOCON peripheral on the chip
pinArray	: Pointer to array of pin mux selections
arrayLength	: Number of entries in pinArray

Returns

Nothing

Chapter 26
RTC: Real Time Clock

26.1 Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC).

26.2 Function groups

The RTC driver supports operating the module as a time counter.

26.2.1 Initialization and deinitialization

The function RTC_Init() initializes the RTC with specified configurations. The function RTC_GetDefault-Config() gets the default configurations.

The function RTC_Deinit() disables the RTC timer and disables the module clock.

26.2.2 Set & Get Datetime

The function RTC_SetDatetime() sets the timer period in seconds. User passes in the details in date & time format by using the below data structure.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/rtc The function RTC_GetDatetime() reads the current timer value in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

26.2.3 Set & Get Alarm

The function RTC_SetAlarm() sets the alarm time period in seconds. User passes in the details in date & time format by using the datetime data structure.

The function RTC_GetAlarm() reads the alarm time in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

26.2.4 Start & Stop timer

The function RTC_StartTimer() starts the RTC time counter.

The function RTC_StopTimer() stops the RTC time counter.

26.2.5 Status

Provides functions to get and clear the RTC status.

26.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

26.2.7 High resolution timer

Provides functions to enable high resolution timer and set and get the wake time.

26.3 Typical use case

26.3.1 RTC tick example

Example to set the RTC current time and trigger an alarm. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/rtc

Files

- file fsl_rtc.h

Data Structures

- struct rtc_datetime_t
  Structure is used to hold the date and time. *More...*

Enumerations

- enum rtc_interrupt_enable_t {
  kRTC_AlarmInterruptEnable = RTC_CTRL_ALARMDPD_EN_MASK,
  kRTC_WakeupInterruptEnable = RTC_CTRL_WAKEDPD_EN_MASK }
  List of RTC interrupts.
- enum rtc_status_flags_t {
  kRTC_AlarmFlag = RTC_CTRL_ALARM1HZ_MASK,
  kRTC_WakeupFlag = RTC_CTRL_WAKE1KHZ_MASK }
  List of RTC flags.

Functions

- static void RTC_SetSecondsTimerMatch (RTC_Type *base, uint32_t matchValue)
  Set the RTC seconds timer (1HZ) MATCH value.
- static uint32_t RTC_GetSecondsTimerMatch (RTC_Type *base)
  Read actual RTC seconds timer (1HZ) MATCH value.

- static void RTC_SetSecondsTimerCount (RTC_Type *base, uint32_t countValue)
  Set the RTC seconds timer (1HZ) COUNT value.
- static uint32_t RTC_GetSecondsTimerCount (RTC_Type *base)
  Read the actual RTC seconds timer (1HZ) COUNT value.
- static void RTC_SetWakeupCount (RTC_Type *base, uint16_t wakeupValue)
  Enable the RTC wake-up timer (1KHZ) and set countdown value to the RTC WAKE register.
- static uint16_t RTC_GetWakeupCount (RTC_Type *base)
  Read the actual value from the WAKE register value in RTC wake-up timer (1KHZ).
- static void RTC_Reset (RTC_Type *base)
  Perform a software reset on the RTC module.

Driver version

- #define FSL_RTC_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))
  Version 2.1.2.

Initialization and deinitialization

- void RTC_Init (RTC_Type *base)
  Un-gate the RTC clock and enable the RTC oscillator.
- static void RTC_Deinit (RTC_Type *base)
  Stop the timer and gate the RTC clock.

Current Time & Alarm

- status_t RTC_SetDatetime (RTC_Type *base, const rtc_datetime_t *datetime)
  Set the RTC date and time according to the given time structure.
- void RTC_GetDatetime (RTC_Type *base, rtc_datetime_t *datetime)
  Get the RTC time and stores it in the given time structure.
- status_t RTC_SetAlarm (RTC_Type *base, const rtc_datetime_t *alarmTime)
  Set the RTC alarm time.
- void RTC_GetAlarm (RTC_Type *base, rtc_datetime_t *datetime)
  Return the RTC alarm time.

RTC wake-up timer (1KHZ) Enable

- static void RTC_EnableWakeupTimer (RTC_Type *base, bool enable)
  Enable the RTC wake-up timer (1KHZ).
- static uint32_t RTC_GetEnabledWakeupTimer (RTC_Type *base)
  Get the enabled status of the RTC wake-up timer (1KHZ).

Interrupt Interface

- static void RTC_EnableWakeUpTimerInterruptFromDPD (RTC_Type *base, bool enable)
  Enable the wake-up timer interrupt from deep power down mode.
- static void RTC_EnableAlarmTimerInterruptFromDPD (RTC_Type *base, bool enable)
  Enable the alarm timer interrupt from deep power down mode.
- static void RTC_EnableInterrupts (RTC_Type *base, uint32_t mask)
  Enables the selected RTC interrupts.
- static void RTC_DisableInterrupts (RTC_Type *base, uint32_t mask)

Disables the selected RTC interrupts.
- static uint32_t RTC_GetEnabledInterrupts (RTC_Type *base)
  Get the enabled RTC interrupts.

Status Interface

- static uint32_t RTC_GetStatusFlags (RTC_Type *base)
  Get the RTC status flags.
- static void RTC_ClearStatusFlags (RTC_Type *base, uint32_t mask)
  Clear the RTC status flags.

Timer Enable

- static void RTC_EnableTimer (RTC_Type *base, bool enable)
  Enable the RTC timer counter.
- static void RTC_StartTimer (RTC_Type *base)
  Starts the RTC time counter.
- static void RTC_StopTimer (RTC_Type *base)
  Stops the RTC time counter.

26.4 Data Structure Documentation

26.4.1 struct rtc_datetime_t

Data Fields

- uint16_t year
  Range from 1970 to 2099.
- uint8_t month
  Range from 1 to 12.
- uint8_t day
  Range from 1 to 31 (depending on month).
- uint8_t hour
  Range from 0 to 23.
- uint8_t minute
  Range from 0 to 59.
- uint8_t second
  Range from 0 to 59.

Field Documentation

(1) uint16_t rtc_datetime_t::year

(2) uint8_t rtc_datetime_t::month

(3) uint8_t rtc_datetime_t::day

(4) uint8_t rtc_datetime_t::hour

(5) uint8_t rtc_datetime_t::minute

(6) uint8_t rtc_datetime_t::second

26.5 Enumeration Type Documentation

26.5.1 enum rtc_interrupt_enable_t

Enumerator

kRTC_AlarmInterruptEnable Alarm interrupt.
kRTC_WakeupInterruptEnable Wake-up interrupt.

26.5.2 enum rtc_status_flags_t

Enumerator

kRTC_AlarmFlag Alarm flag.
kRTC_WakeupFlag 1kHz wake-up timer flag

26.6 Function Documentation

26.6.1 void RTC_Init (RTC_Type * *base*)

Note

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

Parameters

base	RTC peripheral base address

26.6.2 static void RTC_Deinit (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

26.6.3 status_t RTC_SetDatetime (RTC_Type * *base,* const rtc_datetime_t * *datetime*)

The RTC counter must be stopped prior to calling this function as writes to the RTC seconds register will fail if the RTC counter is running.

Parameters

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

Returns

kStatus_Success: Success in setting the time and starting the RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

26.6.4 void RTC_GetDatetime (RTC_Type * *base*, rtc_datetime_t * *datetime*)

Parameters

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

26.6.5 status_t RTC_SetAlarm (RTC_Type * *base*, const rtc_datetime_t * *alarmTime*)

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

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

Returns

kStatus_Success: success in setting the RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

26.6.6 void RTC_GetAlarm (RTC_Type * *base*, rtc_datetime_t * *datetime*)

Parameters

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

26.6.7 static void RTC_EnableWakeupTimer (RTC_Type * *base*, bool *enable*) [inline], [static]

After calling this function, the RTC driver will use/un-use the RTC wake-up (1KHZ) at the same time.

Parameters

base	RTC peripheral base address
enable	Use/Un-use the RTC wake-up timer. • true: Use RTC wake-up timer at the same time. • false: Un-use RTC wake-up timer, RTC only use the normal seconds timer by default.

26.6.8 static uint32_t RTC_GetEnabledWakeupTimer (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

The enabled status of RTC wake-up timer (1KHZ).

26.6.9 static void RTC_SetSecondsTimerMatch (RTC_Type * *base*, uint32_t *matchValue*) [inline], [static]

Parameters

base	RTC peripheral base address
matchValue	The value to be set into the RTC MATCH register

26.6.10 static uint32_t RTC_GetSecondsTimerMatch (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

The actual RTC seconds timer (1HZ) MATCH value.

26.6.11 static void RTC_SetSecondsTimerCount (RTC_Type * *base,* uint32_t *countValue*) [inline], [static]

Parameters

base	RTC peripheral base address
countValue	The value to be loaded into the RTC COUNT register

26.6.12 static uint32_t RTC_GetSecondsTimerCount (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

The actual RTC seconds timer (1HZ) COUNT value.

26.6.13 static void RTC_SetWakeupCount (RTC_Type * *base,* uint16_t *wakeupValue*) [inline], [static]

Parameters

base	RTC peripheral base address
wakeupValue	The value to be loaded into the WAKE register in RTC wake-up timer (1KHZ).

26.6.14 static uint16_t RTC_GetWakeupCount (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

The actual value of the WAKE register value in RTC wake-up timer (1HZ).

26.6.15 static void RTC_EnableWakeUpTimerInterruptFromDPD (RTC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	RTC peripheral base address
enable	Enable/Disable wake-up timer interrupt from deep power down mode. • true: Enable wake-up timer interrupt from deep power down mode. • false: Disable wake-up timer interrupt from deep power down mode.

26.6.16 static void RTC_EnableAlarmTimerInterruptFromDPD (RTC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	RTC peripheral base address

enable	Enable/Disable alarm timer interrupt from deep power down mode. • true: Enable alarm timer interrupt from deep power down mode. • false: Disable alarm timer interrupt from deep power down mode.

26.6.17 static void RTC_EnableInterrupts (RTC_Type * *base*, uint32_t *mask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by RTC_EnableAlarmTimerInterruptFromDPD and RTC_EnableWakeUpTimerInterruptFromDPD

Parameters

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

26.6.18 static void RTC_DisableInterrupts (RTC_Type * *base*, uint32_t *mask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by RTC_EnableAlarmTimerInterruptFromDPD and RTC_EnableWakeUpTimerInterruptFromDPD

Parameters

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

26.6.19 static uint32_t RTC_GetEnabledInterrupts (RTC_Type * *base*) [inline], [static]

Deprecated Do not use this function. It will be deleted in next release version.

Parameters

base	RTC peripheral base address

Returns

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

26.6.20 static uint32_t RTC_GetStatusFlags (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

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

26.6.21 static void RTC_ClearStatusFlags (RTC_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	RTC peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration rtc_status_flags_t

26.6.22 static void RTC_EnableTimer (RTC_Type * *base,* bool *enable*) [inline], [static]

After calling this function, the RTC inner counter increments once a second when only using the RTC seconds timer (1hz), while the RTC inner wake-up timer countdown once a millisecond when using RTC wake-up timer (1KHZ) at the same time. RTC timer contain two timers, one is the RTC normal seconds timer, the other one is the RTC wake-up timer, the RTC enable bit is the master switch for the whole RTC timer, so user can use the RTC seconds (1HZ) timer independly, but they can't use the RTC wake-up timer (1KHZ) independently.

Parameters

base	RTC peripheral base address
enable	Enable/Disable RTC Timer counter. • true: Enable RTC Timer counter. • false: Disable RTC Timer counter.

26.6.23 static void RTC_StartTimer (RTC_Type * *base*) [inline], [static]

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

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

Parameters

base	RTC peripheral base address

26.6.24 static void RTC_StopTimer (RTC_Type * *base*) [inline], [static]

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

RTC's seconds register can be written to only when the timer is stopped.

Parameters

base	RTC peripheral base address

26.6.25 static void RTC_Reset (RTC_Type * *base*) [inline], [static]

This resets all RTC registers to their reset value. The bit is cleared by software explicitly clearing it.

Parameters

base	RTC peripheral base address

Chapter 27
MCAN: Controller Area Network Driver

27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Controller Area Network (MCAN) module of MCUXpresso SDK devices.

Data Structures

- struct mcan_tx_buffer_frame_t
  MCAN Tx Buffer structure. More...
- struct mcan_rx_buffer_frame_t
  MCAN Rx FIFO/Buffer structure. More...
- struct mcan_rx_fifo_config_t
  MCAN Rx FIFO configuration. More...
- struct mcan_rx_buffer_config_t
  MCAN Rx Buffer configuration. More...
- struct mcan_tx_fifo_config_t
  MCAN Tx Event FIFO configuration. More...
- struct mcan_tx_buffer_config_t
  MCAN Tx Buffer configuration. More...
- struct mcan_std_filter_element_config_t
  MCAN Standard Message ID Filter Element. More...
- struct mcan_ext_filter_element_config_t
  MCAN Extended Message ID Filter Element. More...
- struct mcan_frame_filter_config_t
  MCAN Rx filter configuration. More...
- struct mcan_timing_config_t
  MCAN protocol timing characteristic configuration structure. More...
- struct mcan_config_t
  MCAN module configuration structure. More...
- struct mcan_buffer_transfer_t
  MCAN Buffer transfer. More...
- struct mcan_fifo_transfer_t
  MCAN Rx FIFO transfer. More...
- struct mcan_handle_t
  MCAN handle structure. More...

Typedefs

- typedef void(∗ mcan_transfer_callback_t)(CAN_Type ∗base, mcan_handle_t ∗handle, status_t status, uint32_t result, void ∗userData)
  MCAN transfer callback function.

Enumerations

- enum {
  kStatus_MCAN_TxBusy = MAKE_STATUS(kStatusGroup_MCAN, 0),
  kStatus_MCAN_TxIdle = MAKE_STATUS(kStatusGroup_MCAN, 1),
  kStatus_MCAN_RxBusy = MAKE_STATUS(kStatusGroup_MCAN, 2),
  kStatus_MCAN_RxIdle = MAKE_STATUS(kStatusGroup_MCAN, 3),
  kStatus_MCAN_RxFifo0New = MAKE_STATUS(kStatusGroup_MCAN, 4),
  kStatus_MCAN_RxFifo0Idle = MAKE_STATUS(kStatusGroup_MCAN, 5),
  kStatus_MCAN_RxFifo0Watermark = MAKE_STATUS(kStatusGroup_MCAN, 6),
  kStatus_MCAN_RxFifo0Full = MAKE_STATUS(kStatusGroup_MCAN, 7),
  kStatus_MCAN_RxFifo0Lost = MAKE_STATUS(kStatusGroup_MCAN, 8),
  kStatus_MCAN_RxFifo1New = MAKE_STATUS(kStatusGroup_MCAN, 9),
  kStatus_MCAN_RxFifo1Idle = MAKE_STATUS(kStatusGroup_MCAN, 10),
  kStatus_MCAN_RxFifo1Watermark = MAKE_STATUS(kStatusGroup_MCAN, 11),
  kStatus_MCAN_RxFifo1Full = MAKE_STATUS(kStatusGroup_MCAN, 12),
  kStatus_MCAN_RxFifo1Lost = MAKE_STATUS(kStatusGroup_MCAN, 13),
  kStatus_MCAN_RxFifo0Busy = MAKE_STATUS(kStatusGroup_MCAN, 14),
  kStatus_MCAN_RxFifo1Busy = MAKE_STATUS(kStatusGroup_MCAN, 15),
  kStatus_MCAN_ErrorStatus = MAKE_STATUS(kStatusGroup_MCAN, 16),
  kStatus_MCAN_UnHandled = MAKE_STATUS(kStatusGroup_MCAN, 17) }
  MCAN transfer status.
- enum _mcan_flags {
  kMCAN_AccesstoRsvdFlag = CAN_IR_ARA_MASK,
  kMCAN_ProtocolErrDIntFlag = CAN_IR_PED_MASK,
  kMCAN_ProtocolErrAIntFlag = CAN_IR_PEA_MASK,
  kMCAN_BusOffIntFlag = CAN_IR_BO_MASK,
  kMCAN_ErrorWarningIntFlag = CAN_IR_EW_MASK,
  kMCAN_ErrorPassiveIntFlag = CAN_IR_EP_MASK }
  MCAN status flags.
- enum _mcan_rx_fifo_flags {
  kMCAN_RxFifo0NewFlag = CAN_IR_RF0N_MASK,
  kMCAN_RxFifo0WatermarkFlag = CAN_IR_RF0W_MASK,
  kMCAN_RxFifo0FullFlag = CAN_IR_RF0F_MASK,
  kMCAN_RxFifo0LostFlag = CAN_IR_RF0L_MASK,
  kMCAN_RxFifo1NewFlag = CAN_IR_RF1N_MASK,
  kMCAN_RxFifo1WatermarkFlag = CAN_IR_RF1W_MASK,
  kMCAN_RxFifo1FullFlag = CAN_IR_RF1F_MASK,
  kMCAN_RxFifo1LostFlag = CAN_IR_RF1L_MASK }
  MCAN Rx FIFO status flags.
- enum _mcan_tx_flags {

kMCAN_TxTransmitCompleteFlag = CAN_IR_TC_MASK,
kMCAN_TxTransmitCancelFinishFlag = CAN_IR_TCF_MASK,
kMCAN_TxEventFifoLostFlag = CAN_IR_TEFL_MASK,
kMCAN_TxEventFifoFullFlag = CAN_IR_TEFF_MASK,
kMCAN_TxEventFifoWatermarkFlag = CAN_IR_TEFW_MASK,
kMCAN_TxEventFifoNewFlag = CAN_IR_TEFN_MASK,
kMCAN_TxEventFifoEmptyFlag = CAN_IR_TFE_MASK }
MCAN Tx status flags.

- enum _mcan_interrupt_enable {
kMCAN_BusOffInterruptEnable = CAN_IE_BOE_MASK,
kMCAN_ErrorInterruptEnable = CAN_IE_EPE_MASK,
kMCAN_WarningInterruptEnable = CAN_IE_EWE_MASK }
MCAN interrupt configuration structure, default settings all disabled.
- enum mcan_frame_idformat_t {
kMCAN_FrameIDStandard = 0x0U,
kMCAN_FrameIDExtend = 0x1U }
MCAN frame format.
- enum mcan_frame_type_t {
kMCAN_FrameTypeData = 0x0U,
kMCAN_FrameTypeRemote = 0x1U }
MCAN frame type.
- enum mcan_bytes_in_datafield_t {
kMCAN_8ByteDatafield = 0x0U,
kMCAN_12ByteDatafield = 0x1U,
kMCAN_16ByteDatafield = 0x2U,
kMCAN_20ByteDatafield = 0x3U,
kMCAN_24ByteDatafield = 0x4U,
kMCAN_32ByteDatafield = 0x5U,
kMCAN_48ByteDatafield = 0x6U,
kMCAN_64ByteDatafield = 0x7U }
MCAN frame datafield size.
- enum mcan_fifo_type_t {
kMCAN_Fifo0 = 0x0U,
kMCAN_Fifo1 = 0x1U }
MCAN Rx FIFO block number.
- enum mcan_fifo_opmode_config_t {
kMCAN_FifoBlocking = 0x0U,
kMCAN_FifoOverwrite = 0x1U }
MCAN FIFO Operation Mode.
- enum mcan_txmode_config_t {
kMCAN_txFifo = 0x0U,
kMCAN_txQueue = 0x1U }
MCAN Tx FIFO/Queue Mode.
- enum mcan_remote_frame_config_t {
kMCAN_filterFrame = 0x0U,
kMCAN_rejectFrame = 0x1U }
MCAN remote frames treatment.

- enum mcan_nonmasking_frame_config_t {
  kMCAN_acceptinFifo0 = 0x0U,
  kMCAN_acceptinFifo1 = 0x1U,
  kMCAN_reject0 = 0x2U,
  kMCAN_reject1 = 0x3U }
  MCAN non-masking frames treatment.
- enum mcan_fec_config_t {
  kMCAN_disable = 0x0U,
  kMCAN_storeinFifo0 = 0x1U,
  kMCAN_storeinFifo1 = 0x2U,
  kMCAN_reject = 0x3U,
  kMCAN_setprio = 0x4U,
  kMCAN_setpriofifo0 = 0x5U,
  kMCAN_setpriofifo1 = 0x6U,
  kMCAN_storeinbuffer = 0x7U }
  MCAN Filter Element Configuration.
- enum mcan_filter_type_t {
  kMCAN_range = 0x0U,
  kMCAN_dual = 0x1U,
  kMCAN_classic = 0x2U,
  kMCAN_disableORrange2 = 0x3U }
  MCAN Filter Type.

Driver version

- #define FSL_MCAN_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))
  MCAN driver version.

Initialization and deinitialization

- void MCAN_Init (CAN_Type *base, const mcan_config_t *config, uint32_t sourceClock_Hz)
  Initializes an MCAN instance.
- void MCAN_Deinit (CAN_Type *base)
  Deinitializes an MCAN instance.
- void MCAN_GetDefaultConfig (mcan_config_t *config)
  Gets the default configuration structure.
- void MCAN_EnterNormalMode (CAN_Type *base)
  MCAN enters normal mode.

Configuration.

- static void MCAN_SetMsgRAMBase (CAN_Type *base, uint32_t value)
  Sets the MCAN Message RAM base address.
- static uint32_t MCAN_GetMsgRAMBase (CAN_Type *base)
  Gets the MCAN Message RAM base address.
- bool MCAN_CalculateImprovedTimingValues (uint32_t baudRate, uint32_t sourceClock_Hz, mcan_timing_config_t *pconfig)
  Calculates the improved timing values by specific baudrates for classical CAN.

- void MCAN_SetArbitrationTimingConfig (CAN_Type *base, const mcan_timing_config_t *config)
  Sets the MCAN protocol arbitration phase timing characteristic.
- bool MCAN_FDCalculateImprovedTimingValues (uint32_t baudRate, uint32_t baudRateFD, uint32_t sourceClock_Hz, mcan_timing_config_t *pconfig)
  Calculates the improved timing values by specific baudrates for CANFD.
- void MCAN_SetDataTimingConfig (CAN_Type *base, const mcan_timing_config_t *config)
  Sets the MCAN protocol data phase timing characteristic.
- void MCAN_SetRxFifo0Config (CAN_Type *base, const mcan_rx_fifo_config_t *config)
  Configures an MCAN receive fifo 0 buffer.
- void MCAN_SetRxFifo1Config (CAN_Type *base, const mcan_rx_fifo_config_t *config)
  Configures an MCAN receive fifo 1 buffer.
- void MCAN_SetRxBufferConfig (CAN_Type *base, const mcan_rx_buffer_config_t *config)
  Configures an MCAN receive buffer.
- void MCAN_SetTxEventFifoConfig (CAN_Type *base, const mcan_tx_fifo_config_t *config)
  Configures an MCAN transmit event fifo.
- void MCAN_SetTxBufferConfig (CAN_Type *base, const mcan_tx_buffer_config_t *config)
  Configures an MCAN transmit buffer.
- void MCAN_SetFilterConfig (CAN_Type *base, const mcan_frame_filter_config_t *config)
  Set filter configuration.
- void MCAN_SetSTDFilterElement (CAN_Type *base, const mcan_frame_filter_config_t *config, const mcan_std_filter_element_config_t *filter, uint8_t idx)
  Set standard message ID filter element configuration.
- void MCAN_SetEXTFilterElement (CAN_Type *base, const mcan_frame_filter_config_t *config, const mcan_ext_filter_element_config_t *filter, uint8_t idx)
  Set extended message ID filter element configuration.

Status

- static uint32_t MCAN_GetStatusFlag (CAN_Type *base, uint32_t mask)
  Gets the MCAN module interrupt flags.
- static void MCAN_ClearStatusFlag (CAN_Type *base, uint32_t mask)
  Clears the MCAN module interrupt flags.
- static bool MCAN_GetRxBufferStatusFlag (CAN_Type *base, uint8_t idx)
  Gets the new data flag of specific Rx Buffer.
- static void MCAN_ClearRxBufferStatusFlag (CAN_Type *base, uint8_t idx)
  Clears the new data flag of specific Rx Buffer.

Interrupts

- static void MCAN_EnableInterrupts (CAN_Type *base, uint32_t line, uint32_t mask)
  Enables MCAN interrupts according to the provided interrupt line and mask.
- static void MCAN_EnableTransmitBufferInterrupts (CAN_Type *base, uint8_t idx)
  Enables MCAN Tx Buffer interrupts according to the provided index.
- static void MCAN_DisableTransmitBufferInterrupts (CAN_Type *base, uint8_t idx)
  Disables MCAN Tx Buffer interrupts according to the provided index.
- static void MCAN_DisableInterrupts (CAN_Type *base, uint32_t mask)
  Disables MCAN interrupts according to the provided mask.

Bus Operations

- uint32_t MCAN_IsTransmitRequestPending (CAN_Type *base, uint8_t idx)
  Gets the Tx buffer request pending status.
- uint32_t MCAN_IsTransmitOccurred (CAN_Type *base, uint8_t idx)
  Gets the Tx buffer transmission occurred status.
- status_t MCAN_WriteTxBuffer (CAN_Type *base, uint8_t idx, const mcan_tx_buffer_frame_t *pTxFrame)
  Writes an MCAN Message to the Transmit Buffer.
- status_t MCAN_ReadRxBuffer (CAN_Type *base, uint8_t idx, mcan_rx_buffer_frame_t *pRxFrame)
  Reads an MCAN Message from Rx Buffer.
- status_t MCAN_ReadRxFifo (CAN_Type *base, uint8_t fifoBlock, mcan_rx_buffer_frame_t *pRxFrame)
  Reads an MCAN Message from Rx FIFO.

Transactional

- static void MCAN_TransmitAddRequest (CAN_Type *base, uint8_t idx)
  Tx Buffer add request to send message out.
- static void MCAN_TransmitCancelRequest (CAN_Type *base, uint8_t idx)
  Tx Buffer cancel sending request.
- status_t MCAN_TransferSendBlocking (CAN_Type *base, uint8_t idx, mcan_tx_buffer_frame_t *pTxFrame)
  Performs a polling send transaction on the CAN bus.
- status_t MCAN_TransferReceiveBlocking (CAN_Type *base, uint8_t idx, mcan_rx_buffer_frame_t *pRxFrame)
  Performs a polling receive transaction on the CAN bus.
- status_t MCAN_TransferReceiveFifoBlocking (CAN_Type *base, uint8_t fifoBlock, mcan_rx_buffer_frame_t *pRxFrame)
  Performs a polling receive transaction from Rx FIFO on the CAN bus.
- void MCAN_TransferCreateHandle (CAN_Type *base, mcan_handle_t *handle, mcan_transfer_callback_t callback, void *userData)
  Initializes the MCAN handle.
- status_t MCAN_TransferSendNonBlocking (CAN_Type *base, mcan_handle_t *handle, mcan_buffer_transfer_t *xfer)
  Sends a message using IRQ.
- status_t MCAN_TransferReceiveFifoNonBlocking (CAN_Type *base, uint8_t fifoBlock, mcan_handle_t *handle, mcan_fifo_transfer_t *xfer)
  Receives a message from Rx FIFO using IRQ.
- void MCAN_TransferAbortSend (CAN_Type *base, mcan_handle_t *handle, uint8_t bufferIdx)
  Aborts the interrupt driven message send process.
- void MCAN_TransferAbortReceiveFifo (CAN_Type *base, uint8_t fifoBlock, mcan_handle_t *handle)
  Aborts the interrupt driven message receive from Rx FIFO process.
- void MCAN_TransferHandleIRQ (CAN_Type *base, mcan_handle_t *handle)
  MCAN IRQ handle function.

27.2 Data Structure Documentation

27.2.1 struct mcan_tx_buffer_frame_t

Data Fields

- uint8_t size
  classical CAN is 8(bytes), FD is 12/64 such.
- uint32_t id: 29
  CAN Frame Identifier.
- uint32_t rtr: 1
  CAN Frame Type(DATA or REMOTE).
- uint32_t xtd: 1
  CAN Frame Type(STD or EXT).
- uint32_t esi: 1
  CAN Frame Error State Indicator.
- uint32_t dlc: 4
  Data Length Code 9 10 11 12 13 14 15 Number of data bytes 12 16 20 24 32 48 64.
- uint32_t brs: 1
  Bit Rate Switch.
- uint32_t fdf: 1
  CAN FD format.
- uint32_t __pad1__: 1
  Reserved.
- uint32_t efc: 1
  Event FIFO control.
- uint32_t mm: 8
  Message Marker.

Field Documentation

(1) uint32_t mcan_tx_buffer_frame_t::id

(2) uint32_t mcan_tx_buffer_frame_t::rtr

(3) uint32_t mcan_tx_buffer_frame_t::xtd

(4) uint32_t mcan_tx_buffer_frame_t::esi

(5) uint32_t mcan_tx_buffer_frame_t::brs

(6) uint32_t mcan_tx_buffer_frame_t::fdf

(7) uint32_t mcan_tx_buffer_frame_t::__pad1__

(8) uint32_t mcan_tx_buffer_frame_t::efc

(9) uint32_t mcan_tx_buffer_frame_t::mm

(10) uint8_t mcan_tx_buffer_frame_t::size

27.2.2 struct mcan_rx_buffer_frame_t

Data Fields

- uint8_t size
  classical CAN is 8(bytes), FD is 12/64 such.
- uint32_t id: 29
  CAN Frame Identifier.
- uint32_t rtr: 1
  CAN Frame Type(DATA or REMOTE).
- uint32_t xtd: 1
  CAN Frame Type(STD or EXT).
- uint32_t esi: 1
  CAN Frame Error State Indicator.
- uint32_t rxts: 16
  Rx Timestamp.
- uint32_t dlc: 4
  Data Length Code 9 10 11 12 13 14 15 Number of data bytes 12 16 20 24 32 48 64.
- uint32_t brs: 1
  Bit Rate Switch.
- uint32_t fdf: 1
  CAN FD format.
- uint32_t __pad0__: 2
  Reserved.
- uint32_t fidx: 7
  Filter Index.
- uint32_t anmf: 1
  Accepted Non-matching Frame.

Field Documentation

(1) uint32_t mcan_rx_buffer_frame_t::id

(2) uint32_t mcan_rx_buffer_frame_t::rtr

(3) uint32_t mcan_rx_buffer_frame_t::xtd

(4) uint32_t mcan_rx_buffer_frame_t::esi

(5) uint32_t mcan_rx_buffer_frame_t::rxts

(6) uint32_t mcan_rx_buffer_frame_t::brs

(7) uint32_t mcan_rx_buffer_frame_t::fdf

(8) uint32_t mcan_rx_buffer_frame_t::__pad0__

(9) uint32_t mcan_rx_buffer_frame_t::fidx

(10) uint32_t mcan_rx_buffer_frame_t::anmf

(11) uint8_t mcan_rx_buffer_frame_t::size

27.2.3 struct mcan_rx_fifo_config_t

Data Fields

- uint32_t address
  FIFOn start address.
- uint32_t elementSize
  FIFOn element number.
- uint32_t watermark
  FIFOn watermark level.
- mcan_fifo_opmode_config_t opmode
  FIFOn blocking/overwrite mode.
- mcan_bytes_in_datafield_t datafieldSize
  Data field size per frame, size>8 is for CANFD.

Field Documentation

(1) uint32_t mcan_rx_fifo_config_t::address

(2) uint32_t mcan_rx_fifo_config_t::elementSize

(3) uint32_t mcan_rx_fifo_config_t::watermark

(4) mcan_fifo_opmode_config_t mcan_rx_fifo_config_t::opmode

(5) mcan_bytes_in_datafield_t mcan_rx_fifo_config_t::datafieldSize

27.2.4 struct mcan_rx_buffer_config_t

Data Fields

- uint32_t address
  Rx Buffer start address.
- mcan_bytes_in_datafield_t datafieldSize
  Data field size per frame, size>8 is for CANFD.

Field Documentation

(1) uint32_t mcan_rx_buffer_config_t::address

(2) mcan_bytes_in_datafield_t mcan_rx_buffer_config_t::datafieldSize

27.2.5 struct mcan_tx_fifo_config_t

Data Fields

- uint32_t address
  Event fifo start address.
- uint32_t elementSize
  FIFOn element number.
- uint32_t watermark
  FIFOn watermark level.

Field Documentation

(1) uint32_t mcan_tx_fifo_config_t::address

(2) uint32_t mcan_tx_fifo_config_t::elementSize

(3) uint32_t mcan_tx_fifo_config_t::watermark

27.2.6 struct mcan_tx_buffer_config_t

Data Fields

- uint32_t address
  Tx Buffers Start Address.
- uint32_t dedicatedSize
  Number of Dedicated Transmit Buffers.
- uint32_t fqSize
  Transmit FIFO/Queue Size.
- mcan_txmode_config_t mode
  Tx FIFO/Queue Mode.
- mcan_bytes_in_datafield_t datafieldSize
  Data field size per frame, size>8 is for CANFD.

Field Documentation

(1) uint32_t mcan_tx_buffer_config_t::address

(2) uint32_t mcan_tx_buffer_config_t::dedicatedSize

(3) uint32_t mcan_tx_buffer_config_t::fqSize

(4) mcan_txmode_config_t mcan_tx_buffer_config_t::mode

(5) mcan_bytes_in_datafield_t mcan_tx_buffer_config_t::datafieldSize

27.2.7 struct mcan_std_filter_element_config_t

Data Fields

- uint32_t sfid2: 11
  Standard Filter ID 2.
- uint32_t __pad0__: 5
  Reserved.
- uint32_t sfid1: 11
  Standard Filter ID 1.
- uint32_t sfec: 3
  Standard Filter Element Configuration.
- uint32_t sft: 2
  Standard Filter Type.

Field Documentation

(1) uint32_t mcan_std_filter_element_config_t::sfid2

(2) uint32_t mcan_std_filter_element_config_t::__pad0__

(3) uint32_t mcan_std_filter_element_config_t::sfid1

(4) uint32_t mcan_std_filter_element_config_t::sfec

(5) uint32_t mcan_std_filter_element_config_t::sft

27.2.8 struct mcan_ext_filter_element_config_t

Data Fields

- uint32_t efid1: 29
  Extended Filter ID 1.
- uint32_t efec: 3
  Extended Filter Element Configuration.
- uint32_t efid2: 29
  Extended Filter ID 2.
- uint32_t __pad0__: 1
  Reserved.
- uint32_t eft: 2
  Extended Filter Type.

Field Documentation

(1) uint32_t mcan_ext_filter_element_config_t::efid1

(2) uint32_t mcan_ext_filter_element_config_t::efec

(3) uint32_t mcan_ext_filter_element_config_t::efid2

(4) uint32_t mcan_ext_filter_element_config_t::__pad0__

(5) uint32_t mcan_ext_filter_element_config_t::eft

27.2.9 struct mcan_frame_filter_config_t

Data Fields

- uint32_t address
  Filter start address.
- uint32_t listSize
  Filter list size.
- mcan_frame_idformat_t idFormat
  Frame format.
- mcan_remote_frame_config_t remFrame
  Remote frame treatment.
- mcan_nonmasking_frame_config_t nmFrame
  Non-masking frame treatment.

Field Documentation

(1) uint32_t mcan_frame_filter_config_t::address

(2) uint32_t mcan_frame_filter_config_t::listSize

(3) mcan_frame_idformat_t mcan_frame_filter_config_t::idFormat

(4) mcan_remote_frame_config_t mcan_frame_filter_config_t::remFrame

(5) mcan_nonmasking_frame_config_t mcan_frame_filter_config_t::nmFrame

27.2.10 struct mcan_timing_config_t

Data Fields

- uint16_t preDivider
  Nominal Clock Pre-scaler Division Factor.
- uint8_t rJumpwidth
  Nominal Re-sync Jump Width.
- uint8_t seg1
  Nominal Time Segment 1.
- uint8_t seg2
  Nominal Time Segment 2.
- uint16_t datapreDivider
  Data Clock Pre-scaler Division Factor.
- uint8_t datarJumpwidth
  Data Re-sync Jump Width.
- uint8_t dataseg1
  Data Time Segment 1.

- uint8_t dataseg2
  Data Time Segment 2.

Field Documentation

(1) uint16_t mcan_timing_config_t::preDivider

(2) uint8_t mcan_timing_config_t::rJumpwidth

(3) uint8_t mcan_timing_config_t::seg1

(4) uint8_t mcan_timing_config_t::seg2

(5) uint16_t mcan_timing_config_t::datapreDivider

(6) uint8_t mcan_timing_config_t::datarJumpwidth

(7) uint8_t mcan_timing_config_t::dataseg1

(8) uint8_t mcan_timing_config_t::dataseg2

27.2.11 struct mcan_config_t

Data Fields

- uint32_t baudRateA
  Baud rate of Arbitration phase in bps.
- uint32_t baudRateD
  Baud rate of Data phase in bps.
- bool enableCanfdNormal
  Enable or Disable CANFD normal.
- bool enableCanfdSwitch
  Enable or Disable CANFD with baudrate switch.
- bool enableLoopBackInt
  Enable or Disable Internal Back.
- bool enableLoopBackExt
  Enable or Disable External Loop Back.
- bool enableBusMon
  Enable or Disable Bus Monitoring Mode.
- mcan_timing_config_t timingConfig
  Protocol timing .

Field Documentation

(1) uint32_t mcan_config_t::baudRateA

(2) uint32_t mcan_config_t::baudRateD

(3) bool mcan_config_t::enableCanfdNormal

(4) bool mcan_config_t::enableCanfdSwitch

(5) bool mcan_config_t::enableLoopBackInt

(6) bool mcan_config_t::enableLoopBackExt

(7) bool mcan_config_t::enableBusMon

(8) mcan_timing_config_t mcan_config_t::timingConfig

27.2.12 struct mcan_buffer_transfer_t

Data Fields

- mcan_tx_buffer_frame_t * frame
  The buffer of CAN Message to be transfer.
- uint8_t bufferIdx
  The index of Message buffer used to transfer Message.

Field Documentation

(1) mcan_tx_buffer_frame_t* mcan_buffer_transfer_t::frame

(2) uint8_t mcan_buffer_transfer_t::bufferIdx

27.2.13 struct mcan_fifo_transfer_t

Data Fields

- mcan_rx_buffer_frame_t * frame
  The buffer of CAN Message to be received from Rx FIFO.

Field Documentation

(1) mcan_rx_buffer_frame_t* mcan_fifo_transfer_t::frame

27.2.14 struct _mcan_handle

MCAN handle structure definition.

Data Fields

- mcan_transfer_callback_t callback
  Callback function.
- void * userData
  MCAN callback function parameter.
- mcan_tx_buffer_frame_t *volatile bufferFrameBuf [64]

The buffer for received data from Buffers.
- mcan_rx_buffer_frame_t ∗volatile rxFifoFrameBuf
  The buffer for received data from Rx FIFO.
- volatile uint8_t txbufferIdx
  Message Buffer transfer state.
- volatile uint8_t bufferState [64]
  Message Buffer transfer state.
- volatile uint8_t rxFifoState
  Rx FIFO transfer state.

Field Documentation

(1) mcan_transfer_callback_t mcan_handle_t::callback

(2) void∗ mcan_handle_t::userData

(3) mcan_tx_buffer_frame_t∗ volatile mcan_handle_t::bufferFrameBuf[64]

(4) mcan_rx_buffer_frame_t∗ volatile mcan_handle_t::rxFifoFrameBuf

(5) volatile uint8_t mcan_handle_t::txbufferIdx

(6) volatile uint8_t mcan_handle_t::bufferState[64]

(7) volatile uint8_t mcan_handle_t::rxFifoState

27.3 Macro Definition Documentation

27.3.1 #define FSL_MCAN_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))

27.4 Typedef Documentation

27.4.1 typedef void(∗ mcan_transfer_callback_t)(CAN_Type ∗base, mcan_handle_t ∗handle, status_t status, uint32_t result, void ∗userData)

The MCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus_MCAN_ErrorStatus, the result parameter is the Content of MCAN status register which can be used to get the working status(or error status) of MCAN module. If the status equals to other MCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other MCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

27.5 Enumeration Type Documentation

27.5.1 anonymous enum

Enumerator

kStatus_MCAN_TxBusy Tx Buffer is Busy.
kStatus_MCAN_TxIdle Tx Buffer is Idle.

kStatus_MCAN_RxBusy Rx Buffer is Busy.
kStatus_MCAN_RxIdle Rx Buffer is Idle.
kStatus_MCAN_RxFifo0New New message written to Rx FIFO 0.
kStatus_MCAN_RxFifo0Idle Rx FIFO 0 is Idle.
kStatus_MCAN_RxFifo0Watermark Rx FIFO 0 fill level reached watermark.
kStatus_MCAN_RxFifo0Full Rx FIFO 0 full.
kStatus_MCAN_RxFifo0Lost Rx FIFO 0 message lost.
kStatus_MCAN_RxFifo1New New message written to Rx FIFO 1.
kStatus_MCAN_RxFifo1Idle Rx FIFO 1 is Idle.
kStatus_MCAN_RxFifo1Watermark Rx FIFO 1 fill level reached watermark.
kStatus_MCAN_RxFifo1Full Rx FIFO 1 full.
kStatus_MCAN_RxFifo1Lost Rx FIFO 1 message lost.
kStatus_MCAN_RxFifo0Busy Rx FIFO 0 is busy.
kStatus_MCAN_RxFifo1Busy Rx FIFO 1 is busy.
kStatus_MCAN_ErrorStatus MCAN Module Error and Status.
kStatus_MCAN_UnHandled UnHadled Interrupt asserted.

27.5.2 enum _mcan_flags

This provides constants for the MCAN status flags for use in the MCAN functions. Note: The CPU read action clears MCAN_ErrorFlag, therefore user need to read MCAN_ErrorFlag and distinguish which error is occur using _mcan_error_flags enumerations.

Enumerator

kMCAN_AccesstoRsvdFlag CAN Synchronization Status.
kMCAN_ProtocolErrDIntFlag Tx Warning Interrupt Flag.
kMCAN_ProtocolErrAIntFlag Rx Warning Interrupt Flag.
kMCAN_BusOffIntFlag Tx Error Warning Status.
kMCAN_ErrorWarningIntFlag Rx Error Warning Status.
kMCAN_ErrorPassiveIntFlag Rx Error Warning Status.

27.5.3 enum _mcan_rx_fifo_flags

The MCAN Rx FIFO Status enumerations are used to determine the status of the Rx FIFO.

Enumerator

kMCAN_RxFifo0NewFlag Rx FIFO 0 new message flag.
kMCAN_RxFifo0WatermarkFlag Rx FIFO 0 watermark reached flag.
kMCAN_RxFifo0FullFlag Rx FIFO 0 full flag.
kMCAN_RxFifo0LostFlag Rx FIFO 0 message lost flag.
kMCAN_RxFifo1NewFlag Rx FIFO 0 new message flag.

kMCAN_RxFifo1WatermarkFlag Rx FIFO 0 watermark reached flag.
kMCAN_RxFifo1FullFlag Rx FIFO 0 full flag.
kMCAN_RxFifo1LostFlag Rx FIFO 0 message lost flag.

27.5.4 enum _mcan_tx_flags

The MCAN Tx Status enumerations are used to determine the status of the Tx Buffer/Event FIFO.

Enumerator

kMCAN_TxTransmitCompleteFlag Transmission completed flag.
kMCAN_TxTransmitCancelFinishFlag Transmission cancellation finished flag.
kMCAN_TxEventFifoLostFlag Tx Event FIFO element lost.
kMCAN_TxEventFifoFullFlag Tx Event FIFO full.
kMCAN_TxEventFifoWatermarkFlag Tx Event FIFO fill level reached watermark.
kMCAN_TxEventFifoNewFlag Tx Handler wrote Tx Event FIFO element flag.
kMCAN_TxEventFifoEmptyFlag Tx FIFO empty flag.

27.5.5 enum _mcan_interrupt_enable

This structure contains the settings for all of the MCAN Module interrupt configurations.

Enumerator

kMCAN_BusOffInterruptEnable Bus Off interrupt.
kMCAN_ErrorInterruptEnable Error interrupt.
kMCAN_WarningInterruptEnable Rx Warning interrupt.

27.5.6 enum mcan_frame_idformat_t

Enumerator

kMCAN_FrameIDStandard Standard frame format attribute.
kMCAN_FrameIDExtend Extend frame format attribute.

27.5.7 enum mcan_frame_type_t

Enumerator

kMCAN_FrameTypeData Data frame type attribute.
kMCAN_FrameTypeRemote Remote frame type attribute.

27.5.8 enum mcan_bytes_in_datafield_t

Enumerator

kMCAN_8ByteDatafield 8 byte data field.
kMCAN_12ByteDatafield 12 byte data field.
kMCAN_16ByteDatafield 16 byte data field.
kMCAN_20ByteDatafield 20 byte data field.
kMCAN_24ByteDatafield 24 byte data field.
kMCAN_32ByteDatafield 32 byte data field.
kMCAN_48ByteDatafield 48 byte data field.
kMCAN_64ByteDatafield 64 byte data field.

27.5.9 enum mcan_fifo_type_t

Enumerator

kMCAN_Fifo0 CAN Rx FIFO 0.
kMCAN_Fifo1 CAN Rx FIFO 1.

27.5.10 enum mcan_fifo_opmode_config_t

Enumerator

kMCAN_FifoBlocking FIFO blocking mode.
kMCAN_FifoOverwrite FIFO overwrite mode.

27.5.11 enum mcan_txmode_config_t

Enumerator

kMCAN_txFifo Tx FIFO operation.
kMCAN_txQueue Tx Queue operation.

27.5.12 enum mcan_remote_frame_config_t

Enumerator

kMCAN_filterFrame Filter remote frames.
kMCAN_rejectFrame Reject all remote frames.

27.5.13 enum mcan_nonmasking_frame_config_t

Enumerator

kMCAN_acceptinFifo0 Accept non-masking frames in Rx FIFO 0.
kMCAN_acceptinFifo1 Accept non-masking frames in Rx FIFO 1.
kMCAN_reject0 Reject non-masking frames.
kMCAN_reject1 Reject non-masking frames.

27.5.14 enum mcan_fec_config_t

Enumerator

kMCAN_disable Disable filter element.
kMCAN_storeinFifo0 Store in Rx FIFO 0 if filter matches.
kMCAN_storeinFifo1 Store in Rx FIFO 1 if filter matches.
kMCAN_reject Reject ID if filter matches.
kMCAN_setprio Set priority if filter matches.
kMCAN_setpriofifo0 Set priority and store in FIFO 0 if filter matches.
kMCAN_setpriofifo1 Set priority and store in FIFO 1 if filter matches.
kMCAN_storeinbuffer Store into Rx Buffer or as debug message.

27.5.15 enum mcan_filter_type_t

Enumerator

kMCAN_range Range filter from SFID1 to SFID2.
kMCAN_dual Dual ID filter for SFID1 or SFID2.
kMCAN_classic Classic filter: SFID1 = filter, SFID2 = mask.
kMCAN_disableORrange2 Filter element disabled for standard filter or Range filter, XIDAM mask not applied for extended filter.

27.6 Function Documentation

27.6.1 void MCAN_Init (CAN_Type * *base*, const mcan_config_t * *config*, uint32_t *sourceClock_Hz*)

This function initializes the MCAN module with user-defined settings. This example shows how to set up the mcan_config_t parameters and how to call the MCAN_Init function by passing in these parameters.

```
*   mcan_config_t config;
*   config->baudRateA = 500000U;
*   config->baudRateD = 1000000U;
*   config->enableCanfdNormal = false;
```

```
*   config->enableCanfdSwitch = false;
*   config->enableLoopBackInt = false;
*   config->enableLoopBackExt = false;
*   config->enableBusMon = false;
*   MCAN_Init(CANFD0, &config, 8000000UL);
*
```

Parameters

base	MCAN peripheral base address.
config	Pointer to the user-defined configuration structure.
sourceClock_-Hz	MCAN Protocol Engine clock source frequency in Hz.

27.6.2 void MCAN_Deinit (CAN_Type * *base*)

This function deinitializes the MCAN module.

Parameters

base	MCAN peripheral base address.

27.6.3 void MCAN_GetDefaultConfig (mcan_config_t * *config*)

This function initializes the MCAN configuration structure to default values. The default values are as follows. config->baudRateA = 500000U; config->baudRateD = 1000000U; config->enableCanfd-Normal = false; config->enableCanfdSwitch = false; config->enableLoopBackInt = false; config->enableLoopBackExt = false; config->enableBusMon = false;

Parameters

config	Pointer to the MCAN configuration structure.

27.6.4 void MCAN_EnterNormalMode (CAN_Type * *base*)

After initialization, INIT bit in CCCR register must be cleared to enter normal mode thus synchronizes to the CAN bus and ready for communication.

Parameters

base	MCAN peripheral base address.

27.6.5 static void MCAN_SetMsgRAMBase (CAN_Type * *base,* uint32_t *value*) [inline], [static]

This function sets the Message RAM base address.

Parameters

base	MCAN peripheral base address.
value	Desired Message RAM base.

27.6.6 static uint32_t MCAN_GetMsgRAMBase (CAN_Type * *base*) [inline], [static]

This function gets the Message RAM base address.

Parameters

base	MCAN peripheral base address.

Returns

Message RAM base address.

27.6.7 bool MCAN_CalculateImprovedTimingValues (uint32_t *baudRate,* uint32_t *sourceClock_Hz,* mcan_timing_config_t * *pconfig*)

Parameters

baudRate	The classical CAN speed in bps defined by user
sourceClock_Hz	The Source clock data speed in bps. Zero to disable baudrate switching

pconfig	Pointer to the MCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

27.6.8 void MCAN_SetArbitrationTimingConfig (CAN_Type * *base*, const mcan_timing_config_t * *config*)

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the MCAN_Init() and fill the baud rate field with a desired value. This provides the default arbitration phase timing characteristics.

Note that calling MCAN_SetArbitrationTimingConfig() overrides the baud rate set in MCAN_Init().

Parameters

base	MCAN peripheral base address.
config	Pointer to the timing configuration structure.

27.6.9 bool MCAN_FDCalculateImprovedTimingValues (uint32_t *baudRate*, uint32_t *baudRateFD*, uint32_t *sourceClock_Hz*, mcan_timing_config_t * *pconfig*)

Parameters

baudRate	The CANFD bus control speed in bps defined by user
baudRateFD	The CANFD bus data speed in bps defined by user
sourceClock_Hz	The Source clock data speed in bps.
pconfig	Pointer to the MCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

27.6.10 void MCAN_SetDataTimingConfig (CAN_Type * *base*, const mcan_timing_config_t * *config*)

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the MCAN_Init() and fill the baud rate field with a desired value. This provides the default data phase timing characteristics.

Note that calling MCAN_SetArbitrationTimingConfig() overrides the baud rate set in MCAN_Init().

Parameters

base	MCAN peripheral base address.
config	Pointer to the timing configuration structure.

27.6.11 void MCAN_SetRxFifo0Config (CAN_Type * *base*, const mcan_rx_fifo_config_t * *config*)

This function sets start address, element size, watermark, operation mode and datafield size of the recieve fifo 0.

Parameters

base	MCAN peripheral base address.
config	The receive fifo 0 configuration structure.

27.6.12 void MCAN_SetRxFifo1Config (CAN_Type * *base*, const mcan_rx_fifo_config_t * *config*)

This function sets start address, element size, watermark, operation mode and datafield size of the recieve fifo 1.

Parameters

base	MCAN peripheral base address.
config	The receive fifo 1 configuration structure.

27.6.13 void MCAN_SetRxBufferConfig (CAN_Type * *base*, const mcan_rx_buffer_config_t * *config*)

This function sets start address and datafield size of the recieve buffer.

Parameters

base	MCAN peripheral base address.
config	The receive buffer configuration structure.

27.6.14 void MCAN_SetTxEventFifoConfig (CAN_Type * *base,* const mcan_tx_fifo_config_t * *config*)

This function sets start address, element size, watermark of the transmit event fifo.

Parameters

base	MCAN peripheral base address.
config	The transmit event fifo configuration structure.

27.6.15 void MCAN_SetTxBufferConfig (CAN_Type * *base,* const mcan_tx_buffer_config_t * *config*)

This function sets start address, element size, fifo/queue mode and datafield size of the transmit buffer.

Parameters

base	MCAN peripheral base address.
config	The transmit buffer configuration structure.

27.6.16 void MCAN_SetFilterConfig (CAN_Type * *base,* const mcan_frame_filter_config_t * *config*)

This function sets remote and non masking frames in global filter configuration, also the start address, list size in standard/extended ID filter configuration.

Parameters

base	MCAN peripheral base address.

config	The MCAN filter configuration.

27.6.17 void MCAN_SetSTDFilterElement (CAN_Type * *base*, const mcan_frame_filter_config_t * *config*, const mcan_std_filter_element_config_t * *filter*, uint8_t *idx*)

Parameters

base	MCAN peripheral base address.
config	The MCAN filter configuration.
filter	The MCAN standard message ID filter element configuration.
idx	The standard message ID filter element index.

27.6.18 void MCAN_SetEXTFilterElement (CAN_Type * *base*, const mcan_frame_filter_config_t * *config*, const mcan_ext_filter_element_config_t * *filter*, uint8_t *idx*)

Parameters

base	MCAN peripheral base address.
config	The MCAN filter configuration.
filter	The MCAN extended message ID filter element configuration.
idx	The extended message ID filter element index.

27.6.19 static uint32_t MCAN_GetStatusFlag (CAN_Type * *base*, uint32_t *mask*) [inline], [static]

This function gets all MCAN interrupt status flags.

Parameters

base	MCAN peripheral base address.
mask	The ORed MCAN interrupt mask.

Returns

MCAN status flags which are ORed.

27.6.20 static void MCAN_ClearStatusFlag (CAN_Type * *base,* uint32_t *mask*) [inline], [static]

This function clears MCAN interrupt status flags.

Parameters

base	MCAN peripheral base address.
mask	The ORed MCAN interrupt mask.

27.6.21 static bool MCAN_GetRxBufferStatusFlag (CAN_Type * *base,* uint8_t *idx*) [inline], [static]

This function gets new data flag of specific Rx Buffer.

Parameters

base	MCAN peripheral base address.
idx	Rx Buffer index.

Returns

Rx Buffer new data status flag.

27.6.22 static void MCAN_ClearRxBufferStatusFlag (CAN_Type * *base,* uint8_t *idx*) [inline], [static]

This function clears new data flag of specific Rx Buffer.

Parameters

base	MCAN peripheral base address.
idx	Rx Buffer index.

27.6.23 static void MCAN_EnableInterrupts (CAN_Type * *base*, uint32_t *line*, uint32_t *mask*) [inline], [static]

This function enables the MCAN interrupts according to the provided interrupt line and mask. The mask is a logical OR of enumeration members.

Parameters

base	MCAN peripheral base address.
line	Interrupt line number, 0 or 1.
mask	The interrupts to enable.

27.6.24 static void MCAN_EnableTransmitBufferInterrupts (CAN_Type * *base*, uint8_t *idx*) [inline], [static]

This function enables the MCAN Tx Buffer interrupts.

Parameters

base	MCAN peripheral base address.
idx	Tx Buffer index.

27.6.25 static void MCAN_DisableTransmitBufferInterrupts (CAN_Type * *base*, uint8_t *idx*) [inline], [static]

This function disables the MCAN Tx Buffer interrupts.

Parameters

base	MCAN peripheral base address.
idx	Tx Buffer index.

27.6.26 static void MCAN_DisableInterrupts (CAN_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the MCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members.

Parameters

base	MCAN peripheral base address.
mask	The interrupts to disable.

27.6.27 uint32_t MCAN_IsTransmitRequestPending (CAN_Type * *base*, uint8_t *idx*)

This function returns Tx Message Buffer transmission request pending status.

Parameters

base	MCAN peripheral base address.
idx	The MCAN Tx Buffer index.

27.6.28 uint32_t MCAN_IsTransmitOccurred (CAN_Type * *base*, uint8_t *idx*)

This function returns Tx Message Buffer transmission occurred status.

Parameters

base	MCAN peripheral base address.
idx	The MCAN Tx Buffer index.

27.6.29 status_t MCAN_WriteTxBuffer (CAN_Type * *base*, uint8_t *idx*, const mcan_tx_buffer_frame_t * *pTxFrame*)

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

base	MCAN peripheral base address.
idx	The MCAN Tx Buffer index.
pTxFrame	Pointer to CAN message frame to be sent.

27.6.30 status_t MCAN_ReadRxBuffer (CAN_Type * *base*, uint8_t *idx*, mcan_rx_buffer_frame_t * *pRxFrame*)

This function reads a CAN message from the Rx Buffer in the Message RAM.

Parameters

base	MCAN peripheral base address.
idx	The MCAN Rx Buffer index.
pRxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Read Message from Rx Buffer successfully.

27.6.31 status_t MCAN_ReadRxFifo (CAN_Type * *base*, uint8_t *fifoBlock*, mcan_rx_buffer_frame_t * *pRxFrame*)

This function reads a CAN message from the Rx FIFO in the Message RAM.

Parameters

base	MCAN peripheral base address.
fifoBlock	Rx FIFO block 0 or 1.
pRxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Read Message from Rx FIFO successfully.

27.6.32 static void MCAN_TransmitAddRequest (CAN_Type * *base,* uint8_t *idx*) [inline], [static]

This function add sending request to corresponding Tx Buffer.

Parameters

base	MCAN peripheral base address.
idx	Tx Buffer index.

27.6.33 static void MCAN_TransmitCancelRequest (CAN_Type * *base*, uint8_t *idx*) [inline], [static]

This function clears Tx buffer request pending bit.

Parameters

base	MCAN peripheral base address.
idx	Tx Buffer index.

27.6.34 status_t MCAN_TransferSendBlocking (CAN_Type * *base*, uint8_t *idx*, mcan_tx_buffer_frame_t * *pTxFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	MCAN peripheral base pointer.
idx	The MCAN buffer index.
pTxFrame	Pointer to CAN message frame to be sent.

Return values

kStatus_Success	- Write Tx Message Buffer Successfully.
kStatus_Fail	- Tx Message Buffer is currently in use.

27.6.35 status_t MCAN_TransferReceiveBlocking (CAN_Type * *base*, uint8_t *idx*, mcan_rx_buffer_frame_t * *pRxFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	MCAN peripheral base pointer.
idx	The MCAN buffer index.
pRxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Read Rx Message Buffer Successfully.
kStatus_Fail	- No new message.

27.6.36 status_t MCAN_TransferReceiveFifoBlocking (CAN_Type * *base,* uint8_t *fifoBlock,* mcan_rx_buffer_frame_t * *pRxFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	MCAN peripheral base pointer.
fifoBlock	Rx FIFO block, 0 or 1.
pRxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Read Message from Rx FIFO successfully.
kStatus_Fail	- No new message in Rx FIFO.

27.6.37 void MCAN_TransferCreateHandle (CAN_Type * *base,* mcan_handle_t * *handle,* mcan_transfer_callback_t *callback,* void * *userData*)

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

Parameters

base	MCAN peripheral base address.
handle	MCAN handle pointer.
callback	The callback function.
userData	The parameter of the callback function.

27.6.38 status_t MCAN_TransferSendNonBlocking (CAN_Type * *base*, mcan_handle_t * *handle*, mcan_buffer_transfer_t * *xfer*)

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

base	MCAN peripheral base address.
handle	MCAN handle pointer.
xfer	MCAN Buffer transfer structure. See the mcan_buffer_transfer_t.

Return values

kStatus_Success	Start Tx Buffer sending process successfully.
kStatus_Fail	Write Tx Buffer failed.
kStatus_MCAN_TxBusy	Tx Buffer is in use.

27.6.39 status_t MCAN_TransferReceiveFifoNonBlocking (CAN_Type * *base*, uint8_t *fifoBlock*, mcan_handle_t * *handle*, mcan_fifo_transfer_t * *xfer*)

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

base	MCAN peripheral base address.
handle	MCAN handle pointer.
fifoBlock	Rx FIFO block, 0 or 1.
xfer	MCAN Rx FIFO transfer structure. See the mcan_fifo_transfer_t.

Return values

kStatus_Success	- Start Rx FIFO receiving process successfully.
kStatus_MCAN_RxFifo0-Busy	- Rx FIFO 0 is currently in use.
kStatus_MCAN_RxFifo1-Busy	- Rx FIFO 1 is currently in use.

27.6.40 void MCAN_TransferAbortSend (CAN_Type * *base,* mcan_handle_t * *handle,* uint8_t *bufferIdx*)

This function aborts the interrupt driven message send process.

Parameters

base	MCAN peripheral base address.
handle	MCAN handle pointer.
bufferIdx	The MCAN Buffer index.

27.6.41 void MCAN_TransferAbortReceiveFifo (CAN_Type * *base,* uint8_t *fifoBlock,* mcan_handle_t * *handle*)

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

base	MCAN peripheral base address.
fifoBlock	MCAN Fifo block, 0 or 1.
handle	MCAN handle pointer.

27.6.42 void MCAN_TransferHandleIRQ (CAN_Type * *base,* mcan_handle_t * *handle*)

This function handles the MCAN Error, the Buffer, and the Rx FIFO IRQ request.

Parameters

base	MCAN peripheral base address.
handle	MCAN handle pointer.

Chapter 28
MRT: Multi-Rate Timer

28.1 Overview

The MCUXpresso SDK provides a driver for the Multi-Rate Timer (MRT) of MCUXpresso SDK devices.

28.2 Function groups

The MRT driver supports operating the module as a time counter.

28.2.1 Initialization and deinitialization

The function MRT_Init() initializes the MRT with specified configurations. The function MRT_GetDefaultConfig() gets the default configurations. The initialization function configures the MRT operating mode.

The function MRT_Deinit() stops the MRT timers and disables the module clock.

28.2.2 Timer period Operations

The function MRT_UpdateTimerPeriod() is used to update the timer period in units of count. The new value is immediately loaded or will be loaded at the end of the current time interval.

The function MRT_GetCurrentTimerCount() reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. The user can call the utility macros provided in fsl_common.h to convert to microseconds or milliseconds

28.2.3 Start and Stop timer operations

The function MRT_StartTimer() starts the timer counting. After calling this function, the timer loads the period value, counts down to 0 and depending on the timer mode it either loads the respective start value again or stop. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function MRT_StopTimer() stops the timer counting.

28.2.4 Get and release channel

These functions can be used to reserve and release a channel. The function MRT_GetIdleChannel() finds the available channel. This function returns the lowest available channel number. The function MRT_-ReleaseChannel() release the channel when the timer is using the multi-task mode. In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use.

28.2.5 Status

Provides functions to get and clear the PIT status.

28.2.6 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

28.3 Typical use case

28.3.1 MRT tick example

Updates the MRT period and toggles an LED periodically. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/mrt

Files

- file fsl_mrt.h

Data Structures

- struct mrt_config_t
  MRT configuration structure. *More...*

Enumerations

- enum mrt_chnl_t {
  kMRT_Channel_0 = 0U,
  kMRT_Channel_1,
  kMRT_Channel_2,
  kMRT_Channel_3 }
  List of MRT channels.
- enum mrt_timer_mode_t {
  kMRT_RepeatMode = (0 << MRT_CHANNEL_CTRL_MODE_SHIFT),
  kMRT_OneShotMode = (1 << MRT_CHANNEL_CTRL_MODE_SHIFT),
  kMRT_OneShotStallMode = (2 << MRT_CHANNEL_CTRL_MODE_SHIFT) }
  List of MRT timer modes.

- enum mrt_interrupt_enable_t { kMRT_TimerInterruptEnable = MRT_CHANNEL_CTRL_INTEN_MASK }
  List of MRT interrupts.
- enum mrt_status_flags_t {
  kMRT_TimerInterruptFlag = MRT_CHANNEL_STAT_INTFLAG_MASK,
  kMRT_TimerRunFlag = MRT_CHANNEL_STAT_RUN_MASK }
  List of MRT status flags.

Driver version

- #define FSL_MRT_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  Version 2.0.3.

Initialization and deinitialization

- void MRT_Init (MRT_Type *base, const mrt_config_t *config)
  Ungates the MRT clock and configures the peripheral for basic operation.
- void MRT_Deinit (MRT_Type *base)
  Gate the MRT clock.
- static void MRT_GetDefaultConfig (mrt_config_t *config)
  Fill in the MRT config struct with the default settings.
- static void MRT_SetupChannelMode (MRT_Type *base, mrt_chnl_t channel, const mrt_timer_mode_t mode)
  Sets up an MRT channel mode.

Interrupt Interface

- static void MRT_EnableInterrupts (MRT_Type *base, mrt_chnl_t channel, uint32_t mask)
  Enables the MRT interrupt.
- static void MRT_DisableInterrupts (MRT_Type *base, mrt_chnl_t channel, uint32_t mask)
  Disables the selected MRT interrupt.
- static uint32_t MRT_GetEnabledInterrupts (MRT_Type *base, mrt_chnl_t channel)
  Gets the enabled MRT interrupts.

Status Interface

- static uint32_t MRT_GetStatusFlags (MRT_Type *base, mrt_chnl_t channel)
  Gets the MRT status flags.
- static void MRT_ClearStatusFlags (MRT_Type *base, mrt_chnl_t channel, uint32_t mask)
  Clears the MRT status flags.

Read and Write the timer period

- void MRT_UpdateTimerPeriod (MRT_Type *base, mrt_chnl_t channel, uint32_t count, bool immediateLoad)
  Used to update the timer period in units of count.
- static uint32_t MRT_GetCurrentTimerCount (MRT_Type *base, mrt_chnl_t channel)
  Reads the current timer counting value.

Timer Start and Stop

- static void MRT_StartTimer (MRT_Type *base, mrt_chnl_t channel, uint32_t count)
  Starts the timer counting.
- static void MRT_StopTimer (MRT_Type *base, mrt_chnl_t channel)
  Stops the timer counting.

Get & release channel

- static uint32_t MRT_GetIdleChannel (MRT_Type *base)
  Find the available channel.
- static void MRT_ReleaseChannel (MRT_Type *base, mrt_chnl_t channel)
  Release the channel when the timer is using the multi-task mode.

28.4 Data Structure Documentation

28.4.1 struct mrt_config_t

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

The config struct can be made const so it resides in flash

Data Fields

- bool enableMultiTask
  true: Timers run in multi-task mode; false: Timers run in hardware status mode

28.5 Enumeration Type Documentation

28.5.1 enum mrt_chnl_t

Enumerator

kMRT_Channel_0 MRT channel number 0.
kMRT_Channel_1 MRT channel number 1.
kMRT_Channel_2 MRT channel number 2.
kMRT_Channel_3 MRT channel number 3.

28.5.2 enum mrt_timer_mode_t

Enumerator

kMRT_RepeatMode Repeat Interrupt mode.
kMRT_OneShotMode One-shot Interrupt mode.
kMRT_OneShotStallMode One-shot stall mode.

28.5.3 enum mrt_interrupt_enable_t

Enumerator

kMRT_TimerInterruptEnable Timer interrupt enable.

28.5.4 enum mrt_status_flags_t

Enumerator

kMRT_TimerInterruptFlag Timer interrupt flag.
kMRT_TimerRunFlag Indicates state of the timer.

28.6 Function Documentation

28.6.1 void MRT_Init (MRT_Type * *base*, const mrt_config_t * *config*)

Note

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

Parameters

base	Multi-Rate timer peripheral base address
config	Pointer to user's MRT config structure. If MRT has MULTITASK bit field in MODCFG reigster, param config is useless.

28.6.2 void MRT_Deinit (MRT_Type * *base*)

Parameters

base	Multi-Rate timer peripheral base address

28.6.3 static void MRT_GetDefaultConfig (mrt_config_t * *config*) [inline], [static]

The default values are:

```
*     config->enableMultiTask = false;
*
```

Parameters

config	Pointer to user's MRT config structure.

28.6.4 static void MRT_SetupChannelMode (MRT_Type * *base*, mrt_chnl_t *channel*, const mrt_timer_mode_t *mode*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Channel that is being configured.
mode	Timer mode to use for the channel.

28.6.5 static void MRT_EnableInterrupts (MRT_Type * *base*, mrt_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number
mask	The interrupts to enable. This is a logical OR of members of the enumeration mrt_interrupt_enable_t

28.6.6 static void MRT_DisableInterrupts (MRT_Type * *base*, mrt_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number
mask	The interrupts to disable. This is a logical OR of members of the enumeration mrt_interrupt_enable_t

28.6.7 static uint32_t MRT_GetEnabledInterrupts (MRT_Type * *base*, mrt_chnl_t *channel*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number

Returns

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

28.6.8 static uint32_t MRT_GetStatusFlags (MRT_Type * *base*, mrt_chnl_t *channel*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number

Returns

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

28.6.9 static void MRT_ClearStatusFlags (MRT_Type * *base*, mrt_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number
mask	The status flags to clear. This is a logical OR of members of the enumeration mrt_status_flags_t

28.6.10 void MRT_UpdateTimerPeriod (MRT_Type * *base*, mrt_chnl_t *channel*, uint32_t *count*, bool *immediateLoad*)

The new value will be immediately loaded or will be loaded at the end of the current time interval. For one-shot interrupt mode the new value will be immediately loaded.

Note

User can call the utility macros provided in fsl_common.h to convert to ticks

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number
count	Timer period in units of ticks
immediateLoad	true: Load the new value immediately into the TIMER register; false: Load the new value at the end of current timer interval

28.6.11 static uint32_t MRT_GetCurrentTimerCount (MRT_Type * *base*, mrt_chnl_t *channel*) [inline], [static]

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

Note

User can call the utility macros provided in fsl_common.h to convert ticks to usec or msec

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number

Returns

Current timer counting value in ticks

28.6.12 static void MRT_StartTimer (MRT_Type * *base*, mrt_chnl_t *channel*, uint32_t *count*) [inline], [static]

After calling this function, timers load period value, counts down to 0 and depending on the timer mode it will either load the respective start value again or stop.

Note

User can call the utility macros provided in fsl_common.h to convert to ticks

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number.
count	Timer period in units of ticks. Count can contain the LOAD bit, which control the force load feature.

28.6.13 static void MRT_StopTimer (MRT_Type * *base*, mrt_chnl_t *channel*) [inline], [static]

This function stops the timer from counting.

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number.

28.6.14 static uint32_t MRT_GetIdleChannel (MRT_Type * *base*) [inline], [static]

This function returns the lowest available channel number.

Parameters

base	Multi-Rate timer peripheral base address

28.6.15 static void MRT_ReleaseChannel (MRT_Type * *base*, mrt_chnl_t *channel*) [inline], [static]

In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use. The user can hold on to a channel acquired by calling MRT_GetIdleChannel() for as long as it is needed and release it by calling this function. This removes the need to ask for an available channel for every use.

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number.

Chapter 29
OSTIMER: OS Event Timer Driver

29.1 Overview

The MCUXpresso SDK provides a peripheral driver for the OSTIMER module of MCUXpresso SDK devices. OSTIMER driver is created to help user to operate the OSTIMER module. The OSTIMER timer can be used as a low power timer. The APIs can be used to enable the OSTIMER module, initialize it and set the match time, get the current timer count. And the raw value in OS timer register is gray-code type, so both decimal and gray-code format API were added for users. OSTIMER can be used as a wake up source from low power mode.

29.2 Function groups

The OSTIMER driver supports operating the module as a time counter.

29.2.1 Initialization and deinitialization

The OSTIMER_Init() function will initialize the OSTIMER and enable the clock for OSTIMER. The OSTIMER_Deinit() function will shut down the bus clock of OSTIMER.

29.2.2 OSTIMER status

The function OSTIMER_GetStatusFlags() will get the current status flag of OSTIMER. The function OSTIMER_ClearStatusFlag() will help clear the status flags.

29.2.3 OSTIMER set match value

For OSTIMER, allow users set the match in two ways, set match value with raw data(gray code) and st the match value with common data(decimal format). OSTIMER_SetMatchRawValue() is used with gray code and OSTIMER_SetMatchValue() is used together with decimal data.

29.2.4 OSTIMER get timer count

The OSTIMER driver allow users to get the timer count in two ways, getting the gray code value by using OSTIMER_GetCaptureRawValue() and getting the decimal data by using OSTIMER_GetCurrentTimerValue().

29.3 Typical use case

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

Files

- file fsl_ostimer.h

Typedefs

- typedef void(∗ ostimer_callback_t)(void)
  ostimer callback function.

Enumerations

- enum _ostimer_flags { kOSTIMER_MatchInterruptFlag = (OSTIMER_OSEVENT_CTRL_OSTIMER_INTRFLAG_MASK) }
  OSTIMER status flags.

Driver version

- #define FSL_OSTIMER_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  OSTIMER driver version.

Initialization and deinitialization

- void OSTIMER_Init (OSTIMER_Type ∗base)
  Initializes an OSTIMER by turning its bus clock on.
- void OSTIMER_Deinit (OSTIMER_Type ∗base)
  Deinitializes a OSTIMER instance.
- uint64_t OSTIMER_GrayToDecimal (uint64_t gray)
  Translate the value from gray-code to decimal.
- static uint64_t OSTIMER_DecimalToGray (uint64_t dec)
  Translate the value from decimal to gray-code.
- uint32_t OSTIMER_GetStatusFlags (OSTIMER_Type ∗base)
  Get OSTIMER status Flags.
- void OSTIMER_ClearStatusFlags (OSTIMER_Type ∗base, uint32_t mask)
  Clear Status Interrupt Flags.
- status_t OSTIMER_SetMatchRawValue (OSTIMER_Type ∗base, uint64_t count, ostimer_callback_t cb)
  Set the match raw value for OSTIMER.
- status_t OSTIMER_SetMatchValue (OSTIMER_Type ∗base, uint64_t count, ostimer_callback_t cb)
  Set the match value for OSTIMER.
- static void OSTIMER_SetMatchRegister (OSTIMER_Type ∗base, uint64_t value)
  Set value to OSTIMER MATCH register directly.
- static void OSTIMER_EnableMatchInterrupt (OSTIMER_Type ∗base)
  Enable the OSTIMER counter match interrupt.
- static void OSTIMER_DisableMatchInterrupt (OSTIMER_Type ∗base)
  Disable the OSTIMER counter match interrupt.

- static uint64_t OSTIMER_GetCurrentTimerRawValue (OSTIMER_Type *base)
  Get current timer raw count value from OSTIMER.
- uint64_t OSTIMER_GetCurrentTimerValue (OSTIMER_Type *base)
  Get current timer count value from OSTIMER.
- static uint64_t OSTIMER_GetCaptureRawValue (OSTIMER_Type *base)
  Get the capture value from OSTIMER.
- uint64_t OSTIMER_GetCaptureValue (OSTIMER_Type *base)
  Get the capture value from OSTIMER.
- void OSTIMER_HandleIRQ (OSTIMER_Type *base, ostimer_callback_t cb)
  OS timer interrupt Service Handler.

29.4 Macro Definition Documentation

29.4.1 #define FSL_OSTIMER_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

29.5 Typedef Documentation

29.5.1 typedef void(* ostimer_callback_t)(void)

29.6 Enumeration Type Documentation

29.6.1 enum _ostimer_flags

Enumerator

kOSTIMER_MatchInterruptFlag Match interrupt flag bit, sets if the match value was reached.

29.7 Function Documentation

29.7.1 void OSTIMER_Init (OSTIMER_Type * *base*)

29.7.2 void OSTIMER_Deinit (OSTIMER_Type * *base*)

This function shuts down OSTIMER bus clock

Parameters

base	OSTIMER peripheral base address.

29.7.3 uint64_t OSTIMER_GrayToDecimal (uint64_t *gray*)

Parameters

gray	The gray value input.

Returns

The decimal value.

29.7.4 static uint64_t OSTIMER_DecimalToGray (uint64_t *dec*) [inline], [static]

Parameters

dec	The decimal value.

Returns

The gray code of the input value.

29.7.5 uint32_t OSTIMER_GetStatusFlags (OSTIMER_Type * *base*)

This returns the status flag. Currently, only match interrupt flag can be got.

Parameters

base	OSTIMER peripheral base address.

Returns

status register value

29.7.6 void OSTIMER_ClearStatusFlags (OSTIMER_Type * *base,* uint32_t *mask*)

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

Parameters

base	OSTIMER peripheral base address.
mask	Clear bit mask.

Returns

none

29.7.7 status_t OSTIMER_SetMatchRawValue (OSTIMER_Type * *base*, uint64_t *count*, ostimer_callback_t *cb*)

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER. Please note that, the data format is gray-code, if decimal data was desired, please using OSTIMER_SetMatchValue().

Parameters

base	OSTIMER peripheral base address.
count	OSTIMER timer match value.(Value is gray-code format)
cb	OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

kStatus_Success	- Set match raw value and enable interrupt Successfully.
kStatus_Fail	- Set match raw value fail.

29.7.8 status_t OSTIMER_SetMatchValue (OSTIMER_Type * *base*, uint64_t *count*, ostimer_callback_t *cb*)

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central OS TIMER.

Parameters

base	OSTIMER peripheral base address.
count	OSTIMER timer match value.(Value is decimal format, and this value will be translate to Gray code internally.)
cb	OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

kStatus_Success	- Set match value and enable interrupt Successfully.
kStatus_Fail	- Set match value fail.

29.7.9 static void OSTIMER_SetMatchRegister (OSTIMER_Type * *base*, uint64_t *value*) [inline], [static]

This function writes the input value to OSTIMER MATCH register directly, it does not touch any other registers. Note that, the data format is gray-code. The function OSTIMER_DecimalToGray could convert decimal value to gray code.

Parameters

base	OSTIMER peripheral base address.
count	OSTIMER timer match value (Value is gray-code format).

29.7.10 static void OSTIMER_EnableMatchInterrupt (OSTIMER_Type * *base*) [inline], [static]

Enable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

base	OSTIMER peripheral base address.

29.7.11 static void OSTIMER_DisableMatchInterrupt (OSTIMER_Type * *base*) [inline], [static]

Disable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

base	OSTIMER peripheral base address.

29.7.12 static uint64_t OSTIMER_GetCurrentTimerRawValue (OSTIMER_Type * *base*) [inline], [static]

This function will get a gray code type timer count value from OS timer register. The raw value of timer count is gray code format.

Parameters

base	OSTIMER peripheral base address.

Returns

Raw value of OSTIMER, gray code format.

29.7.13 uint64_t OSTIMER_GetCurrentTimerValue (OSTIMER_Type * *base*)

This function will get a decimal timer count value. The RAW value of timer count is gray code format, will be translated to decimal data internally.

Parameters

base	OSTIMER peripheral base address.

Returns

Value of OSTIMER which will be formated to decimal value.

29.7.14 static uint64_t OSTIMER_GetCaptureRawValue (OSTIMER_Type * *base*) [inline], [static]

This function will get a captured gray-code value from OSTIMER. The Raw value of timer capture is gray code format.

Parameters

base	OSTIMER peripheral base address.

Returns

Raw value of capture register, data format is gray code.

29.7.15 uint64_t OSTIMER_GetCaptureValue (OSTIMER_Type * *base*)

This function will get a capture decimal-value from OSTIMER. The RAW value of timer capture is gray code format, will be translated to decimal data internally.

Parameters

base	OSTIMER peripheral base address.

Returns

Value of capture register, data format is decimal.

29.7.16 void OSTIMER_HandleIRQ (OSTIMER_Type * *base*, ostimer_callback_t *cb*)

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in OSTIMER_SetMatchValue()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

base	OS timer peripheral base address.
cb	callback scheduled for this instance of OS timer

Returns

none

Chapter 30
PINT: Pin Interrupt and Pattern Match Driver

30.1 Overview

The MCUXpresso SDK provides a driver for the Pin Interrupt and Pattern match (PINT).

It can configure one or more pins to generate a pin interrupt when the pin or pattern match conditions are met. The pins do not have to be configured as gpio pins however they must be connected to PINT via INPUTMUX. Only the pin interrupt or pattern match function can be active for interrupt generation. If the pin interrupt function is enabled then the pattern match function can be used for wakeup via RXEV.

30.2 Pin Interrupt and Pattern match Driver operation

PINT_PinInterruptConfig() function configures the pins for pin interrupt.

PINT_PatternMatchConfig() function configures the pins for pattern match.

30.2.1 Pin Interrupt use case

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

30.2.2 Pattern match use case

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

Files

- file fsl_pint.h

Typedefs

- typedef void(∗ pint_cb_t)(pint_pin_int_t pintr, uint32_t pmatch_status)
  PINT Callback function.

Enumerations

- enum pint_pin_enable_t {
  kPINT_PinIntEnableNone = 0U,
  kPINT_PinIntEnableRiseEdge = PINT_PIN_RISE_EDGE,
  kPINT_PinIntEnableFallEdge = PINT_PIN_FALL_EDGE,
  kPINT_PinIntEnableBothEdges = PINT_PIN_BOTH_EDGE,
  kPINT_PinIntEnableLowLevel = PINT_PIN_LOW_LEVEL,
  kPINT_PinIntEnableHighLevel = PINT_PIN_HIGH_LEVEL }

PINT Pin Interrupt enable type.

- enum pint_pin_int_t {
  kPINT_PinInt0 = 0U,
  kPINT_PinInt1 = 1U,
  kPINT_PinInt2 = 2U,
  kPINT_PinInt3 = 3U,
  kPINT_PinInt4 = 4U,
  kPINT_PinInt5 = 5U,
  kPINT_PinInt6 = 6U,
  kPINT_PinInt7 = 7U,
  kPINT_SecPinInt0 = 0U,
  kPINT_SecPinInt1 = 1U }

  PINT Pin Interrupt type.
- enum pint_pmatch_input_src_t {
  kPINT_PatternMatchInp0Src = 0U,
  kPINT_PatternMatchInp1Src = 1U,
  kPINT_PatternMatchInp2Src = 2U,
  kPINT_PatternMatchInp3Src = 3U,
  kPINT_PatternMatchInp4Src = 4U,
  kPINT_PatternMatchInp5Src = 5U,
  kPINT_PatternMatchInp6Src = 6U,
  kPINT_PatternMatchInp7Src = 7U,
  kPINT_SecPatternMatchInp0Src = 0U,
  kPINT_SecPatternMatchInp1Src = 1U }

  PINT Pattern Match bit slice input source type.
- enum pint_pmatch_bslice_t {
  kPINT_PatternMatchBSlice0 = 0U,
  kPINT_PatternMatchBSlice1 = 1U,
  kPINT_PatternMatchBSlice2 = 2U,
  kPINT_PatternMatchBSlice3 = 3U,
  kPINT_PatternMatchBSlice4 = 4U,
  kPINT_PatternMatchBSlice5 = 5U,
  kPINT_PatternMatchBSlice6 = 6U,
  kPINT_PatternMatchBSlice7 = 7U,
  kPINT_SecPatternMatchBSlice0 = 0U,
  kPINT_SecPatternMatchBSlice1 = 1U }

  PINT Pattern Match bit slice type.
- enum pint_pmatch_bslice_cfg_t {
  kPINT_PatternMatchAlways = 0U,
  kPINT_PatternMatchStickyRise = 1U,
  kPINT_PatternMatchStickyFall = 2U,
  kPINT_PatternMatchStickyBothEdges = 3U,
  kPINT_PatternMatchHigh = 4U,
  kPINT_PatternMatchLow = 5U,
  kPINT_PatternMatchNever = 6U,
  kPINT_PatternMatchBothEdges = 7U }

PINT Pattern Match configuration type.

Functions

- void PINT_Init (PINT_Type *base)
  Initialize PINT peripheral.
- void PINT_PinInterruptConfig (PINT_Type *base, pint_pin_int_t intr, pint_pin_enable_t enable, pint_cb_t callback)
  Configure PINT peripheral pin interrupt.
- void PINT_PinInterruptGetConfig (PINT_Type *base, pint_pin_int_t pintr, pint_pin_enable_t *enable, pint_cb_t *callback)
  Get PINT peripheral pin interrupt configuration.
- void PINT_PinInterruptClrStatus (PINT_Type *base, pint_pin_int_t pintr)
  Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.
- static uint32_t PINT_PinInterruptGetStatus (PINT_Type *base, pint_pin_int_t pintr)
  Get Selected pin interrupt status.
- void PINT_PinInterruptClrStatusAll (PINT_Type *base)
  Clear all pin interrupts status only when pins were triggered by edge-sensitive.
- static uint32_t PINT_PinInterruptGetStatusAll (PINT_Type *base)
  Get all pin interrupts status.
- static void PINT_PinInterruptClrFallFlag (PINT_Type *base, pint_pin_int_t pintr)
  Clear Selected pin interrupt fall flag.
- static uint32_t PINT_PinInterruptGetFallFlag (PINT_Type *base, pint_pin_int_t pintr)
  Get selected pin interrupt fall flag.
- static void PINT_PinInterruptClrFallFlagAll (PINT_Type *base)
  Clear all pin interrupt fall flags.
- static uint32_t PINT_PinInterruptGetFallFlagAll (PINT_Type *base)
  Get all pin interrupt fall flags.
- static void PINT_PinInterruptClrRiseFlag (PINT_Type *base, pint_pin_int_t pintr)
  Clear Selected pin interrupt rise flag.
- static uint32_t PINT_PinInterruptGetRiseFlag (PINT_Type *base, pint_pin_int_t pintr)
  Get selected pin interrupt rise flag.
- static void PINT_PinInterruptClrRiseFlagAll (PINT_Type *base)
  Clear all pin interrupt rise flags.
- static uint32_t PINT_PinInterruptGetRiseFlagAll (PINT_Type *base)
  Get all pin interrupt rise flags.
- void PINT_PatternMatchConfig (PINT_Type *base, pint_pmatch_bslice_t bslice, pint_pmatch_cfg_t *cfg)
  Configure PINT pattern match.
- void PINT_PatternMatchGetConfig (PINT_Type *base, pint_pmatch_bslice_t bslice, pint_pmatch_cfg_t *cfg)
  Get PINT pattern match configuration.
- static uint32_t PINT_PatternMatchGetStatus (PINT_Type *base, pint_pmatch_bslice_t bslice)
  Get pattern match bit slice status.
- static uint32_t PINT_PatternMatchGetStatusAll (PINT_Type *base)
  Get status of all pattern match bit slices.
- uint32_t PINT_PatternMatchResetDetectLogic (PINT_Type *base)
  Reset pattern match detection logic.
- static void PINT_PatternMatchEnable (PINT_Type *base)
  Enable pattern match function.
- static void PINT_PatternMatchDisable (PINT_Type *base)

Disable pattern match function.
- static void PINT_PatternMatchEnableRXEV (PINT_Type *base)
  Enable RXEV output.
- static void PINT_PatternMatchDisableRXEV (PINT_Type *base)
  Disable RXEV output.
- void PINT_EnableCallback (PINT_Type *base)
  Enable callback.
- void PINT_DisableCallback (PINT_Type *base)
  Disable callback.
- void PINT_Deinit (PINT_Type *base)
  Deinitialize PINT peripheral.
- void PINT_EnableCallbackByIndex (PINT_Type *base, pint_pin_int_t pintIdx)
  enable callback by pin index.
- void PINT_DisableCallbackByIndex (PINT_Type *base, pint_pin_int_t pintIdx)
  disable callback by pin index.

Driver version

- #define FSL_PINT_DRIVER_VERSION (MAKE_VERSION(2, 1, 9))
  Version 2.1.9.

30.3 Typedef Documentation

30.3.1 typedef void(* pint_cb_t)(pint_pin_int_t pintr, uint32_t pmatch_status)

30.4 Enumeration Type Documentation

30.4.1 enum pint_pin_enable_t

Enumerator

kPINT_PinIntEnableNone Do not generate Pin Interrupt.
kPINT_PinIntEnableRiseEdge Generate Pin Interrupt on rising edge.
kPINT_PinIntEnableFallEdge Generate Pin Interrupt on falling edge.
kPINT_PinIntEnableBothEdges Generate Pin Interrupt on both edges.
kPINT_PinIntEnableLowLevel Generate Pin Interrupt on low level.
kPINT_PinIntEnableHighLevel Generate Pin Interrupt on high level.

30.4.2 enum pint_pin_int_t

Enumerator

kPINT_PinInt0 Pin Interrupt 0.
kPINT_PinInt1 Pin Interrupt 1.
kPINT_PinInt2 Pin Interrupt 2.
kPINT_PinInt3 Pin Interrupt 3.
kPINT_PinInt4 Pin Interrupt 4.

kPINT_PinInt5 Pin Interrupt 5.
kPINT_PinInt6 Pin Interrupt 6.
kPINT_PinInt7 Pin Interrupt 7.
kPINT_SecPinInt0 Secure Pin Interrupt 0.
kPINT_SecPinInt1 Secure Pin Interrupt 1.

30.4.3 enum pint_pmatch_input_src_t

Enumerator

kPINT_PatternMatchInp0Src Input source 0.
kPINT_PatternMatchInp1Src Input source 1.
kPINT_PatternMatchInp2Src Input source 2.
kPINT_PatternMatchInp3Src Input source 3.
kPINT_PatternMatchInp4Src Input source 4.
kPINT_PatternMatchInp5Src Input source 5.
kPINT_PatternMatchInp6Src Input source 6.
kPINT_PatternMatchInp7Src Input source 7.
kPINT_SecPatternMatchInp0Src Input source 0.
kPINT_SecPatternMatchInp1Src Input source 1.

30.4.4 enum pint_pmatch_bslice_t

Enumerator

kPINT_PatternMatchBSlice0 Bit slice 0.
kPINT_PatternMatchBSlice1 Bit slice 1.
kPINT_PatternMatchBSlice2 Bit slice 2.
kPINT_PatternMatchBSlice3 Bit slice 3.
kPINT_PatternMatchBSlice4 Bit slice 4.
kPINT_PatternMatchBSlice5 Bit slice 5.
kPINT_PatternMatchBSlice6 Bit slice 6.
kPINT_PatternMatchBSlice7 Bit slice 7.
kPINT_SecPatternMatchBSlice0 Bit slice 0.
kPINT_SecPatternMatchBSlice1 Bit slice 1.

30.4.5 enum pint_pmatch_bslice_cfg_t

Enumerator

kPINT_PatternMatchAlways Always Contributes to product term match.

kPINT_PatternMatchStickyRise Sticky Rising edge.
kPINT_PatternMatchStickyFall Sticky Falling edge.
kPINT_PatternMatchStickyBothEdges Sticky Rising or Falling edge.
kPINT_PatternMatchHigh High level.
kPINT_PatternMatchLow Low level.
kPINT_PatternMatchNever Never contributes to product term match.
kPINT_PatternMatchBothEdges Either rising or falling edge.

30.5 Function Documentation

30.5.1 void PINT_Init (PINT_Type * *base*)

This function initializes the PINT peripheral and enables the clock.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

30.5.2 void PINT_PinInterruptConfig (PINT_Type * *base*, pint_pin_int_t *intr*, pint_pin_enable_t *enable*, pint_cb_t *callback*)

This function configures a given pin interrupt.

Parameters

base	Base address of the PINT peripheral.
intr	Pin interrupt.
enable	Selects detection logic.
callback	Callback.

Return values

None.	

30.5.3 void PINT_PinInterruptGetConfig (PINT_Type * *base*, pint_pin_int_t *pintr*, pint_pin_enable_t * *enable*, pint_cb_t * *callback*)

This function returns the configuration of a given pin interrupt.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.
enable	Pointer to store the detection logic.
callback	Callback.

Return values

None.	

30.5.4 void PINT_PinInterruptClrStatus (PINT_Type * *base*, pint_pin_int_t *pintr*)

This function clears the selected pin interrupt status.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

None.	

30.5.5 static uint32_t PINT_PinInterruptGetStatus (PINT_Type * *base*, pint_pin_int_t *pintr*) [inline], [static]

This function returns the selected pin interrupt status.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

status	= 0 No pin interrupt request. = 1 Selected Pin interrupt request active.

30.5.6 void PINT_PinInterruptClrStatusAll (PINT_Type * *base*)

This function clears the status of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

30.5.7 static uint32_t PINT_PinInterruptGetStatusAll (PINT_Type * *base*) [inline], [static]

This function returns the status of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

status	Each bit position indicates the status of corresponding pin interrupt. = 0 No pin interrupt request. = 1 Pin interrupt request active.

30.5.8 static void PINT_PinInterruptClrFallFlag (PINT_Type * *base*, pint_pin_int_t *pintr*) [inline], [static]

This function clears the selected pin interrupt fall flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

None.	

30.5.9 static uint32_t PINT_PinInterruptGetFallFlag (PINT_Type * *base,* pint_pin_int_t *pintr*) [inline], [static]

This function returns the selected pin interrupt fall flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

flag	= 0 Falling edge has not been detected. = 1 Falling edge has been detected.

30.5.10 static void PINT_PinInterruptClrFallFlagAll (PINT_Type * *base*) [inline], [static]

This function clears the fall flag for all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

30.5.11 static uint32_t PINT_PinInterruptGetFallFlagAll (PINT_Type * *base*) [inline], [static]

This function returns the fall flag of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

flags	Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected.

30.5.12 static void PINT_PinInterruptClrRiseFlag (PINT_Type * *base*, pint_pin_int_t *pintr*) [inline], [static]

This function clears the selected pin interrupt rise flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

None.	

30.5.13 static uint32_t PINT_PinInterruptGetRiseFlag (PINT_Type * *base*, pint_pin_int_t *pintr*) [inline], [static]

This function returns the selected pin interrupt rise flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

flag	= 0 Rising edge has not been detected. = 1 Rising edge has been detected.

30.5.14 static void PINT_PinInterruptClrRiseFlagAll (PINT_Type * *base*) [inline], [static]

This function clears the rise flag for all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

30.5.15 static uint32_t PINT_PinInterruptGetRiseFlagAll (PINT_Type * *base*) [inline], [static]

This function returns the rise flag of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

flags	Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected.

30.5.16 void PINT_PatternMatchConfig (PINT_Type * *base*, pint_pmatch_bslice_t *bslice*, pint_pmatch_cfg_t * *cfg*)

This function configures a given pattern match bit slice.

Parameters

base	Base address of the PINT peripheral.
bslice	Pattern match bit slice number.
cfg	Pointer to bit slice configuration.

Return values

None.	

30.5.17 void PINT_PatternMatchGetConfig (PINT_Type * *base*, pint_pmatch_bslice_t *bslice*, pint_pmatch_cfg_t * *cfg*)

This function returns the configuration of a given pattern match bit slice.

Parameters

base	Base address of the PINT peripheral.
bslice	Pattern match bit slice number.
cfg	Pointer to bit slice configuration.

Return values

None.	

30.5.18 static uint32_t PINT_PatternMatchGetStatus (PINT_Type * *base*, pint_pmatch_bslice_t *bslice*) [inline], [static]

This function returns the status of selected bit slice.

Parameters

base	Base address of the PINT peripheral.
bslice	Pattern match bit slice number.

Return values

status	= 0 Match has not been detected. = 1 Match has been detected.

30.5.19 static uint32_t PINT_PatternMatchGetStatusAll (PINT_Type * *base*) [inline], [static]

This function returns the status of all bit slices.

Parameters

base	Base address of the PINT peripheral.

Return values

status	Each bit position indicates the match status of corresponding bit slice. = 0 Match has not been detected. = 1 Match has been detected.

30.5.20 uint32_t PINT_PatternMatchResetDetectLogic (PINT_Type * *base*)

This function resets the pattern match detection logic if any of the product term is matching.

Parameters

base	Base address of the PINT peripheral.

Return values

pmstatus	Each bit position indicates the match status of corresponding bit slice. = 0 Match was detected. = 1 Match was not detected.

30.5.21 static void PINT_PatternMatchEnable (PINT_Type * *base*) [inline], [static]

This function enables the pattern match function.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

30.5.22 static void PINT_PatternMatchDisable (PINT_Type * *base*) [inline], [static]

This function disables the pattern match function.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

30.5.23 static void PINT_PatternMatchEnableRXEV (PINT_Type * *base*) [inline], [static]

This function enables the pattern match RXEV output.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

30.5.24 static void PINT_PatternMatchDisableRXEV (PINT_Type * *base*) [inline], [static]

This function disables the pattern match RXEV output.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

30.5.25 void PINT_EnableCallback (PINT_Type * *base*)

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

30.5.26 void PINT_DisableCallback (PINT_Type * *base*)

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

base	Base address of the peripheral.

Return values

None.	

30.5.27 void PINT_Deinit (PINT_Type * *base*)

This function disables the PINT clock.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

30.5.28 void PINT_EnableCallbackByIndex (PINT_Type * *base*, pint_pin_int_t *pintIdx*)

This function enables callback by pin index instead of enabling all pins.

Parameters

base	Base address of the peripheral.
pintIdx	pin index.

Return values

None.	

30.5.29 void PINT_DisableCallbackByIndex (PINT_Type * *base*, pint_pin_int_t *pintIdx*)

This function disables callback by pin index instead of disabling all pins.

Parameters

base	Base address of the peripheral.
pintIdx	pin index.

Return values

None.	

Chapter 31
PLU: Programmable Logic Unit

31.1 Overview

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

31.2 Function groups

The PLU driver supports the creation of small combinatorial and/or sequential logic networks including simple state machines.

31.2.1 Initialization and de-initialization

The function PLU_Init() enables the PLU clock and reset the module.

The function PIT_Deinit() gates the PLU clock.

31.2.2 Set input/output source and Truth Table

The function PLU_SetLutInputSource() sets the input source for the LUT element.

The function PLU_SetOutputSource() sets output source of the PLU module.

The function PLU_SetLutTruthTable() sets the truth table for the LUT element.

31.2.3 Read current Output State

The function PLU_ReadOutputState() reads the current state of the 8 designated PLU Outputs.

31.2.4 Wake-up/Interrupt Control

The function PLU_EnableWakeIntRequest() enables the wake-up/interrupt request on a PLU output pin with a optional configuration to eliminate the glitches. The function PLU_GetDefaultWakeIntConfig() gets the default configuration which can be used in a case with a given PLU_CLKIN.

The function PLU_LatchInterrupt() latches the interrupt and it can be cleared by function PLU_ClearLatchedInterrupt().

31.3 Typical use case

31.3.1 PLU combination example

Create a simple combinatorial logic network to control the LED. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/plu/combination

Data Structures

- struct plu_wakeint_config_t
  Wake configuration. *More...*

Enumerations

- enum plu_lut_index_t {
  kPLU_LUT_0 = 0U,
  kPLU_LUT_1 = 1U,
  kPLU_LUT_2 = 2U,
  kPLU_LUT_3 = 3U,
  kPLU_LUT_4 = 4U,
  kPLU_LUT_5 = 5U,
  kPLU_LUT_6 = 6U,
  kPLU_LUT_7 = 7U,
  kPLU_LUT_8 = 8U,
  kPLU_LUT_9 = 9U,
  kPLU_LUT_10 = 10U,
  kPLU_LUT_11 = 11U,
  kPLU_LUT_12 = 12U,
  kPLU_LUT_13 = 13U,
  kPLU_LUT_14 = 14U,
  kPLU_LUT_15 = 15U,
  kPLU_LUT_16 = 16U,
  kPLU_LUT_17 = 17U,
  kPLU_LUT_18 = 18U,
  kPLU_LUT_19 = 19U,
  kPLU_LUT_20 = 20U,
  kPLU_LUT_21 = 21U,
  kPLU_LUT_22 = 22U,
  kPLU_LUT_23 = 23U,
  kPLU_LUT_24 = 24U,
  kPLU_LUT_25 = 25U }
  Index of LUT.
- enum plu_lut_in_index_t {
  kPLU_LUT_IN_0 = 0U,
  kPLU_LUT_IN_1 = 1U,
  kPLU_LUT_IN_2 = 2U,
  kPLU_LUT_IN_3 = 3U,

kPLU_LUT_IN_4 = 4U }
Inputs of LUT.

- enum plu_lut_input_source_t {
kPLU_LUT_IN_SRC_PLU_IN_0 = 0U,
kPLU_LUT_IN_SRC_PLU_IN_1 = 1U,
kPLU_LUT_IN_SRC_PLU_IN_2 = 2U,
kPLU_LUT_IN_SRC_PLU_IN_3 = 3U,
kPLU_LUT_IN_SRC_PLU_IN_4 = 4U,
kPLU_LUT_IN_SRC_PLU_IN_5 = 5U,
kPLU_LUT_IN_SRC_LUT_OUT_0 = 6U,
kPLU_LUT_IN_SRC_LUT_OUT_1 = 7U,
kPLU_LUT_IN_SRC_LUT_OUT_2 = 8U,
kPLU_LUT_IN_SRC_LUT_OUT_3 = 9U,
kPLU_LUT_IN_SRC_LUT_OUT_4 = 10U,
kPLU_LUT_IN_SRC_LUT_OUT_5 = 11U,
kPLU_LUT_IN_SRC_LUT_OUT_6 = 12U,
kPLU_LUT_IN_SRC_LUT_OUT_7 = 13U,
kPLU_LUT_IN_SRC_LUT_OUT_8 = 14U,
kPLU_LUT_IN_SRC_LUT_OUT_9 = 15U,
kPLU_LUT_IN_SRC_LUT_OUT_10 = 16U,
kPLU_LUT_IN_SRC_LUT_OUT_11 = 17U,
kPLU_LUT_IN_SRC_LUT_OUT_12 = 18U,
kPLU_LUT_IN_SRC_LUT_OUT_13 = 19U,
kPLU_LUT_IN_SRC_LUT_OUT_14 = 20U,
kPLU_LUT_IN_SRC_LUT_OUT_15 = 21U,
kPLU_LUT_IN_SRC_LUT_OUT_16 = 22U,
kPLU_LUT_IN_SRC_LUT_OUT_17 = 23U,
kPLU_LUT_IN_SRC_LUT_OUT_18 = 24U,
kPLU_LUT_IN_SRC_LUT_OUT_19 = 25U,
kPLU_LUT_IN_SRC_LUT_OUT_20 = 26U,
kPLU_LUT_IN_SRC_LUT_OUT_21 = 27U,
kPLU_LUT_IN_SRC_LUT_OUT_22 = 28U,
kPLU_LUT_IN_SRC_LUT_OUT_23 = 29U,
kPLU_LUT_IN_SRC_LUT_OUT_24 = 30U,
kPLU_LUT_IN_SRC_LUT_OUT_25 = 31U,
kPLU_LUT_IN_SRC_FLIPFLOP_0 = 32U,
kPLU_LUT_IN_SRC_FLIPFLOP_1 = 33U,
kPLU_LUT_IN_SRC_FLIPFLOP_2 = 34U,
kPLU_LUT_IN_SRC_FLIPFLOP_3 = 35U }
Available sources of LUT input.
- enum plu_output_index_t {

kPLU_OUTPUT_0 = 0U,
kPLU_OUTPUT_1 = 1U,
kPLU_OUTPUT_2 = 2U,
kPLU_OUTPUT_3 = 3U,
kPLU_OUTPUT_4 = 4U,
kPLU_OUTPUT_5 = 5U,
kPLU_OUTPUT_6 = 6U,
kPLU_OUTPUT_7 = 7U }
PLU output multiplexer registers.

- enum plu_output_source_t {
kPLU_OUT_SRC_LUT_0 = 0U,
kPLU_OUT_SRC_LUT_1 = 1U,
kPLU_OUT_SRC_LUT_2 = 2U,
kPLU_OUT_SRC_LUT_3 = 3U,
kPLU_OUT_SRC_LUT_4 = 4U,
kPLU_OUT_SRC_LUT_5 = 5U,
kPLU_OUT_SRC_LUT_6 = 6U,
kPLU_OUT_SRC_LUT_7 = 7U,
kPLU_OUT_SRC_LUT_8 = 8U,
kPLU_OUT_SRC_LUT_9 = 9U,
kPLU_OUT_SRC_LUT_10 = 10U,
kPLU_OUT_SRC_LUT_11 = 11U,
kPLU_OUT_SRC_LUT_12 = 12U,
kPLU_OUT_SRC_LUT_13 = 13U,
kPLU_OUT_SRC_LUT_14 = 14U,
kPLU_OUT_SRC_LUT_15 = 15U,
kPLU_OUT_SRC_LUT_16 = 16U,
kPLU_OUT_SRC_LUT_17 = 17U,
kPLU_OUT_SRC_LUT_18 = 18U,
kPLU_OUT_SRC_LUT_19 = 19U,
kPLU_OUT_SRC_LUT_20 = 20U,
kPLU_OUT_SRC_LUT_21 = 21U,
kPLU_OUT_SRC_LUT_22 = 22U,
kPLU_OUT_SRC_LUT_23 = 23U,
kPLU_OUT_SRC_LUT_24 = 24U,
kPLU_OUT_SRC_LUT_25 = 25U,
kPLU_OUT_SRC_FLIPFLOP_0 = 26U,
kPLU_OUT_SRC_FLIPFLOP_1 = 27U,
kPLU_OUT_SRC_FLIPFLOP_2 = 28U,
kPLU_OUT_SRC_FLIPFLOP_3 = 29U }
Available sources of PLU output.
- enum _plu_interrupt_mask {

kPLU_OUTPUT_0_INTERRUPT_MASK = 1 << 0,
kPLU_OUTPUT_1_INTERRUPT_MASK = 1 << 1,
kPLU_OUTPUT_2_INTERRUPT_MASK = 1 << 2,
kPLU_OUTPUT_3_INTERRUPT_MASK = 1 << 3,
kPLU_OUTPUT_4_INTERRUPT_MASK = 1 << 4,
kPLU_OUTPUT_5_INTERRUPT_MASK = 1 << 5,
kPLU_OUTPUT_6_INTERRUPT_MASK = 1 << 6,
kPLU_OUTPUT_7_INTERRUPT_MASK = 1 << 7 }
The enumerator of PLU Interrupt.

- enum plu_wakeint_filter_mode_t {
kPLU_WAKEINT_FILTER_MODE_BYPASS = 0U,
kPLU_WAKEINT_FILTER_MODE_1_CLK_PERIOD = 1U,
kPLU_WAKEINT_FILTER_MODE_2_CLK_PERIOD = 2U,
kPLU_WAKEINT_FILTER_MODE_3_CLK_PERIOD = 3U }
Control input of the PLU, add filtering for glitch.
- enum plu_wakeint_filter_clock_source_t {
kPLU_WAKEINT_FILTER_CLK_SRC_1MHZ_LPOSC = 0U,
kPLU_WAKEINT_FILTER_CLK_SRC_12MHZ_FRO = 1U,
kPLU_WAKEINT_FILTER_CLK_SRC_ALT = 2U }
Clock source for filter mode.

Driver version

- #define FSL_PLU_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
Version 2.2.1.

Initialization and deinitialization

- void PLU_Init (PLU_Type *base)
Enable the PLU clock and reset the module.
- void PLU_Deinit (PLU_Type *base)
Gate the PLU clock.

Set input/output source and Truth Table

- static void PLU_SetLutInputSource (PLU_Type *base, plu_lut_index_t lutIndex, plu_lut_in_index_t lutInIndex, plu_lut_input_source_t inputSrc)
Set Input source of LUT.
- static void PLU_SetOutputSource (PLU_Type *base, plu_output_index_t outputIndex, plu_output_source_t outputSrc)
Set Output source of PLU.
- static void PLU_SetLutTruthTable (PLU_Type *base, plu_lut_index_t lutIndex, uint32_t truthTable)
Set Truth Table of LUT.

Read current Output State

- static uint32_t PLU_ReadOutputState (PLU_Type *base)
Read the current state of the 8 designated PLU Outputs.

Wake-up/Interrupt Control

- void PLU_GetDefaultWakeIntConfig (plu_wakeint_config_t ∗config)
  Gets an available pre-defined settings for wakeup/interrupt control.
- void PLU_EnableWakeIntRequest (PLU_Type ∗base, uint32_t interruptMask, const plu_wakeint_-config_t ∗config)
  Enable PLU outputs wakeup/interrupt request.
- static void PLU_LatchInterrupt (PLU_Type ∗base)
  Latch an interrupt.
- void PLU_ClearLatchedInterrupt (PLU_Type ∗base)
  Clear the latched interrupt.

31.4 Data Structure Documentation

31.4.1 struct plu_wakeint_config_t

Data Fields

- plu_wakeint_filter_mode_t filterMode
  Filter Mode.
- plu_wakeint_filter_clock_source_t clockSource
  The clock source for filter mode.

Field Documentation

(1) plu_wakeint_filter_mode_t plu_wakeint_config_t::filterMode

(2) plu_wakeint_filter_clock_source_t plu_wakeint_config_t::clockSource

31.5 Enumeration Type Documentation

31.5.1 enum plu_lut_index_t

Enumerator

kPLU_LUT_0 5-input Look-up Table 0
kPLU_LUT_1 5-input Look-up Table 1
kPLU_LUT_2 5-input Look-up Table 2
kPLU_LUT_3 5-input Look-up Table 3
kPLU_LUT_4 5-input Look-up Table 4
kPLU_LUT_5 5-input Look-up Table 5
kPLU_LUT_6 5-input Look-up Table 6
kPLU_LUT_7 5-input Look-up Table 7
kPLU_LUT_8 5-input Look-up Table 8
kPLU_LUT_9 5-input Look-up Table 9
kPLU_LUT_10 5-input Look-up Table 10
kPLU_LUT_11 5-input Look-up Table 11
kPLU_LUT_12 5-input Look-up Table 12
kPLU_LUT_13 5-input Look-up Table 13

kPLU_LUT_14 5-input Look-up Table 14
kPLU_LUT_15 5-input Look-up Table 15
kPLU_LUT_16 5-input Look-up Table 16
kPLU_LUT_17 5-input Look-up Table 17
kPLU_LUT_18 5-input Look-up Table 18
kPLU_LUT_19 5-input Look-up Table 19
kPLU_LUT_20 5-input Look-up Table 20
kPLU_LUT_21 5-input Look-up Table 21
kPLU_LUT_22 5-input Look-up Table 22
kPLU_LUT_23 5-input Look-up Table 23
kPLU_LUT_24 5-input Look-up Table 24
kPLU_LUT_25 5-input Look-up Table 25

31.5.2 enum plu_lut_in_index_t

5 input present for each LUT.

Enumerator

kPLU_LUT_IN_0 LUT input 0.
kPLU_LUT_IN_1 LUT input 1.
kPLU_LUT_IN_2 LUT input 2.
kPLU_LUT_IN_3 LUT input 3.
kPLU_LUT_IN_4 LUT input 4.

31.5.3 enum plu_lut_input_source_t

Enumerator

kPLU_LUT_IN_SRC_PLU_IN_0 Select PLU input 0 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_PLU_IN_1 Select PLU input 1 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_PLU_IN_2 Select PLU input 2 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_PLU_IN_3 Select PLU input 3 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_PLU_IN_4 Select PLU input 4 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_PLU_IN_5 Select PLU input 5 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_0 Select LUT output 0 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_1 Select LUT output 1 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_2 Select LUT output 2 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_3 Select LUT output 3 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_4 Select LUT output 4 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_5 Select LUT output 5 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_6 Select LUT output 6 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_7 Select LUT output 7 to be connected to LUTn Input x.

kPLU_LUT_IN_SRC_LUT_OUT_8 Select LUT output 8 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_9 Select LUT output 9 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_10 Select LUT output 10 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_11 Select LUT output 11 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_12 Select LUT output 12 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_13 Select LUT output 13 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_14 Select LUT output 14 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_15 Select LUT output 15 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_16 Select LUT output 16 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_17 Select LUT output 17 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_18 Select LUT output 18 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_19 Select LUT output 19 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_20 Select LUT output 20 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_21 Select LUT output 21 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_22 Select LUT output 22 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_23 Select LUT output 23 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_24 Select LUT output 24 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_LUT_OUT_25 Select LUT output 25 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_FLIPFLOP_0 Select Flip-Flops state 0 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_FLIPFLOP_1 Select Flip-Flops state 1 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_FLIPFLOP_2 Select Flip-Flops state 2 to be connected to LUTn Input x.
kPLU_LUT_IN_SRC_FLIPFLOP_3 Select Flip-Flops state 3 to be connected to LUTn Input x.

31.5.4 enum plu_output_index_t

Enumerator

kPLU_OUTPUT_0 PLU OUTPUT 0.
kPLU_OUTPUT_1 PLU OUTPUT 1.
kPLU_OUTPUT_2 PLU OUTPUT 2.
kPLU_OUTPUT_3 PLU OUTPUT 3.
kPLU_OUTPUT_4 PLU OUTPUT 4.
kPLU_OUTPUT_5 PLU OUTPUT 5.
kPLU_OUTPUT_6 PLU OUTPUT 6.
kPLU_OUTPUT_7 PLU OUTPUT 7.

31.5.5 enum plu_output_source_t

Enumerator

kPLU_OUT_SRC_LUT_0 Select LUT0 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_1 Select LUT1 output to be connected to PLU output.

kPLU_OUT_SRC_LUT_2 Select LUT2 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_3 Select LUT3 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_4 Select LUT4 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_5 Select LUT5 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_6 Select LUT6 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_7 Select LUT7 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_8 Select LUT8 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_9 Select LUT9 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_10 Select LUT10 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_11 Select LUT11 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_12 Select LUT12 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_13 Select LUT13 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_14 Select LUT14 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_15 Select LUT15 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_16 Select LUT16 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_17 Select LUT17 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_18 Select LUT18 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_19 Select LUT19 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_20 Select LUT20 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_21 Select LUT21 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_22 Select LUT22 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_23 Select LUT23 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_24 Select LUT24 output to be connected to PLU output.
kPLU_OUT_SRC_LUT_25 Select LUT25 output to be connected to PLU output.
kPLU_OUT_SRC_FLIPFLOP_0 Select Flip-Flops state(0) to be connected to PLU output.
kPLU_OUT_SRC_FLIPFLOP_1 Select Flip-Flops state(1) to be connected to PLU output.
kPLU_OUT_SRC_FLIPFLOP_2 Select Flip-Flops state(2) to be connected to PLU output.
kPLU_OUT_SRC_FLIPFLOP_3 Select Flip-Flops state(3) to be connected to PLU output.

31.5.6 enum _plu_interrupt_mask

Enumerator

kPLU_OUTPUT_0_INTERRUPT_MASK Select PLU output 0 contribute to interrupt/wake-up generation.
kPLU_OUTPUT_1_INTERRUPT_MASK Select PLU output 1 contribute to interrupt/wake-up generation.
kPLU_OUTPUT_2_INTERRUPT_MASK Select PLU output 2 contribute to interrupt/wake-up generation.
kPLU_OUTPUT_3_INTERRUPT_MASK Select PLU output 3 contribute to interrupt/wake-up generation.
kPLU_OUTPUT_4_INTERRUPT_MASK Select PLU output 4 contribute to interrupt/wake-up generation.

kPLU_OUTPUT_5_INTERRUPT_MASK Select PLU output 5 contribute to interrupt/wake-up generation.
kPLU_OUTPUT_6_INTERRUPT_MASK Select PLU output 6 contribute to interrupt/wake-up generation.
kPLU_OUTPUT_7_INTERRUPT_MASK Select PLU output 7 contribute to interrupt/wake-up generation.

31.5.7 enum plu_wakeint_filter_mode_t

Enumerator

kPLU_WAKEINT_FILTER_MODE_BYPASS Select Bypass mode.
kPLU_WAKEINT_FILTER_MODE_1_CLK_PERIOD Filter 1 clock period.
kPLU_WAKEINT_FILTER_MODE_2_CLK_PERIOD Filter 2 clock period.
kPLU_WAKEINT_FILTER_MODE_3_CLK_PERIOD Filter 3 clock period.

31.5.8 enum plu_wakeint_filter_clock_source_t

Enumerator

kPLU_WAKEINT_FILTER_CLK_SRC_1MHZ_LPOSC Select the 1MHz low-power oscillator as the filter clock.
kPLU_WAKEINT_FILTER_CLK_SRC_12MHZ_FRO Select the 12MHz FRO as the filer clock.
kPLU_WAKEINT_FILTER_CLK_SRC_ALT Select a third clock source.

31.6 Function Documentation

31.6.1 void PLU_Init (PLU_Type * *base*)

Note

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

Parameters

base	PLU peripheral base address

31.6.2 void PLU_Deinit (PLU_Type * *base*)

Parameters

base	PLU peripheral base address

31.6.3 static void PLU_SetLutInputSource (PLU_Type * *base*, plu_lut_index_t *lutIndex*, plu_lut_in_index_t *lutInIndex*, plu_lut_input_source_t *inputSrc*) [inline], [static]

Note: An external clock must be applied to the PLU_CLKIN input when using FFs. For each LUT, the slot associated with the output from LUTn itself is tied low.

Parameters

base	PLU peripheral base address.
lutIndex	LUT index (see plu_lut_index_t typedef enumeration).
lutInIndex	LUT input index (see plu_lut_in_index_t typedef enumeration).
inputSrc	LUT input source (see plu_lut_input_source_t typedef enumeration).

31.6.4 static void PLU_SetOutputSource (PLU_Type * *base*, plu_output_index_t *outputIndex*, plu_output_source_t *outputSrc*) [inline], [static]

Note: An external clock must be applied to the PLU_CLKIN input when using FFs.

Parameters

base	PLU peripheral base address.
outputIndex	PLU output index (see plu_output_index_t typedef enumeration).
outputSrc	PLU output source (see plu_output_source_t typedef enumeration).

31.6.5 static void PLU_SetLutTruthTable (PLU_Type * *base*, plu_lut_index_t *lutIndex*, uint32_t *truthTable*) [inline], [static]

Parameters

base	PLU peripheral base address.
lutIndex	LUT index (see plu_lut_index_t typedef enumeration).
truthTable	Truth Table value.

31.6.6 static uint32_t PLU_ReadOutputState (PLU_Type * *base*) [inline], [static]

Note: The PLU bus clock must be re-enabled prior to reading the Outpus Register if PLU bus clock is shut-off.

Parameters

base	PLU peripheral base address.

Returns

Current PLU output state value.

31.6.7 void PLU_GetDefaultWakeIntConfig (plu_wakeint_config_t * *config*)

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

```
*   config->filterMode = kPLU_WAKEINT_FILTER_MODE_BYPASS;
*   config->clockSource = kPLU_WAKEINT_FILTER_CLK_SRC_1MHZ_LPOSC;
*
```

Parameters

config	Pointer to configuration structure.

31.6.8 void PLU_EnableWakeIntRequest (PLU_Type * *base*, uint32_t *interruptMask*, const plu_wakeint_config_t * *config*)

This function enables Any of the eight selected PLU outputs to contribute to an asynchronous wake-up or an interrupt request.

Note: If a PLU_CLKIN is provided, the raw wake-up/interrupt request will be set on the rising-edge of the PLU_CLKIN whenever the raw request signal is high. This registered signal will be glitch-free and just use the default wakeint config by PLU_GetDefaultWakeIntConfig(). If not, have to specify the filter

mode and clock source to eliminate the glitches caused by long and widely disparate delays through the network of LUTs making up the PLU. This way may increase power consumption in low-power operating modes and inject delay before the wake-up/interrupt request is generated.

Parameters

base	PLU peripheral base address.
interruptMask	PLU interrupt mask (see _plu_interrupt_mask enumeration).
config	Pointer to configuration structure (see plu_wakeint_config_t typedef enumeration)

31.6.9 static void PLU_LatchInterrupt (PLU_Type * *base*) [inline], [static]

This function latches the interrupt and then it can be cleared with PLU_ClearLatchedInterrupt().

Note: This mode is not compatible with use of the glitch filter. If this bit is set, the FILTER MODE should be set to kPLU_WAKEINT_FILTER_MODE_BYPASS (Bypass Mode) and PLU_CLKIN should be provided. If this bit is set, the wake-up/interrupt request will be set on the rising-edge of PLU_CLKIN whenever the raw wake-up/interrupt signal is high. The request must be cleared by software.

Parameters

base	PLU peripheral base address.

31.6.10 void PLU_ClearLatchedInterrupt (PLU_Type * *base*)

This function clears the wake-up/interrupt request flag latched by PLU_LatchInterrupt()

Note: It is not necessary for the PLU bus clock to be enabled in order to write-to or read-back this bit.

Parameters

base	PLU peripheral base address.

Chapter 32
PRINCE: PRINCE bus crypto engine

32.1 Overview

The MCUXpresso SDK provides a peripheral driver for the PRINCE bus crypto engine module of MCUXpresso SDK devices.

..

This example code shows how to use the PRINCE driver.

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

Enumerations

- enum skboot_status_t {
  kStatus_SKBOOT_Success = 0x5ac3c35au,
  kStatus_SKBOOT_Fail = 0xc35ac35au,
  kStatus_SKBOOT_InvalidArgument = 0xc35a5ac3u,
  kStatus_SKBOOT_KeyStoreMarkerInvalid = 0xc3c35a5au,
  kStatus_SKBOOT_HashcryptFinishedWithStatusSuccess,
  kStatus_SKBOOT_HashcryptFinishedWithStatusFail,
  kStatus_SKBOOT_Success = 0x5ac3c35au,
  kStatus_SKBOOT_Fail = 0xc35ac35au,
  kStatus_SKBOOT_InvalidArgument = 0xc35a5ac3u,
  kStatus_SKBOOT_KeyStoreMarkerInvalid = 0xc3c35a5au }
  Secure status enumeration.
- enum secure_bool_t {
  kSECURE_TRUE = 0xc33cc33cU,
  kSECURE_FALSE = 0x5aa55aa5U,
  kSECURE_CALLPROTECT_SECURITY_FLAGS = 0xc33c5aa5U,
  kSECURE_CALLPROTECT_IS_APP_READY = 0x5aa5c33cU,
  kSECURE_TRACKER_VERIFIED = 0x55aacc33U,
  kSECURE_TRUE = 0xc33cc33cU,
  kSECURE_FALSE = 0x5aa55aa5U }
  Secure boolean enumeration.
- enum prince_region_t {
  kPRINCE_Region0 = 0U,
  kPRINCE_Region1 = 1U,
  kPRINCE_Region2 = 2U }
  Prince region.
- enum prince_lock_t {

kPRINCE_Region0Lock = 1U,
kPRINCE_Region1Lock = 2U,
kPRINCE_Region2Lock = 4U,
kPRINCE_MaskLock = 256U }
Prince lock.

- enum prince_flags_t {
kPRINCE_Flag_None = 0U,
kPRINCE_Flag_EraseCheck = 1U,
kPRINCE_Flag_WriteCheck = 2U }
Prince flag.

Functions

- static void PRINCE_EncryptEnable (PRINCE_Type *base)
Enable data encryption.
- static void PRINCE_EncryptDisable (PRINCE_Type *base)
Disable data encryption.
- static void PRINCE_SetMask (PRINCE_Type *base, uint64_t mask)
Sets PRINCE data mask.
- static void PRINCE_SetLock (PRINCE_Type *base, uint32_t lock)
Locks access for specified region registers or data mask register.
- status_t PRINCE_GenNewIV (prince_region_t region, uint8_t *iv_code, bool store, flash_config_t *flash_context)
Generate new IV code.
- status_t PRINCE_LoadIV (prince_region_t region, uint8_t *iv_code)
Load IV code.
- status_t PRINCE_SetEncryptForAddressRange (prince_region_t region, uint32_t start_address, uint32_t length, flash_config_t *flash_context, bool regenerate_iv)
Allow encryption/decryption for specified address range.
- status_t PRINCE_GetRegionSREnable (PRINCE_Type *base, prince_region_t region, uint32_t *sr-_enable)
Gets the PRINCE Sub-Region Enable register.
- status_t PRINCE_GetRegionBaseAddress (PRINCE_Type *base, prince_region_t region, uint32_t *region_base_addr)
Gets the PRINCE region base address register.
- status_t PRINCE_SetRegionIV (PRINCE_Type *base, prince_region_t region, const uint8_t iv[8])
Sets the PRINCE region IV.
- status_t PRINCE_SetRegionBaseAddress (PRINCE_Type *base, prince_region_t region, uint32_t region_base_addr)
Sets the PRINCE region base address.
- status_t PRINCE_SetRegionSREnable (PRINCE_Type *base, prince_region_t region, uint32_t sr-_enable)
Sets the PRINCE Sub-Region Enable register.
- status_t PRINCE_FlashEraseWithChecker (flash_config_t *config, uint32_t start, uint32_t length-InBytes, uint32_t key)
Erases the flash sectors encompassed by parameters passed into function.
- status_t PRINCE_FlashProgramWithChecker (flash_config_t *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)
Programs flash with data at locations passed in through parameters.

Driver version

- #define FSL_PRINCE_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))
  PRINCE driver version 2.3.2.

32.2 Macro Definition Documentation

32.2.1 #define FSL_PRINCE_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))

Current version: 2.3.2

Change log:

- Version 2.0.0
  - Initial version.
- Version 2.1.0
  - Update for the A1 rev. of LPC55Sxx serie.
- Version 2.2.0
  - Add runtime checking of the A0 and A1 rev. of LPC55Sxx serie to support both silicone revisions.
- Version 2.3.0
  - Add support for LPC55S1x and LPC55S2x series
- Version 2.3.0
  - Fix MISRA-2012 issues.
- Version 2.3.1
  - Add support for LPC55S0x series
- Version 2.3.2
  - Fix documentation of enumeration. Extend PRINCE example.

32.3 Enumeration Type Documentation

32.3.1 enum skboot_status_t

Enumerator

kStatus_SKBOOT_Success SKBOOT return success status.
kStatus_SKBOOT_Fail SKBOOT return fail status.
kStatus_SKBOOT_InvalidArgument SKBOOT return invalid argument status.
kStatus_SKBOOT_KeyStoreMarkerInvalid SKBOOT return Keystore invalid Marker status.
kStatus_SKBOOT_HashcryptFinishedWithStatusSuccess SKBOOT return Hashcrypt finished with the success status.
kStatus_SKBOOT_HashcryptFinishedWithStatusFail SKBOOT return Hashcrypt finished with the fail status.
kStatus_SKBOOT_Success PRINCE Success.
kStatus_SKBOOT_Fail PRINCE Fail.
kStatus_SKBOOT_InvalidArgument PRINCE Invalid argument.
kStatus_SKBOOT_KeyStoreMarkerInvalid PRINCE Invalid marker.

32.3.2 enum secure_bool_t

Enumerator

kSECURE_TRUE Secure true flag.
kSECURE_FALSE Secure false flag.
kSECURE_CALLPROTECT_SECURITY_FLAGS Secure call protect the security flag.
kSECURE_CALLPROTECT_IS_APP_READY Secure call protect the app is ready flag.
kSECURE_TRACKER_VERIFIED Secure tracker verified flag.
kSECURE_TRUE PRINCE true.
kSECURE_FALSE PRINCE false.

32.3.3 enum prince_region_t

Enumerator

kPRINCE_Region0 PRINCE region 0.
kPRINCE_Region1 PRINCE region 1.
kPRINCE_Region2 PRINCE region 2.

32.3.4 enum prince_lock_t

Enumerator

kPRINCE_Region0Lock PRINCE region 0 lock.
kPRINCE_Region1Lock PRINCE region 1 lock.
kPRINCE_Region2Lock PRINCE region 2 lock.
kPRINCE_MaskLock PRINCE mask register lock.

32.3.5 enum prince_flags_t

Enumerator

kPRINCE_Flag_None PRINCE Flag None.
kPRINCE_Flag_EraseCheck PRINCE Flag Erase check.
kPRINCE_Flag_WriteCheck PRINCE Flag Write check.

32.4 Function Documentation

32.4.1 static void PRINCE_EncryptEnable (PRINCE_Type * *base*) [inline], [static]

This function enables PRINCE on-the-fly data encryption.

Parameters

base	PRINCE peripheral address.

32.4.2 static void PRINCE_EncryptDisable (PRINCE_Type * *base*) [inline], [static]

This function disables PRINCE on-the-fly data encryption.

Parameters

base	PRINCE peripheral address.

32.4.3 static void PRINCE_SetMask (PRINCE_Type * *base,* uint64_t *mask*) [inline], [static]

This function sets the PRINCE mask that is used to mask decrypted data.

Parameters

base	PRINCE peripheral address.
mask	64-bit data mask value.

32.4.4 static void PRINCE_SetLock (PRINCE_Type * *base,* uint32_t *lock*) [inline], [static]

This function sets lock on specified region registers or mask register.

Parameters

base	PRINCE peripheral address.
lock	registers to lock. This is a logical OR of members of the enumeration prince_lock_t

32.4.5 status_t PRINCE_GenNewIV (prince_region_t *region,* uint8_t * *iv_code,* bool *store,* flash_config_t * *flash_context*)

This function generates new IV code and stores it into the persistent memory. Ensure about 800 bytes free space on the stack when calling this routine with the store parameter set to true!

Parameters

region	PRINCE region index.
iv_code	IV code pointer used for storing the newly generated 52 bytes long IV code.
store	flag to allow storing the newly generated IV code into the persistent memory (FFR).
flash_context	pointer to the flash driver context structure.

Returns

kStatus_Success upon success
kStatus_Fail otherwise, kStatus_Fail is also returned if the key code for the particular PRINCE region is not present in the keystore (though new IV code has been provided)

32.4.6 status_t PRINCE_LoadIV (prince_region_t *region,* uint8_t * *iv_code*)

This function enables IV code loading into the PRINCE bus encryption engine.

Parameters

region	PRINCE region index.
iv_code	IV code pointer used for passing the IV code.

Returns

kStatus_Success upon success
kStatus_Fail otherwise

32.4.7 status_t PRINCE_SetEncryptForAddressRange (prince_region_t *region,* uint32_t *start_address,* uint32_t *length,* flash_config_t * *flash_context,* bool *regenerate_iv*)

This function sets the encryption/decryption for specified address range. The SR mask value for the selected Prince region is calculated from provided start_address and length parameters. This calculated value is OR'ed with the actual SR mask value and stored into the PRINCE SR_ENABLE register and also into the persistent memory (FFR) to be used after the device reset. It is possible to define several nonadjacent encrypted areas within one Prince region when calling this function repeatedly. If the length parameter is set to 0, the SR mask value is set to 0 and thus the encryption/decryption for the whole selected Prince region is disabled. Ensure about 800 bytes free space on the stack when calling this routine!

Parameters

region	PRINCE region index.
start_address	start address of the area to be encrypted/decrypted.
length	length of the area to be encrypted/decrypted.
flash_context	pointer to the flash driver context structure.
regenerate_iv	flag to allow IV code regenerating, storing into the persistent memory (FFR) and loading into the PRINCE engine

Returns

kStatus_Success upon success
kStatus_Fail otherwise

32.4.8 status_t PRINCE_GetRegionSREnable (PRINCE_Type * *base*, prince_region_t *region*, uint32_t * *sr_enable*)

This function gets PRINCE SR_ENABLE register.

Parameters

base	PRINCE peripheral address.
region	PRINCE region index.
sr_enable	Sub-Region Enable register pointer.

Returns

kStatus_Success upon success
kStatus_InvalidArgument

32.4.9 status_t PRINCE_GetRegionBaseAddress (PRINCE_Type * *base*, prince_region_t *region*, uint32_t * *region_base_addr*)

This function gets PRINCE BASE_ADDR register.

Parameters

base	PRINCE peripheral address.
region	PRINCE region index.
region_base_addr	Region base address pointer.

Returns

kStatus_Success upon success
kStatus_InvalidArgument

32.4.10 status_t PRINCE_SetRegionIV (PRINCE_Type * *base*, prince_region_t *region*, const uint8_t *iv[8]*)

This function sets specified AES IV for the given region.

Parameters

base	PRINCE peripheral address.
region	Selection of the PRINCE region to be configured.
iv	64-bit AES IV in little-endian byte order.

32.4.11 status_t PRINCE_SetRegionBaseAddress (PRINCE_Type * *base*, prince_region_t *region*, uint32_t *region_base_addr*)

This function configures PRINCE region base address.

Parameters

base	PRINCE peripheral address.
region	Selection of the PRINCE region to be configured.
region_base_addr	Base Address for region.

32.4.12 status_t PRINCE_SetRegionSREnable (PRINCE_Type * *base*, prince_region_t *region*, uint32_t *sr_enable*)

This function configures PRINCE SR_ENABLE register.

Parameters

base	PRINCE peripheral address.
region	Selection of the PRINCE region to be configured.
sr_enable	Sub-Region Enable register value.

32.4.13 status_t PRINCE_FlashEraseWithChecker (flash_config_t * *config*, uint32_t *start*, uint32_t *lengthInBytes*, uint32_t *key*)

This function erases the appropriate number of flash sectors based on the desired start address and length. It deals with the flash erase function complenentary to the standard erase API of the IAP1 driver. This implementation additionally checks if the whole encrypted PRINCE subregions are erased at once to avoid secrets revealing. The checker implementation is limited to one contiguous PRINCE-controlled memory area.

Parameters

config	The pointer to the flash driver context structure.
start	The start address of the desired flash memory to be erased. The start address needs to be prince-sburegion-aligned.
lengthInBytes	The length, given in bytes (not words or long-words) to be erased. Must be prince-sburegion-size-aligned.
key	The value used to validate all flash erase APIs.

Returns

kStatus_FLASH_Success API was executed successfully.
kStatus_FLASH_InvalidArgument An invalid argument is provided.
kStatus_FLASH_AlignmentError The parameter is not aligned with the specified baseline.
kStatus_FLASH_AddressError The address is out of range.
kStatus_FLASH_EraseKeyError The API erase key is invalid.
kStatus_FLASH_CommandFailure Run-time error during the command execution.
kStatus_FLASH_CommandNotSupported Flash API is not supported.
kStatus_FLASH_EccError A correctable or uncorrectable error during command execution.
kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce Encrypted flash subregions are not erased at once.

32.4.14 status_t PRINCE_FlashProgramWithChecker (flash_config_t * *config*, uint32_t *start*, uint8_t * *src*, uint32_t *lengthInBytes*)

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length. It deals with the flash program function complenentary to the standard program API of the IAP1 driver. This implementation additionally checks if the whole PRINCE subregions are programmed at once to avoid secrets revealing. The checker implementation is limited to one contiguous PRINCE-controlled memory area.

Parameters

config	The pointer to the flash driver context structure.
start	The start address of the desired flash memory to be programmed. Must be prince-sburegion-aligned.
src	A pointer to the source buffer of data that is to be programmed into the flash.
lengthInBytes	The length, given in bytes (not words or long-words), to be programmed. Must be prince-sburegion-size-aligned.

Returns

kStatus_FLASH_Success API was executed successfully.
kStatus_FLASH_InvalidArgument An invalid argument is provided.
kStatus_FLASH_AlignmentError Parameter is not aligned with the specified baseline.
kStatus_FLASH_AddressError Address is out of range.
kStatus_FLASH_AccessError Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_CommandFailure Run-time error during the command execution.
kStatus_FLASH_CommandFailure Run-time error during the command execution.
kStatus_FLASH_CommandNotSupported Flash API is not supported.
kStatus_FLASH_EccError A correctable or uncorrectable error during command execution.
kStatus_FLASH_SizeError Encrypted flash subregions are not programmed at once.

Chapter 33
PUF: Physical Unclonable Function

33.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Physical Unclonable Function (PUF) module of MCUXpresso SDK devices. The PUF controller provides a secure key storage without injecting or provisioning device unique PUF root key.

Blocking synchronous APIs are provided for generating the activation code, intrinsic key generation, storing and reconstructing keys using PUF hardware. The PUF operations are complete (and results are made availabe for further usage) when a function returns. When called, these functions do not return until an PUF operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

33.2 PUF Driver Initialization and deinitialization

PUF Driver is initialized by calling the PUF_Init() function, it resets the PUF module, enables it's clock and enables power to PUF SRAM. PUF Driver is deinitialized by calling the PUF_Deinit() function, it disables PUF module clock, asserts peripheral reset and disables power to PUF SRAM.

33.3 Comments about API usage in RTOS

PUF operations provided by this driver are not re-entrant. Thus, application software shall ensure the PUF module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

33.4 Comments about API usage in interrupt handler

All APIs can be used from interrupt handler although execution time shall be considered (interrupt latency of equal and lower priority interrupts increases).

33.5 PUF Driver Examples

33.5.1 Simple examples

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

Macros

- #define PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(x) ((160u + (((((x) << 3) + 255u) >> 8) << 8)) >> 3)
  Get Key Code size in bytes from key size in bytes at compile time.

Enumerations

- enum puf_key_slot_t {
  kPUF_KeySlot0 = 0U,
  kPUF_KeySlot1 = 1U }
  PUF key slot.
- enum
  PUF status return codes.

Driver version

- #define FSL_PUF_DRIVER_VERSION (MAKE_VERSION(2, 1, 6))
  PUF driver version.

33.6 Macro Definition Documentation

33.6.1 #define FSL_PUF_DRIVER_VERSION (MAKE_VERSION(2, 1, 6))

Version 2.1.6.

Current version: 2.1.6

Change log:

- 2.0.0
  - Initial version.
- 2.0.1
  - Fixed puf_wait_usec function optimization issue.
- 2.0.2
  - Add PUF configuration structure and support for PUF SRAM controller. Remove magic constants.
- 2.0.3
  - Fix MISRA C-2012 issue.
- 2.1.0
  - Align driver with PUF SRAM controller registers on LPCXpresso55s16.
  - Update initizalition logic .
- 2.1.1
  - Fix ARMGCC build warning .
- 2.1.2
  - Update: Add automatic big to little endian swap for user (pre-shared) keys destinated to secret hardware bus (PUF key index 0).
- 2.1.3
  - Fix MISRA C-2012 issue.
- 2.1.4
  - Replace register uint32_t ticksCount with volatile uint32_t ticksCount in puf_wait_usec() to prevent optimization out delay loop.
- 2.1.5
  - Use common SDK delay in puf_wait_usec()

- 2.1.6
  - Changed wait time in PUF_Init(), when initialization fails it will try PUF_Powercycle() with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.

33.6.2 #define PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(*x*) ((160u + (((((x) << 3) + 255u) >> 8) << 8)) >> 3)

33.7 Enumeration Type Documentation

33.7.1 enum puf_key_slot_t

Enumerator

kPUF_KeySlot0 PUF key slot 0.
kPUF_KeySlot1 PUF key slot 1.

33.7.2 anonymous enum

Chapter 34
RNG: Random Number Generator

34.1 Overview

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

The Random Number Generator is a hardware module that generates 32-bit random numbers. A typical consumer is a pseudo random number generator (PRNG) which can be implemented to achieve both true randomness and cryptographic strength random numbers using the RNG output as its entropy seed. The data generated by a RNG is intended for direct use by functions that generate secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

34.2 Get random data from RNG

1. RNG_GetRandomData() function gets random data from the RNG module.

This example code shows how to get 128-bit random data from the RNG driver.

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

Functions

- void RNG_Init (RNG_Type *base)
  Initializes the RNG.
- void RNG_Deinit (RNG_Type *base)
  Shuts down the RNG.
- status_t RNG_GetRandomData (RNG_Type *base, void *data, size_t dataSize)
  Gets random data.
- static uint32_t RNG_GetRandomWord (RNG_Type *base)
  Returns random 32-bit number.

Driver version

- #define FSL_RNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  RNG driver version.

34.3 Macro Definition Documentation

34.3.1 #define FSL_RNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

Version 2.0.3.

Current version: 2.0.3

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Fix MISRA C-2012 issue.
- Version 2.0.2
  - Add RESET_PeripheralReset function inside RNG_Init and RNG_Deinit functions.
- Version 2.0.3
  - Modified RNG_Init and RNG_GetRandomData functions, added rng_accumulateEntropy and rng_readEntropy functions.
  - These changes are reflecting recommended usage of RNG according to device UM.

34.4 Function Documentation

34.4.1 void RNG_Init (RNG_Type * *base*)

This function initializes the RNG. When called, the RNG module and ring oscillator is enabled.

Parameters

base	RNG base address

Returns

If successful, returns the kStatus_RNG_Success. Otherwise, it returns an error.

34.4.2 void RNG_Deinit (RNG_Type * *base*)

This function shuts down the RNG.

Parameters

base	RNG base address.

34.4.3 status_t RNG_GetRandomData (RNG_Type * *base,* void * *data,* size_t *dataSize*)

This function gets random data from the RNG.

Parameters

base	RNG base address.
data	Pointer address used to store random data.
dataSize	Size of the buffer pointed by the data parameter.

Returns

random data

34.4.4 static uint32_t RNG_GetRandomWord (RNG_Type * *base*) [inline], [static]

This function gets random number from the RNG.

Parameters

base	RNG base address.

Returns

random number

Chapter 35
SCTimer: SCTimer/PWM (SCT)

35.1 Overview

The MCUXpresso SDK provides a driver for the SCTimer Module (SCT) of MCUXpresso SDK devices.

35.2 Function groups

The SCTimer driver supports the generation of PWM signals. The driver also supports enabling events in various states of the SCTimer and the actions that will be triggered when an event occurs.

35.2.1 Initialization and deinitialization

The function SCTIMER_Init() initializes the SCTimer with specified configurations. The function SCTIMER_GetDefaultConfig() gets the default configurations.

The function SCTIMER_Deinit() halts the SCTimer counter and turns off the module clock.

35.2.2 PWM Operations

The function SCTIMER_SetupPwm() sets up SCTimer channels for PWM output. The function can set up the PWM signal properties duty cycle and level-mode (active low or high) to use. However, the same PWM period and PWM mode (edge or center-aligned) is applied to all channels requesting the PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 1 and 100.

The function SCTIMER_UpdatePwmDutycycle() updates the PWM signal duty cycle of a particular SCTimer channel.

35.2.3 Status

Provides functions to get and clear the SCTimer status.

35.2.4 Interrupt

Provides functions to enable/disable SCTimer interrupts and get current enabled interrupts.

35.3 SCTimer State machine and operations

The SCTimer has 10 states and each state can have a set of events enabled that can trigger a user specified action when the event occurs.

35.3.1 SCTimer event operations

The user can create an event and enable it in the current state using the functions SCTIMER_CreateAndScheduleEvent() and SCTIMER_ScheduleEvent(). SCTIMER_CreateAndScheduleEvent() creates a new event based on the users preference and enables it in the current state. SCTIMER_ScheduleEvent() enables an event created earlier in the current state.

35.3.2 SCTimer state operations

The user can get the current state number by calling SCTIMER_GetCurrentState(), he can use this state number to set state transitions when a particular event is triggered.

Once the user has created and enabled events for the current state he can go to the next state by calling the function SCTIMER_IncreaseState(). The user can then start creating events to be enabled in this new state.

35.3.3 SCTimer action operations

There are a set of functions that decide what action should be taken when an event is triggered. SCTIMER_SetupCaptureAction() sets up which counter to capture and which capture register to read on event trigger. SCTIMER_SetupNextStateAction() sets up which state the SCTimer state machine should transition to on event trigger. SCTIMER_SetupOutputSetAction() sets up which pin to set on event trigger. SCTIMER_SetupOutputClearAction() sets up which pin to clear on event trigger. SCTIMER_SetupOutputToggleAction() sets up which pin to toggle on event trigger. SCTIMER_SetupCounterLimitAction() sets up which counter will be limited on event trigger. SCTIMER_SetupCounterStopAction() sets up which counter will be stopped on event trigger. SCTIMER_SetupCounterStartAction() sets up which counter will be started on event trigger. SCTIMER_SetupCounterHaltAction() sets up which counter will be halted on event trigger. SCTIMER_SetupDmaTriggerAction() sets up which DMA request will be activated on event trigger.

35.4 16-bit counter mode

The SCTimer is configurable to run as two 16-bit counters via the enableCounterUnify flag that is available in the configuration structure passed in to the SCTIMER_Init() function.

When operating in 16-bit mode, it is important the user specify the appropriate counter to use when working with the functions: SCTIMER_StartTimer(), SCTIMER_StopTimer(), SCTIMER_CreateAndScheduleEvent(), SCTIMER_SetupCaptureAction(), SCTIMER_SetupCounterLimitAction(), SCTIM-

ER_SetupCounterStopAction(), SCTIMER_SetupCounterStartAction(), and SCTIMER_SetupCounter-HaltAction().

35.5 Typical use case

35.5.1 PWM output

Output a PWM signal on 2 SCTimer channels with different duty cycles. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/sctimer

Files

- file fsl_sctimer.h

Data Structures

- struct sctimer_pwm_signal_param_t
  Options to configure a SCTimer PWM signal. More...
- struct sctimer_config_t
  SCTimer configuration structure. More...

Typedefs

- typedef void(∗ sctimer_event_callback_t)(void)
  SCTimer callback typedef.

Enumerations

- enum sctimer_pwm_mode_t {
  kSCTIMER_EdgeAlignedPwm = 0U,
  kSCTIMER_CenterAlignedPwm }
  SCTimer PWM operation modes.
- enum sctimer_counter_t {
  kSCTIMER_Counter_L = (1U << 0),
  kSCTIMER_Counter_H = (1U << 1),
  kSCTIMER_Counter_U = (1U << 2) }
  SCTimer counters type.
- enum sctimer_input_t {
  kSCTIMER_Input_0 = 0U,
  kSCTIMER_Input_1,
  kSCTIMER_Input_2,
  kSCTIMER_Input_3,
  kSCTIMER_Input_4,
  kSCTIMER_Input_5,
  kSCTIMER_Input_6,
  kSCTIMER_Input_7 }
  List of SCTimer input pins.

- enum sctimer_out_t {
  kSCTIMER_Out_0 = 0U,
  kSCTIMER_Out_1,
  kSCTIMER_Out_2,
  kSCTIMER_Out_3,
  kSCTIMER_Out_4,
  kSCTIMER_Out_5,
  kSCTIMER_Out_6,
  kSCTIMER_Out_7,
  kSCTIMER_Out_8,
  kSCTIMER_Out_9 }
  List of SCTimer output pins.
- enum sctimer_pwm_level_select_t {
  kSCTIMER_LowTrue = 0U,
  kSCTIMER_HighTrue }
  SCTimer PWM output pulse mode: high-true, low-true or no output.
- enum sctimer_clock_mode_t {
  kSCTIMER_System_ClockMode = 0U,
  kSCTIMER_Sampled_ClockMode,
  kSCTIMER_Input_ClockMode,
  kSCTIMER_Asynchronous_ClockMode }
  SCTimer clock mode options.
- enum sctimer_clock_select_t {
  kSCTIMER_Clock_On_Rise_Input_0 = 0U,
  kSCTIMER_Clock_On_Fall_Input_0,
  kSCTIMER_Clock_On_Rise_Input_1,
  kSCTIMER_Clock_On_Fall_Input_1,
  kSCTIMER_Clock_On_Rise_Input_2,
  kSCTIMER_Clock_On_Fall_Input_2,
  kSCTIMER_Clock_On_Rise_Input_3,
  kSCTIMER_Clock_On_Fall_Input_3,
  kSCTIMER_Clock_On_Rise_Input_4,
  kSCTIMER_Clock_On_Fall_Input_4,
  kSCTIMER_Clock_On_Rise_Input_5,
  kSCTIMER_Clock_On_Fall_Input_5,
  kSCTIMER_Clock_On_Rise_Input_6,
  kSCTIMER_Clock_On_Fall_Input_6,
  kSCTIMER_Clock_On_Rise_Input_7,
  kSCTIMER_Clock_On_Fall_Input_7 }
  SCTimer clock select options.
- enum sctimer_conflict_resolution_t {
  kSCTIMER_ResolveNone = 0U,
  kSCTIMER_ResolveSet,
  kSCTIMER_ResolveClear,
  kSCTIMER_ResolveToggle }
  SCTimer output conflict resolution options.

- enum sctimer_event_active_direction_t {
  kSCTIMER_ActiveIndependent = 0U,
  kSCTIMER_ActiveInCountUp,
  kSCTIMER_ActiveInCountDown }
  List of SCTimer event generation active direction when the counters are operating in BIDIR mode.
- enum sctimer_event_t
  List of SCTimer event types.
- enum sctimer_interrupt_enable_t {
  kSCTIMER_Event0InterruptEnable = (1U << 0),
  kSCTIMER_Event1InterruptEnable = (1U << 1),
  kSCTIMER_Event2InterruptEnable = (1U << 2),
  kSCTIMER_Event3InterruptEnable = (1U << 3),
  kSCTIMER_Event4InterruptEnable = (1U << 4),
  kSCTIMER_Event5InterruptEnable = (1U << 5),
  kSCTIMER_Event6InterruptEnable = (1U << 6),
  kSCTIMER_Event7InterruptEnable = (1U << 7),
  kSCTIMER_Event8InterruptEnable = (1U << 8),
  kSCTIMER_Event9InterruptEnable = (1U << 9),
  kSCTIMER_Event10InterruptEnable = (1U << 10),
  kSCTIMER_Event11InterruptEnable = (1U << 11),
  kSCTIMER_Event12InterruptEnable = (1U << 12) }
  List of SCTimer interrupts.
- enum sctimer_status_flags_t {
  kSCTIMER_Event0Flag = (1U << 0),
  kSCTIMER_Event1Flag = (1U << 1),
  kSCTIMER_Event2Flag = (1U << 2),
  kSCTIMER_Event3Flag = (1U << 3),
  kSCTIMER_Event4Flag = (1U << 4),
  kSCTIMER_Event5Flag = (1U << 5),
  kSCTIMER_Event6Flag = (1U << 6),
  kSCTIMER_Event7Flag = (1U << 7),
  kSCTIMER_Event8Flag = (1U << 8),
  kSCTIMER_Event9Flag = (1U << 9),
  kSCTIMER_Event10Flag = (1U << 10),
  kSCTIMER_Event11Flag = (1U << 11),
  kSCTIMER_Event12Flag = (1U << 12),
  kSCTIMER_BusErrorLFlag,
  kSCTIMER_BusErrorHFlag }
  List of SCTimer flags.

Driver version

- #define FSL_SCTIMER_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
  Version.

Initialization and deinitialization

- status_t SCTIMER_Init (SCT_Type *base, const sctimer_config_t *config)
  Ungates the SCTimer clock and configures the peripheral for basic operation.
- void SCTIMER_Deinit (SCT_Type *base)
  Gates the SCTimer clock.
- void SCTIMER_GetDefaultConfig (sctimer_config_t *config)
  Fills in the SCTimer configuration structure with the default settings.

PWM setup operations

- status_t SCTIMER_SetupPwm (SCT_Type *base, const sctimer_pwm_signal_param_t *pwmParams, sctimer_pwm_mode_t mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint32_t *event)
  Configures the PWM signal parameters.
- void SCTIMER_UpdatePwmDutycycle (SCT_Type *base, sctimer_out_t output, uint8_t dutyCyclePercent, uint32_t event)
  Updates the duty cycle of an active PWM signal.

Interrupt Interface

- static void SCTIMER_EnableInterrupts (SCT_Type *base, uint32_t mask)
  Enables the selected SCTimer interrupts.
- static void SCTIMER_DisableInterrupts (SCT_Type *base, uint32_t mask)
  Disables the selected SCTimer interrupts.
- static uint32_t SCTIMER_GetEnabledInterrupts (SCT_Type *base)
  Gets the enabled SCTimer interrupts.

Status Interface

- static uint32_t SCTIMER_GetStatusFlags (SCT_Type *base)
  Gets the SCTimer status flags.
- static void SCTIMER_ClearStatusFlags (SCT_Type *base, uint32_t mask)
  Clears the SCTimer status flags.

Counter Start and Stop

- static void SCTIMER_StartTimer (SCT_Type *base, uint32_t countertoStart)
  Starts the SCTimer counter.
- static void SCTIMER_StopTimer (SCT_Type *base, uint32_t countertoStop)
  Halts the SCTimer counter.

Functions to create a new event and manage the state logic

- status_t SCTIMER_CreateAndScheduleEvent (SCT_Type *base, sctimer_event_t howToMonitor, uint32_t matchValue, uint32_t whichIO, sctimer_counter_t whichCounter, uint32_t *event)
  Create an event that is triggered on a match or IO and schedule in current state.
- void SCTIMER_ScheduleEvent (SCT_Type *base, uint32_t event)
  Enable an event in the current state.
- status_t SCTIMER_IncreaseState (SCT_Type *base)

Increase the state by 1.
- uint32_t SCTIMER_GetCurrentState (SCT_Type *base)
  Provides the current state.
- static void SCTIMER_SetCounterState (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t state)
  Set the counter current state.
- static uint16_t SCTIMER_GetCounterState (SCT_Type *base, sctimer_counter_t whichCounter)
  Get the counter current state value.

Actions to take in response to an event

- status_t SCTIMER_SetupCaptureAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t *captureRegister, uint32_t event)
  Setup capture of the counter value on trigger of a selected event.
- void SCTIMER_SetCallback (SCT_Type *base, sctimer_event_callback_t callback, uint32_t event)
  Receive noticification when the event trigger an interrupt.
- static void SCTIMER_SetupStateLdMethodAction (SCT_Type *base, uint32_t event, bool fgLoad)
  Change the load method of transition to the specified state.
- static void SCTIMER_SetupNextStateActionwithLdMethod (SCT_Type *base, uint32_t nextState, uint32_t event, bool fgLoad)
  Transition to the specified state with Load method.
- static void SCTIMER_SetupNextStateAction (SCT_Type *base, uint32_t nextState, uint32_t event)
  Transition to the specified state.
- static void SCTIMER_SetupEventActiveDirection (SCT_Type *base, sctimer_event_active_direction_t activeDirection, uint32_t event)
  Setup event active direction when the counters are operating in BIDIR mode.
- static void SCTIMER_SetupOutputSetAction (SCT_Type *base, uint32_t whichIO, uint32_t event)
  Set the Output.
- static void SCTIMER_SetupOutputClearAction (SCT_Type *base, uint32_t whichIO, uint32_t event)
  Clear the Output.
- void SCTIMER_SetupOutputToggleAction (SCT_Type *base, uint32_t whichIO, uint32_t event)
  Toggle the output level.
- static void SCTIMER_SetupCounterLimitAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)
  Limit the running counter.
- static void SCTIMER_SetupCounterStopAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)
  Stop the running counter.
- static void SCTIMER_SetupCounterStartAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)
  Re-start the stopped counter.
- static void SCTIMER_SetupCounterHaltAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)
  Halt the running counter.
- static void SCTIMER_SetupDmaTriggerAction (SCT_Type *base, uint32_t dmaNumber, uint32_t event)
  Generate a DMA request.
- static void SCTIMER_SetCOUNTValue (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t value)

Set the value of counter.
- static uint32_t SCTIMER_GetCOUNTValue (SCT_Type *base, sctimer_counter_t whichCounter)
  Get the value of counter.
- static void SCTIMER_SetEventInState (SCT_Type *base, uint32_t event, uint32_t state)
  Set the state mask bit field of EV_STATE register.
- static void SCTIMER_ClearEventInState (SCT_Type *base, uint32_t event, uint32_t state)
  Clear the state mask bit field of EV_STATE register.
- static bool SCTIMER_GetEventInState (SCT_Type *base, uint32_t event, uint32_t state)
  Get the state mask bit field of EV_STATE register.
- void SCTIMER_EventHandleIRQ (SCT_Type *base)
  SCTimer interrupt handler.

35.6 Data Structure Documentation

35.6.1 struct sctimer_pwm_signal_param_t

Data Fields

- sctimer_out_t output
  The output pin to use to generate the PWM signal.
- sctimer_pwm_level_select_t level
  PWM output active level select.
- uint8_t dutyCyclePercent
  PWM pulse width, value should be between 0 to 100 0 = always inactive signal (0% duty cycle) 100 = always active signal (100% duty cycle).

Field Documentation

(1) sctimer_pwm_level_select_t sctimer_pwm_signal_param_t::level

(2) uint8_t sctimer_pwm_signal_param_t::dutyCyclePercent

35.6.2 struct sctimer_config_t

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

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

Data Fields

- bool enableCounterUnify
  true: SCT operates as a unified 32-bit counter; false: SCT operates as two 16-bit counters.
- sctimer_clock_mode_t clockMode
  SCT clock mode value.
- sctimer_clock_select_t clockSelect
  SCT clock select value.

- bool enableBidirection_l
  true: Up-down count mode for the L or unified counter false: Up count mode only for the L or unified counter
- bool enableBidirection_h
  true: Up-down count mode for the H or unified counter false: Up count mode only for the H or unified counter.
- uint8_t prescale_l
  Prescale value to produce the L or unified counter clock.
- uint8_t prescale_h
  Prescale value to produce the H counter clock.
- uint8_t outInitState
  Defines the initial output value.
- uint8_t inputsync
  SCT INSYNC value, INSYNC field in the CONFIG register, from bit9 to bit 16.

Field Documentation

(1) bool sctimer_config_t::enableCounterUnify

User can use the 16-bit low counter and the 16-bit high counters at the same time; for Hardware limit, user can not use unified 32-bit counter and any 16-bit low/high counter at the same time.

(2) bool sctimer_config_t::enableBidirection_h

This field is used only if the enableCounterUnify is set to false

(3) uint8_t sctimer_config_t::prescale_h

This field is used only if the enableCounterUnify is set to false

(4) uint8_t sctimer_config_t::inputsync

it is used to define synchronization for input N: bit 9 = input 0 bit 10 = input 1 bit 11 = input 2 bit 12 = input 3 All other bits are reserved (bit13 ∼bit 16). How User to set the the value for the member inputsync. IE: delay for input0, and input 1, bypasses for input 2 and input 3 MACRO definition in user level. #define INPUTSYNC0 (0U) #define INPUTSYNC1 (1U) #define INPUTSYNC2 (2U) #define INPUTSYNC3 (3U) User Code. sctimerInfo.inputsync = (1 << INPUTSYNC2) | (1 << INPUTSYNC3);

35.7 Typedef Documentation

35.7.1 typedef void(∗ sctimer_event_callback_t)(void)

35.8 Enumeration Type Documentation

35.8.1 enum sctimer_pwm_mode_t

Enumerator

kSCTIMER_EdgeAlignedPwm Edge-aligned PWM.
kSCTIMER_CenterAlignedPwm Center-aligned PWM.

35.8.2 enum sctimer_counter_t

Enumerator

kSCTIMER_Counter_L 16-bit Low counter.
kSCTIMER_Counter_H 16-bit High counter.
kSCTIMER_Counter_U 32-bit Unified counter.

35.8.3 enum sctimer_input_t

Enumerator

kSCTIMER_Input_0 SCTIMER input 0.
kSCTIMER_Input_1 SCTIMER input 1.
kSCTIMER_Input_2 SCTIMER input 2.
kSCTIMER_Input_3 SCTIMER input 3.
kSCTIMER_Input_4 SCTIMER input 4.
kSCTIMER_Input_5 SCTIMER input 5.
kSCTIMER_Input_6 SCTIMER input 6.
kSCTIMER_Input_7 SCTIMER input 7.

35.8.4 enum sctimer_out_t

Enumerator

kSCTIMER_Out_0 SCTIMER output 0.
kSCTIMER_Out_1 SCTIMER output 1.
kSCTIMER_Out_2 SCTIMER output 2.
kSCTIMER_Out_3 SCTIMER output 3.
kSCTIMER_Out_4 SCTIMER output 4.
kSCTIMER_Out_5 SCTIMER output 5.
kSCTIMER_Out_6 SCTIMER output 6.
kSCTIMER_Out_7 SCTIMER output 7.
kSCTIMER_Out_8 SCTIMER output 8.
kSCTIMER_Out_9 SCTIMER output 9.

35.8.5 enum sctimer_pwm_level_select_t

Enumerator

kSCTIMER_LowTrue Low true pulses.
kSCTIMER_HighTrue High true pulses.

35.8.6 enum sctimer_clock_mode_t

Enumerator

kSCTIMER_System_ClockMode System Clock Mode.
kSCTIMER_Sampled_ClockMode Sampled System Clock Mode.
kSCTIMER_Input_ClockMode SCT Input Clock Mode.
kSCTIMER_Asynchronous_ClockMode Asynchronous Mode.

35.8.7 enum sctimer_clock_select_t

Enumerator

kSCTIMER_Clock_On_Rise_Input_0 Rising edges on input 0.
kSCTIMER_Clock_On_Fall_Input_0 Falling edges on input 0.
kSCTIMER_Clock_On_Rise_Input_1 Rising edges on input 1.
kSCTIMER_Clock_On_Fall_Input_1 Falling edges on input 1.
kSCTIMER_Clock_On_Rise_Input_2 Rising edges on input 2.
kSCTIMER_Clock_On_Fall_Input_2 Falling edges on input 2.
kSCTIMER_Clock_On_Rise_Input_3 Rising edges on input 3.
kSCTIMER_Clock_On_Fall_Input_3 Falling edges on input 3.
kSCTIMER_Clock_On_Rise_Input_4 Rising edges on input 4.
kSCTIMER_Clock_On_Fall_Input_4 Falling edges on input 4.
kSCTIMER_Clock_On_Rise_Input_5 Rising edges on input 5.
kSCTIMER_Clock_On_Fall_Input_5 Falling edges on input 5.
kSCTIMER_Clock_On_Rise_Input_6 Rising edges on input 6.
kSCTIMER_Clock_On_Fall_Input_6 Falling edges on input 6.
kSCTIMER_Clock_On_Rise_Input_7 Rising edges on input 7.
kSCTIMER_Clock_On_Fall_Input_7 Falling edges on input 7.

35.8.8 enum sctimer_conflict_resolution_t

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

Enumerator

kSCTIMER_ResolveNone No change.
kSCTIMER_ResolveSet Set output.
kSCTIMER_ResolveClear Clear output.
kSCTIMER_ResolveToggle Toggle output.

35.8.9 enum sctimer_event_active_direction_t

Enumerator

kSCTIMER_ActiveIndependent This event is triggered regardless of the count direction.
kSCTIMER_ActiveInCountUp This event is triggered only during up-counting when BIDIR = 1.
kSCTIMER_ActiveInCountDown This event is triggered only during down-counting when BIDIR = 1.

35.8.10 enum sctimer_interrupt_enable_t

Enumerator

kSCTIMER_Event0InterruptEnable Event 0 interrupt.
kSCTIMER_Event1InterruptEnable Event 1 interrupt.
kSCTIMER_Event2InterruptEnable Event 2 interrupt.
kSCTIMER_Event3InterruptEnable Event 3 interrupt.
kSCTIMER_Event4InterruptEnable Event 4 interrupt.
kSCTIMER_Event5InterruptEnable Event 5 interrupt.
kSCTIMER_Event6InterruptEnable Event 6 interrupt.
kSCTIMER_Event7InterruptEnable Event 7 interrupt.
kSCTIMER_Event8InterruptEnable Event 8 interrupt.
kSCTIMER_Event9InterruptEnable Event 9 interrupt.
kSCTIMER_Event10InterruptEnable Event 10 interrupt.
kSCTIMER_Event11InterruptEnable Event 11 interrupt.
kSCTIMER_Event12InterruptEnable Event 12 interrupt.

35.8.11 enum sctimer_status_flags_t

Enumerator

kSCTIMER_Event0Flag Event 0 Flag.
kSCTIMER_Event1Flag Event 1 Flag.
kSCTIMER_Event2Flag Event 2 Flag.
kSCTIMER_Event3Flag Event 3 Flag.
kSCTIMER_Event4Flag Event 4 Flag.
kSCTIMER_Event5Flag Event 5 Flag.
kSCTIMER_Event6Flag Event 6 Flag.
kSCTIMER_Event7Flag Event 7 Flag.
kSCTIMER_Event8Flag Event 8 Flag.
kSCTIMER_Event9Flag Event 9 Flag.
kSCTIMER_Event10Flag Event 10 Flag.
kSCTIMER_Event11Flag Event 11 Flag.

kSCTIMER_Event12Flag Event 12 Flag.
kSCTIMER_BusErrorLFlag Bus error due to write when L counter was not halted.
kSCTIMER_BusErrorHFlag Bus error due to write when H counter was not halted.

35.9 Function Documentation

35.9.1 status_t SCTIMER_Init (SCT_Type * *base*, const sctimer_config_t * *config*)

Note

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

Parameters

base	SCTimer peripheral base address
config	Pointer to the user configuration structure.

Returns

kStatus_Success indicates success; Else indicates failure.

35.9.2 void SCTIMER_Deinit (SCT_Type * *base*)

Parameters

base	SCTimer peripheral base address

35.9.3 void SCTIMER_GetDefaultConfig (sctimer_config_t * *config*)

The default values are:

```
*  config->enableCounterUnify = true;
*  config->clockMode = kSCTIMER_System_ClockMode;
*  config->clockSelect = kSCTIMER_Clock_On_Rise_Input_0;
*  config->enableBidirection_l = false;
*  config->enableBidirection_h = false;
*  config->prescale_l = 0U;
*  config->prescale_h = 0U;
*  config->outInitState = 0U;
*  config->inputsync  = 0xFU;
*
```

Parameters

config	Pointer to the user configuration structure.

35.9.4 status_t SCTIMER_SetupPwm (SCT_Type * *base*, const sctimer_pwm_signal_param_t * *pwmParams*, sctimer_pwm_mode_t *mode*, uint32_t *pwmFreq_Hz*, uint32_t *srcClock_Hz*, uint32_t * *event*)

Call this function to configure the PWM signal period, mode, duty cycle, and edge. This function will create 2 events; one of the events will trigger on match with the pulse value and the other will trigger when the counter matches the PWM period. The PWM period event is also used as a limit event to reset the counter or change direction. Both events are enabled for the same state. The state number can be retrieved by calling the function SCTIMER_GetCurrentStateNumber(). The counter is set to operate as one 32-bit counter (unify bit is set to 1). The counter operates in bi-directional mode when generating a center-aligned PWM.

Note

When setting PWM output from multiple output pins, they all should use the same PWM mode i.e all PWM's should be either edge-aligned or center-aligned. When using this API, the PWM signal frequency of all the initialized channels must be the same. Otherwise all the initialized channels' PWM signal frequency is equal to the last call to the API's pwmFreq_Hz.

Parameters

base	SCTimer peripheral base address
pwmParams	PWM parameters to configure the output
mode	PWM operation mode, options available in enumeration sctimer_pwm_mode_t
pwmFreq_Hz	PWM signal frequency in Hz
srcClock_Hz	SCTimer counter clock in Hz
event	Pointer to a variable where the PWM period event number is stored

Returns

kStatus_Success on success kStatus_Fail If we have hit the limit in terms of number of events created or if an incorrect PWM dutycylce is passed in.

35.9.5 void SCTIMER_UpdatePwmDutycycle (SCT_Type * *base*, sctimer_out_t *output*, uint8_t *dutyCyclePercent*, uint32_t *event*)

Before calling this function, the counter is set to operate as one 32-bit counter (unify bit is set to 1).

Parameters

base	SCTimer peripheral base address
output	The output to configure
dutyCycle-Percent	New PWM pulse width; the value should be between 1 to 100
event	Event number associated with this PWM signal. This was returned to the user by the function SCTIMER_SetupPwm().

35.9.6 static void SCTIMER_EnableInterrupts (SCT_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SCTimer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration sctimer-_interrupt_enable_t

35.9.7 static void SCTIMER_DisableInterrupts (SCT_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SCTimer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration sctimer-_interrupt_enable_t

35.9.8 static uint32_t SCTIMER_GetEnabledInterrupts (SCT_Type * *base*) [inline], [static]

Parameters

base	SCTimer peripheral base address

Returns

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

35.9.9 static uint32_t SCTIMER_GetStatusFlags (SCT_Type * *base*) [inline], [static]

Parameters

base	SCTimer peripheral base address

Returns

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

35.9.10 static void SCTIMER_ClearStatusFlags (SCT_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SCTimer peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration sctimer_status_flags_t

35.9.11 static void SCTIMER_StartTimer (SCT_Type * *base*, uint32_t *countertoStart*) [inline], [static]

Note

In 16-bit mode, we can enable both Counter_L and Counter_H, In 32-bit mode, we only can select Counter_U.

Parameters

base	SCTimer peripheral base address
countertoStart	The SCTimer counters to enable. This is a logical OR of members of the enumeration sctimer_counter_t.

35.9.12 static void SCTIMER_StopTimer (SCT_Type * *base*, uint32_t *countertoStop*) [inline], [static]

Parameters

base	SCTimer peripheral base address
countertoStop	The SCTimer counters to stop. This is a logical OR of members of the enumeration sctimer_counter_t.

35.9.13 status_t SCTIMER_CreateAndScheduleEvent (SCT_Type * *base*, sctimer_event_t *howToMonitor*, uint32_t *matchValue*, uint32_t *whichIO*, sctimer_counter_t *whichCounter*, uint32_t * *event*)

This function will configure an event using the options provided by the user. If the event type uses the counter match, then the function will set the user provided match value into a match register and put this match register number into the event control register. The event is enabled for the current state and the event number is increased by one at the end. The function returns the event number; this event number can be used to configure actions to be done when this event is triggered.

Parameters

base	SCTimer peripheral base address
howToMonitor	Event type; options are available in the enumeration sctimer_interrupt_enable_t
matchValue	The match value that will be programmed to a match register
whichIO	The input or output that will be involved in event triggering. This field is ignored if the event type is "match only"

whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
event	Pointer to a variable where the new event number is stored

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of events created or if we have reached the limit in terms of number of match registers

35.9.14 void SCTIMER_ScheduleEvent (SCT_Type * *base*, uint32_t *event*)

This function will allow the event passed in to trigger in the current state. The event must be created earlier by either calling the function SCTIMER_SetupPwm() or function SCTIMER_CreateAndScheduleEvent() .

Parameters

base	SCTimer peripheral base address
event	Event number to enable in the current state

35.9.15 status_t SCTIMER_IncreaseState (SCT_Type * *base*)

All future events created by calling the function SCTIMER_ScheduleEvent() will be enabled in this new state.

Parameters

base	SCTimer peripheral base address

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of states used

35.9.16 uint32_t SCTIMER_GetCurrentState (SCT_Type * *base*)

User can use this to set the next state by calling the function SCTIMER_SetupNextStateAction().

Parameters

base	SCTimer peripheral base address

Returns

The current state

35.9.17 static void SCTIMER_SetCounterState (SCT_Type * *base*, sctimer_counter_t *whichCounter*, uint32_t *state*) [inline], [static]

The function is to set the state variable bit field of STATE register. Writing to the STATE_L, STATE_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
state	The counter current state number (only support range from 0∼31).

35.9.18 static uint16_t SCTIMER_GetCounterState (SCT_Type * *base*, sctimer_counter_t *whichCounter*) [inline], [static]

The function is to get the state variable bit field of STATE register.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The the counter current state value.

35.9.19 status_t SCTIMER_SetupCaptureAction (SCT_Type * *base*, sctimer_counter_t *whichCounter*, uint32_t * *captureRegister*, uint32_t *event*)

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
captureRegister	Pointer to a variable where the capture register number will be returned. User can read the captured value from this register when the specified event is triggered.
event	Event number that will trigger the capture

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of match/capture registers available

35.9.20 void SCTIMER_SetCallback (SCT_Type * *base,* sctimer_event_callback_t *callback,* uint32_t *event*)

If the interrupt for the event is enabled by the user, then a callback can be registered which will be invoked when the event is triggered

Parameters

base	SCTimer peripheral base address
event	Event number that will trigger the interrupt
callback	Function to invoke when the event is triggered

35.9.21 static void SCTIMER_SetupStateLdMethodAction (SCT_Type * *base,* uint32_t *event,* bool *fgLoad*) [inline], [static]

Change the load method of transition, it will be triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address

event	Event number that will change the method to trigger the state transition
fgLoad	The method to load highest-numbered event occurring for that state to the STATE register. • true: Load the STATEV value to STATE when the event occurs to be the next state. • false: Add the STATEV value to STATE when the event occurs to be the next state.

35.9.22 static void SCTIMER_SetupNextStateActionwithLdMethod (SCT_Type * *base,* uint32_t *nextState,* uint32_t *event,* bool *fgLoad*) [inline], [static]

This transition will be triggered by the event number that is passed in by the user, the method decide how to load the highest-numbered event occurring for that state to the STATE register.

Parameters

base	SCTimer peripheral base address
nextState	The next state SCTimer will transition to
event	Event number that will trigger the state transition
fgLoad	The method to load the highest-numbered event occurring for that state to the STATE register. • true: Load the STATEV value to STATE when the event occurs to be the next state. • false: Add the STATEV value to STATE when the event occurs to be the next state.

35.9.23 static void SCTIMER_SetupNextStateAction (SCT_Type * *base,* uint32_t *nextState,* uint32_t *event*) [inline], [static]

Deprecated Do not use this function. It has been superceded by SCTIMER_SetupNextStateActionwith-LdMethod

This transition will be triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address
nextState	The next state SCTimer will transition to
event	Event number that will trigger the state transition

35.9.24 static void SCTIMER_SetupEventActiveDirection (SCT_Type * *base*, sctimer_event_active_direction_t *activeDirection*, uint32_t *event*) [inline], [static]

Parameters

base	SCTimer peripheral base address
activeDirection	Event generation active direction, see sctimer_event_active_direction_t.
event	Event number that need setup the active direction.

35.9.25 static void SCTIMER_SetupOutputSetAction (SCT_Type * *base*, uint32_t *whichIO*, uint32_t *event*) [inline], [static]

This output will be set when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichIO	The output to set
event	Event number that will trigger the output change

35.9.26 static void SCTIMER_SetupOutputClearAction (SCT_Type * *base*, uint32_t *whichIO*, uint32_t *event*) [inline], [static]

This output will be cleared when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichIO	The output to clear
event	Event number that will trigger the output change

35.9.27 void SCTIMER_SetupOutputToggleAction (SCT_Type * *base,* uint32_t *whichIO,* uint32_t *event*)

This change in the output level is triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address
whichIO	The output to toggle
event	Event number that will trigger the output change

35.9.28 static void SCTIMER_SetupCounterLimitAction (SCT_Type * *base,* sctimer_counter_t *whichCounter,* uint32_t *event*) [inline], [static]

The counter is limited when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
event	Event number that will trigger the counter to be limited

35.9.29 static void SCTIMER_SetupCounterStopAction (SCT_Type * *base,* sctimer_counter_t *whichCounter,* uint32_t *event*) [inline], [static]

The counter is stopped when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
event	Event number that will trigger the counter to be stopped

35.9.30 static void SCTIMER_SetupCounterStartAction (SCT_Type * *base*, sctimer_counter_t *whichCounter*, uint32_t *event*) [inline], [static]

The counter will re-start when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
event	Event number that will trigger the counter to re-start

35.9.31 static void SCTIMER_SetupCounterHaltAction (SCT_Type * *base*, sctimer_counter_t *whichCounter*, uint32_t *event*) [inline], [static]

The counter is disabled (halted) when the event number that is passed in by the user is triggered. When the counter is halted, all further events are disabled. The HALT condition can only be removed by calling the SCTIMER_StartTimer() function.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
event	Event number that will trigger the counter to be halted

35.9.32 static void SCTIMER_SetupDmaTriggerAction (SCT_Type * *base*, uint32_t *dmaNumber*, uint32_t *event*) [inline], [static]

DMA request will be triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address
dmaNumber	The DMA request to generate
event	Event number that will trigger the DMA request

35.9.33 static void SCTIMER_SetCOUNTValue (SCT_Type * *base,* sctimer_counter_t *whichCounter,* uint32_t *value*) [inline], [static]

The function is to set the value of Count register, Writing to the COUNT_L, COUNT_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
value	the counter value update to the COUNT register.

35.9.34 static uint32_t SCTIMER_GetCOUNTValue (SCT_Type * *base,* sctimer_counter_t *whichCounter*) [inline], [static]

The function is to read the value of Count register, software can read the counter registers at any time..

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The value of counter selected.

35.9.35 static void SCTIMER_SetEventInState (SCT_Type * *base,* uint32_t *event,* uint32_t *state*) [inline], [static]

Parameters

base	SCTimer peripheral base address
event	The EV_STATE register be set.
state	The state value in which the event is enabled to occur.

35.9.36 static void SCTIMER_ClearEventInState (SCT_Type * *base,* uint32_t *event,* uint32_t *state*) [inline], [static]

Parameters

base	SCTimer peripheral base address
event	The EV_STATE register be clear.
state	The state value in which the event is disabled to occur.

35.9.37 static bool SCTIMER_GetEventInState (SCT_Type * *base*, uint32_t *event*, uint32_t *state*) [inline], [static]

Note

This function is to check whether the event is enabled in a specific state.

Parameters

base	SCTimer peripheral base address
event	The EV_STATE register be read.
state	The state value.

Returns

The the state mask bit field of EV_STATE register.
- true: The event is enable in state.
- false: The event is disable in state.

35.9.38 void SCTIMER_EventHandleIRQ (SCT_Type * *base*)

Parameters

base	SCTimer peripheral base address.

Chapter 36
SYSCTL: I2S bridging and signal sharing Configuration

36.1 Overview

The MCUXpresso SDK provides a peripheral driver for the SYSCTL module of MCUXpresso SDK devices. For furter details, see the corresponding chapter.

Files

- file fsl_sysctl.h
- file fsl_sysctl.h

Enumerations

- enum _sysctl_share_set_index {
  kSYSCTL_ShareSet0 = 0,
  kSYSCTL_ShareSet1 = 1 }
  SYSCTL share set.
- enum sysctl_fcctrlsel_signal_t {
  kSYSCTL_FlexcommSignalSCK = SYSCTL_FCCTRLSEL_SCKINSEL_SHIFT,
  kSYSCTL_FlexcommSignalWS = SYSCTL_FCCTRLSEL_WSINSEL_SHIFT,
  kSYSCTL_FlexcommSignalDataIn = SYSCTL_FCCTRLSEL_DATAINSEL_SHIFT,
  kSYSCTL_FlexcommSignalDataOut = SYSCTL_FCCTRLSEL_DATAOUTSEL_SHIFT }
  SYSCTL flexcomm signal.
- enum _sysctl_share_src {
  kSYSCTL_Flexcomm0 = 0,
  kSYSCTL_Flexcomm1 = 1,
  kSYSCTL_Flexcomm2 = 2,
  kSYSCTL_Flexcomm4 = 4,
  kSYSCTL_Flexcomm5 = 5,
  kSYSCTL_Flexcomm6 = 6,
  kSYSCTL_Flexcomm7 = 7 }
  SYSCTL flexcomm index.
- enum _sysctl_dataout_mask {
  kSYSCTL_Flexcomm0DataOut = SYSCTL_SHAREDCTRLSET_FC0DATAOUTEN_MASK,
  kSYSCTL_Flexcomm1DataOut = SYSCTL_SHAREDCTRLSET_FC1DATAOUTEN_MASK,
  kSYSCTL_Flexcomm2DataOut = SYSCTL_SHAREDCTRLSET_FC2DATAOUTEN_MASK,
  kSYSCTL_Flexcomm4DataOut = SYSCTL_SHAREDCTRLSET_FC4DATAOUTEN_MASK,
  kSYSCTL_Flexcomm5DataOut = SYSCTL_SHAREDCTRLSET_FC5DATAOUTEN_MASK,
  kSYSCTL_Flexcomm6DataOut = SYSCTL_SHAREDCTRLSET_FC6DATAOUTEN_MASK,
  kSYSCTL_Flexcomm7DataOut = SYSCTL_SHAREDCTRLSET_FC7DATAOUTEN_MASK }
  SYSCTL shared data out mask.

- enum sysctl_sharedctrlset_signal_t {
  kSYSCTL_SharedCtrlSignalSCK = SYSCTL_SHAREDCTRLSET_SHAREDSCKSEL_SHIFT,
  kSYSCTL_SharedCtrlSignalWS = SYSCTL_SHAREDCTRLSET_SHAREDWSSEL_SHIFT,
  kSYSCTL_SharedCtrlSignalDataIn = SYSCTL_SHAREDCTRLSET_SHAREDDATASEL_SHIFT,
  kSYSCTL_SharedCtrlSignalDataOut = SYSCTL_SHAREDCTRLSET_FC0DATAOUTEN_SHIFT }
  SYSCTL flexcomm signal.

Driver version

- #define FSL_SYSCTL_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))
  Group sysctl driver version for SDK.

Initialization and deinitialization

- void SYSCTL_Init (SYSCTL_Type ∗base)
  SYSCTL initial.
- void SYSCTL_Deinit (SYSCTL_Type ∗base)
  SYSCTL deinit.

SYSCTL share signal configure

- void SYSCTL_SetFlexcommShareSet (SYSCTL_Type ∗base, uint32_t flexCommIndex, uint32_t sckSet, uint32_t wsSet, uint32_t dataInSet, uint32_t dataOutSet)
  SYSCTL share set configure for flexcomm.
- void SYSCTL_SetShareSet (SYSCTL_Type ∗base, uint32_t flexCommIndex, sysctl_fcctrlsel_signal_t signal, uint32_t set)
  SYSCTL share set configure for separate signal.
- void SYSCTL_SetShareSetSrc (SYSCTL_Type ∗base, uint32_t setIndex, uint32_t sckShareSrc, uint32_t wsShareSrc, uint32_t dataInShareSrc, uint32_t dataOutShareSrc)
  SYSCTL share set source configure.
- void SYSCTL_SetShareSignalSrc (SYSCTL_Type ∗base, uint32_t setIndex, sysctl_sharedctrlset_signal_t signal, uint32_t shareSrc)
  SYSCTL sck source configure.

36.2 Macro Definition Documentation

36.2.1 #define FSL_SYSCTL_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

Version 2.0.5.

36.3 Enumeration Type Documentation

36.3.1 enum _sysctl_share_set_index

Enumerator

kSYSCTL_ShareSet0 share set 0

kSYSCTL_ShareSet1 share set 1

36.3.2 enum sysctl_fcctrlsel_signal_t

Enumerator

kSYSCTL_FlexcommSignalSCK SCK signal.
kSYSCTL_FlexcommSignalWS WS signal.
kSYSCTL_FlexcommSignalDataIn Data in signal.
kSYSCTL_FlexcommSignalDataOut Data out signal.

36.3.3 enum _sysctl_share_src

Enumerator

kSYSCTL_Flexcomm0 share set 0
kSYSCTL_Flexcomm1 share set 1
kSYSCTL_Flexcomm2 share set 2
kSYSCTL_Flexcomm4 share set 4
kSYSCTL_Flexcomm5 share set 5
kSYSCTL_Flexcomm6 share set 6
kSYSCTL_Flexcomm7 share set 7

36.3.4 enum _sysctl_dataout_mask

Enumerator

kSYSCTL_Flexcomm0DataOut share set 0
kSYSCTL_Flexcomm1DataOut share set 1
kSYSCTL_Flexcomm2DataOut share set 2
kSYSCTL_Flexcomm4DataOut share set 4
kSYSCTL_Flexcomm5DataOut share set 5
kSYSCTL_Flexcomm6DataOut share set 6
kSYSCTL_Flexcomm7DataOut share set 7

36.3.5 enum sysctl_sharedctrlset_signal_t

Enumerator

kSYSCTL_SharedCtrlSignalSCK SCK signal.

kSYSCTL_SharedCtrlSignalWS WS signal.
kSYSCTL_SharedCtrlSignalDataIn Data in signal.
kSYSCTL_SharedCtrlSignalDataOut Data out signal.

36.4 Function Documentation

36.4.1 void SYSCTL_Init (SYSCTL_Type * *base*)

Parameters

base	Base address of the SYSCTL peripheral.

36.4.2 void SYSCTL_Deinit (SYSCTL_Type * *base*)

Parameters

base	Base address of the SYSCTL peripheral.

36.4.3 void SYSCTL_SetFlexcommShareSet (SYSCTL_Type * *base*, uint32_t *flexCommIndex*, uint32_t *sckSet*, uint32_t *wsSet*, uint32_t *dataInSet*, uint32_t *dataOutSet*)

Parameters

base	Base address of the SYSCTL peripheral.
flexCommIndex	index of flexcomm, reference _sysctl_share_src
sckSet	share set for sck,reference _sysctl_share_set_index
wsSet	share set for ws, reference _sysctl_share_set_index
dataInSet	share set for data in, reference _sysctl_share_set_index
dataOutSet	share set for data out, reference _sysctl_dataout_mask

36.4.4 void SYSCTL_SetShareSet (SYSCTL_Type * *base*, uint32_t *flexCommIndex*, sysctl_fcctrlsel_signal_t *signal*, uint32_t *set*)

Parameters

base	Base address of the SYSCTL peripheral
flexCommIndex	index of flexcomm,reference _sysctl_share_src
signal	FCCTRLSEL signal shift
set	share set for sck, reference _sysctl_share_set_index

36.4.5 void SYSCTL_SetShareSetSrc (SYSCTL_Type * *base*, uint32_t *setIndex*, uint32_t *sckShareSrc*, uint32_t *wsShareSrc*, uint32_t *dataInShareSrc*, uint32_t *dataOutShareSrc*)

Parameters

base	Base address of the SYSCTL peripheral
setIndex	index of share set, reference _sysctl_share_set_index
sckShareSrc	sck source for this share set,reference _sysctl_share_src
wsShareSrc	ws source for this share set,reference _sysctl_share_src
dataInShareSrc	data in source for this share set,reference _sysctl_share_src
dataOutShare-Src	data out source for this share set,reference _sysctl_dataout_mask

36.4.6 void SYSCTL_SetShareSignalSrc (SYSCTL_Type * *base*, uint32_t *setIndex*, sysctl_sharedctrlset_signal_t *signal*, uint32_t *shareSrc*)

Parameters

base	Base address of the SYSCTL peripheral
setIndex	index of share set, reference _sysctl_share_set_index
signal	FCCTRLSEL signal shift
shareSrc	sck source fro this share set,reference _sysctl_share_src

Chapter 37
UTICK: MictoTick Timer Driver

37.1 Overview

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

UTICK driver is created to help user to operate the UTICK module. The UTICK timer can be used as a low power timer. The APIs can be used to enable the UTICK module, initialize it and set the time. UTICK can be used as a wake up source from low power mode.

37.2 Typical use case

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

Files

- file fsl_utick.h

Typedefs

- typedef void(∗ utick_callback_t)(void)
  UTICK callback function.

Enumerations

- enum utick_mode_t {
  kUTICK_Onetime = 0x0U,
  kUTICK_Repeat = 0x1U }
  UTICK timer operational mode.

Driver version

- #define FSL_UTICK_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))
  UTICK driver version 2.0.5.

Initialization and deinitialization

- void UTICK_Init (UTICK_Type ∗base)
  Initializes an UTICK by turning its bus clock on.
- void UTICK_Deinit (UTICK_Type ∗base)
  Deinitializes a UTICK instance.
- uint32_t UTICK_GetStatusFlags (UTICK_Type ∗base)
  Get Status Flags.
- void UTICK_ClearStatusFlags (UTICK_Type ∗base)
  Clear Status Interrupt Flags.

- void UTICK_SetTick (UTICK_Type ∗base, utick_mode_t mode, uint32_t count, utick_callback_t cb)
  Starts UTICK.
- void UTICK_HandleIRQ (UTICK_Type ∗base, utick_callback_t cb)
  UTICK Interrupt Service Handler.

37.3 Macro Definition Documentation

37.3.1 #define FSL_UTICK_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

37.4 Typedef Documentation

37.4.1 typedef void(∗ utick_callback_t)(void)

37.5 Enumeration Type Documentation

37.5.1 enum utick_mode_t

Enumerator

kUTICK_Onetime Trigger once.
kUTICK_Repeat Trigger repeatedly.

37.6 Function Documentation

37.6.1 void UTICK_Init (UTICK_Type ∗ *base*)

37.6.2 void UTICK_Deinit (UTICK_Type ∗ *base*)

This function shuts down Utick bus clock

Parameters

base	UTICK peripheral base address.

37.6.3 uint32_t UTICK_GetStatusFlags (UTICK_Type ∗ *base*)

This returns the status flag

Parameters

base	UTICK peripheral base address.

Returns

status register value

37.6.4 void UTICK_ClearStatusFlags (UTICK_Type * *base*)

This clears intr status flag

Parameters

base	UTICK peripheral base address.

Returns

none

37.6.5 void UTICK_SetTick (UTICK_Type * *base*, utick_mode_t *mode*, uint32_t *count*, utick_callback_t *cb*)

This function starts a repeat/onetime countdown with an optional callback

Parameters

base	UTICK peripheral base address.
mode	UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
count	UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
cb	UTICK callback (can be left as NULL if none, otherwise should be a void func(void))

Returns

none

37.6.6 void UTICK_HandleIRQ (UTICK_Type * *base*, utick_callback_t *cb*)

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in UTICK_SetTick()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

base	UTICK peripheral base address.
cb	callback scheduled for this instance of UTICK

Returns

none

Chapter 38
WWDT: Windowed Watchdog Timer Driver

38.1 Overview

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

38.2 Function groups

38.2.1 Initialization and deinitialization

The function WWDT_Init() initializes the watchdog timer with specified configurations. The configurations include timeout value and whether to enable watchdog after init. The function WWDT_GetDefaultConfig() gets the default configurations.

The function WWDT_Deinit() disables the watchdog and the module clock.

38.2.2 Status

Provides functions to get and clear the WWDT status.

38.2.3 Interrupt

Provides functions to enable/disable WWDT interrupts and get current enabled interrupts.

38.2.4 Watch dog Refresh

The function WWDT_Refresh() feeds the WWDT.

38.3 Typical use case

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

Files

- file fsl_wwdt.h

Data Structures

- struct wwdt_config_t
  Describes WWDT configuration structure. *More...*

Enumerations

- enum _wwdt_status_flags_t {
  kWWDT_TimeoutFlag = WWDT_MOD_WDTOF_MASK,
  kWWDT_WarningFlag = WWDT_MOD_WDINT_MASK }
  WWDT status flags.

Driver version

- #define FSL_WWDT_DRIVER_VERSION (MAKE_VERSION(2, 1, 9))
  Defines WWDT driver version.

Refresh sequence

- #define WWDT_FIRST_WORD_OF_REFRESH (0xAAU)
  First word of refresh sequence.
- #define WWDT_SECOND_WORD_OF_REFRESH (0x55U)
  Second word of refresh sequence.

WWDT Initialization and De-initialization

- void WWDT_GetDefaultConfig (wwdt_config_t *config)
  Initializes WWDT configure structure.
- void WWDT_Init (WWDT_Type *base, const wwdt_config_t *config)
  Initializes the WWDT.
- void WWDT_Deinit (WWDT_Type *base)
  Shuts down the WWDT.

WWDT Functional Operation

- static void WWDT_Enable (WWDT_Type *base)
  Enables the WWDT module.
- static void WWDT_Disable (WWDT_Type *base)
  Disables the WWDT module.
- static uint32_t WWDT_GetStatusFlags (WWDT_Type *base)
  Gets all WWDT status flags.
- void WWDT_ClearStatusFlags (WWDT_Type *base, uint32_t mask)
  Clear WWDT flag.
- static void WWDT_SetWarningValue (WWDT_Type *base, uint32_t warningValue)
  Set the WWDT warning value.
- static void WWDT_SetTimeoutValue (WWDT_Type *base, uint32_t timeoutCount)
  Set the WWDT timeout value.
- static void WWDT_SetWindowValue (WWDT_Type *base, uint32_t windowValue)
  Sets the WWDT window value.
- void WWDT_Refresh (WWDT_Type *base)
  Refreshes the WWDT timer.

38.4 Data Structure Documentation

38.4.1 struct wwdt_config_t

Data Fields

- bool enableWwdt
  Enables or disables WWDT.
- bool enableWatchdogReset
  true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset
- bool enableWatchdogProtect
  true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time
- uint32_t windowValue
  Window value, set this to 0xFFFFFF if windowing is not in effect.
- uint32_t timeoutValue
  Timeout value.
- uint32_t warningValue
  Watchdog time counter value that will generate a warning interrupt.
- uint32_t clockFreq_Hz
  Watchdog clock source frequency.

Field Documentation

(1) uint32_t wwdt_config_t::warningValue

Set this to 0 for no warning

(2) uint32_t wwdt_config_t::clockFreq_Hz

38.5 Macro Definition Documentation

38.5.1 #define FSL_WWDT_DRIVER_VERSION (MAKE_VERSION(2, 1, 9))

38.6 Enumeration Type Documentation

38.6.1 enum _wwdt_status_flags_t

This structure contains the WWDT status flags for use in the WWDT functions.

Enumerator

kWWDT_TimeoutFlag Time-out flag, set when the timer times out.
kWWDT_WarningFlag Warning interrupt flag, set when timer is below the value WDWARNINT.

38.7 Function Documentation

38.7.1 void WWDT_GetDefaultConfig (wwdt_config_t * *config*)

This function initializes the WWDT configure structure to default value. The default value are:

```
*  config->enableWwdt = true;
*  config->enableWatchdogReset = false;
*  config->enableWatchdogProtect = false;
*  config->enableLockOscillator = false;
*  config->windowValue = 0xFFFFFFU;
*  config->timeoutValue = 0xFFFFFFU;
*  config->warningValue = 0;
*
```

Parameters

config	Pointer to WWDT config structure.

See Also

wwdt_config_t

38.7.2 void WWDT_Init (WWDT_Type * *base*, const wwdt_config_t * *config*)

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
*   wwdt_config_t config;
*   WWDT_GetDefaultConfig(&config);
*   config.timeoutValue = 0x7ffU;
*   WWDT_Init(wwdt_base,&config);
*
```

Parameters

base	WWDT peripheral base address
config	The configuration of WWDT

38.7.3 void WWDT_Deinit (WWDT_Type * *base*)

This function shuts down the WWDT.

Parameters

base	WWDT peripheral base address

38.7.4 static void WWDT_Enable (WWDT_Type * *base*) [inline], [static]

This function write value into WWDT_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

Parameters

base	WWDT peripheral base address

38.7.5 static void WWDT_Disable (WWDT_Type * *base*) [inline], [static]

Deprecated Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT_MOD register to disable the WWDT.

Parameters

base	WWDT peripheral base address

38.7.6 static uint32_t WWDT_GetStatusFlags (WWDT_Type * *base*) [inline], [static]

This function gets all status flags.

Example for getting Timeout Flag:

```
*   uint32_t status;
*   status = WWDT_GetStatusFlags(wwdt_base) &
      kWWDT_TimeoutFlag;
*
```

Parameters

base	WWDT peripheral base address

Returns

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

38.7.7 void WWDT_ClearStatusFlags (WWDT_Type * *base*, uint32_t *mask*)

This function clears WWDT status flag.

Example for clearing warning flag:

```
*   WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
*
```

Parameters

base	WWDT peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration _wwdt_status_flags_t

38.7.8 static void WWDT_SetWarningValue (WWDT_Type * *base*, uint32_t *warningValue*) [inline], [static]

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

Parameters

base	WWDT peripheral base address
warningValue	WWDT warning value.

38.7.9 static void WWDT_SetTimeoutValue (WWDT_Type * *base*, uint32_t *timeoutCount*) [inline], [static]

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC register. Thus the minimum time-out interval is TWDCLK*256*4. If enableWatchdogProtect flag is true in wwdt_config_t config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

Parameters

base	WWDT peripheral base address
timeoutCount	WWDT timeout value, count of WWDT clock tick.

38.7.10 static void WWDT_SetWindowValue (WWDT_Type * *base*, uint32_t *windowValue*) [inline], [static]

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set windowValue to 0xFFFFFF (maximum possible timer value) so windowing is not in effect.

Parameters

base	WWDT peripheral base address
windowValue	WWDT window value.

38.7.11 void WWDT_Refresh (WWDT_Type * *base*)

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

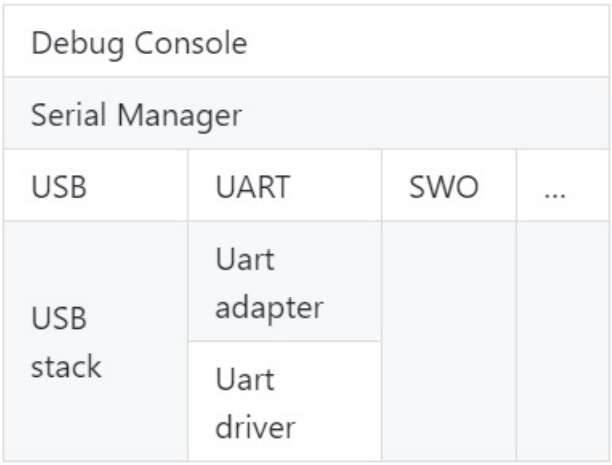
base	WWDT peripheral base address

Chapter 39
Debug Console

39.1 Overview

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

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



Debug console overview

39.2 Function groups

39.2.1 Initialization

To initialize the debug console, call the DbgConsole_Init() function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate,
      serial_port_type_t device, uint32_t clkSrcFreq);
```

Select the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
    kSerialPort_Uart = 1U,
    kSerialPort_UsbCdc,
    kSerialPort_Swo,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral.

This example shows how to call the DbgConsole_Init() given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUDRATE, BOARD_DEBUG_UART_TYPE,
                                BOARD_DEBUG_UART_CLK_FREQ);
```

39.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype " %[flags][width][.precision][length]specifier", which is explained below

flags	Description
-	Left-justified within the given field width. Right-justified is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is written, a blank space is inserted before the value.
#	Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

Width	Description
(number)	A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	Description
.number	For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

length	Description
Do not support	

specifier	Description
d or i	Signed decimal integer
f	Decimal floating point
F	Decimal floating point capital letters
x	Unsigned hexadecimal integer
X	Unsigned hexadecimal integer capital letters
o	Signed octal
b	Binary value
p	Pointer address
u	Unsigned decimal integer
c	Character
s	String of characters
n	Nothing printed

- Support a format specifier for SCANF following this prototype " %[∗][width][length]specifier", which is explained below

∗	Description
An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument.	

width	Description
This specifies the maximum number of characters to be read in the current reading operation.	

length	Description
hh	The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).
h	The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).
l	The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
ll	The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
L	The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).
j or z or t	Not supported

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char *
i	Integer: : Number optionally preceded with a + or - sign	int *
d	Decimal integer: Number optionally preceded with a + or - sign	int *
a, A, e, E, f, F, g, G	Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	float *
o	Octal Integer:	int *
s	String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).	char *
u	Unsigned decimal integer.	unsigned int *

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

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

```
#if SDK_DEBUGCONSOLE == DEBUGCONSOLE_DISABLE /* Disable debug console */
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
```

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

39.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART

There are two macros SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART added to configure PRINTF and low level output perihperal.

- The macro SDK_DEBUGCONSOLE is used for forntend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The fucntion can be set by the macro SDK_DEBUGCONSOLE.
- The macro SDK_DEBUGCONSOLE_UART is used for backend. It is use to decide whether provide low level IO implementation to toolchain printf and scanf. For example, within MCU-Xpresso, if the macro SDK_DEBUGCONSOLE_UART is defined, __sys_write and __sys_readc will be used when __REDLIB__ is defined; _write and _read will be used in other cases.The macro does not specifically refer to the perihpheral "UART". It refers to the external perihperal similar to UART, like as USB CDC, UART, SWO, etc. So if the macro SDK_DEBUGCONSOLE_UART is not defined when tool-chain printf is calling, the semihosting will be used.

The following the matrix show the effects of SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_-UART on PRINTF and printf. The green mark is the default setting of the debug console.

SDK_DEBUGCONSOLE	SDK_DEBUGCONSOLE_UART	PRINTF	printf
DEBUGCONSOLE_-REDIRECT_TO_SDK	defined	Low level peripheral∗	Low level peripheral
DEBUGCONSOLE_-REDIRECT_TO_SDK	undefined	Low level peripheral∗	semihost
DEBUGCONSOLE_-REDIRECT_TO_TO-OLCHAIN	defined	Low level peripheral∗	Low level peripheral
DEBUGCONSOLE_-REDIRECT_TO_TO-OLCHAIN	undefined	semihost	semihost
DEBUGCONSOLE_-DISABLE	defined	No ouput	Low level peripheral
DEBUGCONSOLE_-DISABLE	undefined	No ouput	semihost

* the **low level peripheral** could be USB CDC, UART, or SWO, and so on.

39.3 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 DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U
  Definition select redirect toolchain printf, scanf to uart or not.
- #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U
  Select SDK version printf, scanf.
- #define DEBUGCONSOLE_DISABLE 2U
  Disable debugconsole function.
- #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK
  Definition to select sdk or toolchain printf, scanf.
- #define PRINTF DbgConsole_Printf
  Definition to select redirect toolchain printf, scanf to uart or not.

Typedefs

- typedef void(∗ printfCb)(char ∗buf, int32_t ∗indicator, char val, int len)
  A function pointer which is used when format printf log.

Functions

- int StrFormatPrintf (const char ∗fmt, va_list ap, char ∗buf, printfCb cb)
  This function outputs its parameters according to a formatted string.
- int StrFormatScanf (const char ∗line_ptr, char ∗format, va_list args_ptr)
  Converts an input line of ASCII characters based upon a provided string format.

Variables

- serial_handle_t g_serialHandle
  serial manager handle

Initialization

- status_t DbgConsole_Init (uint8_t instance, uint32_t baudRate, serial_port_type_t device, uint32_t clkSrcFreq)
  Initializes the peripheral used for debug messages.
- status_t DbgConsole_Deinit (void)
  De-initializes the peripheral used for debug messages.
- status_t DbgConsole_EnterLowpower (void)
  Prepares to enter low power consumption.
- status_t DbgConsole_ExitLowpower (void)
  Restores from low power consumption.
- int DbgConsole_Printf (const char ∗fmt_s,...)
  Writes formatted output to the standard output stream.
- int DbgConsole_Vprintf (const char ∗fmt_s, va_list formatStringArg)
  Writes formatted output to the standard output stream.
- int DbgConsole_Putchar (int ch)

Writes a character to stdout.

- int DbgConsole_Scanf (char *fmt_s,...)
  Reads formatted data from the standard input stream.
- int DbgConsole_Getchar (void)
  Reads a character from standard input.
- int DbgConsole_BlockingPrintf (const char *fmt_s,...)
  Writes formatted output to the standard output stream with the blocking mode.
- int DbgConsole_BlockingVprintf (const char *fmt_s, va_list formatStringArg)
  Writes formatted output to the standard output stream with the blocking mode.
- status_t DbgConsole_Flush (void)
  Debug console flush.

39.4 Macro Definition Documentation

39.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

39.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

39.4.3 #define DEBUGCONSOLE_DISABLE 2U

39.4.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

The macro only support to be redefined in project setting.

39.4.5 #define PRINTF DbgConsole_Printf

if SDK_DEBUGCONSOLE defined to 0,it represents select toolchain printf, scanf. if SDK_DEBUGCONSOLE defined to 1,it represents select SDK version printf, scanf. if SDK_DEBUGCONSOLE defined to 2,it represents disable debugconsole function.

39.5 Function Documentation

39.5.1 status_t DbgConsole_Init (uint8_t *instance*, uint32_t *baudRate*, serial_port_type_t *device*, uint32_t *clkSrcFreq*)

Call this function to enable debug log messages to be output via the specified peripheral initialized by the serial manager module. After this function has returned, stdout and stdin are connected to the selected peripheral.

Parameters

instance	The instance of the module.If the device is kSerialPort_Uart, the instance is UART peripheral instance. The UART hardware peripheral type is determined by UART adapter. For example, if the instance is 1, if the lpuart_adapter.c is added to the current project, the UART periheral is LPUART1. If the uart_adapter.c is added to the current project, the UART periheral is UART1.
baudRate	The desired baud rate in bits per second.
device	Low level device type for the debug console, can be one of the following. • kSerialPort_Uart, • kSerialPort_UsbCdc
clkSrcFreq	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

kStatus_Success	Execution successfully

39.5.2 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.

39.5.3 status_t DbgConsole_EnterLowpower (void)

This function is used to prepare to enter low power consumption.

Returns

Indicates whether de-initialization was successful or not.

39.5.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

39.5.5 int DbgConsole_Printf (const char * *fmt_s*, ...)

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

Parameters

fmt_s	Format control string.

Returns

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

39.5.6 int DbgConsole_Vprintf (const char * *fmt_s*, va_list *formatStringArg*)

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

Parameters

fmt_s	Format control string.
formatString-Arg	Format arguments.

Returns

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

39.5.7 int DbgConsole_Putchar (int *ch*)

Call this function to write a character to stdout.

Parameters

ch	Character to be written.

Returns

Returns the character written.

39.5.8 int DbgConsole_Scanf (char * *fmt_s, ...*)

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

Note

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

Parameters

fmt_s	Format control string.

Returns

Returns the number of fields successfully converted and assigned.

39.5.9 int DbgConsole_Getchar (void)

Call this function to read a character from standard input.

Note

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

Returns

Returns the character read.

39.5.10 int DbgConsole_BlockingPrintf (const char * *fmt_s*, ...)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set or not. The function could be used in system ISR mode with DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set.

Parameters

fmt_s	Format control string.

Returns

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

39.5.11 int DbgConsole_BlockingVprintf (const char * *fmt_s*, va_list *formatStringArg*)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set or not. The function could be used in system ISR mode with DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set.

Parameters

fmt_s	Format control string.
formatString-Arg	Format arguments.

Returns

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

39.5.12 status_t DbgConsole_Flush (void)

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

Returns

Indicates whether wait idle was successful or not.

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

Note

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

Parameters

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

Returns

Number of characters to be print

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

Parameters

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

Returns

Number of input items converted and assigned.

Return values

IO_EOF	When line_ptr is empty string "".

39.6 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.

39.6.1 Guide Semihosting for IAR

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

Step 1: Setting up the environment

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

Step 2: Building the project

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

Step 3: Starting semihosting

1. Choose "Semihosting_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting. Please Make sure the SDK_DEBUGCONSOLE_UART is not defined in project settings.
4. Start the project by choosing Project>Download and Debug.
5. Choose View>Terminal I/O to display the output from the I/O operations.

39.6.2 Guide Semihosting for Keil µVision

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

39.6.3 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.

39.6.4 Guide Semihosting for ARMGCC

Step 1: Setting up the environment

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

Add to "CMakeLists.txt"

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

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

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

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

Step 2: Building the project

1. Change "CMakeLists.txt":
   Change "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=nano.specs")"
   to "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=rdimon.specs")"
   Replace paragraph
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")
   To
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --specs=rdimon.specs ")
   Remove
   target_link_libraries(semihosting_ARMGCC.elf debug nosys)
2. Run "build_debug.bat" to build project

Step 3: Starting semihosting

1. Download the image and set as follows.

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

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

39.7 SWO

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

39.7.1 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 SERIAL_PORT_TYPE_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(instance, baudRate, kSerialPort_Swo, clkSrcFreq);
```

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

Step 2: Building the project

Step 3: Download and run project

39.7.1.1 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. There are three cases for this SDK_DEBUGCONSOLE_UART whether or not defined. a: if use uppercase PRINTF to output log,The SDK_DEBUGCONSOLE_UART defined or not defined will not effect debug function. b: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to zero,then debug function ok. c: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to one,then debug function ok.

NOTE: Case a or c only apply at example which enable swo function,the SDK_DEBUGCONSOLE_UART definition in fsl_debug_console.h. For case a and c, Do and not do the above third step will be not affect function.

1. 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.

39.7.2 Guide SWO for Keil µVision

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

Step 1: Setting up the environment

1. There are three cases for this SDK_DEBUGCONSOLE_UART whether or not defined. a: if use uppercase PRINTF to output log,the SDK_DEBUGCONSOLE_UART definition does not affect the functionality and skip the second step directly. b: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to zero,then start the second step. c: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to one,then skip the second step directly.

NOTE: Case a or c only apply at example which enable swo function,the SDK_DEBUGCONSOLE_UART definition in fsl_debug_console.h.

1. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
2. Open Project>Options for target or using Alt+F7 or click.
3. Select “Debug” tab, select “J-Link/J-Trace Cortex” and click “Setting button”.
4. Select “Debug” tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click OK, 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.

39.7.3 Guide SWO for MCUXpresso IDE

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

39.7.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.

Chapter 40
Notification Framework

40.1 Overview

This section describes the programming interface of the Notifier driver.

40.2 Notifier Overview

The Notifier provides a configuration dynamic change service. Based on this service, applications can switch between pre-defined configurations. The Notifier enables drivers and applications to register callback functions to this framework. Each time that the configuration is changed, drivers and applications receive a notification and change their settings. To simplify, the Notifier only supports the static callback registration. This means that, for applications, all callback functions are collected into a static table and passed to the Notifier.

These are the steps for the configuration transition.

1. Before configuration transition, the Notifier sends a "BEFORE" message to the callback table. When this message is received, IP drivers should check whether any current processes can be stopped and stop them. If the processes cannot be stopped, the callback function returns an error.
   The Notifier supports two types of transition policies, a graceful policy and a forceful policy. When the graceful policy is used, if some callbacks return an error while sending a "BEFORE" message, the configuration transition stops and the Notifier sends a "RECOVER" message to all drivers that have stopped. Then, these drivers can recover the previous status and continue to work. When the forceful policy is used, drivers are stopped forcefully.
2. After the "BEFORE" message is processed successfully, the system switches to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application.

```
#include "fsl_notifier.h"

// Definition of the Power Manager callback.
status_t callback0(notifier_notification_block_t *notify, void *data)
{

    status_t ret = kStatus_Success;

    ...
    ...
    ...

    return ret;
}
// Definition of the Power Manager user function.
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *
      userData)
```

```
{
    ...
    ...
    ...
}
...
...
...
...
...
// Main function.
int main(void)
{
    // Define a notifier handle.
    notifier_handle_t powerModeHandle;

    // Callback configuration.
    user_callback_data_t callbackData0;

    notifier_callback_config_t callbackCfg0 = {callback0,
                kNOTIFIER_CallbackBeforeAfter,
                (void *)&callbackData0};

    notifier_callback_config_t callbacks[] = {callbackCfg0};

    // Power mode configurations.
    power_user_config_t vlprConfig;
    power_user_config_t stopConfig;

    notifier_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

    // Definition of a transition to and out the power modes.
    vlprConfig.mode = kAPP_PowerModeVlpr;
    vlprConfig.enableLowPowerWakeUpOnInterrupt = false;

    stopConfig = vlprConfig;
    stopConfig.mode = kAPP_PowerModeStop;

    // Create Notifier handle.
    NOTIFIER_CreateHandle(&powerModeHandle, powerConfigs, 2U, callbacks, 1U,
      APP_PowerModeSwitch, NULL);
    ...
    ...
    // Power mode switch.
    NOTIFIER_switchConfig(&powerModeHandle, targetConfigIndex,
      kNOTIFIER_PolicyAgreement);
}
```

Data Structures

- struct notifier_notification_block_t
  notification block passed to the registered callback function. More...
- struct notifier_callback_config_t
  Callback configuration structure. More...
- struct notifier_handle_t
  Notifier handle structure. More...

Typedefs

- typedef void notifier_user_config_t
  Notifier user configuration type.
- typedef status_t(* notifier_user_function_t)(notifier_user_config_t *targetConfig, void *userData)

Notifier user function prototype Use this function to execute specific operations in configuration switch.

- typedef status_t(∗ notifier_callback_t)(notifier_notification_block_t ∗notify, void ∗data)
  Callback prototype.

Enumerations

- enum _notifier_status {
  kStatus_NOTIFIER_ErrorNotificationBefore,
  kStatus_NOTIFIER_ErrorNotificationAfter }
  Notifier error codes.
- enum notifier_policy_t {
  kNOTIFIER_PolicyAgreement,
  kNOTIFIER_PolicyForcible }
  Notifier policies.
- enum notifier_notification_type_t {
  kNOTIFIER_NotifyRecover = 0x00U,
  kNOTIFIER_NotifyBefore = 0x01U,
  kNOTIFIER_NotifyAfter = 0x02U }
  Notification type.
- enum notifier_callback_type_t {
  kNOTIFIER_CallbackBefore = 0x01U,
  kNOTIFIER_CallbackAfter = 0x02U,
  kNOTIFIER_CallbackBeforeAfter = 0x03U }
  The callback type, which indicates kinds of notification the callback handles.

Functions

- status_t NOTIFIER_CreateHandle (notifier_handle_t ∗notifierHandle, notifier_user_config_t ∗∗configs, uint8_t configsNumber, notifier_callback_config_t ∗callbacks, uint8_t callbacksNumber, notifier_user_function_t userFunction, void ∗userData)
  Creates a Notifier handle.
- status_t NOTIFIER_SwitchConfig (notifier_handle_t ∗notifierHandle, uint8_t configIndex, notifier_policy_t policy)
  Switches the configuration according to a pre-defined structure.
- uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t ∗notifierHandle)
  This function returns the last failed notification callback.

40.3 Data Structure Documentation

40.3.1 struct notifier_notification_block_t

Data Fields

- notifier_user_config_t ∗ targetConfig
  Pointer to target configuration.
- notifier_policy_t policy
  Configure transition policy.
- notifier_notification_type_t notifyType

Configure notification type.

Field Documentation

(1) notifier_user_config_t* notifier_notification_block_t::targetConfig

(2) notifier_policy_t notifier_notification_block_t::policy

(3) notifier_notification_type_t notifier_notification_block_t::notifyType

40.3.2 struct notifier_callback_config_t

This structure holds the configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains the following application-defined data. callback - pointer to the callback function callbackType - specifies when the callback is called callbackData - pointer to the data passed to the callback.

Data Fields

- notifier_callback_t callback
  Pointer to the callback function.
- notifier_callback_type_t callbackType
  Callback type.
- void * callbackData
  Pointer to the data passed to the callback.

Field Documentation

(1) notifier_callback_t notifier_callback_config_t::callback

(2) notifier_callback_type_t notifier_callback_config_t::callbackType

(3) void* notifier_callback_config_t::callbackData

40.3.3 struct notifier_handle_t

Notifier handle structure. Contains data necessary for the Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data, and other internal data. NOTIFIER_CreateHandle() must be called to initialize this handle.

Data Fields

- notifier_user_config_t ** configsTable
  Pointer to configure table.
- uint8_t configsNumber
  Number of configurations.

- notifier_callback_config_t * callbacksTable
  Pointer to callback table.
- uint8_t callbacksNumber
  Maximum number of callback configurations.
- uint8_t errorCallbackIndex
  Index of callback returns error.
- uint8_t currentConfigIndex
  Index of current configuration.
- notifier_user_function_t userFunction
  User function.
- void * userData
  User data passed to user function.

Field Documentation

(1) notifier_user_config_t notifier_handle_t::configsTable**

(2) uint8_t notifier_handle_t::configsNumber

(3) notifier_callback_config_t* notifier_handle_t::callbacksTable

(4) uint8_t notifier_handle_t::callbacksNumber

(5) uint8_t notifier_handle_t::errorCallbackIndex

(6) uint8_t notifier_handle_t::currentConfigIndex

(7) notifier_user_function_t notifier_handle_t::userFunction

(8) void* notifier_handle_t::userData

40.4 Typedef Documentation

40.4.1 typedef void notifier_user_config_t

Reference of the user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

40.4.2 typedef status_t(* notifier_user_function_t)(notifier_user_config_t *targetConfig, void *userData)

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, NOTIFIER_SwitchConfig() exits.

Parameters

targetConfig	target Configuration.
userData	Refers to other specific data passed to user function.

Returns

An error code or kStatus_Success.

40.4.3 typedef status_t(∗ notifier_callback_t)(notifier_notification_block_t ∗notify, void ∗data)

Declaration of a callback. It is common for registered callbacks. Reference to function of this type is part of the notifier_callback_config_t callback configuration structure. Depending on callback type, function of this prototype is called (see NOTIFIER_SwitchConfig()) before configuration switch, after it or in both use cases to notify about the switch progress (see notifier_callback_type_t). When called, the type of the notification is passed as a parameter along with the reference to the target configuration structure (see notifier_notification_block_t) and any data passed during the callback registration. When notified before the configuration switch, depending on the configuration switch policy (see notifier_policy_t), the callback may deny the execution of the user function by returning an error code different than kStatus_Success (see NOTIFIER_SwitchConfig()).

Parameters

notify	Notification block.
data	Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information.

Returns

An error code or kStatus_Success.

40.5 Enumeration Type Documentation

40.5.1 enum _notifier_status

Used as return value of Notifier functions.

Enumerator

kStatus_NOTIFIER_ErrorNotificationBefore An error occurs during send "BEFORE" notification.

kStatus_NOTIFIER_ErrorNotificationAfter An error occurs during send "AFTER" notification.

40.5.2 enum notifier_policy_t

Defines whether the user function execution is forced or not. For kNOTIFIER_PolicyForcible, the user function is executed regardless of the callback results, while kNOTIFIER_PolicyAgreement policy is used to exit NOTIFIER_SwitchConfig() when any of the callbacks returns error code. See also NOTIFIER_SwitchConfig() description.

Enumerator

kNOTIFIER_PolicyAgreement NOTIFIER_SwitchConfig() method is exited when any of the callbacks returns error code.
kNOTIFIER_PolicyForcible The user function is executed regardless of the results.

40.5.3 enum notifier_notification_type_t

Used to notify registered callbacks

Enumerator

kNOTIFIER_NotifyRecover Notify IP to recover to previous work state.
kNOTIFIER_NotifyBefore Notify IP that configuration setting is going to change.
kNOTIFIER_NotifyAfter Notify IP that configuration setting has been changed.

40.5.4 enum notifier_callback_type_t

Used in the callback configuration structure (notifier_callback_config_t) to specify when the registered callback is called during configuration switch initiated by the NOTIFIER_SwitchConfig(). Callback can be invoked in following situations.

- Before the configuration switch (Callback return value can affect NOTIFIER_SwitchConfig() execution. See the NOTIFIER_SwitchConfig() and notifier_policy_t documentation).
- After an unsuccessful attempt to switch configuration
- After a successful configuration switch

Enumerator

kNOTIFIER_CallbackBefore Callback handles BEFORE notification.
kNOTIFIER_CallbackAfter Callback handles AFTER notification.
kNOTIFIER_CallbackBeforeAfter Callback handles BEFORE and AFTER notification.

40.6 Function Documentation

40.6.1 status_t NOTIFIER_CreateHandle (notifier_handle_t * *notifierHandle,* notifier_user_config_t ** *configs,* uint8_t *configsNumber,* notifier_callback_config_t * *callbacks,* uint8_t *callbacksNumber,* notifier_user_function_t *userFunction,* void * *userData*)

Parameters

notifierHandle	A pointer to the notifier handle.
configs	A pointer to an array with references to all configurations which is handled by the Notifier.
configsNumber	Number of configurations. Size of the configuration array.
callbacks	A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value.
callbacks-Number	Number of registered callbacks. Size of the callbacks array.
userFunction	User function.
userData	User data passed to user function.

Returns

An error Code or kStatus_Success.

40.6.2 status_t NOTIFIER_SwitchConfig (notifier_handle_t * *notifierHandle,* uint8_t *configIndex,* notifier_policy_t *policy*)

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER_PolicyForcible) or exited (kNOTIFIER_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER_Get-ErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when NOTIFIER_SwitchConfig() exits.

Parameters

notifierHandle	pointer to notifier handle
configIndex	Index of the target configuration.
policy	Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible.

Returns

An error code or kStatus_Success.

40.6.3 uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t * *notifierHandle*)

This function returns an index of the last callback that failed during the configuration switch while the last NOTIFIER_SwitchConfig() was called. If the last NOTIFIER_SwitchConfig() call ended successfully value equal to callbacks number is returned. The returned value represents an index in the array of static call-backs.

Parameters

notifierHandle	Pointer to the notifier handle

Returns

Callback Index of the last failed callback or value equal to callbacks count.

Chapter 41
Shell

41.1 Overview

This section describes the programming interface of the Shell middleware.

Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

41.2 Function groups

41.2.1 Initialization

To initialize the Shell middleware, call the SHELL_Init() function with these parameters. This function automatically enables the middleware.

```
shell_status_t SHELL_Init(shell_handle_t shellHandle,
      serial_handle_t serialHandle, char *prompt);
```

Then, after the initialization was successful, call a command to control MCUs.

This example shows how to call the SHELL_Init() given the user configuration structure.

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");
```

41.2.2 Advanced Feature

- Support to get a character from standard input devices.

  ```
  static shell_status_t SHELL_GetChar(shell_context_handle_t *shellContextHandle, uint8_t *ch);
  ```

Commands	Description
help	List all the registered commands.
exit	Exit program.

41.2.3 Shell Operation

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");
SHELL_Task((s_shellHandle);
```

Data Structures

- struct shell_command_t
  User command data configuration structure. More...

Macros

- #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE
  Whether use non-blocking mode.
- #define SHELL_AUTO_COMPLETE (1U)
  Macro to set on/off auto-complete feature.
- #define SHELL_BUFFER_SIZE (64U)
  Macro to set console buffer size.
- #define SHELL_MAX_ARGS (8U)
  Macro to set maximum arguments in command.
- #define SHELL_HISTORY_COUNT (3U)
  Macro to set maximum count of history commands.
- #define SHELL_IGNORE_PARAMETER_COUNT (0xFF)
  Macro to bypass arguments check.
- #define SHELL_HANDLE_SIZE
  The handle size of the shell module.
- #define SHELL_USE_COMMON_TASK (0U)
  Macro to determine whether use common task.
- #define SHELL_TASK_PRIORITY (2U)
  Macro to set shell task priority.
- #define SHELL_TASK_STACK_SIZE (1000U)
  Macro to set shell task stack size.
- #define SHELL_HANDLE_DEFINE(name) uint32_t name[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the shell handle.
- #define SHELL_COMMAND_DEFINE(command, descriptor, callback, paramCount)
  Defines the shell command structure.
- #define SHELL_COMMAND(command) &g_shellCommand##command
  Gets the shell command pointer.

Typedefs

- typedef void * shell_handle_t
  The handle of the shell module.
- typedef shell_status_t(* cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char **argv)
  User command function prototype.

Enumerations

- enum shell_status_t {
  kStatus_SHELL_Success = kStatus_Success,
  kStatus_SHELL_Error = MAKE_STATUS(kStatusGroup_SHELL, 1),
  kStatus_SHELL_OpenWriteHandleFailed = MAKE_STATUS(kStatusGroup_SHELL, 2),
  kStatus_SHELL_OpenReadHandleFailed = MAKE_STATUS(kStatusGroup_SHELL, 3) }
  Shell status.

Shell functional operation

- shell_status_t SHELL_Init (shell_handle_t shellHandle, serial_handle_t serialHandle, char *prompt)
  Initializes the shell module.
- shell_status_t SHELL_RegisterCommand (shell_handle_t shellHandle, shell_command_t *shell-Command)
  Registers the shell command.
- shell_status_t SHELL_UnregisterCommand (shell_command_t *shellCommand)
  Unregisters the shell command.
- shell_status_t SHELL_Write (shell_handle_t shellHandle, const char *buffer, uint32_t length)
  Sends data to the shell output stream.
- int SHELL_Printf (shell_handle_t shellHandle, const char *formatString,...)
  Writes formatted output to the shell output stream.
- shell_status_t SHELL_WriteSynchronization (shell_handle_t shellHandle, const char *buffer, uint32_t length)
  Sends data to the shell output stream with OS synchronization.
- int SHELL_PrintfSynchronization (shell_handle_t shellHandle, const char *formatString,...)
  Writes formatted output to the shell output stream with OS synchronization.
- void SHELL_ChangePrompt (shell_handle_t shellHandle, char *prompt)
  Change shell prompt.
- void SHELL_PrintPrompt (shell_handle_t shellHandle)
  Print shell prompt.
- void SHELL_Task (shell_handle_t shellHandle)
  The task function for Shell.
- static bool SHELL_checkRunningInIsr (void)
  Check if code is running in ISR.

41.3 Data Structure Documentation

41.3.1 struct shell_command_t

Data Fields

- const char * pcCommand
  The command that is executed.
- char * pcHelpString
  String that describes how to use the command.
- const cmd_function_t pFuncCallBack
  A pointer to the callback function that returns the output generated by the command.
- uint8_t cExpectedNumberOfParameters
  Commands expect a fixed number of parameters, which may be zero.
- list_element_t link
  link of the element

Field Documentation

(1) const char* shell_command_t::pcCommand

For example "help". It must be all lower case.

(2) char* shell_command_t::pcHelpString

It should start with the command itself, and end with "\r\n". For example "help: Returns a list of all the commands\r\n".

(3) const cmd_function_t shell_command_t::pFuncCallBack

(4) uint8_t shell_command_t::cExpectedNumberOfParameters

41.4 Macro Definition Documentation

41.4.1 #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE

41.4.2 #define SHELL_AUTO_COMPLETE (1U)

41.4.3 #define SHELL_BUFFER_SIZE (64U)

41.4.4 #define SHELL_MAX_ARGS (8U)

41.4.5 #define SHELL_HISTORY_COUNT (3U)

41.4.6 #define SHELL_HANDLE_SIZE

Value:

```
(160U + SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE +
      SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + \
     SERIAL_MANAGER_WRITE_HANDLE_SIZE)
```

It is the sum of the SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE + SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + SERIAL_MANAGER_WRITE_HANDLE_SIZE

41.4.7 #define SHELL_USE_COMMON_TASK (0U)

41.4.8 #define SHELL_TASK_PRIORITY (2U)

41.4.9 #define SHELL_TASK_STACK_SIZE (1000U)

41.4.10 #define SHELL_HANDLE_DEFINE(*name*) uint32_t name[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned shell handle. Then use "(shell_handle_t)name" to get the shell handle.

The macro should be global and could be optional. You could also define shell handle by yourself.

This is an example,

```
* SHELL_HANDLE_DEFINE(shellHandle);
*
```

Parameters

name	The name string of the shell handle.

41.4.11 #define SHELL_COMMAND_DEFINE(*command, descriptor, callback, paramCount*)

Value:

```
\
    shell_command_t g_shellCommand##command = {                         \
        (#command), (descriptor), (callback), (paramCount), {0},        \
    }
```

This macro is used to define the shell command structure shell_command_t. And then uses the macro SHELL_COMMAND to get the command structure pointer. The macro should not be used in any function.

This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

command	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".

descriptor	The description of the command is used for showing the command usage when "help" is typing.
callback	The callback of the command is used to handle the command line when the input command is matched.
paramCount	The max parameter count of the current command.

41.4.12 #define SHELL_COMMAND(*command*) &g_shellCommand##command

This macro is used to get the shell command pointer. The macro should not be used before the macro SHELL_COMMAND_DEFINE is used.

Parameters

command	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".

41.5 Typedef Documentation

41.5.1 typedef shell_status_t(∗ cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char ∗∗argv)

41.6 Enumeration Type Documentation

41.6.1 enum shell_status_t

Enumerator

kStatus_SHELL_Success Success.
kStatus_SHELL_Error Failed.
kStatus_SHELL_OpenWriteHandleFailed Open write handle failed.
kStatus_SHELL_OpenReadHandleFailed Open read handle failed.

41.7 Function Documentation

41.7.1 shell_status_t SHELL_Init (shell_handle_t *shellHandle*, serial_handle_t *serialHandle*, char ∗ *prompt*)

This function must be called before calling all other Shell functions. Call operation the Shell commands with user-defined settings. The example below shows how to set up the Shell and how to call the SHELL-_Init function by passing in these parameters. This is an example.

```
*   static SHELL_HANDLE_DEFINE(s_shellHandle);
*   SHELL_Init((shell_handle_t)s_shellHandle, (
      serial_handle_t)s_serialHandle, "Test@SHELL>");
*
```

Parameters

shellHandle	Pointer to point to a memory space of size SHELL_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SHELL_HANDLE_DEFINE(shellHandle); or uint32_t shellHandle[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];
serialHandle	The serial manager module handle pointer.
prompt	The string prompt pointer of Shell. Only the global variable can be passed.

Return values

kStatus_SHELL_Success	The shell initialization succeed.
kStatus_SHELL_Error	An error occurred when the shell is initialized.
kStatus_SHELL_OpenWriteHandleFailed	Open the write handle failed.
kStatus_SHELL_OpenReadHandleFailed	Open the read handle failed.

41.7.2 shell_status_t SHELL_RegisterCommand (shell_handle_t *shellHandle*, shell_command_t * *shellCommand*)

This function is used to register the shell command by using the command configuration shell_command_config_t. This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

shellHandle	The shell module handle pointer.
shellCommand	The command element.

Return values

kStatus_SHELL_Success	Successfully register the command.
kStatus_SHELL_Error	An error occurred.

41.7.3 shell_status_t SHELL_UnregisterCommand (shell_command_t * *shellCommand*)

This function is used to unregister the shell command.

Parameters

shellCommand	The command element.

Return values

kStatus_SHELL_Success	Successfully unregister the command.

41.7.4 shell_status_t SHELL_Write (shell_handle_t *shellHandle,* const char * *buffer,* uint32_t *length*)

This function is used to send data to the shell output stream.

Parameters

shellHandle	The shell module handle pointer.
buffer	Start address of the data to write.
length	Length of the data to write.

Return values

kStatus_SHELL_Success	Successfully send data.
kStatus_SHELL_Error	An error occurred.

41.7.5 int SHELL_Printf (shell_handle_t *shellHandle,* const char * *formatString,* ...)

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

Parameters

shellHandle	The shell module handle pointer.
formatString	Format string.

Returns

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

41.7.6 shell_status_t SHELL_WriteSynchronization (shell_handle_t *shellHandle,* const char * *buffer,* uint32_t *length*)

This function is used to send data to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

shellHandle	The shell module handle pointer.
buffer	Start address of the data to write.
length	Length of the data to write.

Return values

kStatus_SHELL_Success	Successfully send data.
kStatus_SHELL_Error	An error occurred.

41.7.7 int SHELL_PrintfSynchronization (shell_handle_t *shellHandle,* const char * *formatString,* ...)

Call this function to write a formatted output to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

shellHandle	The shell module handle pointer.

formatString	Format string.

Returns

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

41.7.8 void SHELL_ChangePrompt (shell_handle_t *shellHandle,* char * *prompt*)

Call this function to change shell prompt.

Parameters

shellHandle	The shell module handle pointer.
prompt	The string which will be used for command prompt

Returns

NULL.

41.7.9 void SHELL_PrintPrompt (shell_handle_t *shellHandle*)

Call this function to print shell prompt.

Parameters

shellHandle	The shell module handle pointer.

Returns

NULL.

41.7.10 void SHELL_Task (shell_handle_t *shellHandle*)

The task function for Shell; The function should be polled by upper layer. This function does not return until Shell command exit was called.

Parameters

shellHandle	The shell module handle pointer.

41.7.11 static bool SHELL_checkRunningInIsr (void) [inline], [static]

This function is used to check if code running in ISR.

Return values

TRUE	if code runing in ISR.

Chapter 42
CODEC Driver

42.1 Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

Modules

- CODEC Common Driver
- CODEC I2C Driver
- CS42888 Driver
- DA7212 Driver
- SGTL5000 Driver
- WM8904 Driver
- WM8960 Driver

42.2 CODEC Common Driver

42.2.1 Overview

The codec common driver provides a codec control abstraction interface.

Modules

- CODEC Adapter
- CS42888 Adapter
- DA7212 Adapter
- SGTL5000 Adapter
- WM8904 Adapter
- WM8960 Adapter

Data Structures

- struct codec_config_t
  Initialize structure of the codec. More...
- struct codec_capability_t
  codec capability More...
- struct codec_handle_t
  Codec handle definition. More...

Macros

- #define CODEC_VOLUME_MAX_VALUE (100U)
  codec maximum volume range

Enumerations

- enum {
  kStatus_CODEC_NotSupport = MAKE_STATUS(kStatusGroup_CODEC, 0U),
  kStatus_CODEC_DeviceNotRegistered = MAKE_STATUS(kStatusGroup_CODEC, 1U),
  kStatus_CODEC_I2CBusInitialFailed,
  kStatus_CODEC_I2CCommandTransferFailed }
  CODEC status.
- enum codec_audio_protocol_t {
  kCODEC_BusI2S = 0U,
  kCODEC_BusLeftJustified = 1U,
  kCODEC_BusRightJustified = 2U,
  kCODEC_BusPCMA = 3U,
  kCODEC_BusPCMB = 4U,
  kCODEC_BusTDM = 5U }

AUDIO format definition.

- enum {
  kCODEC_AudioSampleRate8KHz = 8000U,
  kCODEC_AudioSampleRate11025Hz = 11025U,
  kCODEC_AudioSampleRate12KHz = 12000U,
  kCODEC_AudioSampleRate16KHz = 16000U,
  kCODEC_AudioSampleRate22050Hz = 22050U,
  kCODEC_AudioSampleRate24KHz = 24000U,
  kCODEC_AudioSampleRate32KHz = 32000U,
  kCODEC_AudioSampleRate44100Hz = 44100U,
  kCODEC_AudioSampleRate48KHz = 48000U,
  kCODEC_AudioSampleRate96KHz = 96000U,
  kCODEC_AudioSampleRate192KHz = 192000U,
  kCODEC_AudioSampleRate384KHz = 384000U }
  audio sample rate definition
- enum {
  kCODEC_AudioBitWidth16bit = 16U,
  kCODEC_AudioBitWidth20bit = 20U,
  kCODEC_AudioBitWidth24bit = 24U,
  kCODEC_AudioBitWidth32bit = 32U }
  audio bit width
- enum codec_module_t {
  kCODEC_ModuleADC = 0U,
  kCODEC_ModuleDAC = 1U,
  kCODEC_ModulePGA = 2U,
  kCODEC_ModuleHeadphone = 3U,
  kCODEC_ModuleSpeaker = 4U,
  kCODEC_ModuleLinein = 5U,
  kCODEC_ModuleLineout = 6U,
  kCODEC_ModuleVref = 7U,
  kCODEC_ModuleMicbias = 8U,
  kCODEC_ModuleMic = 9U,
  kCODEC_ModuleI2SIn = 10U,
  kCODEC_ModuleI2SOut = 11U,
  kCODEC_ModuleMixer = 12U }
  audio codec module
- enum codec_module_ctrl_cmd_t { kCODEC_ModuleSwitchI2SInInterface = 0U }
  audio codec module control cmd
- enum {
  kCODEC_ModuleI2SInInterfacePCM = 0U,
  kCODEC_ModuleI2SInInterfaceDSD = 1U }
  audio codec module digital interface
- enum {

kCODEC_RecordSourceDifferentialLine = 1U,
kCODEC_RecordSourceLineInput = 2U,
kCODEC_RecordSourceDifferentialMic = 4U,
kCODEC_RecordSourceDigitalMic = 8U,
kCODEC_RecordSourceSingleEndMic = 16U }
audio codec module record source value

- enum {
kCODEC_RecordChannelLeft1 = 1U,
kCODEC_RecordChannelLeft2 = 2U,
kCODEC_RecordChannelLeft3 = 4U,
kCODEC_RecordChannelRight1 = 1U,
kCODEC_RecordChannelRight2 = 2U,
kCODEC_RecordChannelRight3 = 4U,
kCODEC_RecordChannelDifferentialPositive1 = 1U,
kCODEC_RecordChannelDifferentialPositive2 = 2U,
kCODEC_RecordChannelDifferentialPositive3 = 4U,
kCODEC_RecordChannelDifferentialNegative1 = 8U,
kCODEC_RecordChannelDifferentialNegative2 = 16U,
kCODEC_RecordChannelDifferentialNegative3 = 32U }
audio codec record channel
- enum {
kCODEC_PlaySourcePGA = 1U,
kCODEC_PlaySourceInput = 2U,
kCODEC_PlaySourceDAC = 4U,
kCODEC_PlaySourceMixerIn = 1U,
kCODEC_PlaySourceMixerInLeft = 2U,
kCODEC_PlaySourceMixerInRight = 4U,
kCODEC_PlaySourceAux = 8U }
audio codec module play source value
- enum {
kCODEC_PlayChannelHeadphoneLeft = 1U,
kCODEC_PlayChannelHeadphoneRight = 2U,
kCODEC_PlayChannelSpeakerLeft = 4U,
kCODEC_PlayChannelSpeakerRight = 8U,
kCODEC_PlayChannelLineOutLeft = 16U,
kCODEC_PlayChannelLineOutRight = 32U,
kCODEC_PlayChannelLeft0 = 1U,
kCODEC_PlayChannelRight0 = 2U,
kCODEC_PlayChannelLeft1 = 4U,
kCODEC_PlayChannelRight1 = 8U,
kCODEC_PlayChannelLeft2 = 16U,
kCODEC_PlayChannelRight2 = 32U,
kCODEC_PlayChannelLeft3 = 64U,
kCODEC_PlayChannelRight3 = 128U }
codec play channel
- enum {

kCODEC_VolumeHeadphoneLeft = 1U,
kCODEC_VolumeHeadphoneRight = 2U,
kCODEC_VolumeSpeakerLeft = 4U,
kCODEC_VolumeSpeakerRight = 8U,
kCODEC_VolumeLineOutLeft = 16U,
kCODEC_VolumeLineOutRight = 32U,
kCODEC_VolumeLeft0 = 1UL << 0U,
kCODEC_VolumeRight0 = 1UL << 1U,
kCODEC_VolumeLeft1 = 1UL << 2U,
kCODEC_VolumeRight1 = 1UL << 3U,
kCODEC_VolumeLeft2 = 1UL << 4U,
kCODEC_VolumeRight2 = 1UL << 5U,
kCODEC_VolumeLeft3 = 1UL << 6U,
kCODEC_VolumeRight3 = 1UL << 7U,
kCODEC_VolumeDAC = 1UL << 8U }

codec volume setting

- enum {

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

audio codec capability

Functions

- status_t CODEC_Init (codec_handle_t *handle, codec_config_t *config)
  Codec initilization.
- status_t CODEC_Deinit (codec_handle_t *handle)
  Codec de-initilization.
- status_t CODEC_SetFormat (codec_handle_t *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t CODEC_ModuleControl (codec_handle_t *handle, codec_module_ctrl_cmd_t cmd, uint32_t data)
  codec module control.
- status_t CODEC_SetVolume (codec_handle_t *handle, uint32_t channel, uint32_t volume)
  set audio codec pl volume.
- status_t CODEC_SetMute (codec_handle_t *handle, uint32_t channel, bool mute)
  set audio codec module mute.
- status_t CODEC_SetPower (codec_handle_t *handle, codec_module_t module, bool powerOn)
  set audio codec power.
- status_t CODEC_SetRecord (codec_handle_t *handle, uint32_t recordSource)
  codec set record source.
- status_t CODEC_SetRecordChannel (codec_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t CODEC_SetPlay (codec_handle_t *handle, uint32_t playSource)
  codec set play source.

Driver version

- #define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  CLOCK driver version 2.3.0.

42.2.2 Data Structure Documentation

42.2.2.1 struct codec_config_t

Data Fields

- uint32_t codecDevType
  codec type
- void * codecDevConfig
  Codec device specific configuration.

42.2.2.2 struct codec_capability_t

Data Fields

- uint32_t codecModuleCapability
  codec module capability
- uint32_t codecPlayCapability
  codec play capability
- uint32_t codecRecordCapability
  codec record capability
- uint32_t codecVolumeCapability
  codec volume capability

42.2.2.3 struct _codec_handle

codec handle declaration

- Application should allocate a buffer with CODEC_HANDLE_SIZE for handle definition, such as uint8_t codecHandleBuffer[CODEC_HANDLE_SIZE]; codec_handle_t ∗codecHandle = codecHandleBuffer;

Data Fields

- codec_config_t ∗ codecConfig
  codec configuration function pointer
- const codec_capability_t ∗ codecCapability
  codec capability
- uint8_t codecDevHandle [HAL_CODEC_HANDLER_SIZE]
  codec device handle

42.2.3 Macro Definition Documentation

42.2.3.1 #define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

42.2.4 Enumeration Type Documentation

42.2.4.1 anonymous enum

Enumerator

kStatus_CODEC_NotSupport CODEC not support status.
kStatus_CODEC_DeviceNotRegistered CODEC device register failed status.
kStatus_CODEC_I2CBusInitialFailed CODEC i2c bus initialization failed status.
kStatus_CODEC_I2CCommandTransferFailed CODEC i2c bus command transfer failed status.

42.2.4.2 enum codec_audio_protocol_t

Enumerator

kCODEC_BusI2S I2S type.
kCODEC_BusLeftJustified Left justified mode.
kCODEC_BusRightJustified Right justified mode.
kCODEC_BusPCMA DSP/PCM A mode.
kCODEC_BusPCMB DSP/PCM B mode.
kCODEC_BusTDM TDM mode.

42.2.4.3 anonymous enum

Enumerator

kCODEC_AudioSampleRate8KHz Sample rate 8000 Hz.
kCODEC_AudioSampleRate11025Hz Sample rate 11025 Hz.
kCODEC_AudioSampleRate12KHz Sample rate 12000 Hz.
kCODEC_AudioSampleRate16KHz Sample rate 16000 Hz.
kCODEC_AudioSampleRate22050Hz Sample rate 22050 Hz.
kCODEC_AudioSampleRate24KHz Sample rate 24000 Hz.
kCODEC_AudioSampleRate32KHz Sample rate 32000 Hz.
kCODEC_AudioSampleRate44100Hz Sample rate 44100 Hz.
kCODEC_AudioSampleRate48KHz Sample rate 48000 Hz.
kCODEC_AudioSampleRate96KHz Sample rate 96000 Hz.
kCODEC_AudioSampleRate192KHz Sample rate 192000 Hz.
kCODEC_AudioSampleRate384KHz Sample rate 384000 Hz.

42.2.4.4 anonymous enum

Enumerator

kCODEC_AudioBitWidth16bit audio bit width 16
kCODEC_AudioBitWidth20bit audio bit width 20
kCODEC_AudioBitWidth24bit audio bit width 24
kCODEC_AudioBitWidth32bit audio bit width 32

42.2.4.5 enum codec_module_t

Enumerator

kCODEC_ModuleADC codec module ADC
kCODEC_ModuleDAC codec module DAC
kCODEC_ModulePGA codec module PGA
kCODEC_ModuleHeadphone codec module headphone

kCODEC_ModuleSpeaker codec module speaker
kCODEC_ModuleLinein codec module linein
kCODEC_ModuleLineout codec module lineout
kCODEC_ModuleVref codec module VREF
kCODEC_ModuleMicbias codec module MIC BIAS
kCODEC_ModuleMic codec module MIC
kCODEC_ModuleI2SIn codec module I2S in
kCODEC_ModuleI2SOut codec module I2S out
kCODEC_ModuleMixer codec module mixer

42.2.4.6 enum codec_module_ctrl_cmd_t

Enumerator

kCODEC_ModuleSwitchI2SInInterface module digital interface siwtch.

42.2.4.7 anonymous enum

Enumerator

kCODEC_ModuleI2SInInterfacePCM Pcm interface.
kCODEC_ModuleI2SInInterfaceDSD DSD interface.

42.2.4.8 anonymous enum

Enumerator

kCODEC_RecordSourceDifferentialLine record source from differential line
kCODEC_RecordSourceLineInput record source from line input
kCODEC_RecordSourceDifferentialMic record source from differential mic
kCODEC_RecordSourceDigitalMic record source from digital microphone
kCODEC_RecordSourceSingleEndMic record source from single microphone

42.2.4.9 anonymous enum

Enumerator

kCODEC_RecordChannelLeft1 left record channel 1
kCODEC_RecordChannelLeft2 left record channel 2
kCODEC_RecordChannelLeft3 left record channel 3
kCODEC_RecordChannelRight1 right record channel 1
kCODEC_RecordChannelRight2 right record channel 2
kCODEC_RecordChannelRight3 right record channel 3
kCODEC_RecordChannelDifferentialPositive1 differential positive record channel 1

kCODEC_RecordChannelDifferentialPositive2 differential positive record channel 2
kCODEC_RecordChannelDifferentialPositive3 differential positive record channel 3
kCODEC_RecordChannelDifferentialNegative1 differential negative record channel 1
kCODEC_RecordChannelDifferentialNegative2 differential negative record channel 2
kCODEC_RecordChannelDifferentialNegative3 differential negative record channel 3

42.2.4.10 anonymous enum

Enumerator

kCODEC_PlaySourcePGA play source PGA, bypass ADC
kCODEC_PlaySourceInput play source Input3
kCODEC_PlaySourceDAC play source DAC
kCODEC_PlaySourceMixerIn play source mixer in
kCODEC_PlaySourceMixerInLeft play source mixer in left
kCODEC_PlaySourceMixerInRight play source mixer in right
kCODEC_PlaySourceAux play source mixer in AUx

42.2.4.11 anonymous enum

Enumerator

kCODEC_PlayChannelHeadphoneLeft play channel headphone left
kCODEC_PlayChannelHeadphoneRight play channel headphone right
kCODEC_PlayChannelSpeakerLeft play channel speaker left
kCODEC_PlayChannelSpeakerRight play channel speaker right
kCODEC_PlayChannelLineOutLeft play channel lineout left
kCODEC_PlayChannelLineOutRight play channel lineout right
kCODEC_PlayChannelLeft0 play channel left0
kCODEC_PlayChannelRight0 play channel right0
kCODEC_PlayChannelLeft1 play channel left1
kCODEC_PlayChannelRight1 play channel right1
kCODEC_PlayChannelLeft2 play channel left2
kCODEC_PlayChannelRight2 play channel right2
kCODEC_PlayChannelLeft3 play channel left3
kCODEC_PlayChannelRight3 play channel right3

42.2.4.12 anonymous enum

Enumerator

kCODEC_VolumeHeadphoneLeft headphone left volume
kCODEC_VolumeHeadphoneRight headphone right volume
kCODEC_VolumeSpeakerLeft speaker left volume
kCODEC_VolumeSpeakerRight speaker right volume

kCODEC_VolumeLineOutLeft lineout left volume
kCODEC_VolumeLineOutRight lineout right volume
kCODEC_VolumeLeft0 left0 volume
kCODEC_VolumeRight0 right0 volume
kCODEC_VolumeLeft1 left1 volume
kCODEC_VolumeRight1 right1 volume
kCODEC_VolumeLeft2 left2 volume
kCODEC_VolumeRight2 right2 volume
kCODEC_VolumeLeft3 left3 volume
kCODEC_VolumeRight3 right3 volume
kCODEC_VolumeDAC dac volume

42.2.4.13 anonymous enum

Enumerator

kCODEC_SupportModuleADC codec capability of module ADC
kCODEC_SupportModuleDAC codec capability of module DAC
kCODEC_SupportModulePGA codec capability of module PGA
kCODEC_SupportModuleHeadphone codec capability of module headphone
kCODEC_SupportModuleSpeaker codec capability of module speaker
kCODEC_SupportModuleLinein codec capability of module linein
kCODEC_SupportModuleLineout codec capability of module lineout
kCODEC_SupportModuleVref codec capability of module vref
kCODEC_SupportModuleMicbias codec capability of module mic bias
kCODEC_SupportModuleMic codec capability of module mic bias
kCODEC_SupportModuleI2SIn codec capability of module I2S in
kCODEC_SupportModuleI2SOut codec capability of module I2S out
kCODEC_SupportModuleMixer codec capability of module mixer
kCODEC_SupportModuleI2SInSwitchInterface codec capability of module I2S in switch interface

kCODEC_SupportPlayChannelLeft0 codec capability of play channel left 0
kCODEC_SupportPlayChannelRight0 codec capability of play channel right 0
kCODEC_SupportPlayChannelLeft1 codec capability of play channel left 1
kCODEC_SupportPlayChannelRight1 codec capability of play channel right 1
kCODEC_SupportPlayChannelLeft2 codec capability of play channel left 2
kCODEC_SupportPlayChannelRight2 codec capability of play channel right 2
kCODEC_SupportPlayChannelLeft3 codec capability of play channel left 3
kCODEC_SupportPlayChannelRight3 codec capability of play channel right 3
kCODEC_SupportPlaySourcePGA codec capability of set playback source PGA
kCODEC_SupportPlaySourceInput codec capability of set playback source INPUT
kCODEC_SupportPlaySourceDAC codec capability of set playback source DAC
kCODEC_SupportPlaySourceMixerIn codec capability of set play source Mixer in
kCODEC_SupportPlaySourceMixerInLeft codec capability of set play source Mixer in left
kCODEC_SupportPlaySourceMixerInRight codec capability of set play source Mixer in right

kCODEC_SupportPlaySourceAux codec capability of set play source aux
kCODEC_SupportRecordSourceDifferentialLine codec capability of record source differential line

kCODEC_SupportRecordSourceLineInput codec capability of record source line input
kCODEC_SupportRecordSourceDifferentialMic codec capability of record source differential mic

kCODEC_SupportRecordSourceDigitalMic codec capability of record digital mic
kCODEC_SupportRecordSourceSingleEndMic codec capability of single end mic
kCODEC_SupportRecordChannelLeft1 left record channel 1
kCODEC_SupportRecordChannelLeft2 left record channel 2
kCODEC_SupportRecordChannelLeft3 left record channel 3
kCODEC_SupportRecordChannelRight1 right record channel 1
kCODEC_SupportRecordChannelRight2 right record channel 2
kCODEC_SupportRecordChannelRight3 right record channel 3

42.2.5 Function Documentation

42.2.5.1 status_t CODEC_Init (codec_handle_t * *handle,* codec_config_t * *config*)

Parameters

handle	codec handle.
config	codec configurations.

Returns

kStatus_Success is success, else de-initial failed.

42.2.5.2 status_t CODEC_Deinit (codec_handle_t * *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.2.5.3 status_t CODEC_SetFormat (codec_handle_t * *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.2.5.4 status_t CODEC_ModuleControl (codec_handle_t * *handle*, codec_module_ctrl_cmd_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature.

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

42.2.5.5 status_t CODEC_SetVolume (codec_handle_t * *handle*, uint32_t *channel*, uint32_t *volume*)

Parameters

handle	codec handle.
channel	audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel.
volume	volume value, support 0 ∼ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.2.5.6 status_t CODEC_SetMute (codec_handle_t * *handle*, uint32_t *channel*, bool *mute*)

Parameters

handle	codec handle.
channel	audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel.
mute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.2.5.7 status_t CODEC_SetPower (codec_handle_t * *handle*, codec_module_t *module*, bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.2.5.8 status_t CODEC_SetRecord (codec_handle_t * *handle*, uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.2.5.9 status_t CODEC_SetRecordChannel (codec_handle_t * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.2.5.10 status_t CODEC_SetPlay (codec_handle_t * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.3 CODEC I2C Driver

42.3.1 Overview

The codec common driver provides a codec control abstraction interface.

Data Structures

- struct codec_i2c_config_t
  CODEC I2C configurations structure. More...

Macros

- #define CODEC_I2C_MASTER_HANDLER_SIZE HAL_I2C_MASTER_HANDLE_SIZE
  codec i2c handler

Enumerations

- enum codec_reg_addr_t {
  kCODEC_RegAddr8Bit = 1U,
  kCODEC_RegAddr16Bit = 2U }
  CODEC device register address type.
- enum codec_reg_width_t {
  kCODEC_RegWidth8Bit = 1U,
  kCODEC_RegWidth16Bit = 2U,
  kCODEC_RegWidth32Bit = 4U }
  CODEC device register width.

Functions

- status_t CODEC_I2C_Init (void ∗handle, uint32_t i2cInstance, uint32_t i2cBaudrate, uint32_t i2cSourceClockHz)
  Codec i2c bus initilization.
- status_t CODEC_I2C_Deinit (void ∗handle)
  Codec i2c de-initilization.
- status_t CODEC_I2C_Send (void ∗handle, uint8_t deviceAddress, uint32_t subAddress, uint8_t subaddressSize, uint8_t ∗txBuff, uint8_t txBuffSize)
  codec i2c send function.
- status_t CODEC_I2C_Receive (void ∗handle, uint8_t deviceAddress, uint32_t subAddress, uint8_t subaddressSize, uint8_t ∗rxBuff, uint8_t rxBuffSize)
  codec i2c receive function.

42.3.2 Data Structure Documentation

42.3.2.1 struct codec_i2c_config_t

Data Fields

- uint32_t codecI2CInstance
  i2c bus instance
- uint32_t codecI2CSourceClock
  i2c bus source clock frequency

42.3.3 Enumeration Type Documentation

42.3.3.1 enum codec_reg_addr_t

Enumerator

kCODEC_RegAddr8Bit 8-bit register address.
kCODEC_RegAddr16Bit 16-bit register address.

42.3.3.2 enum codec_reg_width_t

Enumerator

kCODEC_RegWidth8Bit 8-bit register width.
kCODEC_RegWidth16Bit 16-bit register width.
kCODEC_RegWidth32Bit 32-bit register width.

42.3.4 Function Documentation

42.3.4.1 status_t CODEC_I2C_Init (void * *handle,* uint32_t *i2cInstance,* uint32_t *i2cBaudrate,* uint32_t *i2cSourceClockHz*)

Parameters

handle	i2c master handle.
i2cInstance	instance number of the i2c bus, such as 0 is corresponding to I2C0.

i2cBaudrate	i2c baudrate.
i2cSource-ClockHz	i2c source clock frequency.

Returns

kStatus_HAL_I2cSuccess is success, else initial failed.

42.3.4.2 status_t CODEC_I2C_Deinit (void * *handle*)

Parameters

handle	i2c master handle.

Returns

kStatus_HAL_I2cSuccess is success, else deinitial failed.

42.3.4.3 status_t CODEC_I2C_Send (void * *handle,* uint8_t *deviceAddress,* uint32_t *subAddress,* uint8_t *subaddressSize,* uint8_t * *txBuff,* uint8_t *txBuffSize*)

Parameters

handle	i2c master handle.
deviceAddress	codec device address.
subAddress	register address.
subaddressSize	register address width.
txBuff	tx buffer pointer.
txBuffSize	tx buffer size.

Returns

kStatus_HAL_I2cSuccess is success, else send failed.

42.3.4.4 status_t CODEC_I2C_Receive (void * *handle,* uint8_t *deviceAddress,* uint32_t *subAddress,* uint8_t *subaddressSize,* uint8_t * *rxBuff,* uint8_t *rxBuffSize*)

Parameters

handle	i2c master handle.
deviceAddress	codec device address.
subAddress	register address.
subaddressSize	register address width.
rxBuff	rx buffer pointer.
rxBuffSize	rx buffer size.

Returns

kStatus_HAL_I2cSuccess is success, else receive failed.

42.4 CS42888 Driver

42.4.1 Overview

The cs42888 driver provides a codec control interface.

Data Structures

- struct cs42888_audio_format_t
  cs42888 audio format More...
- struct cs42888_config_t
  Initialize structure of CS42888. More...
- struct cs42888_handle_t
  cs42888 handler More...

Macros

- #define CS42888_I2C_HANDLER_SIZE CODEC_I2C_MASTER_HANDLER_SIZE
  CS42888 handle size.
- #define CS42888_ID 0x01U
  Define the register address of CS42888.
- #define CS42888_AOUT_MAX_VOLUME_VALUE 0xFFU
  CS42888 volume setting range.
- #define CS42888_CACHEREGNUM 28U
  Cache register number.
- #define CS42888_I2C_ADDR 0x48U
  CS42888 I2C address.
- #define CS42888_I2C_BITRATE (100000U)
  CS42888 I2C baudrate.

Typedefs

- typedef void(∗ cs42888_reset)(bool state)
  cs42888 reset function pointer

Enumerations

- enum cs42888_func_mode {
  kCS42888_ModeMasterSSM = 0x0,
  kCS42888_ModeMasterDSM = 0x1,
  kCS42888_ModeMasterQSM = 0x2,
  kCS42888_ModeSlave = 0x3 }
  CS42888 support modes.

- enum cs42888_module_t {
  kCS42888_ModuleDACPair1 = 0x2,
  kCS42888_ModuleDACPair2 = 0x4,
  kCS42888_ModuleDACPair3 = 0x8,
  kCS42888_ModuleDACPair4 = 0x10,
  kCS42888_ModuleADCPair1 = 0x20,
  kCS42888_ModuleADCPair2 = 0x40 }
  Modules in CS42888 board.
- enum cs42888_bus_t {
  kCS42888_BusLeftJustified = 0x0,
  kCS42888_BusI2S = 0x1,
  kCS42888_BusRightJustified = 0x2,
  kCS42888_BusOL1 = 0x4,
  kCS42888_BusOL2 = 0x5,
  kCS42888_BusTDM = 0x6 }
  CS42888 supported audio bus type.
- enum {
  kCS42888_AOUT1 = 1U,
  kCS42888_AOUT2 = 2U,
  kCS42888_AOUT3 = 3U,
  kCS42888_AOUT4 = 4U,
  kCS42888_AOUT5 = 5U,
  kCS42888_AOUT6 = 6U,
  kCS42888_AOUT7 = 7U,
  kCS42888_AOUT8 = 8U }
  CS428888 play channel.

Functions

- status_t CS42888_Init (cs42888_handle_t *handle, cs42888_config_t *config)
  CS42888 initialize function.
- status_t CS42888_Deinit (cs42888_handle_t *handle)
  Deinit the CS42888 codec.
- status_t CS42888_SetProtocol (cs42888_handle_t *handle, cs42888_bus_t protocol, uint32_t bitWidth)
  Set the audio transfer protocol.
- void CS42888_SetFuncMode (cs42888_handle_t *handle, cs42888_func_mode mode)
  Set CS42888 to differernt working mode.
- status_t CS42888_SelectFunctionalMode (cs42888_handle_t *handle, cs42888_func_mode adcMode, cs42888_func_mode dacMode)
  Set CS42888 to differernt functional mode.
- status_t CS42888_SetAOUTVolume (cs42888_handle_t *handle, uint8_t channel, uint8_t volume)
  Set the volume of different modules in CS42888.
- status_t CS42888_SetAINVolume (cs42888_handle_t *handle, uint8_t channel, uint8_t volume)
  Set the volume of different modules in CS42888.
- uint8_t CS42888_GetAOUTVolume (cs42888_handle_t *handle, uint8_t channel)

Get the volume of different AOUT channel in CS42888.
- uint8_t CS42888_GetAINVolume (cs42888_handle_t *handle, uint8_t channel)
  Get the volume of different AIN channel in CS42888.
- status_t CS42888_SetMute (cs42888_handle_t *handle, uint8_t channelMask)
  Mute modules in CS42888.
- status_t CS42888_SetChannelMute (cs42888_handle_t *handle, uint8_t channel, bool isMute)
  Mute channel modules in CS42888.
- status_t CS42888_SetModule (cs42888_handle_t *handle, cs42888_module_t module, bool isEnabled)
  Enable/disable expected devices.
- status_t CS42888_ConfigDataFormat (cs42888_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)
  Configure the data format of audio data.
- status_t CS42888_WriteReg (cs42888_handle_t *handle, uint8_t reg, uint8_t val)
  Write register to CS42888 using I2C.
- status_t CS42888_ReadReg (cs42888_handle_t *handle, uint8_t reg, uint8_t *val)
  Read register from CS42888 using I2C.
- status_t CS42888_ModifyReg (cs42888_handle_t *handle, uint8_t reg, uint8_t mask, uint8_t val)
  Modify some bits in the register using I2C.

Driver version

- #define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))
  cs42888 driver version 2.1.3.

42.4.2 Data Structure Documentation

42.4.2.1 struct cs42888_audio_format_t

Data Fields

- uint32_t mclk_HZ
  master clock frequency
- uint32_t sampleRate
  sample rate
- uint32_t bitWidth
  bit width

42.4.2.2 struct cs42888_config_t

Data Fields

- cs42888_bus_t bus
  Audio transfer protocol.
- cs42888_audio_format_t format
  cs42888 audio format

- cs42888_func_mode ADCMode
  CS42888 ADC function mode.
- cs42888_func_mode DACMode
  CS42888 DAC function mode.
- bool master
  true is master, false is slave
- codec_i2c_config_t i2cConfig
  i2c bus configuration
- uint8_t slaveAddress
  slave address
- cs42888_reset reset
  reset function pointer

Field Documentation

(1) cs42888_func_mode cs42888_config_t::ADCMode

(2) cs42888_func_mode cs42888_config_t::DACMode

42.4.2.3 struct cs42888_handle_t

Data Fields

- cs42888_config_t * config
  cs42888 config pointer
- uint8_t i2cHandle [CS42888_I2C_HANDLER_SIZE]
  i2c handle pointer

42.4.3 Macro Definition Documentation

42.4.3.1 #define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

42.4.3.2 #define CS42888_ID 0x01U

42.4.3.3 #define CS42888_I2C_ADDR 0x48U

42.4.4 Enumeration Type Documentation

42.4.4.1 enum cs42888_func_mode

Enumerator

kCS42888_ModeMasterSSM master single speed mode
kCS42888_ModeMasterDSM master dual speed mode
kCS42888_ModeMasterQSM master quad speed mode
kCS42888_ModeSlave master single speed mode

42.4.4.2 enum cs42888_module_t

Enumerator

kCS42888_ModuleDACPair1 DAC pair1 (AOUT1 and AOUT2) module in CS42888.
kCS42888_ModuleDACPair2 DAC pair2 (AOUT3 and AOUT4) module in CS42888.
kCS42888_ModuleDACPair3 DAC pair3 (AOUT5 and AOUT6) module in CS42888.
kCS42888_ModuleDACPair4 DAC pair4 (AOUT7 and AOUT8) module in CS42888.
kCS42888_ModuleADCPair1 ADC pair1 (AIN1 and AIN2) module in CS42888.
kCS42888_ModuleADCPair2 ADC pair2 (AIN3 and AIN4) module in CS42888.

42.4.4.3 enum cs42888_bus_t

Enumerator

kCS42888_BusLeftJustified Left justified format, up to 24 bits.
kCS42888_BusI2S I2S format, up to 24 bits.
kCS42888_BusRightJustified Right justified, can support 16bits and 24 bits.
kCS42888_BusOL1 One-Line #1 mode.
kCS42888_BusOL2 One-Line #2 mode.
kCS42888_BusTDM TDM mode.

42.4.4.4 anonymous enum

Enumerator

kCS42888_AOUT1 aout1
kCS42888_AOUT2 aout2
kCS42888_AOUT3 aout3
kCS42888_AOUT4 aout4
kCS42888_AOUT5 aout5
kCS42888_AOUT6 aout6
kCS42888_AOUT7 aout7
kCS42888_AOUT8 aout8

42.4.5 Function Documentation

42.4.5.1 status_t CS42888_Init (cs42888_handle_t * *handle,* cs42888_config_t * *config*)

The second parameter is NULL to CS42888 in this version. If users want to change the settings, they have to use cs42888_write_reg() or cs42888_modify_reg() to set the register value of CS42888. Note: If the codec_config is NULL, it would initialize CS42888 using default settings. The default setting: codec_config->bus = kCS42888_BusI2S codec_config->ADCmode = kCS42888_ModeSlave codec_config->DACmode = kCS42888_ModeSlave

Parameters

handle	CS42888 handle structure.
config	CS42888 configuration structure.

42.4.5.2 status_t CS42888_Deinit (cs42888_handle_t * *handle*)

This function close all modules in CS42888 to save power.

Parameters

handle	CS42888 handle structure pointer.

42.4.5.3 status_t CS42888_SetProtocol (cs42888_handle_t * *handle*, cs42888_bus_t *protocol*, uint32_t *bitWidth*)

CS42888 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

handle	CS42888 handle structure.
protocol	Audio data transfer protocol.
bitWidth	bit width

42.4.5.4 void CS42888_SetFuncMode (cs42888_handle_t * *handle*, cs42888_func_mode *mode*)

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

Parameters

handle	CS42888 handle structure.
mode	differenht working mode of CS42888.

42.4.5.5 status_t CS42888_SelectFunctionalMode (cs42888_handle_t * *handle*, cs42888_func_mode *adcMode*, cs42888_func_mode *dacMode*)

Parameters

handle	CS42888 handle structure.
adcMode	differenht working mode of CS42888.
dacMode	differenht working mode of CS42888.

42.4.5.6 status_t CS42888_SetAOUTVolume (cs42888_handle_t * *handle*, uint8_t *channel*, uint8_t *volume*)

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AOUT channel, it shall be 1∼8.
volume	Volume value need to be set.

42.4.5.7 status_t CS42888_SetAINVolume (cs42888_handle_t * *handle*, uint8_t *channel*, uint8_t *volume*)

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AIN channel, it shall be 1∼4.
volume	Volume value need to be set.

42.4.5.8 uint8_t CS42888_GetAOUTVolume (cs42888_handle_t * *handle*, uint8_t *channel*)

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AOUT channel, it shall be 1∼8.

42.4.5.9 uint8_t CS42888_GetAINVolume (cs42888_handle_t * *handle,* uint8_t *channel*)

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AIN channel, it shall be 1∼4.

42.4.5.10 status_t CS42888_SetMute (cs42888_handle_t * *handle,* uint8_t *channelMask*)

Parameters

handle	CS42888 handle structure.
channelMask	Channel mask for mute. Mute channel 0, it shall be 0x1, while mute channel 0 and 1, it shall be 0x3. Mute all channel, it shall be 0xFF. Each bit represent one channel, 1 means mute, 0 means unmute.

42.4.5.11 status_t CS42888_SetChannelMute (cs42888_handle_t * *handle,* uint8_t *channel,* bool *isMute*)

Parameters

handle	CS42888 handle structure.
channel	reference _cs42888_play_channel.
isMute	true is mute, falase is unmute.

42.4.5.12 status_t CS42888_SetModule (cs42888_handle_t * *handle,* cs42888_module_t *module,* bool *isEnabled*)

Parameters

handle	CS42888 handle structure.
module	Module expected to enable.
isEnabled	Enable or disable moudles.

42.4.5.13 status_t CS42888_ConfigDataFormat (cs42888_handle_t * *handle*, uint32_t *mclk*, uint32_t *sample_rate*, uint32_t *bits*)

This function would configure the registers about the sample rate, bit depths.

Parameters

handle	CS42888 handle structure pointer.
mclk	Master clock frequency of I2S.
sample_rate	Sample rate of audio file running in CS42888. CS42888 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
bits	Bit depth of audio file (CS42888 only supports 16bit, 20bit, 24bit and 32 bit in HW).

42.4.5.14 status_t CS42888_WriteReg (cs42888_handle_t * *handle*, uint8_t *reg*, uint8_t *val*)

Parameters

handle	CS42888 handle structure.
reg	The register address in CS42888.
val	Value needs to write into the register.

42.4.5.15 status_t CS42888_ReadReg (cs42888_handle_t * *handle*, uint8_t *reg*, uint8_t * *val*)

Parameters

handle	CS42888 handle structure.
reg	The register address in CS42888.
val	Value written to.

42.4.5.16 status_t CS42888_ModifyReg (cs42888_handle_t * *handle,* uint8_t *reg,* uint8_t *mask,* uint8_t *val*)

Parameters

handle	CS42888 handle structure.
reg	The register address in CS42888.
mask	The mask code for the bits want to write. The bit you want to write should be 0.
val	Value needs to write into the register.

42.4.6 CS42888 Adapter

42.4.6.1 Overview

The cs42888 adapter provides a codec unify control interface.

Macros

- #define HAL_CODEC_CS42888_HANDLER_SIZE (CS42888_I2C_HANDLER_SIZE + 4)
  codec handler size

Functions

- status_t HAL_CODEC_CS42888_Init (void *handle, void *config)
  Codec initilization.
- status_t HAL_CODEC_CS42888_Deinit (void *handle)
  Codec de-initilization.
- status_t HAL_CODEC_CS42888_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_CS42888_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_CS42888_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_CS42888_SetPower (void *handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_CS42888_SetRecord (void *handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_CS42888_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_CS42888_SetPlay (void *handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_CS42888_ModuleControl (void *handle, uint32_t cmd, uint32_t data)
  codec module control.
- static status_t HAL_CODEC_Init (void *handle, void *config)
  Codec initilization.
- static status_t HAL_CODEC_Deinit (void *handle)
  Codec de-initilization.
- static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)

set audio codec module power.

- static status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- static status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- static status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- static status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

42.4.6.2 Function Documentation

42.4.6.2.1 status_t HAL_CODEC_CS42888_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

42.4.6.2.2 status_t HAL_CODEC_CS42888_Deinit (void ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.4.6.2.3 status_t HAL_CODEC_CS42888_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.4 status_t HAL_CODEC_CS42888_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.5 status_t HAL_CODEC_CS42888_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.6 status_t HAL_CODEC_CS42888_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.7 status_t HAL_CODEC_CS42888_SetRecord (void * *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.8 status_t HAL_CODEC_CS42888_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.9 status_t HAL_CODEC_CS42888_SetPlay (void * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.10 status_t HAL_CODEC_CS42888_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.11 static status_t HAL_CODEC_Init (void * *handle,* void * *config*) [inline], [static]

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

42.4.6.2.12 static status_t HAL_CODEC_Deinit (void * *handle*) [inline], [static]

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.4.6.2.13 static status_t HAL_CODEC_SetFormat (void * *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*) [inline], [static]

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.14 static status_t HAL_CODEC_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.15 static status_t HAL_CODEC_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.16 static status_t HAL_CODEC_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*) [inline], [static]

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.17 static status_t HAL_CODEC_SetRecord (void * *handle,* uint32_t *recordSource*) [inline], [static]

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.18 static status_t HAL_CODEC_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*) [inline], [static]

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.19 static status_t HAL_CODEC_SetPlay (void * *handle,* uint32_t *playSource*) [inline], [static]

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.4.6.2.20 static status_t HAL_CODEC_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*) [inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

42.5 DA7212 Driver

42.5.1 Overview

The da7212 driver provides a codec control interface.

Data Structures

- struct da7212_pll_config_t
  da7212 pll configuration More...
- struct da7212_audio_format_t
  da7212 audio format More...
- struct da7212_config_t
  DA7212 configure structure. More...
- struct da7212_handle_t
  da7212 codec handler More...

Macros

- #define DA7212_I2C_HANDLER_SIZE CODEC_I2C_MASTER_HANDLER_SIZE
  da7212 handle size
- #define DA7212_ADDRESS (0x1A)
  DA7212 I2C address.
- #define DA7212_HEADPHONE_MAX_VOLUME_VALUE 0x3FU
  da7212 volume setting range

Enumerations

- enum da7212_Input_t {
  kDA7212_Input_AUX = 0x0,
  kDA7212_Input_MIC1_Dig,
  kDA7212_Input_MIC1_An,
  kDA7212_Input_MIC2 }
  DA7212 input source select.
- enum _da7212_play_channel {
  kDA7212_HeadphoneLeft = 1U,
  kDA7212_HeadphoneRight = 2U,
  kDA7212_Speaker = 4U }
  da7212 play channel
- enum da7212_Output_t {
  kDA7212_Output_HP = 0x0,
  kDA7212_Output_SP }
  DA7212 output device select.

- enum _da7212_module {
  kDA7212_ModuleADC,
  kDA7212_ModuleDAC,
  kDA7212_ModuleHeadphone,
  kDA7212_ModuleSpeaker }
  DA7212 module.
- enum da7212_dac_source_t {
  kDA7212_DACSourceADC = 0x0U,
  kDA7212_DACSourceInputStream = 0x3U }
  DA7212 functionality.
- enum da7212_volume_t {
  kDA7212_DACGainMute = 0x7,
  kDA7212_DACGainM72DB = 0x17,
  kDA7212_DACGainM60DB = 0x1F,
  kDA7212_DACGainM54DB = 0x27,
  kDA7212_DACGainM48DB = 0x2F,
  kDA7212_DACGainM42DB = 0x37,
  kDA7212_DACGainM36DB = 0x3F,
  kDA7212_DACGainM30DB = 0x47,
  kDA7212_DACGainM24DB = 0x4F,
  kDA7212_DACGainM18DB = 0x57,
  kDA7212_DACGainM12DB = 0x5F,
  kDA7212_DACGainM6DB = 0x67,
  kDA7212_DACGain0DB = 0x6F,
  kDA7212_DACGain6DB = 0x77,
  kDA7212_DACGain12DB = 0x7F }
  DA7212 volume.
- enum da7212_protocol_t {
  kDA7212_BusI2S = 0x0,
  kDA7212_BusLeftJustified,
  kDA7212_BusRightJustified,
  kDA7212_BusDSPMode }
  The audio data transfer protocol choice.
- enum da7212_sys_clk_source_t {
  kDA7212_SysClkSourceMCLK = 0U,
  kDA7212_SysClkSourcePLL = 1U << 14 }
  da7212 system clock source
- enum da7212_pll_clk_source_t { kDA7212_PLLClkSourceMCLK = 0U }
  DA7212 pll clock source.
- enum da7212_pll_out_clk_t {
  kDA7212_PLLOutputClk11289600 = 11289600U,
  kDA7212_PLLOutputClk12288000 = 12288000U }
  DA7212 output clock frequency.
- enum da7212_master_bits_t {

kDA7212_MasterBits32PerFrame = 0U,
kDA7212_MasterBits64PerFrame = 1U,
kDA7212_MasterBits128PerFrame = 2U,
kDA7212_MasterBits256PerFrame = 3U }
master mode bits per frame

Functions

- status_t DA7212_Init (da7212_handle_t *handle, da7212_config_t *codecConfig)
  DA7212 initialize function.
- status_t DA7212_ConfigAudioFormat (da7212_handle_t *handle, uint32_t masterClock_Hz, uint32_t sampleRate_Hz, uint32_t dataBits)
  Set DA7212 audio format.
- status_t DA7212_SetPLLConfig (da7212_handle_t *handle, da7212_pll_config_t *config)
  DA7212 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.
- void DA7212_ChangeHPVolume (da7212_handle_t *handle, da7212_volume_t volume)
  Set DA7212 playback volume.
- void DA7212_Mute (da7212_handle_t *handle, bool isMuted)
  Mute or unmute DA7212.
- void DA7212_ChangeInput (da7212_handle_t *handle, da7212_Input_t DA7212_Input)
  Set the input data source of DA7212.
- void DA7212_ChangeOutput (da7212_handle_t *handle, da7212_Output_t DA7212_Output)
  Set the output device of DA7212.
- status_t DA7212_SetChannelVolume (da7212_handle_t *handle, uint32_t channel, uint32_t volume)
  Set module volume.
- status_t DA7212_SetChannelMute (da7212_handle_t *handle, uint32_t channel, bool isMute)
  Set module mute.
- status_t DA7212_SetProtocol (da7212_handle_t *handle, da7212_protocol_t protocol)
  Set protocol for DA7212.
- status_t DA7212_SetMasterModeBits (da7212_handle_t *handle, uint32_t bitWidth)
  Set master mode bits per frame for DA7212.
- status_t DA7212_WriteRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t u8RegisterData)
  Write a register for DA7212.
- status_t DA7212_ReadRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t *pu8RegisterData)
  Get a register value of DA7212.
- status_t DA7212_Deinit (da7212_handle_t *handle)
  Deinit DA7212.

Driver version

- #define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))
  CLOCK driver version 2.2.2.

42.5.2 Data Structure Documentation

42.5.2.1 struct da7212_pll_config_t

Data Fields

- da7212_pll_clk_source_t source
  pll reference clock source
- uint32_t refClock_HZ
  pll reference clock frequency
- da7212_pll_out_clk_t outputClock_HZ
  pll output clock frequency

42.5.2.2 struct da7212_audio_format_t

Data Fields

- uint32_t mclk_HZ
  master clock frequency
- uint32_t sampleRate
  sample rate
- uint32_t bitWidth
  bit width
- bool isBclkInvert
  bit clock intervet

42.5.2.3 struct da7212_config_t

Data Fields

- bool isMaster
  If DA7212 is master, true means master, false means slave.
- da7212_protocol_t protocol
  Audio bus format, can be I2S, LJ, RJ or DSP mode.
- da7212_dac_source_t dacSource
  DA7212 data source.
- da7212_audio_format_t format
  audio format
- uint8_t slaveAddress
  device address
- codec_i2c_config_t i2cConfig
  i2c configuration
- da7212_sys_clk_source_t sysClkSource
  system clock source
- da7212_pll_config_t * pll
  pll configuration

Field Documentation

(1) bool da7212_config_t::isMaster

(2) da7212_protocol_t da7212_config_t::protocol

(3) da7212_dac_source_t da7212_config_t::dacSource

42.5.2.4 struct da7212_handle_t

Data Fields

- da7212_config_t * config
  da7212 config pointer
- uint8_t i2cHandle [DA7212_I2C_HANDLER_SIZE]
  i2c handle

42.5.3 Macro Definition Documentation

42.5.3.1 #define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))

42.5.4 Enumeration Type Documentation

42.5.4.1 enum da7212_Input_t

Enumerator

kDA7212_Input_AUX Input from AUX.
kDA7212_Input_MIC1_Dig Input from MIC1 Digital.
kDA7212_Input_MIC1_An Input from Mic1 Analog.
kDA7212_Input_MIC2 Input from MIC2.

42.5.4.2 enum _da7212_play_channel

Enumerator

kDA7212_HeadphoneLeft headphone left
kDA7212_HeadphoneRight headphone right
kDA7212_Speaker speaker channel

42.5.4.3 enum da7212_Output_t

Enumerator

kDA7212_Output_HP Output to headphone.
kDA7212_Output_SP Output to speaker.

42.5.4.4 enum _da7212_module

Enumerator

kDA7212_ModuleADC module ADC
kDA7212_ModuleDAC module DAC
kDA7212_ModuleHeadphone module headphone
kDA7212_ModuleSpeaker module speaker

42.5.4.5 enum da7212_dac_source_t

Enumerator

kDA7212_DACSourceADC DAC source from ADC.
kDA7212_DACSourceInputStream DAC source from.

42.5.4.6 enum da7212_volume_t

Enumerator

kDA7212_DACGainMute Mute DAC.
kDA7212_DACGainM72DB DAC volume -72db.
kDA7212_DACGainM60DB DAC volume -60db.
kDA7212_DACGainM54DB DAC volume -54db.
kDA7212_DACGainM48DB DAC volume -48db.
kDA7212_DACGainM42DB DAC volume -42db.
kDA7212_DACGainM36DB DAC volume -36db.
kDA7212_DACGainM30DB DAC volume -30db.
kDA7212_DACGainM24DB DAC volume -24db.
kDA7212_DACGainM18DB DAC volume -18db.
kDA7212_DACGainM12DB DAC volume -12db.
kDA7212_DACGainM6DB DAC volume -6bb.
kDA7212_DACGain0DB DAC volume +0db.
kDA7212_DACGain6DB DAC volume +6db.
kDA7212_DACGain12DB DAC volume +12db.

42.5.4.7 enum da7212_protocol_t

Enumerator

kDA7212_BusI2S I2S Type.
kDA7212_BusLeftJustified Left justified.
kDA7212_BusRightJustified Right Justified.
kDA7212_BusDSPMode DSP mode.

42.5.4.8 enum da7212_sys_clk_source_t

Enumerator

kDA7212_SysClkSourceMCLK da7212 system clock soure from MCLK
kDA7212_SysClkSourcePLL da7212 system clock soure from pLL

42.5.4.9 enum da7212_pll_clk_source_t

Enumerator

kDA7212_PLLClkSourceMCLK DA7212 PLL clock source from MCLK.

42.5.4.10 enum da7212_pll_out_clk_t

Enumerator

kDA7212_PLLOutputClk11289600 output 112896000U
kDA7212_PLLOutputClk12288000 output 12288000U

42.5.4.11 enum da7212_master_bits_t

Enumerator

kDA7212_MasterBits32PerFrame master mode bits32 per frame
kDA7212_MasterBits64PerFrame master mode bits64 per frame
kDA7212_MasterBits128PerFrame master mode bits128 per frame
kDA7212_MasterBits256PerFrame master mode bits256 per frame

42.5.5 Function Documentation

42.5.5.1 status_t DA7212_Init (da7212_handle_t * *handle*, da7212_config_t * *codecConfig*)

Parameters

handle	DA7212 handle pointer.

codecConfig	Codec configure structure. This parameter can be NULL, if NULL, set as default settings. The default setting: * sgtl_init_t codec_config * codec_config.route = kDA7212_RoutePlayback * codec_config.bus = kDA7212_BusI2S * codec_config.isMaster = false *

42.5.5.2 status_t DA7212_ConfigAudioFormat (da7212_handle_t * *handle*, uint32_t *masterClock_Hz*, uint32_t *sampleRate_Hz*, uint32_t *dataBits*)

Parameters

handle	DA7212 handle pointer.
masterClock_Hz	Master clock frequency in Hz. If DA7212 is slave, use the frequency of master, if DA7212 as master, it should be 1228000 while sample rate frequency is 8k/12K/16K/24K/32K/48K/96K, 11289600 whie sample rate is 11.025K/22.05K/44.1K
sampleRate_Hz	Sample rate frequency in Hz.
dataBits	How many bits in a word of a audio frame, DA7212 only supports 16/20/24/32 bits.

42.5.5.3 status_t DA7212_SetPLLConfig (da7212_handle_t * *handle*, da7212_pll_config_t * *config*)

Parameters

handle	DA7212 handler pointer.
config	PLL configuration pointer.

42.5.5.4 void DA7212_ChangeHPVolume (da7212_handle_t * *handle*, da7212_volume_t *volume*)

Parameters

handle	DA7212 handle pointer.
volume	The volume of playback.

42.5.5.5 void DA7212_Mute (da7212_handle_t * *handle,* bool *isMuted*)

Parameters

handle	DA7212 handle pointer.
isMuted	True means mute, false means unmute.

42.5.5.6 void DA7212_ChangeInput (da7212_handle_t * *handle,* da7212_Input_t *DA7212_Input*)

Parameters

handle	DA7212 handle pointer.
DA7212_Input	Input data source.

42.5.5.7 void DA7212_ChangeOutput (da7212_handle_t * *handle,* da7212_Output_t *DA7212_Output*)

Parameters

handle	DA7212 handle pointer.
DA7212_-Output	Output device of DA7212.

42.5.5.8 status_t DA7212_SetChannelVolume (da7212_handle_t * *handle,* uint32_t *channel,* uint32_t *volume*)

Parameters

handle	DA7212 handle pointer.
channel	shoule be a value of _da7212_channel.
volume	volume range 0 - 0x3F mapped to range -57dB - 6dB.

42.5.5.9 status_t DA7212_SetChannelMute (da7212_handle_t * *handle,* uint32_t *channel,* bool *isMute*)

Parameters

handle	DA7212 handle pointer.
channel	shoule be a value of _da7212_channel.
isMute	true is mute, false is unmute.

42.5.5.10 status_t DA7212_SetProtocol (da7212_handle_t * *handle,* da7212_protocol_t *protocol*)

Parameters

handle	DA7212 handle pointer.
protocol	da7212_protocol_t.

42.5.5.11 status_t DA7212_SetMasterModeBits (da7212_handle_t * *handle,* uint32_t *bitWidth*)

Parameters

handle	DA7212 handle pointer.
bitWidth	audio data bitwidth.

42.5.5.12 status_t DA7212_WriteRegister (da7212_handle_t * *handle,* uint8_t *u8Register,* uint8_t *u8RegisterData*)

Parameters

handle	DA7212 handle pointer.
u8Register	DA7212 register address to be written.
u8RegisterData	Data to be written into regsiter

42.5.5.13 status_t DA7212_ReadRegister (da7212_handle_t * *handle,* uint8_t *u8Register,* uint8_t * *pu8RegisterData*)

Parameters

handle	DA7212 handle pointer.
u8Register	DA7212 register address to be read.
pu8Register-Data	Pointer where the read out value to be stored.

42.5.5.14 status_t DA7212_Deinit (da7212_handle_t * *handle*)

Parameters

handle	DA7212 handle pointer.

42.5.6 DA7212 Adapter

42.5.6.1 Overview

The da7212 adapter provides a codec unify control interface.

Macros

- #define HAL_CODEC_DA7212_HANDLER_SIZE (DA7212_I2C_HANDLER_SIZE + 4)
  codec handler size

Functions

- status_t HAL_CODEC_DA7212_Init (void *handle, void *config)
  Codec initilization.
- status_t HAL_CODEC_DA7212_Deinit (void *handle)
  Codec de-initilization.
- status_t HAL_CODEC_DA7212_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_DA7212_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_DA7212_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_DA7212_SetPower (void *handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_DA7212_SetRecord (void *handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_DA7212_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_DA7212_SetPlay (void *handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_DA7212_ModuleControl (void *handle, uint32_t cmd, uint32_t data)
  codec module control.
- static status_t HAL_CODEC_Init (void *handle, void *config)
  Codec initilization.
- static status_t HAL_CODEC_Deinit (void *handle)
  Codec de-initilization.
- static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)

set audio codec module power.

- static status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- static status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- static status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- static status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

42.5.6.2 Function Documentation

42.5.6.2.1 status_t HAL_CODEC_DA7212_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

42.5.6.2.2 status_t HAL_CODEC_DA7212_Deinit (void ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.5.6.2.3 status_t HAL_CODEC_DA7212_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.4 status_t HAL_CODEC_DA7212_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.5 status_t HAL_CODEC_DA7212_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.6 status_t HAL_CODEC_DA7212_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.7 status_t HAL_CODEC_DA7212_SetRecord (void * *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.8 status_t HAL_CODEC_DA7212_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.9 status_t HAL_CODEC_DA7212_SetPlay (void * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.10 status_t HAL_CODEC_DA7212_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.11 static status_t HAL_CODEC_Init (void * *handle,* void * *config*) [inline], [static]

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

42.5.6.2.12 static status_t HAL_CODEC_Deinit (void * *handle*) [inline], [static]

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.5.6.2.13 static status_t HAL_CODEC_SetFormat (void * *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*) [inline], [static]

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.14 static status_t HAL_CODEC_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.15 static status_t HAL_CODEC_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.16 static status_t HAL_CODEC_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*) [inline], [static]

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.17 static status_t HAL_CODEC_SetRecord (void * *handle,* uint32_t *recordSource*) [inline], [static]

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.18 static status_t HAL_CODEC_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*) [inline], [static]

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.19 static status_t HAL_CODEC_SetPlay (void * *handle,* uint32_t *playSource*) [inline], [static]

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.5.6.2.20 static status_t HAL_CODEC_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*) [inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

42.6 SGTL5000 Driver

42.6.1 Overview

The sgtl5000 driver provides a codec control interface.

Data Structures

- struct sgtl_audio_format_t
  Audio format configuration. More...
- struct sgtl_config_t
  Initailize structure of sgtl5000. More...
- struct sgtl_handle_t
  SGTL codec handler. More...

Macros

- #define CHIP_ID 0x0000U
  Define the register address of sgtl5000.
- #define SGTL5000_HEADPHONE_MAX_VOLUME_VALUE 0x7FU
  SGTL5000 volume setting range.
- #define SGTL5000_I2C_ADDR 0x0A
  SGTL5000 I2C address.
- #define SGTL_I2C_HANDLER_SIZE CODEC_I2C_MASTER_HANDLER_SIZE
  sgtl handle size
- #define SGTL_I2C_BITRATE 100000U
  sgtl i2c baudrate

Enumerations

- enum sgtl_module_t {
  kSGTL_ModuleADC = 0x0,
  kSGTL_ModuleDAC,
  kSGTL_ModuleDAP,
  kSGTL_ModuleHP,
  kSGTL_ModuleI2SIN,
  kSGTL_ModuleI2SOUT,
  kSGTL_ModuleLineIn,
  kSGTL_ModuleLineOut,
  kSGTL_ModuleMicin }
  Modules in Sgtl5000 board.
- enum sgtl_route_t {

kSGTL_RouteBypass = 0x0,
kSGTL_RoutePlayback,
kSGTL_RoutePlaybackandRecord,
kSGTL_RoutePlaybackwithDAP,
kSGTL_RoutePlaybackwithDAPandRecord,
kSGTL_RouteRecord }
Sgtl5000 data route.

- enum sgtl_protocol_t {
kSGTL_BusI2S = 0x0,
kSGTL_BusLeftJustified,
kSGTL_BusRightJustified,
kSGTL_BusPCMA,
kSGTL_BusPCMB }
The audio data transfer protocol choice.
- enum {
kSGTL_HeadphoneLeft = 0,
kSGTL_HeadphoneRight = 1,
kSGTL_LineoutLeft = 2,
kSGTL_LineoutRight = 3 }
sgtl play channel
- enum {
kSGTL_RecordSourceLineIn = 0U,
kSGTL_RecordSourceMic = 1U }
sgtl record source _sgtl_record_source
- enum {
kSGTL_PlaySourceLineIn = 0U,
kSGTL_PlaySourceDAC = 1U }
sgtl play source _stgl_play_source
- enum sgtl_sclk_edge_t {
kSGTL_SclkValidEdgeRising = 0U,
kSGTL_SclkValidEdgeFailling = 1U }
SGTL SCLK valid edge.

Functions

- status_t SGTL_Init (sgtl_handle_t *handle, sgtl_config_t *config)
sgtl5000 initialize function.
- status_t SGTL_SetDataRoute (sgtl_handle_t *handle, sgtl_route_t route)
Set audio data route in sgtl5000.
- status_t SGTL_SetProtocol (sgtl_handle_t *handle, sgtl_protocol_t protocol)
Set the audio transfer protocol.
- void SGTL_SetMasterSlave (sgtl_handle_t *handle, bool master)
Set sgtl5000 as master or slave.
- status_t SGTL_SetVolume (sgtl_handle_t *handle, sgtl_module_t module, uint32_t volume)
Set the volume of different modules in sgtl5000.
- uint32_t SGTL_GetVolume (sgtl_handle_t *handle, sgtl_module_t module)
Get the volume of different modules in sgtl5000.

- status_t SGTL_SetMute (sgtl_handle_t *handle, sgtl_module_t module, bool mute)
  Mute/unmute modules in sgtl5000.
- status_t SGTL_EnableModule (sgtl_handle_t *handle, sgtl_module_t module)
  Enable expected devices.
- status_t SGTL_DisableModule (sgtl_handle_t *handle, sgtl_module_t module)
  Disable expected devices.
- status_t SGTL_Deinit (sgtl_handle_t *handle)
  Deinit the sgtl5000 codec.
- status_t SGTL_ConfigDataFormat (sgtl_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)
  Configure the data format of audio data.
- status_t SGTL_SetPlay (sgtl_handle_t *handle, uint32_t playSource)
  select SGTL codec play source.
- status_t SGTL_SetRecord (sgtl_handle_t *handle, uint32_t recordSource)
  select SGTL codec record source.
- status_t SGTL_WriteReg (sgtl_handle_t *handle, uint16_t reg, uint16_t val)
  Write register to sgtl using I2C.
- status_t SGTL_ReadReg (sgtl_handle_t *handle, uint16_t reg, uint16_t *val)
  Read register from sgtl using I2C.
- status_t SGTL_ModifyReg (sgtl_handle_t *handle, uint16_t reg, uint16_t clr_mask, uint16_t val)
  Modify some bits in the register using I2C.

Driver version

- #define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  CLOCK driver version 2.1.1.

42.6.2 Data Structure Documentation

42.6.2.1 struct sgtl_audio_format_t

Data Fields

- uint32_t mclk_HZ
  master clock
- uint32_t sampleRate
  Sample rate.
- uint32_t bitWidth
  Bit width.
- sgtl_sclk_edge_t sclkEdge
  sclk valid edge

42.6.2.2 struct sgtl_config_t

Data Fields

- sgtl_route_t route

Audio data route.
- sgtl_protocol_t bus
  Audio transfer protocol.
- bool master_slave
  Master or slave.
- sgtl_audio_format_t format
  audio format
- uint8_t slaveAddress
  code device slave address
- codec_i2c_config_t i2cConfig
  i2c bus configuration

Field Documentation

(1) sgtl_route_t sgtl_config_t::route

(2) bool sgtl_config_t::master_slave

True means master, false means slave.

42.6.2.3 struct sgtl_handle_t

Data Fields

- sgtl_config_t * config
  sgtl config pointer
- uint8_t i2cHandle [SGTL_I2C_HANDLER_SIZE]
  i2c handle

42.6.3 Macro Definition Documentation

42.6.3.1 #define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

42.6.3.2 #define CHIP_ID 0x0000U

42.6.3.3 #define SGTL5000_I2C_ADDR 0x0A

42.6.4 Enumeration Type Documentation

42.6.4.1 enum sgtl_module_t

Enumerator

kSGTL_ModuleADC ADC module in SGTL5000.
kSGTL_ModuleDAC DAC module in SGTL5000.
kSGTL_ModuleDAP DAP module in SGTL5000.
kSGTL_ModuleHP Headphone module in SGTL5000.

kSGTL_ModuleI2SIN I2S-IN module in SGTL5000.
kSGTL_ModuleI2SOUT I2S-OUT module in SGTL5000.
kSGTL_ModuleLineIn Line-in moudle in SGTL5000.
kSGTL_ModuleLineOut Line-out module in SGTL5000.
kSGTL_ModuleMicin Micphone module in SGTL5000.

42.6.4.2 enum sgtl_route_t

Note

Only provide some typical data route, not all route listed. Users cannot combine any routes, once a new route is set, the precios one would be replaced.

Enumerator

kSGTL_RouteBypass LINEIN->Headphone.
kSGTL_RoutePlayback I2SIN->DAC->Headphone.
kSGTL_RoutePlaybackandRecord I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.
kSGTL_RoutePlaybackwithDAP I2SIN->DAP->DAC->Headphone.
kSGTL_RoutePlaybackwithDAPandRecord I2SIN->DAP->DAC->HP, LINEIN->ADC->I2SOUT.
kSGTL_RouteRecord LINEIN->ADC->I2SOUT.

42.6.4.3 enum sgtl_protocol_t

Sgtl5000 only supports I2S format and PCM format.

Enumerator

kSGTL_BusI2S I2S Type.
kSGTL_BusLeftJustified Left justified.
kSGTL_BusRightJustified Right Justified.
kSGTL_BusPCMA PCMA.
kSGTL_BusPCMB PCMB.

42.6.4.4 anonymous enum

Enumerator

kSGTL_HeadphoneLeft headphone left channel
kSGTL_HeadphoneRight headphone right channel
kSGTL_LineoutLeft lineout left channel
kSGTL_LineoutRight lineout right channel

42.6.4.5 anonymous enum

Enumerator

kSGTL_RecordSourceLineIn record source line in
kSGTL_RecordSourceMic record source single end

42.6.4.6 anonymous enum

Enumerator

kSGTL_PlaySourceLineIn play source line in
kSGTL_PlaySourceDAC play source line in

42.6.4.7 enum sgtl_sclk_edge_t

Enumerator

kSGTL_SclkValidEdgeRising SCLK valid edge.
kSGTL_SclkValidEdgeFailling SCLK failling edge.

42.6.5 Function Documentation

42.6.5.1 status_t SGTL_Init (sgtl_handle_t * *handle*, sgtl_config_t * *config*)

This function calls SGTL_I2CInit(), and in this function, some configurations are fixed. The second parameter can be NULL. If users want to change the SGTL5000 settings, a configure structure should be prepared.

Note

If the codec_config is NULL, it would initialize sgtl5000 using default settings. The default setting:

```
* sgtl_init_t codec_config
* codec_config.route = kSGTL_RoutePlaybackandRecord
* codec_config.bus = kSGTL_BusI2S
* codec_config.master = slave
*
```

Parameters

handle	Sgtl5000 handle structure.
config	sgtl5000 configuration structure. If this pointer equals to NULL, it means using the default configuration.

Returns

Initialization status

42.6.5.2 status_t SGTL_SetDataRoute (sgtl_handle_t * *handle*, sgtl_route_t *route*)

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules.

Note

If a new route is set, the previous route would not work.

Parameters

handle	Sgtl5000 handle structure.
route	Audio data route in sgtl5000.

42.6.5.3 status_t SGTL_SetProtocol (sgtl_handle_t * *handle*, sgtl_protocol_t *protocol*)

Sgtl5000 only supports I2S, I2S left, I2S right, PCM A, PCM B format.

Parameters

handle	Sgtl5000 handle structure.
protocol	Audio data transfer protocol.

42.6.5.4 void SGTL_SetMasterSlave (sgtl_handle_t * *handle*, bool *master*)

Parameters

handle	Sgtl5000 handle structure.
master	1 represent master, 0 represent slave.

42.6.5.5 status_t SGTL_SetVolume (sgtl_handle_t * *handle*, sgtl_module_t *module*, uint32_t *volume*)

This function would set the volume of sgtl5000 modules. This interface set module volume. The function assume that left channel and right channel has the same volume.

kSGTL_ModuleADC volume range: 0 - 0xF, 0dB - 22.5dB kSGTL_ModuleDAC volume range: 0x3C - 0xF0, 0dB - -90dB kSGTL_ModuleHP volume range: 0 - 0x7F, 12dB - -51.5dB kSGTL_ModuleLineOut volume range: 0 - 0x1F, 0.5dB steps

Parameters

handle	Sgtl5000 handle structure.
module	Sgtl5000 module, such as DAC, ADC and etc.
volume	Volume value need to be set. The value is the exact value in register.

42.6.5.6 uint32_t SGTL_GetVolume (sgtl_handle_t * *handle*, sgtl_module_t *module*)

This function gets the volume of sgtl5000 modules. This interface get DAC module volume. The function assume that left channel and right channel has the same volume.

Parameters

handle	Sgtl5000 handle structure.
module	Sgtl5000 module, such as DAC, ADC and etc.

Returns

Module value, the value is exact value in register.

42.6.5.7 status_t SGTL_SetMute (sgtl_handle_t * *handle*, sgtl_module_t *module*, bool *mute*)

Parameters

handle	Sgtl5000 handle structure.
module	Sgtl5000 module, such as DAC, ADC and etc.
mute	True means mute, and false means unmute.

42.6.5.8 status_t SGTL_EnableModule (sgtl_handle_t * *handle,* sgtl_module_t *module*)

Parameters

handle	Sgtl5000 handle structure.
module	Module expected to enable.

42.6.5.9 status_t SGTL_DisableModule (sgtl_handle_t * *handle,* sgtl_module_t *module*)

Parameters

handle	Sgtl5000 handle structure.
module	Module expected to enable.

42.6.5.10 status_t SGTL_Deinit (sgtl_handle_t * *handle*)

Shut down Sgtl5000 modules.

Parameters

handle	Sgtl5000 handle structure pointer.

42.6.5.11 status_t SGTL_ConfigDataFormat (sgtl_handle_t * *handle,* uint32_t *mclk,* uint32_t *sample_rate,* uint32_t *bits*)

This function would configure the registers about the sample rate, bit depths.

Parameters

handle	Sgtl5000 handle structure pointer.
mclk	Master clock frequency of I2S.
sample_rate	Sample rate of audio file running in sgtl5000. Sgtl5000 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
bits	Bit depth of audio file (Sgtl5000 only supports 16bit, 20bit, 24bit and 32 bit in HW).

42.6.5.12 status_t SGTL_SetPlay (sgtl_handle_t * *handle,* uint32_t *playSource*)

Parameters

handle	Sgtl5000 handle structure pointer.
playSource	play source value, reference _sgtl_play_source.

Returns

kStatus_Success, else failed.

42.6.5.13 status_t SGTL_SetRecord (sgtl_handle_t * *handle,* uint32_t *recordSource*)

Parameters

handle	Sgtl5000 handle structure pointer.
recordSource	record source value, reference _sgtl_record_source.

Returns

kStatus_Success, else failed.

42.6.5.14 status_t SGTL_WriteReg (sgtl_handle_t * *handle,* uint16_t *reg,* uint16_t *val*)

Parameters

handle	Sgtl5000 handle structure.

reg	The register address in sgtl.
val	Value needs to write into the register.

42.6.5.15 status_t SGTL_ReadReg (sgtl_handle_t * *handle*, uint16_t *reg*, uint16_t * *val*)

Parameters

handle	Sgtl5000 handle structure.
reg	The register address in sgtl.
val	Value written to.

42.6.5.16 status_t SGTL_ModifyReg (sgtl_handle_t * *handle*, uint16_t *reg*, uint16_t *clr_mask*, uint16_t *val*)

Parameters

handle	Sgtl5000 handle structure.
reg	The register address in sgtl.
clr_mask	The mask code for the bits want to write. The bit you want to write should be 0.
val	Value needs to write into the register.

42.6.6 SGTL5000 Adapter

42.6.6.1 Overview

The sgtl5000 adapter provides a codec unify control interface.

Macros

- #define HAL_CODEC_SGTL_HANDLER_SIZE (SGTL_I2C_HANDLER_SIZE + 4)
  codec handler size

Functions

- status_t HAL_CODEC_SGTL5000_Init (void *handle, void *config)
  Codec initilization.
- status_t HAL_CODEC_SGTL5000_Deinit (void *handle)
  Codec de-initilization.
- status_t HAL_CODEC_SGTL5000_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_SGTL5000_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_SGTL5000_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_SGTL5000_SetPower (void *handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_SGTL5000_SetRecord (void *handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_SGTL5000_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_SGTL5000_SetPlay (void *handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_SGTL5000_ModuleControl (void *handle, uint32_t cmd, uint32_t data)
  codec module control.
- static status_t HAL_CODEC_Init (void *handle, void *config)
  Codec initilization.
- static status_t HAL_CODEC_Deinit (void *handle)
  Codec de-initilization.
- static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)

set audio codec module power.

- static status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- static status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- static status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- static status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

42.6.6.2 Function Documentation

42.6.6.2.1 status_t HAL_CODEC_SGTL5000_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

42.6.6.2.2 status_t HAL_CODEC_SGTL5000_Deinit (void ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.6.6.2.3 status_t HAL_CODEC_SGTL5000_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.4 status_t HAL_CODEC_SGTL5000_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.5 status_t HAL_CODEC_SGTL5000_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.6 status_t HAL_CODEC_SGTL5000_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.7 status_t HAL_CODEC_SGTL5000_SetRecord (void * *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.8 status_t HAL_CODEC_SGTL5000_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.9 status_t HAL_CODEC_SGTL5000_SetPlay (void * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.10 status_t HAL_CODEC_SGTL5000_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.11 static status_t HAL_CODEC_Init (void * *handle*, void * *config*) [inline], [static]

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

42.6.6.2.12 static status_t HAL_CODEC_Deinit (void * *handle*) [inline], [static]

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.6.6.2.13 static status_t HAL_CODEC_SetFormat (void * *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*) [inline], [static]

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.14 static status_t HAL_CODEC_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.15 static status_t HAL_CODEC_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.16 static status_t HAL_CODEC_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*) [inline], [static]

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.17 static status_t HAL_CODEC_SetRecord (void * *handle,* uint32_t *recordSource*) [inline], [static]

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.18 static status_t HAL_CODEC_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*) [inline], [static]

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.19 static status_t HAL_CODEC_SetPlay (void * *handle,* uint32_t *playSource*) [inline], [static]

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.6.6.2.20 static status_t HAL_CODEC_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*) [inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

42.7 WM8960 Driver

42.7.1 Overview

The wm8960 driver provides a codec control interface.

Data Structures

- struct wm8960_audio_format_t
  wm8960 audio format More...
- struct wm8960_master_sysclk_config_t
  wm8960 master system clock configuration More...
- struct wm8960_config_t
  Initialize structure of WM8960. More...
- struct wm8960_handle_t
  wm8960 codec handler More...

Macros

- #define WM8960_I2C_HANDLER_SIZE CODEC_I2C_MASTER_HANDLER_SIZE
  wm8960 handle size
- #define WM8960_LINVOL 0x0U
  Define the register address of WM8960.
- #define WM8960_CACHEREGNUM 56U
  Cache register number.
- #define WM8960_CLOCK2_BCLK_DIV_MASK 0xFU
  WM8960 CLOCK2 bits.
- #define WM8960_IFACE1_FORMAT_MASK 0x03U
  WM8960_IFACE1 FORMAT bits.
- #define WM8960_IFACE1_WL_MASK 0x0CU
  WM8960_IFACE1 WL bits.
- #define WM8960_IFACE1_LRP_MASK 0x10U
  WM8960_IFACE1 LRP bit.
- #define WM8960_IFACE1_DLRSWAP_MASK 0x20U
  WM8960_IFACE1 DLRSWAP bit.
- #define WM8960_IFACE1_MS_MASK 0x40U
  WM8960_IFACE1 MS bit.
- #define WM8960_IFACE1_BCLKINV_MASK 0x80U
  WM8960_IFACE1 BCLKINV bit.
- #define WM8960_IFACE1_ALRSWAP_MASK 0x100U
  WM8960_IFACE1 ALRSWAP bit.
- #define WM8960_POWER1_VREF_MASK 0x40U
  WM8960_POWER1.
- #define WM8960_POWER2_DACL_MASK 0x100U
  WM8960_POWER2.
- #define WM8960_I2C_ADDR 0x1A
  WM8960 I2C address.
- #define WM8960_I2C_BAUDRATE (100000U)

WM8960 I2C baudrate.

- #define WM8960_ADC_MAX_VOLUME_vALUE 0xFFU
  WM8960 maximum volume value.

Enumerations

- enum wm8960_module_t {
  kWM8960_ModuleADC = 0,
  kWM8960_ModuleDAC = 1,
  kWM8960_ModuleVREF = 2,
  kWM8960_ModuleHP = 3,
  kWM8960_ModuleMICB = 4,
  kWM8960_ModuleMIC = 5,
  kWM8960_ModuleLineIn = 6,
  kWM8960_ModuleLineOut = 7,
  kWM8960_ModuleSpeaker = 8,
  kWM8960_ModuleOMIX = 9 }
  Modules in WM8960 board.
- enum {
  kWM8960_HeadphoneLeft = 1,
  kWM8960_HeadphoneRight = 2,
  kWM8960_SpeakerLeft = 4,
  kWM8960_SpeakerRight = 8 }
  wm8960 play channel
- enum wm8960_play_source_t {
  kWM8960_PlaySourcePGA = 1,
  kWM8960_PlaySourceInput = 2,
  kWM8960_PlaySourceDAC = 4 }
  wm8960 play source
- enum wm8960_route_t {
  kWM8960_RouteBypass = 0,
  kWM8960_RoutePlayback = 1,
  kWM8960_RoutePlaybackandRecord = 2,
  kWM8960_RouteRecord = 5 }
  WM8960 data route.
- enum wm8960_protocol_t {
  kWM8960_BusI2S = 2,
  kWM8960_BusLeftJustified = 1,
  kWM8960_BusRightJustified = 0,
  kWM8960_BusPCMA = 3,
  kWM8960_BusPCMB = 3 | (1 << 4) }
  The audio data transfer protocol choice.
- enum wm8960_input_t {

kWM8960_InputClosed = 0,
kWM8960_InputSingleEndedMic = 1,
kWM8960_InputDifferentialMicInput2 = 2,
kWM8960_InputDifferentialMicInput3 = 3,
kWM8960_InputLineINPUT2 = 4,
kWM8960_InputLineINPUT3 = 5 }
wm8960 input source

- enum {
kWM8960_AudioSampleRate8KHz = 8000U,
kWM8960_AudioSampleRate11025Hz = 11025U,
kWM8960_AudioSampleRate12KHz = 12000U,
kWM8960_AudioSampleRate16KHz = 16000U,
kWM8960_AudioSampleRate22050Hz = 22050U,
kWM8960_AudioSampleRate24KHz = 24000U,
kWM8960_AudioSampleRate32KHz = 32000U,
kWM8960_AudioSampleRate44100Hz = 44100U,
kWM8960_AudioSampleRate48KHz = 48000U,
kWM8960_AudioSampleRate96KHz = 96000U,
kWM8960_AudioSampleRate192KHz = 192000U,
kWM8960_AudioSampleRate384KHz = 384000U }
audio sample rate definition
- enum {
kWM8960_AudioBitWidth16bit = 16U,
kWM8960_AudioBitWidth20bit = 20U,
kWM8960_AudioBitWidth24bit = 24U,
kWM8960_AudioBitWidth32bit = 32U }
audio bit width
- enum wm8960_sysclk_source_t {
kWM8960_SysClkSourceMclk = 0U,
kWM8960_SysClkSourceInternalPLL = 1U }
wm8960 sysclk source

Functions

- status_t WM8960_Init (wm8960_handle_t *handle, const wm8960_config_t *config)
WM8960 initialize function.
- status_t WM8960_Deinit (wm8960_handle_t *handle)
Deinit the WM8960 codec.
- status_t WM8960_SetDataRoute (wm8960_handle_t *handle, wm8960_route_t route)
Set audio data route in WM8960.
- status_t WM8960_SetLeftInput (wm8960_handle_t *handle, wm8960_input_t input)
Set left audio input source in WM8960.
- status_t WM8960_SetRightInput (wm8960_handle_t *handle, wm8960_input_t input)
Set right audio input source in WM8960.
- status_t WM8960_SetProtocol (wm8960_handle_t *handle, wm8960_protocol_t protocol)
Set the audio transfer protocol.

- void WM8960_SetMasterSlave (wm8960_handle_t *handle, bool master)
  Set WM8960 as master or slave.
- status_t WM8960_SetVolume (wm8960_handle_t *handle, wm8960_module_t module, uint32_t volume)
  Set the volume of different modules in WM8960.
- uint32_t WM8960_GetVolume (wm8960_handle_t *handle, wm8960_module_t module)
  Get the volume of different modules in WM8960.
- status_t WM8960_SetMute (wm8960_handle_t *handle, wm8960_module_t module, bool isEnabled)
  Mute modules in WM8960.
- status_t WM8960_SetModule (wm8960_handle_t *handle, wm8960_module_t module, bool isEnabled)
  Enable/disable expected devices.
- status_t WM8960_SetPlay (wm8960_handle_t *handle, uint32_t playSource)
  SET the WM8960 play source.
- status_t WM8960_ConfigDataFormat (wm8960_handle_t *handle, uint32_t sysclk, uint32_t sample_rate, uint32_t bits)
  Configure the data format of audio data.
- status_t WM8960_SetJackDetect (wm8960_handle_t *handle, bool isEnabled)
  Enable/disable jack detect feature.
- status_t WM8960_WriteReg (wm8960_handle_t *handle, uint8_t reg, uint16_t val)
  Write register to WM8960 using I2C.
- status_t WM8960_ReadReg (uint8_t reg, uint16_t *val)
  Read register from WM8960 using I2C.
- status_t WM8960_ModifyReg (wm8960_handle_t *handle, uint8_t reg, uint16_t mask, uint16_t val)
  Modify some bits in the register using I2C.

Driver version

- #define FSL_WM8960_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  CLOCK driver version 2.2.0.

42.7.2 Data Structure Documentation

42.7.2.1 struct wm8960_audio_format_t

Data Fields

- uint32_t mclk_HZ
  master clock frequency
- uint32_t sampleRate
  sample rate
- uint32_t bitWidth
  bit width

42.7.2.2 struct wm8960_master_sysclk_config_t

Data Fields

- wm8960_sysclk_source_t sysclkSource
  sysclk source
- uint32_t sysclkFreq
  PLL output frequency value.

42.7.2.3 struct wm8960_config_t

Data Fields

- wm8960_route_t route
  Audio data route.
- wm8960_protocol_t bus
  Audio transfer protocol.
- wm8960_audio_format_t format
  Audio format.
- bool master_slave
  Master or slave.
- wm8960_master_sysclk_config_t masterClock
  master clock configurations
- bool enableSpeaker
  True means enable class D speaker as output, false means no.
- wm8960_input_t leftInputSource
  Left input source for WM8960.
- wm8960_input_t rightInputSource
  Right input source for wm8960.
- wm8960_play_source_t playSource
  play source
- uint8_t slaveAddress
  wm8960 device address
- codec_i2c_config_t i2cConfig
  i2c configuration

Field Documentation

(1) wm8960_route_t wm8960_config_t::route

(2) bool wm8960_config_t::master_slave

42.7.2.4 struct wm8960_handle_t

Data Fields

- const wm8960_config_t * config
  wm8904 config pointer
- uint8_t i2cHandle [WM8960_I2C_HANDLER_SIZE]
  i2c handle

42.7.3 Macro Definition Documentation

42.7.3.1 #define WM8960_LINVOL 0x0U

42.7.3.2 #define WM8960_I2C_ADDR 0x1A

42.7.4 Enumeration Type Documentation

42.7.4.1 enum wm8960_module_t

Enumerator

kWM8960_ModuleADC ADC module in WM8960.
kWM8960_ModuleDAC DAC module in WM8960.
kWM8960_ModuleVREF VREF module.
kWM8960_ModuleHP Headphone.
kWM8960_ModuleMICB Mic bias.
kWM8960_ModuleMIC Input Mic.
kWM8960_ModuleLineIn Analog in PGA.
kWM8960_ModuleLineOut Line out module.
kWM8960_ModuleSpeaker Speaker module.
kWM8960_ModuleOMIX Output mixer.

42.7.4.2 anonymous enum

Enumerator

kWM8960_HeadphoneLeft wm8960 headphone left channel
kWM8960_HeadphoneRight wm8960 headphone right channel
kWM8960_SpeakerLeft wm8960 speaker left channel
kWM8960_SpeakerRight wm8960 speaker right channel

42.7.4.3 enum wm8960_play_source_t

Enumerator

kWM8960_PlaySourcePGA wm8960 play source PGA
kWM8960_PlaySourceInput wm8960 play source Input
kWM8960_PlaySourceDAC wm8960 play source DAC

42.7.4.4 enum wm8960_route_t

Only provide some typical data route, not all route listed. Note: Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

kWM8960_RouteBypass LINEIN->Headphone.
kWM8960_RoutePlayback I2SIN->DAC->Headphone.
kWM8960_RoutePlaybackandRecord I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.
kWM8960_RouteRecord LINEIN->ADC->I2SOUT.

42.7.4.5 enum wm8960_protocol_t

WM8960 only supports I2S format and PCM format.

Enumerator

kWM8960_BusI2S I2S type.
kWM8960_BusLeftJustified Left justified mode.
kWM8960_BusRightJustified Right justified mode.
kWM8960_BusPCMA PCM A mode.
kWM8960_BusPCMB PCM B mode.

42.7.4.6 enum wm8960_input_t

Enumerator

kWM8960_InputClosed Input device is closed.
kWM8960_InputSingleEndedMic Input as single ended mic, only use L/RINPUT1.
kWM8960_InputDifferentialMicInput2 Input as differential mic, use L/RINPUT1 and L/RINPUT2.
kWM8960_InputDifferentialMicInput3 Input as differential mic, use L/RINPUT1 and L/RINPUT3.
kWM8960_InputLineINPUT2 Input as line input, only use L/RINPUT2.
kWM8960_InputLineINPUT3 Input as line input, only use L/RINPUT3.

42.7.4.7 anonymous enum

Enumerator

kWM8960_AudioSampleRate8KHz Sample rate 8000 Hz.
kWM8960_AudioSampleRate11025Hz Sample rate 11025 Hz.
kWM8960_AudioSampleRate12KHz Sample rate 12000 Hz.
kWM8960_AudioSampleRate16KHz Sample rate 16000 Hz.
kWM8960_AudioSampleRate22050Hz Sample rate 22050 Hz.
kWM8960_AudioSampleRate24KHz Sample rate 24000 Hz.
kWM8960_AudioSampleRate32KHz Sample rate 32000 Hz.
kWM8960_AudioSampleRate44100Hz Sample rate 44100 Hz.
kWM8960_AudioSampleRate48KHz Sample rate 48000 Hz.

kWM8960_AudioSampleRate96KHz Sample rate 96000 Hz.
kWM8960_AudioSampleRate192KHz Sample rate 192000 Hz.
kWM8960_AudioSampleRate384KHz Sample rate 384000 Hz.

42.7.4.8 anonymous enum

Enumerator

kWM8960_AudioBitWidth16bit audio bit width 16
kWM8960_AudioBitWidth20bit audio bit width 20
kWM8960_AudioBitWidth24bit audio bit width 24
kWM8960_AudioBitWidth32bit audio bit width 32

42.7.4.9 enum wm8960_sysclk_source_t

Enumerator

kWM8960_SysClkSourceMclk sysclk source from external MCLK
kWM8960_SysClkSourceInternalPLL sysclk source from internal PLL

42.7.5 Function Documentation

42.7.5.1 status_t WM8960_Init (wm8960_handle_t * *handle*, const wm8960_config_t * *config*)

The second parameter is NULL to WM8960 in this version. If users want to change the settings, they have to use wm8960_write_reg() or wm8960_modify_reg() to set the register value of WM8960. Note: : If the codec_config is NULL, it would initialize WM8960 using default settings. The default setting: codec_config->route = kWM8960_RoutePlaybackandRecord codec_config->bus = kWM8960_BusI2S codec_config->master = slave

Parameters

handle	WM8960 handle structure.
config	WM8960 configuration structure.

42.7.5.2 status_t WM8960_Deinit (wm8960_handle_t * *handle*)

This function close all modules in WM8960 to save power.

Parameters

handle	WM8960 handle structure pointer.

42.7.5.3 status_t WM8960_SetDataRoute (wm8960_handle_t * *handle*, wm8960_route_t *route*)

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules. Note: If a new route is set, the previous route would not work.

Parameters

handle	WM8960 handle structure.
route	Audio data route in WM8960.

42.7.5.4 status_t WM8960_SetLeftInput (wm8960_handle_t * *handle*, wm8960_input_t *input*)

Parameters

handle	WM8960 handle structure.
input	Audio input source.

42.7.5.5 status_t WM8960_SetRightInput (wm8960_handle_t * *handle*, wm8960_input_t *input*)

Parameters

handle	WM8960 handle structure.
input	Audio input source.

42.7.5.6 status_t WM8960_SetProtocol (wm8960_handle_t * *handle*, wm8960_protocol_t *protocol*)

WM8960 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

handle	WM8960 handle structure.
protocol	Audio data transfer protocol.

42.7.5.7 void WM8960_SetMasterSlave (wm8960_handle_t * *handle*, bool *master*)

Parameters

handle	WM8960 handle structure.
master	1 represent master, 0 represent slave.

42.7.5.8 status_t WM8960_SetVolume (wm8960_handle_t * *handle*, wm8960_module_t *module*, uint32_t *volume*)

This function would set the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Module:kWM8960_ModuleADC, volume range value: 0 is mute, 1-255 is -97db to 30db Module:kWM8960_ModuleDAC, volume range value: 0 is mute, 1-255 is -127db to 0db Module:kWM8960_ModuleHP, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db Module:kWM8960_ModuleLineIn, volume range value: 0 - 0x3F is -17.25db to 30db Module:kWM8960_ModuleSpeaker, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db

Parameters

handle	WM8960 handle structure.
module	Module to set volume, it can be ADC, DAC, Headphone and so on.
volume	Volume value need to be set.

42.7.5.9 uint32_t WM8960_GetVolume (wm8960_handle_t * *handle*, wm8960_module_t *module*)

This function gets the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	WM8960 handle structure.
module	Module to set volume, it can be ADC, DAC, Headphone and so on.

Returns

Volume value of the module.

42.7.5.10 status_t WM8960_SetMute (wm8960_handle_t * *handle*, wm8960_module_t *module*, bool *isEnabled*)

Parameters

handle	WM8960 handle structure.
module	Modules need to be mute.
isEnabled	Mute or unmute, 1 represent mute.

42.7.5.11 status_t WM8960_SetModule (wm8960_handle_t * *handle*, wm8960_module_t *module*, bool *isEnabled*)

Parameters

handle	WM8960 handle structure.
module	Module expected to enable.
isEnabled	Enable or disable moudles.

42.7.5.12 status_t WM8960_SetPlay (wm8960_handle_t * *handle*, uint32_t *playSource*)

Parameters

handle	WM8960 handle structure.
playSource	play source , can be a value combine of kWM8960_ModuleHeadphoneSourcePGA, kWM8960_ModuleHeadphoneSourceDAC, kWM8960_ModulePlaySourceInput, kWM8960_ModulePlayMonoRight, kWM8960_ModulePlayMonoLeft.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

42.7.5.13 status_t WM8960_ConfigDataFormat (wm8960_handle_t * *handle,* uint32_t *sysclk,* uint32_t *sample_rate,* uint32_t *bits*)

This function would configure the registers about the sample rate, bit depths.

Parameters

handle	WM8960 handle structure pointer.
sysclk	system clock of the codec which can be generated by MCLK or PLL output.
sample_rate	Sample rate of audio file running in WM8960. WM8960 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
bits	Bit depth of audio file (WM8960 only supports 16bit, 20bit, 24bit and 32 bit in HW).

42.7.5.14 status_t WM8960_SetJackDetect (wm8960_handle_t * *handle*, bool *isEnabled*)

Parameters

handle	WM8960 handle structure.
isEnabled	Enable or disable moudles.

42.7.5.15 status_t WM8960_WriteReg (wm8960_handle_t * *handle*, uint8_t *reg*, uint16_t *val*)

Parameters

handle	WM8960 handle structure.
reg	The register address in WM8960.
val	Value needs to write into the register.

42.7.5.16 status_t WM8960_ReadReg (uint8_t *reg*, uint16_t * *val*)

Parameters

reg	The register address in WM8960.
val	Value written to.

42.7.5.17 status_t WM8960_ModifyReg (wm8960_handle_t * *handle*, uint8_t *reg*, uint16_t *mask*, uint16_t *val*)

Parameters

handle	WM8960 handle structure.
reg	The register address in WM8960.
mask	The mask code for the bits want to write. The bit you want to write should be 0.
val	Value needs to write into the register.

42.7.6 WM8960 Adapter

42.7.6.1 Overview

The wm8960 adapter provides a codec unify control interface.

Macros

- #define HAL_CODEC_WM8960_HANDLER_SIZE (WM8960_I2C_HANDLER_SIZE + 4)
  codec handler size

Functions

- status_t HAL_CODEC_WM8960_Init (void *handle, void *config)
  Codec initilization.
- status_t HAL_CODEC_WM8960_Deinit (void *handle)
  Codec de-initilization.
- status_t HAL_CODEC_WM8960_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_WM8960_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_WM8960_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_WM8960_SetPower (void *handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_WM8960_SetRecord (void *handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_WM8960_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_WM8960_SetPlay (void *handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_WM8960_ModuleControl (void *handle, uint32_t cmd, uint32_t data)
  codec module control.
- static status_t HAL_CODEC_Init (void *handle, void *config)
  Codec initilization.
- static status_t HAL_CODEC_Deinit (void *handle)
  Codec de-initilization.
- static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)

set audio codec module power.

- static status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- static status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- static status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- static status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

42.7.6.2 Function Documentation

42.7.6.2.1 status_t HAL_CODEC_WM8960_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

42.7.6.2.2 status_t HAL_CODEC_WM8960_Deinit (void ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.7.6.2.3 status_t HAL_CODEC_WM8960_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.4 status_t HAL_CODEC_WM8960_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.5 status_t HAL_CODEC_WM8960_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.6 status_t HAL_CODEC_WM8960_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.7 status_t HAL_CODEC_WM8960_SetRecord (void * *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.8 status_t HAL_CODEC_WM8960_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.9 status_t HAL_CODEC_WM8960_SetPlay (void * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.10 status_t HAL_CODEC_WM8960_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.11 static status_t HAL_CODEC_Init (void * *handle,* void * *config*) [inline], [static]

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

42.7.6.2.12 static status_t HAL_CODEC_Deinit (void * *handle*) [inline], [static]

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.7.6.2.13 static status_t HAL_CODEC_SetFormat (void * *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*) [inline], [static]

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.14 static status_t HAL_CODEC_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.15 static status_t HAL_CODEC_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.16 static status_t HAL_CODEC_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*) [inline], [static]

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.17 static status_t HAL_CODEC_SetRecord (void * *handle,* uint32_t *recordSource*) [inline], [static]

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.18 static status_t HAL_CODEC_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*) [inline], [static]

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.19 static status_t HAL_CODEC_SetPlay (void * *handle,* uint32_t *playSource*) [inline], [static]

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.7.6.2.20 static status_t HAL_CODEC_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*) [inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

42.8 WM8904 Driver

42.8.1 Overview

The wm8904 driver provides a codec control interface.

Data Structures

- struct wm8904_fll_config_t
  wm8904 fll configuration More...
- struct wm8904_audio_format_t
  Audio format configuration. More...
- struct wm8904_config_t
  Configuration structure of WM8904. More...
- struct wm8904_handle_t
  wm8904 codec handler More...

Macros

- #define WM8904_I2C_HANDLER_SIZE (CODEC_I2C_MASTER_HANDLER_SIZE)
  wm8904 handle size
- #define WM8904_DEBUG_REGISTER 0
  wm8904 debug macro
- #define WM8904_RESET (0x00)
  WM8904 register map.
- #define WM8904_I2C_ADDRESS (0x1A)
  WM8904 I2C address.
- #define WM8904_I2C_BITRATE (400000U)
  WM8904 I2C bit rate.
- #define WM8904_MAP_HEADPHONE_LINEOUT_MAX_VOLUME 0x3FU
  WM8904 maximum volume.

Enumerations

- enum {
  kStatus_WM8904_Success = 0x0,
  kStatus_WM8904_Fail = 0x1 }
  WM8904 status return codes.
- enum {
  kWM8904_LRCPolarityNormal = 0U,
  kWM8904_LRCPolarityInverted = 1U << 4U }
  WM8904 lrc polarity.
- enum wm8904_module_t {

kWM8904_ModuleADC = 0,
kWM8904_ModuleDAC = 1,
kWM8904_ModulePGA = 2,
kWM8904_ModuleHeadphone = 3,
kWM8904_ModuleLineout = 4 }
wm8904 module value

- enum
  wm8904 play channel
- enum wm8904_timeslot_t {
  kWM8904_TimeSlot0 = 0U,
  kWM8904_TimeSlot1 = 1U }
  WM8904 time slot.
- enum wm8904_protocol_t {
  kWM8904_ProtocolI2S = 0x2,
  kWM8904_ProtocolLeftJustified = 0x1,
  kWM8904_ProtocolRightJustified = 0x0,
  kWM8904_ProtocolPCMA = 0x3,
  kWM8904_ProtocolPCMB = 0x3 | (1 << 4) }
  The audio data transfer protocol.
- enum wm8904_fs_ratio_t {
  kWM8904_FsRatio64X = 0x0,
  kWM8904_FsRatio128X = 0x1,
  kWM8904_FsRatio192X = 0x2,
  kWM8904_FsRatio256X = 0x3,
  kWM8904_FsRatio384X = 0x4,
  kWM8904_FsRatio512X = 0x5,
  kWM8904_FsRatio768X = 0x6,
  kWM8904_FsRatio1024X = 0x7,
  kWM8904_FsRatio1408X = 0x8,
  kWM8904_FsRatio1536X = 0x9 }
  The SYSCLK / fs ratio.
- enum wm8904_sample_rate_t {
  kWM8904_SampleRate8kHz = 0x0,
  kWM8904_SampleRate12kHz = 0x1,
  kWM8904_SampleRate16kHz = 0x2,
  kWM8904_SampleRate24kHz = 0x3,
  kWM8904_SampleRate32kHz = 0x4,
  kWM8904_SampleRate48kHz = 0x5,
  kWM8904_SampleRate11025Hz = 0x6,
  kWM8904_SampleRate22050Hz = 0x7,
  kWM8904_SampleRate44100Hz = 0x8 }
  Sample rate.
- enum wm8904_bit_width_t {
  kWM8904_BitWidth16 = 0x0,
  kWM8904_BitWidth20 = 0x1,
  kWM8904_BitWidth24 = 0x2,

kWM8904_BitWidth32 = 0x3 }
Bit width.
- enum {
kWM8904_RecordSourceDifferentialLine = 1U,
kWM8904_RecordSourceLineInput = 2U,
kWM8904_RecordSourceDifferentialMic = 4U,
kWM8904_RecordSourceDigitalMic = 8U }
wm8904 record source
- enum {
kWM8904_RecordChannelLeft1 = 1U,
kWM8904_RecordChannelLeft2 = 2U,
kWM8904_RecordChannelLeft3 = 4U,
kWM8904_RecordChannelRight1 = 1U,
kWM8904_RecordChannelRight2 = 2U,
kWM8904_RecordChannelRight3 = 4U,
kWM8904_RecordChannelDifferentialPositive1 = 1U,
kWM8904_RecordChannelDifferentialPositive2 = 2U,
kWM8904_RecordChannelDifferentialPositive3 = 4U,
kWM8904_RecordChannelDifferentialNegative1 = 8U,
kWM8904_RecordChannelDifferentialNegative2 = 16U,
kWM8904_RecordChannelDifferentialNegative3 = 32U }
wm8904 record channel
- enum {
kWM8904_PlaySourcePGA = 1U,
kWM8904_PlaySourceDAC = 4U }
wm8904 play source
- enum wm8904_sys_clk_source_t {
kWM8904_SysClkSourceMCLK = 0U,
kWM8904_SysClkSourceFLL = 1U << 14 }
wm8904 system clock source
- enum wm8904_fll_clk_source_t { kWM8904_FLLClkSourceMCLK = 0U }
wm8904 fll clock source

Functions

- status_t WM8904_WriteRegister (wm8904_handle_t *handle, uint8_t reg, uint16_t value)
WM8904 write register.
- status_t WM8904_ReadRegister (wm8904_handle_t *handle, uint8_t reg, uint16_t *value)
WM8904 write register.
- status_t WM8904_ModifyRegister (wm8904_handle_t *handle, uint8_t reg, uint16_t mask, uint16_t value)
WM8904 modify register.
- status_t WM8904_Init (wm8904_handle_t *handle, wm8904_config_t *wm8904Config)
Initializes WM8904.
- status_t WM8904_Deinit (wm8904_handle_t *handle)
Deinitializes the WM8904 codec.
- void WM8904_GetDefaultConfig (wm8904_config_t *config)

Fills the configuration structure with default values.

- status_t WM8904_SetMasterSlave (wm8904_handle_t *handle, bool master)
  Sets WM8904 as master or slave.
- status_t WM8904_SetMasterClock (wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth)
  Sets WM8904 master clock configuration.
- status_t WM8904_SetFLLConfig (wm8904_handle_t *handle, wm8904_fll_config_t *config)
  WM8904 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.
- status_t WM8904_SetProtocol (wm8904_handle_t *handle, wm8904_protocol_t protocol)
  Sets the audio data transfer protocol.
- status_t WM8904_SetAudioFormat (wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth)
  Sets the audio data format.
- status_t WM8904_CheckAudioFormat (wm8904_handle_t *handle, wm8904_audio_format_t *format, uint32_t mclkFreq)
  check and update the audio data format.
- status_t WM8904_SetVolume (wm8904_handle_t *handle, uint16_t volumeLeft, uint16_t volumeRight)
  Sets the module output volume.
- status_t WM8904_SetMute (wm8904_handle_t *handle, bool muteLeft, bool muteRight)
  Sets the headphone output mute.
- status_t WM8904_SelectLRCPolarity (wm8904_handle_t *handle, uint32_t polarity)
  Select LRC polarity.
- status_t WM8904_EnableDACTDMMode (wm8904_handle_t *handle, wm8904_timeslot_t timeSlot)
  Enable WM8904 DAC time slot.
- status_t WM8904_EnableADCTDMMode (wm8904_handle_t *handle, wm8904_timeslot_t timeSlot)
  Enable WM8904 ADC time slot.
- status_t WM8904_SetModulePower (wm8904_handle_t *handle, wm8904_module_t module, bool isEnabled)
  SET the module output power.
- status_t WM8904_SetDACVolume (wm8904_handle_t *handle, uint8_t volume)
  SET the DAC module volume.
- status_t WM8904_SetChannelVolume (wm8904_handle_t *handle, uint32_t channel, uint32_t volume)
  Sets the channel output volume.
- status_t WM8904_SetRecord (wm8904_handle_t *handle, uint32_t recordSource)
  SET the WM8904 record source.
- status_t WM8904_SetRecordChannel (wm8904_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  SET the WM8904 record source.
- status_t WM8904_SetPlay (wm8904_handle_t *handle, uint32_t playSource)
  SET the WM8904 play source.
- status_t WM8904_SetChannelMute (wm8904_handle_t *handle, uint32_t channel, bool isMute)
  Sets the channel mute.

Driver version

- #define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))
  WM8904 driver version 2.5.0.

42.8.2 Data Structure Documentation

42.8.2.1 struct wm8904_fll_config_t

Data Fields

- wm8904_fll_clk_source_t source
  fll reference clock source
- uint32_t refClock_HZ
  fll reference clock frequency
- uint32_t outputClock_HZ
  fll output clock frequency

42.8.2.2 struct wm8904_audio_format_t

Data Fields

- wm8904_fs_ratio_t fsRatio
  SYSCLK / fs ratio.
- wm8904_sample_rate_t sampleRate
  Sample rate.
- wm8904_bit_width_t bitWidth
  Bit width.

42.8.2.3 struct wm8904_config_t

Data Fields

- bool master
  Master or slave.
- wm8904_sys_clk_source_t sysClkSource
  system clock source
- wm8904_fll_config_t * fll
  fll configuration
- wm8904_protocol_t protocol
  Audio transfer protocol.
- wm8904_audio_format_t format
  Audio format.
- uint32_t mclk_HZ
  MCLK frequency value.
- uint16_t recordSource
  record source

- uint16_t recordChannelLeft
  record channel
- uint16_t recordChannelRight
  record channel
- uint16_t playSource
  play source
- uint8_t slaveAddress
  code device slave address
- codec_i2c_config_t i2cConfig
  i2c bus configuration

42.8.2.4 struct wm8904_handle_t

Data Fields

- wm8904_config_t * config
  wm8904 config pointer
- uint8_t i2cHandle [WM8904_I2C_HANDLER_SIZE]
  i2c handle

42.8.3 Macro Definition Documentation

42.8.3.1 #define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))

42.8.3.2 #define WM8904_I2C_ADDRESS (0x1A)

42.8.3.3 #define WM8904_I2C_BITRATE (400000U)

42.8.4 Enumeration Type Documentation

42.8.4.1 anonymous enum

Enumerator

kStatus_WM8904_Success Success.
kStatus_WM8904_Fail Failure.

42.8.4.2 anonymous enum

Enumerator

kWM8904_LRCPolarityNormal LRC polarity normal.
kWM8904_LRCPolarityInverted LRC polarity inverted.

42.8.4.3 enum wm8904_module_t

Enumerator

kWM8904_ModuleADC moduel ADC
kWM8904_ModuleDAC module DAC
kWM8904_ModulePGA module PGA
kWM8904_ModuleHeadphone module headphone
kWM8904_ModuleLineout module line out

42.8.4.4 anonymous enum

42.8.4.5 enum wm8904_timeslot_t

Enumerator

kWM8904_TimeSlot0 time slot0
kWM8904_TimeSlot1 time slot1

42.8.4.6 enum wm8904_protocol_t

Enumerator

kWM8904_ProtocolI2S I2S type.
kWM8904_ProtocolLeftJustified Left justified mode.
kWM8904_ProtocolRightJustified Right justified mode.
kWM8904_ProtocolPCMA PCM A mode.
kWM8904_ProtocolPCMB PCM B mode.

42.8.4.7 enum wm8904_fs_ratio_t

Enumerator

kWM8904_FsRatio64X SYSCLK is 64 ∗ sample rate ∗ frame width.
kWM8904_FsRatio128X SYSCLK is 128 ∗ sample rate ∗ frame width.
kWM8904_FsRatio192X SYSCLK is 192 ∗ sample rate ∗ frame width.
kWM8904_FsRatio256X SYSCLK is 256 ∗ sample rate ∗ frame width.
kWM8904_FsRatio384X SYSCLK is 384 ∗ sample rate ∗ frame width.
kWM8904_FsRatio512X SYSCLK is 512 ∗ sample rate ∗ frame width.
kWM8904_FsRatio768X SYSCLK is 768 ∗ sample rate ∗ frame width.
kWM8904_FsRatio1024X SYSCLK is 1024 ∗ sample rate ∗ frame width.
kWM8904_FsRatio1408X SYSCLK is 1408 ∗ sample rate ∗ frame width.
kWM8904_FsRatio1536X SYSCLK is 1536 ∗ sample rate ∗ frame width.

42.8.4.8 enum wm8904_sample_rate_t

Enumerator

kWM8904_SampleRate8kHz 8 kHz
kWM8904_SampleRate12kHz 12kHz
kWM8904_SampleRate16kHz 16kHz
kWM8904_SampleRate24kHz 24kHz
kWM8904_SampleRate32kHz 32kHz
kWM8904_SampleRate48kHz 48kHz
kWM8904_SampleRate11025Hz 11.025kHz
kWM8904_SampleRate22050Hz 22.05kHz
kWM8904_SampleRate44100Hz 44.1kHz

42.8.4.9 enum wm8904_bit_width_t

Enumerator

kWM8904_BitWidth16 16 bits
kWM8904_BitWidth20 20 bits
kWM8904_BitWidth24 24 bits
kWM8904_BitWidth32 32 bits

42.8.4.10 anonymous enum

Enumerator

kWM8904_RecordSourceDifferentialLine record source from differential line
kWM8904_RecordSourceLineInput record source from line input
kWM8904_RecordSourceDifferentialMic record source from differential mic
kWM8904_RecordSourceDigitalMic record source from digital microphone

42.8.4.11 anonymous enum

Enumerator

kWM8904_RecordChannelLeft1 left record channel 1
kWM8904_RecordChannelLeft2 left record channel 2
kWM8904_RecordChannelLeft3 left record channel 3
kWM8904_RecordChannelRight1 right record channel 1
kWM8904_RecordChannelRight2 right record channel 2
kWM8904_RecordChannelRight3 right record channel 3
kWM8904_RecordChannelDifferentialPositive1 differential positive record channel 1
kWM8904_RecordChannelDifferentialPositive2 differential positive record channel 2
kWM8904_RecordChannelDifferentialPositive3 differential positive record channel 3

kWM8904_RecordChannelDifferentialNegative1 differential negative record channel 1
kWM8904_RecordChannelDifferentialNegative2 differential negative record channel 2
kWM8904_RecordChannelDifferentialNegative3 differential negative record channel 3

42.8.4.12 anonymous enum

Enumerator

kWM8904_PlaySourcePGA play source PGA, bypass ADC
kWM8904_PlaySourceDAC play source Input3

42.8.4.13 enum wm8904_sys_clk_source_t

Enumerator

kWM8904_SysClkSourceMCLK wm8904 system clock soure from MCLK
kWM8904_SysClkSourceFLL wm8904 system clock soure from FLL

42.8.4.14 enum wm8904_fll_clk_source_t

Enumerator

kWM8904_FLLClkSourceMCLK wm8904 FLL clock source from MCLK

42.8.5 Function Documentation

42.8.5.1 status_t WM8904_WriteRegister (wm8904_handle_t * *handle*, uint8_t *reg*, uint16_t *value*)

Parameters

handle	WM8904 handle structure.
reg	register address.
value	value to write.

Returns

kStatus_Success, else failed.

42.8.5.2 status_t WM8904_ReadRegister (wm8904_handle_t * *handle*, uint8_t *reg*, uint16_t * *value*)

Parameters

handle	WM8904 handle structure.
reg	register address.
value	value to read.

Returns

kStatus_Success, else failed.

42.8.5.3 status_t WM8904_ModifyRegister (wm8904_handle_t * *handle*, uint8_t *reg*, uint16_t *mask*, uint16_t *value*)

Parameters

handle	WM8904 handle structure.
reg	register address.
mask	register bits mask.
value	value to write.

Returns

kStatus_Success, else failed.

42.8.5.4 status_t WM8904_Init (wm8904_handle_t * *handle*, wm8904_config_t * *wm8904Config*)

Parameters

handle	WM8904 handle structure.
wm8904Config	WM8904 configuration structure.

42.8.5.5 status_t WM8904_Deinit (wm8904_handle_t * *handle*)

This function resets WM8904.

Parameters

handle	WM8904 handle structure.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.6 void WM8904_GetDefaultConfig (wm8904_config_t * *config*)

The default values are:

master = false; protocol = kWM8904_ProtocolI2S; format.fsRatio = kWM8904_FsRatio64X; format.sampleRate = kWM8904_SampleRate48kHz; format.bitWidth = kWM8904_BitWidth16;

Parameters

config	default configurations of wm8904.

42.8.5.7 status_t WM8904_SetMasterSlave (wm8904_handle_t * *handle*, bool *master*)

Deprecated DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY WM8904_SetMasterClock

Parameters

handle	WM8904 handle structure.
master	true for master, false for slave.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.8 status_t WM8904_SetMasterClock (wm8904_handle_t * *handle*, uint32_t *sysclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

User should pay attention to the sysclk parameter ,When using external MCLK as system clock source, the value should be frequency of MCLK, when using FLL as system clock source, the value should be frequency of the output of FLL.

Parameters

handle	WM8904 handle structure.
sysclk	system clock source frequency.
sampleRate	sample rate
bitWidth	bit width

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.9 status_t WM8904_SetFLLConfig (wm8904_handle_t * *handle*, wm8904_fll_config_t * *config*)

Parameters

handle	wm8904 handler pointer.
config	FLL configuration pointer.

42.8.5.10 status_t WM8904_SetProtocol (wm8904_handle_t * *handle*, wm8904_protocol_t *protocol*)

Parameters

handle	WM8904 handle structure.
protocol	Audio transfer protocol.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.11 status_t WM8904_SetAudioFormat (wm8904_handle_t * *handle*, uint32_t *sysclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

User should pay attention to the sysclk parameter ,When using external MCLK as system clock source, the value should be frequency of MCLK, when using FLL as system clock source, the value should be frequency of the output of FLL.

Parameters

handle	WM8904 handle structure.
sysclk	system clock source frequency.
sampleRate	Sample rate frequency in Hz.
bitWidth	Audio data bit width.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.12 status_t WM8904_CheckAudioFormat (wm8904_handle_t * *handle*, wm8904_audio_format_t * *format*, uint32_t *mclkFreq*)

This api is used check the fsRatio setting based on the mclk and sample rate, if fsRatio setting is not correct, it will correct it according to mclk and sample rate.

Parameters

handle	WM8904 handle structure.
format	audio data format
mclkFreq	mclk frequency

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.13 status_t WM8904_SetVolume (wm8904_handle_t * *handle*, uint16_t *volumeLeft*, uint16_t *volumeRight*)

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57DB, 63 for 6DB.

Parameters

handle	WM8904 handle structure.

volumeLeft	left channel volume.
volumeRight	right channel volume.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.14 status_t WM8904_SetMute (wm8904_handle_t * *handle*, bool *muteLeft*, bool *muteRight*)

Parameters

handle	WM8904 handle structure.
muteLeft	true to mute left channel, false to unmute.
muteRight	true to mute right channel, false to unmute.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.15 status_t WM8904_SelectLRCPolarity (wm8904_handle_t * *handle*, uint32_t *polarity*)

Parameters

handle	WM8904 handle structure.
polarity	LRC clock polarity.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.16 status_t WM8904_EnableDACTDMMode (wm8904_handle_t * *handle*, wm8904_timeslot_t *timeSlot*)

Parameters

handle	WM8904 handle structure.
timeSlot	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.17 status_t WM8904_EnableADCTDMMode (wm8904_handle_t * *handle*, wm8904_timeslot_t *timeSlot*)

Parameters

handle	WM8904 handle structure.
timeSlot	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.18 status_t WM8904_SetModulePower (wm8904_handle_t * *handle*, wm8904_module_t *module*, bool *isEnabled*)

Parameters

handle	WM8904 handle structure.
module	wm8904 module.
isEnabled,true	is power on, false is power down.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

42.8.5.19 status_t WM8904_SetDACVolume (wm8904_handle_t * *handle*, uint8_t *volume*)

Parameters

handle	WM8904 handle structure.
volume	volume to be configured.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

42.8.5.20 status_t WM8904_SetChannelVolume (wm8904_handle_t * *handle,* uint32_t *channel,* uint32_t *volume*)

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57dB, 63 for 6DB.

Parameters

handle	codec handle structure.
channel	codec channel.
volume	volume value from 0 -63.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.21 status_t WM8904_SetRecord (wm8904_handle_t * *handle,* uint32_t *recordSource*)

Parameters

handle	WM8904 handle structure.
recordSource	record source , can be a value of kCODEC_ModuleRecordSourceDifferential-Line, kCODEC_ModuleRecordSourceDifferentialMic, kCODEC_ModuleRecord-SourceSingleEndMic, kCODEC_ModuleRecordSourceDigitalMic.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.5.22 status_t WM8904_SetRecordChannel (wm8904_handle_t * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	WM8904 handle structure.
leftRecord-Channel	channel number of left record channel when using differential source, channel number of single end left channel when using single end source, channel number of digital mic when using digital mic source.
rightRecord-Channel	channel number of right record channel when using differential source, channel number of single end right channel when using single end source.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

42.8.5.23 status_t WM8904_SetPlay (wm8904_handle_t * *handle,* uint32_t *playSource*)

Parameters

handle	WM8904 handle structure.
playSource	play source , can be a value of kCODEC_ModuleHeadphoneSourcePGA, kCODEC-_ModuleHeadphoneSourceDAC, kCODEC_ModuleLineoutSourcePGA, kCODEC-_ModuleLineoutSourceDAC.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

42.8.5.24 status_t WM8904_SetChannelMute (wm8904_handle_t * *handle,* uint32_t *channel,* bool *isMute*)

Parameters

handle	codec handle structure.
channel	codec module name.
isMute	true is mute, false unmute.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

42.8.6 WM8904 Adapter

42.8.6.1 Overview

The wm8904 adapter provides a codec unify control interface.

Macros

- #define HAL_CODEC_WM8904_HANDLER_SIZE (WM8904_I2C_HANDLER_SIZE + 4)
  codec handler size

Functions

- status_t HAL_CODEC_WM8904_Init (void *handle, void *config)
  Codec initilization.
- status_t HAL_CODEC_WM8904_Deinit (void *handle)
  Codec de-initilization.
- status_t HAL_CODEC_WM8904_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_WM8904_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_WM8904_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_WM8904_SetPower (void *handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_WM8904_SetRecord (void *handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_WM8904_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_WM8904_SetPlay (void *handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_WM8904_ModuleControl (void *handle, uint32_t cmd, uint32_t data)
  codec module control.
- static status_t HAL_CODEC_Init (void *handle, void *config)
  Codec initilization.
- static status_t HAL_CODEC_Deinit (void *handle)
  Codec de-initilization.
- static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)

set audio codec module power.

- static status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- static status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- static status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- static status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

42.8.6.2 Function Documentation

42.8.6.2.1 status_t HAL_CODEC_WM8904_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

42.8.6.2.2 status_t HAL_CODEC_WM8904_Deinit (void ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.8.6.2.3 status_t HAL_CODEC_WM8904_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.4 status_t HAL_CODEC_WM8904_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.5 status_t HAL_CODEC_WM8904_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.6 status_t HAL_CODEC_WM8904_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.7 status_t HAL_CODEC_WM8904_SetRecord (void * *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.8 status_t HAL_CODEC_WM8904_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.9 status_t HAL_CODEC_WM8904_SetPlay (void * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.10 status_t HAL_CODEC_WM8904_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.11 static status_t HAL_CODEC_Init (void * *handle,* void * *config*) [inline], [static]

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

42.8.6.2.12 static status_t HAL_CODEC_Deinit (void * *handle*) [inline], [static]

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

42.8.6.2.13 static status_t HAL_CODEC_SetFormat (void * *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*) [inline], [static]

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.14 static status_t HAL_CODEC_SetVolume (void * *handle,* uint32_t *playChannel,* uint32_t *volume*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 $\sim$ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.15 static status_t HAL_CODEC_SetMute (void * *handle,* uint32_t *playChannel,* bool *isMute*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.16 static status_t HAL_CODEC_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*) [inline], [static]

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.17 static status_t HAL_CODEC_SetRecord (void * *handle,* uint32_t *recordSource*) [inline], [static]

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.18 static status_t HAL_CODEC_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*) [inline], [static]

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.19 static status_t HAL_CODEC_SetPlay (void * *handle,* uint32_t *playSource*) [inline], [static]

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

42.8.6.2.20 static status_t HAL_CODEC_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*) [inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

Chapter 43
Serial Manager

43.1 Overview

This chapter describes the programming interface of the serial manager component.

The serial manager component provides a series of APIs to operate different serial port types. The port types it supports are UART, USB CDC and SWO.

Modules

- Serial Port SWO
- Serial Port USB
- Serial Port Uart

Data Structures

- struct serial_manager_config_t
  serial manager config structure More...
- struct serial_manager_callback_message_t
  Callback message structure. More...

Macros

- #define SERIAL_MANAGER_NON_BLOCKING_MODE (0U)
  Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)
- #define SERIAL_MANAGER_RING_BUFFER_FLOWCONTROL (0U)
  Enable or ring buffer flow control (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_UART (0U)
  Enable or disable uart port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_UART_DMA (0U)
  Enable or disable uart dma port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_USBCDC (0U)
  Enable or disable USB CDC port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_SWO (0U)
  Enable or disable SWO port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_VIRTUAL (0U)
  Enable or disable USB CDC virtual port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_RPMSG (0U)
  Enable or disable rPMSG port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_SPI_MASTER (0U)
  Enable or disable SPI Master port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_SPI_SLAVE (0U)
  Enable or disable SPI Slave port (1 - enable, 0 - disable)
- #define SERIAL_MANAGER_TASK_HANDLE_TX (0U)
  Enable or disable SerialManager_Task() handle TX to prevent recursive calling.

- #define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)
  Set the default delay time in ms used by SerialManager_WriteTimeDelay().
- #define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)
  Set the default delay time in ms used by SerialManager_ReadTimeDelay().
- #define SERIAL_MANAGER_TASK_HANDLE_RX_AVAILABLE_NOTIFY (0U)
  Enable or disable SerialManager_Task() handle RX data available notify.
- #define SERIAL_MANAGER_WRITE_HANDLE_SIZE (4U)
  Set serial manager write handle size.
- #define SERIAL_MANAGER_USE_COMMON_TASK (0U)
  SERIAL_PORT_UART_HANDLE_SIZE/SERIAL_PORT_USB_CDC_HANDLE_SIZE + serial manager dedicated size.
- #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)
  Macro to determine whether use common task.
- #define SERIAL_MANAGER_HANDLE_DEFINE(name) uint32_t name[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the serial manager handle.
- #define SERIAL_MANAGER_WRITE_HANDLE_DEFINE(name) uint32_t name[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the serial manager write handle.
- #define SERIAL_MANAGER_READ_HANDLE_DEFINE(name) uint32_t name[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the serial manager read handle.
- #define SERIAL_MANAGER_TASK_PRIORITY (2U)
  Macro to set serial manager task priority.
- #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)
  Macro to set serial manager task stack size.

Typedefs

- typedef void * serial_handle_t
  The handle of the serial manager module.
- typedef void * serial_write_handle_t
  The write handle of the serial manager module.
- typedef void * serial_read_handle_t
  The read handle of the serial manager module.
- typedef void(* serial_manager_callback_t)(void *callbackParam, serial_manager_callback_message_t *message, serial_manager_status_t status)
  callback function

Enumerations

- enum serial_port_type_t {
  kSerialPort_Uart = 1U,
  kSerialPort_UsbCdc,
  kSerialPort_Swo,
  kSerialPort_Virtual,
  kSerialPort_Rpmsg,
  kSerialPort_UartDma,
  kSerialPort_SpiMaster,
  kSerialPort_SpiSlave,
  kSerialPort_None }
  serial port type
- enum serial_manager_type_t {
  kSerialManager_NonBlocking = 0x0U,
  kSerialManager_Blocking = 0x8F41U }
  serial manager type
- enum serial_manager_status_t {
  kStatus_SerialManager_Success = kStatus_Success,
  kStatus_SerialManager_Error = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1),
  kStatus_SerialManager_Busy = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2),
  kStatus_SerialManager_Notify = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3),
  kStatus_SerialManager_Canceled,
  kStatus_SerialManager_HandleConflict = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5),
  kStatus_SerialManager_RingBufferOverflow,
  kStatus_SerialManager_NotConnected = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7) }
  serial manager error code

Functions

- serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, const serial_manager_config_t *config)
  Initializes a serial manager module with the serial manager handle and the user configuration structure.
- serial_manager_status_t SerialManager_Deinit (serial_handle_t serialHandle)
  De-initializes the serial manager module instance.
- serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t serialHandle, serial_write_handle_t writeHandle)
  Opens a writing handle for the serial manager module.
- serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t writeHandle)
  Closes a writing handle for the serial manager module.
- serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t serialHandle, serial_read_handle_t readHandle)
  Opens a reading handle for the serial manager module.
- serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t readHandle)
  Closes a reading for the serial manager module.

- serial_manager_status_t SerialManager_WriteBlocking (serial_write_handle_t writeHandle, uint8_t *buffer, uint32_t length)
  Transmits data with the blocking mode.
- serial_manager_status_t SerialManager_ReadBlocking (serial_read_handle_t readHandle, uint8_t *buffer, uint32_t length)
  Reads data with the blocking mode.
- serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t serialHandle)
  Prepares to enter low power consumption.
- serial_manager_status_t SerialManager_ExitLowpower (serial_handle_t serialHandle)
  Restores from low power consumption.
- static bool SerialManager_needPollingIsr (void)
  Check if need polling ISR.

43.2 Data Structure Documentation

43.2.1 struct serial_manager_config_t

Data Fields

- uint8_t * ringBuffer
  Ring buffer address, it is used to buffer data received by the hardware.
- uint32_t ringBufferSize
  The size of the ring buffer.
- serial_port_type_t type
  Serial port type.
- serial_manager_type_t blockType
  Serial manager port type.
- void * portConfig
  Serial port configuration.

Field Documentation

(1) uint8_t* serial_manager_config_t::ringBuffer

Besides, the memory space cannot be free during the lifetime of the serial manager module.

43.2.2 struct serial_manager_callback_message_t

Data Fields

- uint8_t * buffer
  Transferred buffer.
- uint32_t length
  Transferred data length.

43.3 Macro Definition Documentation

43.3.1 #define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)

43.3.2 #define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)

43.3.3 #define SERIAL_MANAGER_USE_COMMON_TASK (0U)

Macro to determine whether use common task.

43.3.4 #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)

Definition of serial manager handle size.

43.3.5 #define SERIAL_MANAGER_HANDLE_DEFINE(*name*) uint32_t name[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned serial manager handle. Then use "(serial_handle_t)name" to get the serial manager handle.

The macro should be global and could be optional. You could also define serial manager handle by yourself.

This is an example,

```
* SERIAL_MANAGER_HANDLE_DEFINE(serialManagerHandle);
*
```

Parameters

name	The name string of the serial manager handle.

43.3.6 #define SERIAL_MANAGER_WRITE_HANDLE_DEFINE(*name*) uint32_t name[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned serial manager write handle. Then use "(serial_write_handle_t)name" to get the serial manager write handle.

The macro should be global and could be optional. You could also define serial manager write handle by yourself.

This is an example,

```
* SERIAL_MANAGER_WRITE_HANDLE_DEFINE(serialManagerwriteHandle);
*
```

Parameters

name	The name string of the serial manager write handle.

43.3.7 #define SERIAL_MANAGER_READ_HANDLE_DEFINE(*name*) uint32_t name[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned serial manager read handle. Then use "(serial_read_handle-_t)name" to get the serial manager read handle.

The macro should be global and could be optional. You could also define serial manager read handle by yourself.

This is an example,

```
* SERIAL_MANAGER_READ_HANDLE_DEFINE(serialManagerReadHandle);
*
```

Parameters

name	The name string of the serial manager read handle.

43.3.8 #define SERIAL_MANAGER_TASK_PRIORITY (2U)

43.3.9 #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)

43.4 Enumeration Type Documentation

43.4.1 enum serial_port_type_t

Enumerator

kSerialPort_Uart Serial port UART.
kSerialPort_UsbCdc Serial port USB CDC.
kSerialPort_Swo Serial port SWO.
kSerialPort_Virtual Serial port Virtual.
kSerialPort_Rpmsg Serial port RPMSG.
kSerialPort_UartDma Serial port UART DMA.
kSerialPort_SpiMaster Serial port SPIMASTER.

kSerialPort_SpiSlave Serial port SPISLAVE.
kSerialPort_None Serial port is none.

43.4.2 enum serial_manager_type_t

Enumerator

kSerialManager_NonBlocking None blocking handle.
kSerialManager_Blocking Blocking handle.

43.4.3 enum serial_manager_status_t

Enumerator

kStatus_SerialManager_Success Success.
kStatus_SerialManager_Error Failed.
kStatus_SerialManager_Busy Busy.
kStatus_SerialManager_Notify Ring buffer is not empty.
kStatus_SerialManager_Canceled the non-blocking request is canceled
kStatus_SerialManager_HandleConflict The handle is opened.
kStatus_SerialManager_RingBufferOverflow The ring buffer is overflowed.
kStatus_SerialManager_NotConnected The host is not connected.

43.5 Function Documentation

43.5.1 serial_manager_status_t SerialManager_Init (serial_handle_t *serialHandle*, const serial_manager_config_t * *config*)

This function configures the Serial Manager module with user-defined settings. The user can configure the configuration structure. The parameter serialHandle is a pointer to point to a memory space of size SERIAL_MANAGER_HANDLE_SIZE allocated by the caller. The Serial Manager module supports three types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc), USB CDC and swo. Please refer to serial_port_type_t for serial port setting. These three types can be set by using serial_manager_config_t.

Example below shows how to use this API to configure the Serial Manager. For UART,

```
*   #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
*   static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*   static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
*   serial_manager_config_t config;
*   serial_port_uart_config_t uartConfig;
*   config.type = kSerialPort_Uart;
*   config.ringBuffer = &s_ringBuffer[0];
*   config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
*   uartConfig.instance = 0;
```

```
*   uartConfig.clockRate = 24000000;
*   uartConfig.baudRate = 115200;
*   uartConfig.parityMode = kSerialManager_UartParityDisabled;
*   uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
*   uartConfig.enableRx = 1;
*   uartConfig.enableTx = 1;
*   uartConfig.enableRxRTS = 0;
*   uartConfig.enableTxCTS = 0;
*   config.portConfig = &uartConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*
```

For USB CDC,

```
*   #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
*   static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*   static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
*   serial_manager_config_t config;
*   serial_port_usb_cdc_config_t usbCdcConfig;
*   config.type = kSerialPort_UsbCdc;
*   config.ringBuffer = &s_ringBuffer[0];
*   config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
*   usbCdcConfig.controllerIndex =
      kSerialManager_UsbControllerKhci0;
*   config.portConfig = &usbCdcConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*
```

Parameters

serialHandle	Pointer to point to a memory space of size SERIAL_MANAGER_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_HANDLE_DEFINE(serialHandle); or uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];
config	Pointer to user-defined configuration structure.

Return values

kStatus_SerialManager_Error	An error occurred.
kStatus_SerialManager_Success	The Serial Manager module initialization succeed.

43.5.2 serial_manager_status_t SerialManager_Deinit (serial_handle_t *serialHandle*)

This function de-initializes the serial manager module instance. If the opened writing or reading handle is not closed, the function will return kStatus_SerialManager_Busy.

Parameters

serialHandle	The serial manager module handle pointer.

Return values

kStatus_SerialManager_Success	The serial manager de-initialization succeed.
kStatus_SerialManager_Busy	Opened reading or writing handle is not closed.

43.5.3 serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t *serialHandle*, serial_write_handle_t *writeHandle*)

This function Opens a writing handle for the serial manager module. If the serial manager needs to be used in different tasks, the task should open a dedicated write handle for itself by calling SerialManager_OpenWriteHandle. Since there can only one buffer for transmission for the writing handle at the same time, multiple writing handles need to be opened when the multiple transmission is needed for a task.

Parameters

serialHandle	The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
writeHandle	The serial manager module writing handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_WRITE_HANDLE_DEFINE(writeHandle); or uint32_t writeHandle[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];

Return values

kStatus_SerialManager_Error	An error occurred.
kStatus_SerialManager_HandleConflict	The writing handle was opened.

kStatus_SerialManager_Success	The writing handle is opened.

Example below shows how to use this API to write data. For task 1,

```
*   static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle1);
*   static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
*   SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
      , (serial_write_handle_t)s_serialWriteHandle1);
*   SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle1,
*                                   Task1_SerialManagerTxCallback,
*                                   s_serialWriteHandle1);
*   SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle1,
*                                  s_nonBlockingWelcome1,
*                                  sizeof(s_nonBlockingWelcome1) - 1U);
*
```

For task 2,

```
*   static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle2);
*   static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
*   SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
      , (serial_write_handle_t)s_serialWriteHandle2);
*   SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle2,
*                                   Task2_SerialManagerTxCallback,
*                                   s_serialWriteHandle2);
*   SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle2,
*                                  s_nonBlockingWelcome2,
*                                  sizeof(s_nonBlockingWelcome2) - 1U);
*
```

43.5.4 serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t *writeHandle*)

This function Closes a writing handle for the serial manager module.

Parameters

writeHandle	The serial manager module writing handle pointer.

Return values

kStatus_SerialManager_Success	The writing handle is closed.

43.5.5 serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t *serialHandle*, serial_read_handle_t *readHandle*)

This function Opens a reading handle for the serial manager module. The reading handle can not be opened multiple at the same time. The error code kStatus_SerialManager_Busy would be returned when

the previous reading handle is not closed. And there can only be one buffer for receiving for the reading handle at the same time.

Parameters

serialHandle	The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
readHandle	The serial manager module reading handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle); or uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];

Return values

kStatus_SerialManager_Error	An error occurred.
kStatus_SerialManager_Success	The reading handle is opened.
kStatus_SerialManager_Busy	Previous reading handle is not closed.

Example below shows how to use this API to read data.

```
*   static SERIAL_MANAGER_READ_HANDLE_DEFINE(s_serialReadHandle);
*   SerialManager_OpenReadHandle((serial_handle_t)serialHandle,
      (serial_read_handle_t)s_serialReadHandle);
*   static uint8_t s_nonBlockingBuffer[64];
*   SerialManager_InstallRxCallback((serial_read_handle_t)s_serialReadHandle,
*                                   APP_SerialManagerRxCallback,
*                                   s_serialReadHandle);
*   SerialManager_ReadNonBlocking((serial_read_handle_t)s_serialReadHandle,
*                                 s_nonBlockingBuffer,
*                                 sizeof(s_nonBlockingBuffer));
*
```

43.5.6 serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t *readHandle*)

This function Closes a reading for the serial manager module.

Parameters

readHandle	The serial manager module reading handle pointer.

Return values

kStatus_SerialManager_Success	The reading handle is closed.

43.5.7 serial_manager_status_t SerialManager_WriteBlocking (serial_write_handle_t *writeHandle*, uint8_t * *buffer*, uint32_t *length*)

This is a blocking function, which polls the sending queue, waits for the sending queue to be empty. This function sends data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for transmission for the writing handle at the same time.

Note

The function SerialManager_WriteBlocking and the function SerialManager_WriteNonBlocking cannot be used at the same time. And, the function SerialManager_CancelWriting cannot be used to abort the transmission of this function.

Parameters

writeHandle	The serial manager module handle pointer.
buffer	Start address of the data to write.
length	Length of the data to write.

Return values

kStatus_SerialManager_Success	Successfully sent all data.
kStatus_SerialManager_Busy	Previous transmission still not finished; data not all sent yet.
kStatus_SerialManager_Error	An error occurred.

43.5.8 serial_manager_status_t SerialManager_ReadBlocking (serial_read_handle_t *readHandle*, uint8_t * *buffer*, uint32_t *length*)

This is a blocking function, which polls the receiving buffer, waits for the receiving buffer to be full. This function receives data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for receiving for the reading handle at the same time.

Note

The function SerialManager_ReadBlocking and the function SerialManager_ReadNonBlocking cannot be used at the same time. And, the function SerialManager_CancelReading cannot be used to abort the transmission of this function.

Parameters

readHandle	The serial manager module handle pointer.
buffer	Start address of the data to store the received data.
length	The length of the data to be received.

Return values

kStatus_SerialManager_Success	Successfully received all data.
kStatus_SerialManager_Busy	Previous transmission still not finished; data not all received yet.
kStatus_SerialManager_Error	An error occurred.

43.5.9 serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t *serialHandle*)

This function is used to prepare to enter low power consumption.

Parameters

serialHandle	The serial manager module handle pointer.

Return values

kStatus_SerialManager_Success	Successful operation.

43.5.10 serial_manager_status_t SerialManager_ExitLowpower (serial_handle_t *serialHandle*)

This function is used to restore from low power consumption.

Parameters

serialHandle	The serial manager module handle pointer.

Return values

kStatus_SerialManager_-Success	Successful operation.

43.5.11 static bool SerialManager_needPollingIsr (void) [inline], [static]

This function is used to check if need polling ISR.

Return values

TRUE	if need polling.

43.6 Serial Port Uart

43.6.1 Overview

Macros

- #define SERIAL_PORT_UART_DMA_RECEIVE_DATA_LENGTH (64U)
  serial port uart handle size
- #define SERIAL_USE_CONFIGURE_STRUCTURE (0U)
  Enable or disable the confgure structure pointer.

Enumerations

- enum serial_port_uart_parity_mode_t {
  kSerialManager_UartParityDisabled = 0x0U,
  kSerialManager_UartParityEven = 0x2U,
  kSerialManager_UartParityOdd = 0x3U }
  serial port uart parity mode
- enum serial_port_uart_stop_bit_count_t {
  kSerialManager_UartOneStopBit = 0U,
  kSerialManager_UartTwoStopBit = 1U }
  serial port uart stop bit count

43.6.2 Enumeration Type Documentation

43.6.2.1 enum serial_port_uart_parity_mode_t

Enumerator

kSerialManager_UartParityDisabled Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

43.6.2.2 enum serial_port_uart_stop_bit_count_t

Enumerator

kSerialManager_UartOneStopBit One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

43.7 Serial Port USB

43.7.1 Overview

Modules

- USB Device Configuration

Data Structures

- struct serial_port_usb_cdc_config_t
  serial port usb config struct More...

Macros

- #define SERIAL_PORT_USB_CDC_HANDLE_SIZE (72U)
  serial port usb handle size
- #define USB_DEVICE_INTERRUPT_PRIORITY (3U)
  USB interrupt priority.

Enumerations

- enum serial_port_usb_cdc_controller_index_t {
  kSerialManager_UsbControllerKhci0 = 0U,
  kSerialManager_UsbControllerKhci1 = 1U,
  kSerialManager_UsbControllerEhci0 = 2U,
  kSerialManager_UsbControllerEhci1 = 3U,
  kSerialManager_UsbControllerLpcIp3511Fs0 = 4U,
  kSerialManager_UsbControllerLpcIp3511Fs1 = 5U,
  kSerialManager_UsbControllerLpcIp3511Hs0 = 6U,
  kSerialManager_UsbControllerLpcIp3511Hs1 = 7U,
  kSerialManager_UsbControllerOhci0 = 8U,
  kSerialManager_UsbControllerOhci1 = 9U,
  kSerialManager_UsbControllerIp3516Hs0 = 10U,
  kSerialManager_UsbControllerIp3516Hs1 = 11U }
  USB controller ID.

43.7.2 Data Structure Documentation

43.7.2.1 struct serial_port_usb_cdc_config_t

Data Fields

- serial_port_usb_cdc_controller_index_t controllerIndex
  controller index

43.7.3 Enumeration Type Documentation

43.7.3.1 enum serial_port_usb_cdc_controller_index_t

Enumerator

kSerialManager_UsbControllerKhci0 KHCI 0U.

kSerialManager_UsbControllerKhci1 KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerEhci0 EHCI 0U.

kSerialManager_UsbControllerEhci1 EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerLpcIp3511Fs0 LPC USB IP3511 FS controller 0.

kSerialManager_UsbControllerLpcIp3511Fs1 LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerLpcIp3511Hs0 LPC USB IP3511 HS controller 0.

kSerialManager_UsbControllerLpcIp3511Hs1 LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerOhci0 OHCI 0U.

kSerialManager_UsbControllerOhci1 OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerIp3516Hs0 IP3516HS 0U.

kSerialManager_UsbControllerIp3516Hs1 IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.

43.7.4 USB Device Configuration

43.8 Serial Port SWO

43.8.1 Overview

Data Structures

- struct serial_port_swo_config_t
  serial port swo config struct More...

Macros

- #define SERIAL_PORT_SWO_HANDLE_SIZE (12U)
  serial port swo handle size

Enumerations

- enum serial_port_swo_protocol_t {
  kSerialManager_SwoProtocolManchester = 1U,
  kSerialManager_SwoProtocolNrz = 2U }
  serial port swo protocol

43.8.2 Data Structure Documentation

43.8.2.1 struct serial_port_swo_config_t

Data Fields

- uint32_t clockRate
  clock rate
- uint32_t baudRate
  baud rate
- uint32_t port
  Port used to transfer data.
- serial_port_swo_protocol_t protocol
  SWO protocol.

43.8.3 Enumeration Type Documentation

43.8.3.1 enum serial_port_swo_protocol_t

Enumerator

kSerialManager_SwoProtocolManchester SWO Manchester protocol.
kSerialManager_SwoProtocolNrz SWO UART/NRZ protocol.

Chapter 44
CDOG

44.1 Overview

Files

- file fsl_cdog.h

Driver version

- #define FSL_CDOG_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  Defines CDOG driver version 2.1.0.

CDOG Functional Operation

- status_t CDOG_Init (CDOG_Type *base, cdog_config_t *conf)
  Initialize CDOG.
- void CDOG_Deinit (CDOG_Type *base)
  Deinitialize CDOG.
- void CDOG_GetDefaultConfig (cdog_config_t *conf)
  Sets the default configuration of CDOG.
- void CDOG_Stop (CDOG_Type *base, uint32_t stop)
  Stops secure counter and instruction timer.
- void CDOG_Start (CDOG_Type *base, uint32_t reload, uint32_t start)
  Sets secure counter and instruction timer values.
- void CDOG_Check (CDOG_Type *base, uint32_t check)
  Checks secure counter.
- void CDOG_Set (CDOG_Type *base, uint32_t stop, uint32_t reload, uint32_t start)
  Sets secure counter and instruction timer values.
- void CDOG_Add (CDOG_Type *base, uint32_t add)
  Add value to secure counter.
- void CDOG_Add1 (CDOG_Type *base)
  Add 1 to secure counter.
- void CDOG_Add16 (CDOG_Type *base)
  Add 16 to secure counter.
- void CDOG_Add256 (CDOG_Type *base)
  Add 256 to secure counter.
- void CDOG_Sub (CDOG_Type *base, uint32_t sub)
  brief Substract value to secure counter
- void CDOG_Sub1 (CDOG_Type *base)
  Substract 1 from secure counter.
- void CDOG_Sub16 (CDOG_Type *base)
  Substract 16 from secure counter.
- void CDOG_Sub256 (CDOG_Type *base)
  Substract 256 from secure counter.
- void CDOG_WritePersistent (CDOG_Type *base, uint32_t value)

Set the CDOG persistent word.

- uint32_t CDOG_ReadPersistent (CDOG_Type *base)
  Get the CDOG persistent word.

44.2 Macro Definition Documentation

44.2.1 #define FSL_CDOG_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

Change log:

- Version 2.1.0
  - Rename CWT to CDOG
- Version 2.0.2
  - Fix MISRA-2012 issues
- Version 2.0.1
  - Fix doxygen issues
- Version 2.0.0
  - initial version

44.3 Function Documentation

44.3.1 status_t CDOG_Init (CDOG_Type * *base*, cdog_config_t * *conf*)

This function initializes CDOG block and setting.

Parameters

base	CDOG peripheral base address
conf	CDOG configuration structure

Returns

Status of the init operation

44.3.2 void CDOG_Deinit (CDOG_Type * *base*)

This function deinitializes CDOG secure counter.

Parameters

base	CDOG peripheral base address

44.3.3 void CDOG_GetDefaultConfig (cdog_config_t * *conf*)

This function initialize CDOG config structure to default values.

Parameters

conf	CDOG configuration structure

44.3.4 void CDOG_Stop (CDOG_Type * *base,* uint32_t *stop*)

This function stops instruction timer and secure counter. This also change state od CDOG to IDLE.

Parameters

base	CDOG peripheral base address
stop	expected value which will be compared with value of secure counter

44.3.5 void CDOG_Start (CDOG_Type * *base,* uint32_t *reload,* uint32_t *start*)

This function sets value in RELOAD and START registers for instruction timer and secure counter

Parameters

base	CDOG peripheral base address
reload	reload value
start	start value

44.3.6 void CDOG_Check (CDOG_Type * *base,* uint32_t *check*)

This function compares stop value in handler with secure counter value by writting to RELOAD refister.

Parameters

base	CDOG peripheral base address
check	expected (stop) value

44.3.7 void CDOG_Set (CDOG_Type * *base*, uint32_t *stop*, uint32_t *reload*, uint32_t *start*)

This function sets value in STOP, RELOAD and START registers for instruction timer and secure counter.

Parameters

base	CDOG peripheral base address
stop	expected value which will be compared with value of secure counter
reload	reload value for instruction timer
start	start value for secure timer

44.3.8 void CDOG_Add (CDOG_Type * *base*, uint32_t *add*)

This function add specified value to secure counter.

Parameters

base	CDOG peripheral base address.
add	Value to be added.

44.3.9 void CDOG_Add1 (CDOG_Type * *base*)

This function add 1 to secure counter.

Parameters

base	CDOG peripheral base address.

44.3.10 void CDOG_Add16 (CDOG_Type * *base*)

This function add 16 to secure counter.

Parameters

base	CDOG peripheral base address.

44.3.11 void CDOG_Add256 (CDOG_Type * *base*)

This function add 256 to secure counter.

Parameters

base	CDOG peripheral base address.

44.3.12 void CDOG_Sub (CDOG_Type * *base,* uint32_t *sub*)

This function substract specified value to secure counter.

param base CDOG peripheral base address. param sub Value to be substracted.

44.3.13 void CDOG_Sub1 (CDOG_Type * *base*)

This function substract specified 1 from secure counter.

Parameters

base	CDOG peripheral base address.

44.3.14 void CDOG_Sub16 (CDOG_Type * *base*)

This function substract specified 16 from secure counter.

Parameters

base	CDOG peripheral base address.

44.3.15 void CDOG_Sub256 (CDOG_Type * *base*)

This function substract specified 256 from secure counter.

Parameters

base	CDOG peripheral base address.

44.3.16 void CDOG_WritePersistent (CDOG_Type * *base,* uint32_t *value*)

Parameters

base	CDOG peripheral base address.
value	The value to be written.

44.3.17 uint32_t CDOG_ReadPersistent (CDOG_Type * *base*)

Parameters

base	CDOG peripheral base address.

Returns

The persistent word.

Chapter 45
I2s_dma_driver

45.1 Overview

Data Structures

- struct i2s_dma_handle_t
  i2s dma handle More...

Typedefs

- typedef void(∗ i2s_dma_transfer_callback_t)(I2S_Type ∗base, i2s_dma_handle_t ∗handle, status_t completionStatus, void ∗userData)
  Callback function invoked from DMA API on completion.

Driver version

- #define FSL_I2S_DMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  I2S DMA driver version 2.3.0.

DMA API

- void I2S_TxTransferCreateHandleDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle, dma_handle_t ∗dmaHandle, i2s_dma_transfer_callback_t callback, void ∗userData)
  Initializes handle for transfer of audio data.
- status_t I2S_TxTransferSendDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle, i2s_transfer_t transfer)
  Begins or queue sending of the given data.
- void I2S_TransferAbortDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle)
  Aborts transfer of data.
- void I2S_RxTransferCreateHandleDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle, dma_handle_t ∗dmaHandle, i2s_dma_transfer_callback_t callback, void ∗userData)
  Initializes handle for reception of audio data.
- status_t I2S_RxTransferReceiveDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle, i2s_transfer_t transfer)
  Begins or queue reception of data into given buffer.
- void I2S_DMACallback (dma_handle_t ∗handle, void ∗userData, bool transferDone, uint32_t tcds)
  Invoked from DMA interrupt handler.
- void I2S_TransferInstallLoopDMADescriptorMemory (i2s_dma_handle_t ∗handle, void ∗dmaDescriptorAddr, size_t dmaDescriptorNum)
  Install DMA descriptor memory for loop transfer only.
- status_t I2S_TransferSendLoopDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle, i2s_transfer_t ∗xfer, uint32_t loopTransferCount)
  Send link transfer data using DMA.

- status_t I2S_TransferReceiveLoopDMA (I2S_Type *base, i2s_dma_handle_t *handle, i2s_transfer_t *xfer, uint32_t loopTransferCount)
  Receive link transfer data using DMA.

45.2 Data Structure Documentation

45.2.1 struct _i2s_dma_handle

Members not to be accessed / modified outside of the driver.

Data Fields

- uint32_t state
  Internal state of I2S DMA transfer.
- uint8_t bytesPerFrame
  bytes per frame
- i2s_dma_transfer_callback_t completionCallback
  Callback function pointer.
- void * userData
  Application data passed to callback.
- dma_handle_t * dmaHandle
  DMA handle.
- volatile i2s_transfer_t i2sQueue [I2S_NUM_BUFFERS]
  Transfer queue storing transfer buffers.
- volatile uint8_t queueUser
  Queue index where user's next transfer will be stored.
- volatile uint8_t queueDriver
  Queue index of buffer actually used by the driver.
- dma_descriptor_t * i2sLoopDMADescriptor
  descriptor pool pointer
- size_t i2sLoopDMADescriptorNum
  number of descriptor in descriptors pool

45.3 Macro Definition Documentation

45.3.1 #define FSL_I2S_DMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

45.4 Typedef Documentation

45.4.1 typedef void(* i2s_dma_transfer_callback_t)(I2S_Type *base, i2s_dma_handle_t *handle, status_t completionStatus, void *userData)

Parameters

base	I2S base pointer.
handle	pointer to I2S transaction.
completion-Status	status of the transaction.
userData	optional pointer to user arguments data.

45.5 Function Documentation

45.5.1 void I2S_TxTransferCreateHandleDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*, dma_handle_t * *dmaHandle*, i2s_dma_transfer_callback_t *callback*, void * *userData*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
dmaHandle	pointer to dma handle structure.
callback	function to be called back when transfer is done or fails.
userData	pointer to data passed to callback.

45.5.2 status_t I2S_TxTransferSendDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*, i2s_transfer_t *transfer*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
transfer	data buffer.

Return values

kStatus_Success	

kStatus_I2S_Busy	if all queue slots are occupied with unsent buffers.

45.5.3 void I2S_TransferAbortDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.

45.5.4 void I2S_RxTransferCreateHandleDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*, dma_handle_t * *dmaHandle*, i2s_dma_transfer_callback_t *callback*, void * *userData*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
dmaHandle	pointer to dma handle structure.
callback	function to be called back when transfer is done or fails.
userData	pointer to data passed to callback.

45.5.5 status_t I2S_RxTransferReceiveDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*, i2s_transfer_t *transfer*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
transfer	data buffer.

Return values

kStatus_Success	
kStatus_I2S_Busy	if all queue slots are occupied with buffers which are not full.

45.5.6 void I2S_DMACallback (dma_handle_t * *handle*, void * *userData*, bool *transferDone*, uint32_t *tcds*)

Parameters

handle	pointer to DMA handle structure.
userData	argument for user callback.
transferDone	if transfer was done.
tcds	

45.5.7 void I2S_TransferInstallLoopDMADescriptorMemory (i2s_dma_handle_t * *handle*, void * *dmaDescriptorAddr*, size_t *dmaDescriptorNum*)

This function used to register DMA descriptor memory for the i2s loop dma transfer.

It must be callbed before I2S_TransferSendLoopDMA/I2S_TransferReceiveLoopDMA and after I2S_RxTransferCreateHandleDMA/I2S_TxTransferCreateHandleDMA.

User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

Parameters

handle	Pointer to i2s DMA transfer handle.
dmaDescriptorAddr	DMA descriptor start address.
dmaDescriptorNum	DMA descriptor number.

45.5.8 status_t I2S_TransferSendLoopDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*, i2s_transfer_t * *xfer*, uint32_t *loopTransferCount*)

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

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S_InstallDMADescriptor-Memory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S_TransferAbortDMA to stop the loop transfer.

Parameters

base	I2S peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
xfer	I2S DMA transfer structure. See i2s_transfer_t.
loopTransfer-Count	loop count

Return values

kStatus_Success	

45.5.9 status_t I2S_TransferReceiveLoopDMA (I2S_Type * *base*, i2s_dma_-handle_t * *handle*, i2s_transfer_t * *xfer*, uint32_t *loopTransferCount*)

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

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S_InstallDMADescriptor-Memory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S_TransferAbortDMA to stop the loop transfer.

Parameters

base	I2S peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
xfer	I2S DMA transfer structure. See i2s_transfer_t.
loopTransfer-Count	loop count

Return values

kStatus_Success	

Chapter 46
Hashcrypt_driver

46.1 Overview

Macros

- #define HASHCRYPT_MODE_SHA1 0x1
  Algorithm definitions correspond with the values for Mode field in Control register !

Enumerations

- enum hashcrypt_algo_t {
  kHASHCRYPT_Sha1 = HASHCRYPT_MODE_SHA1,
  kHASHCRYPT_Sha256 = HASHCRYPT_MODE_SHA256,
  kHASHCRYPT_Aes = HASHCRYPT_MODE_AES }
  Algorithm used for Hashcrypt operation.

Functions

- void HASHCRYPT_Init (HASHCRYPT_Type ∗base)
  Enables clock and disables reset for HASHCRYPT peripheral.
- void HASHCRYPT_Deinit (HASHCRYPT_Type ∗base)
  Disables clock for HASHCRYPT peripheral.

Driver version

- #define FSL_HASHCRYPT_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))
  HASHCRYPT driver version.

46.2 Macro Definition Documentation

46.2.1 #define FSL_HASHCRYPT_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))

Version 2.2.3.

Current version: 2.2.3

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Support loading AES key from unaligned address
- Version 2.0.2
  - Support loading AES key from unaligned address for different compiler and core variants

- Version 2.0.3
  - Remove SHA512 and AES ICB algorithm definitions
- Version 2.0.4
  - Add SHA context switch support
- Version 2.1.0
  - Update the register name and macro to align with new header.
- Version 2.1.1
  - Fix MISRA C-2012.
- Version 2.1.2
  - Support loading AES input data from unaligned address.
- Version 2.1.3
  - Fix MISRA C-2012.
- Version 2.1.4
  - Fix context switch cannot work when switching from AES.
- Version 2.1.5
  - Add data synchronization barriere inside hashcrypt_sha_ldm_stm_16_words() to prevent possible optimization issue.
- Version 2.2.0
  - Add AES-OFB and AES-CFB mixed IP/SW modes.
- Version 2.2.1
  - Add data synchronization barrier inside hashcrypt_sha_ldm_stm_16_words() prevent compiler from reordering memory write when -O2 or higher is used.
- Version 2.2.2
  - Add data synchronization barrier inside hashcrypt_sha_ldm_stm_16_words() to fix optimization issue
- Version 2.2.3
  - Added check for size in hashcrypt_aes_one_block to prevent overflowing COUNT field in MEMCTRL register, if its bigger than COUNT field do a multiple runs.

46.3 Enumeration Type Documentation

46.3.1 enum hashcrypt_algo_t

Enumerator

kHASHCRYPT_Sha1 SHA_1.
kHASHCRYPT_Sha256 SHA_256.
kHASHCRYPT_Aes AES.

46.4 Function Documentation

46.4.1 void HASHCRYPT_Init (HASHCRYPT_Type * *base*)

Enable clock and disable reset for HASHCRYPT.

Parameters

base	HASHCRYPT base address

46.4.2 void HASHCRYPT_Deinit (HASHCRYPT_Type * *base*)

Disable clock and enable reset.

Parameters

base	HASHCRYPT base address

Chapter 47
Skboot_authenticate

47.1 Overview

Enumerations

- enum skboot_status_t {
  kStatus_SKBOOT_Success = 0x5ac3c35au,
  kStatus_SKBOOT_Fail = 0xc35ac35au,
  kStatus_SKBOOT_InvalidArgument = 0xc35a5ac3u,
  kStatus_SKBOOT_KeyStoreMarkerInvalid = 0xc3c35a5au,
  kStatus_SKBOOT_HashcryptFinishedWithStatusSuccess,
  kStatus_SKBOOT_HashcryptFinishedWithStatusFail,
  kStatus_SKBOOT_Success = 0x5ac3c35au,
  kStatus_SKBOOT_Fail = 0xc35ac35au,
  kStatus_SKBOOT_InvalidArgument = 0xc35a5ac3u,
  kStatus_SKBOOT_KeyStoreMarkerInvalid = 0xc3c35a5au }
  SKBOOT return status.
- enum secure_bool_t {
  kSECURE_TRUE = 0xc33cc33cU,
  kSECURE_FALSE = 0x5aa55aa5U,
  kSECURE_CALLPROTECT_SECURITY_FLAGS = 0xc33c5aa5U,
  kSECURE_CALLPROTECT_IS_APP_READY = 0x5aa5c33cU,
  kSECURE_TRACKER_VERIFIED = 0x55aacc33U,
  kSECURE_TRUE = 0xc33cc33cU,
  kSECURE_FALSE = 0x5aa55aa5U }
  Secure bool flag.

Functions

- skboot_status_t skboot_authenticate (const uint8_t *imageStartAddr, secure_bool_t *isSignVerified)
  Authenticate entry function with ARENA allocator init.
- void HASH_IRQHandler (void)
  Interface for image authentication API.

47.2 Enumeration Type Documentation

47.2.1 enum skboot_status_t

Enumerator

kStatus_SKBOOT_Success SKBOOT return success status.

kStatus_SKBOOT_Fail SKBOOT return fail status.
kStatus_SKBOOT_InvalidArgument SKBOOT return invalid argument status.
kStatus_SKBOOT_KeyStoreMarkerInvalid SKBOOT return Keystore invalid Marker status.
kStatus_SKBOOT_HashcryptFinishedWithStatusSuccess SKBOOT return Hashcrypt finished with the success status.
kStatus_SKBOOT_HashcryptFinishedWithStatusFail SKBOOT return Hashcrypt finished with the fail status.
kStatus_SKBOOT_Success PRINCE Success.
kStatus_SKBOOT_Fail PRINCE Fail.
kStatus_SKBOOT_InvalidArgument PRINCE Invalid argument.
kStatus_SKBOOT_KeyStoreMarkerInvalid PRINCE Invalid marker.

47.2.2 enum secure_bool_t

Enumerator

kSECURE_TRUE Secure true flag.
kSECURE_FALSE Secure false flag.
kSECURE_CALLPROTECT_SECURITY_FLAGS Secure call protect the security flag.
kSECURE_CALLPROTECT_IS_APP_READY Secure call protect the app is ready flag.
kSECURE_TRACKER_VERIFIED Secure tracker verified flag.
kSECURE_TRUE PRINCE true.
kSECURE_FALSE PRINCE false.

47.3 Function Documentation

47.3.1 skboot_status_t skboot_authenticate (const uint8_t * *imageStartAddr*, secure_bool_t * *isSignVerified*)

This is called by ROM boot or by ROM API g_skbootAuthenticateInterface

47.3.2 CODEC Adapter

47.3.2.1 Overview

Enumerations

- enum {
  kCODEC_WM8904,
  kCODEC_WM8960,
  kCODEC_WM8524,
  kCODEC_SGTL5000,
  kCODEC_DA7212,
  kCODEC_CS42888,
  kCODEC_CS42448,
  kCODEC_AK4497,
  kCODEC_AK4458,
  kCODEC_TFA9XXX,
  kCODEC_TFA9896 }
  codec type

47.3.2.2 Enumeration Type Documentation

47.3.2.2.1 anonymous enum

Enumerator

kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960
kCODEC_WM8524 wm8524
kCODEC_SGTL5000 sgtl5000
kCODEC_DA7212 da7212
kCODEC_CS42888 CS42888.
kCODEC_CS42448 CS42448.
kCODEC_AK4497 AK4497.
kCODEC_AK4458 ak4458
kCODEC_TFA9XXX tfa9xxx
kCODEC_TFA9896 tfa9896

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support