Document Number: MCUXSDKAPIRM
Rev 2.14.2
Jan 2024

MCUXpresso SDK API Reference Manual

NXP Semiconductors

Contents

Title

Section No. Page No.

Chapter 1
Introduction

The MCUXpresso Software Development Kit (MCUXpresso SDK) is a collection of software enablement for NXP Microcontrollers that includes peripheral drivers, multicore support and integrated RTOS support for FreeRTOS™ . 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.

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.

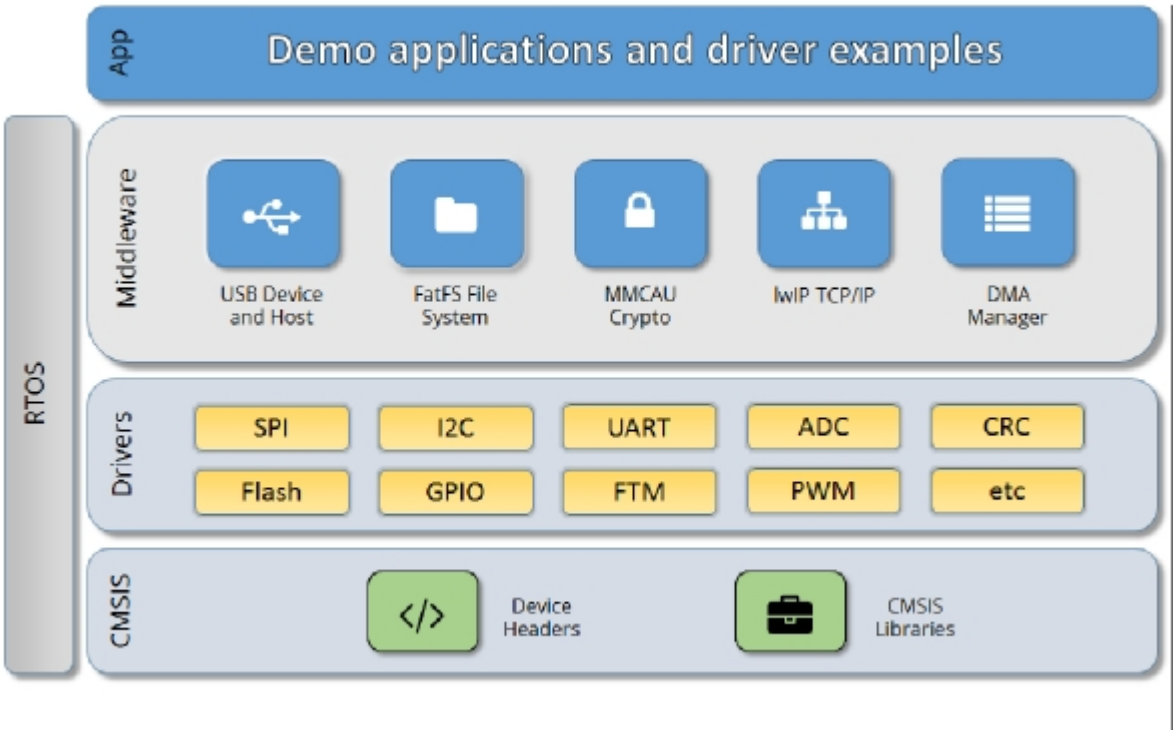
Chapter 3
Architectural Overview

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 configuration
- Clock mux and divider configuration
- Getting clock frequency

Files

- file fsl_clock.h

Data Structures

- struct firc_trim_config_t
  firc trim configuration. More...
- struct sirc_trim_config_t
  sirc trim configuration. More...

Macros

- #define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0U
  Configure whether driver controls clock.
- #define CLK_GATE_REG_OFFSET(value) (((uint32_t)(value)) >> 16U)
  Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.
- #define AOI_CLOCKS
  Clock ip name array for AOI.
- #define CRC_CLOCKS
  Clock ip name array for CRC.
- #define CTIMER_CLOCKS
  Clock ip name array for CTIMER.
- #define DMA_CLOCKS
  Clock ip name array for DMA.
- #define EDMA_CLOCKS
  Clock gate name array for EDMA.
- #define ERM_CLOCKS
  Clock ip name array for ERM.
- #define EIM_CLOCKS
  Clock ip name array for EIM.
- #define FREQME_CLOCKS
  Clock ip name array for FREQME.
- #define GPIO_CLOCKS

Clock ip name array for GPIO.

- #define I3C_CLOCKS
  Clock ip name array for I3C.
- #define INPUTMUX_CLOCKS
  Clock ip name array for INPUTMUX.
- #define LPCMP_CLOCKS
  Clock ip name array for GPIO.
- #define LPADC_CLOCKS
  Clock ip name array for LPADC.
- #define LPUART_CLOCKS
  Clock ip name array for LPUART.
- #define LPI2C_CLOCKS
  Clock ip name array for LPI2C.
- #define LPSPI_CLOCKS
  Clock ip name array for LSPI.
- #define MTR_CLOCKS
  Clock ip name array for MTR.
- #define OSTIMER_CLOCKS
  Clock ip name array for OSTIMER.
- #define PWM_CLOCKS
  Clock ip name array for PWM.
- #define QDC_CLOCKS
  Clock ip name array for QDC.
- #define UTICK_CLOCKS
  Clock ip name array for UTICK.
- #define WWDT_CLOCKS
  Clock ip name array for WWDT.
- #define BUS_CLK kCLOCK_BusClk
  Peripherals clock source definition.
- #define CLK_ATTACH_REG_OFFSET(value) (((uint32_t)(value)) >> 16U)
  Clock Mux Switches The encoding is as follows each connection identified is 32bits wide while 24bits are valuable starting from LSB upwards.

Enumerations

- enum clock_ip_name_t {
  kCLOCK_GateINPUTMUX0 = (0x00000U | (0U)),
  kCLOCK_InputMux = (0x00000U | (0U)),
  kCLOCK_GateI3C0 = (0x00000U | (1U)),
  kCLOCK_GateCTIMER0 = (0x00000U | (2U)),
  kCLOCK_GateCTIMER1 = (0x00000U | (3U)),
  kCLOCK_GateCTIMER2 = (0x00000U | (4U)),
  kCLOCK_GateFREQME = (0x00000U | (5U)),
  kCLOCK_GateUTICK0 = (0x00000U | (6U)),
  kCLOCK_GateWWDT0 = (0x00000U | (7U)),
  kCLOCK_GateDMA = (0x00000U | (8U)),
  kCLOCK_GateAOI0 = (0x00000U | (9U)),
  kCLOCK_GateCRC = (0x00000U | (10U)),
  kCLOCK_Crc0 = (0x00000U | (10U)),
  kCLOCK_GateEIM = (0x00000U | (11U)),
  kCLOCK_GateERM = (0x00000U | (12U)),
  kCLOCK_GateLPI2C0 = (0x00000U | (16U)),
  kCLOCK_GateLPSPI0 = (0x00000U | (17U)),
  kCLOCK_GateLPSPI1 = (0x00000U | (18U)),
  kCLOCK_GateLPUART0 = (0x00000U | (19U)),
  kCLOCK_GateLPUART1 = (0x00000U | (20U)),
  kCLOCK_GateLPUART2 = (0x00000U | (21U)),
  kCLOCK_GateUSB0 = (0x00000U | (22U)),
  kCLOCK_GateQDC0 = (0x00000U | (23U)),
  kCLOCK_GateFLEXPWM0 = (0x00000U | (24U)),
  kCLOCK_GateOSTIMER0 = (0x00000U | (25U)),
  kCLOCK_GateADC0 = (0x00000U | (26U)),
  kCLOCK_GateCMP0 = (0x00000U | (27U)),
  kCLOCK_GateCMP1 = (0x00000U | (28U)),
  kCLOCK_GatePORT0 = (0x00000U | (29U)),
  kCLOCK_GatePORT1 = (0x00000U | (30U)),
  kCLOCK_GatePORT2 = (0x00000U | (31U)),
  kCLOCK_GatePORT3 = ((0x10U << 16U) | (0U)),
  kCLOCK_GateATX0 = ((0x10U << 16U) | (1U)),
  kCLOCK_GateMTR = ((0x10U << 16U) | (2U)),
  kCLOCK_GateTCU = ((0x10U << 16U) | (3U)),
  kCLOCK_GateEZRAMC_RAMA = ((0x10U << 16U) | (4U)),
  kCLOCK_GateGPIO0 = ((0x10U << 16U) | (5U)),
  kCLOCK_GateGPIO1 = ((0x10U << 16U) | (6U)),
  kCLOCK_GateGPIO2 = ((0x10U << 16U) | (7U)),
  kCLOCK_GateGPIO3 = ((0x10U << 16U) | (8U)),
  kCLOCK_GateROMCP = ((0x10U << 16U) | (9U)),
  kCLOCK_GatePWMSM0 = ((REG_PWM0SUBCTL << 16U) | (0U)),
  kCLOCK_GatePWMSM1 = ((REG_PWM0SUBCTL << 16U) | (1U)),
  kCLOCK_GatePWMSM2 = ((REG_PWM0SUBCTL << 16U) | (2U)),

kCLOCK_GateNotAvail = (0xFFFFFFFFU) }
Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

- enum clock_name_t {
  kCLOCK_MainClk,
  kCLOCK_CoreSysClk,
  kCLOCK_SYSTEM_CLK,
  kCLOCK_BusClk,
  kCLOCK_ExtClk,
  kCLOCK_FroHf,
  kCLOCK_FroHfDiv,
  kCLOCK_Clk48M,
  kCLOCK_Fro12M,
  kCLOCK_Clk1M,
  kCLOCK_Fro16K,
  kCLOCK_Clk16K0,
  kCLOCK_Clk16K1,
  kCLOCK_SLOW_CLK }
  Clock name used to get clock frequency.
- enum clock_select_name_t {
  kCLOCK_SelI3C0_FCLK = (0x0A0U),
  kCLOCK_SelCTIMER0 = (0x0A8U),
  kCLOCK_SelCTIMER1 = (0x0B0U),
  kCLOCK_SelCTIMER2 = (0x0B8U),
  kCLOCK_SelLPI2C0 = (0x0C8U),
  kCLOCK_SelLPSPI0 = (0x0D0U),
  kCLOCK_SelLPSPI1 = (0x0D8U),
  kCLOCK_SelLPUART0 = (0x0E0U),
  kCLOCK_SelLPUART1 = (0x0E8U),
  kCLOCK_SelLPUART2 = (0x0F0U),
  kCLOCK_SelUSB0 = (0x0F8U),
  kCLOCK_SelLPTMR0 = (0x100U),
  kCLOCK_SelOSTIMER0 = (0x108U),
  kCLOCK_SelADC0 = (0x110U),
  kCLOCK_SelCMP0_RR = (0x120U),
  kCLOCK_SelCMP1_RR = (0x130U),
  kCLOCK_SelTRACE = (0x138U),
  kCLOCK_SelCLKOUT = (0x140U),
  kCLOCK_SelSYSTICK = (0x148U),
  kCLOCK_SelSCGSCS = (0x200U),
  kCLOCK_SelMax = (0x200U) }
  Clock name used to get clock frequency.
- enum clock_attach_id_t {

kCLK_IN_to_MAIN_CLK = CLK_ATTACH_MUX(kCLOCK_SelSCGSCS, 1U),
kFRO12M_to_MAIN_CLK = CLK_ATTACH_MUX(kCLOCK_SelSCGSCS, 2U),
kFRO_HF_to_MAIN_CLK = CLK_ATTACH_MUX(kCLOCK_SelSCGSCS, 3U),
kCLK_16K_to_MAIN_CLK = CLK_ATTACH_MUX(kCLOCK_SelSCGSCS, 4U),
kNONE_to_MAIN_CLK = CLK_ATTACH_MUX(kCLOCK_SelSCGSCS, 7U),
kFRO12M_to_I3C0FCLK = CLK_ATTACH_MUX(kCLOCK_SelI3C0_FCLK, 0U),
kFRO_HF_DIV_to_I3C0FCLK = CLK_ATTACH_MUX(kCLOCK_SelI3C0_FCLK, 2U),
kCLK_IN_to_I3C0FCLK = CLK_ATTACH_MUX(kCLOCK_SelI3C0_FCLK, 3U),
kCLK_1M_to_I3C0FCLK = CLK_ATTACH_MUX(kCLOCK_SelI3C0_FCLK, 5U),
kNONE_to_I3C0FCLK = CLK_ATTACH_MUX(kCLOCK_SelI3C0_FCLK, 7U),
kFRO12M_to_CTIMER0 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER0, 0U),
kFRO_HF_to_CTIMER0 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER0, 1U),
kCLK_IN_to_CTIMER0 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER0, 3U),
kCLK_16K_to_CTIMER0 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER0, 4U),
kCLK_1M_to_CTIMER0 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER0, 5U),
kNONE_to_CTIMER0 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER0, 7U),
kFRO12M_to_CTIMER1 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER1, 0U),
kFRO_HF_to_CTIMER1 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER1, 1U),
kCLK_IN_to_CTIMER1 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER1, 3U),
kCLK_16K_to_CTIMER1 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER1, 4U),
kCLK_1M_to_CTIMER1 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER1, 5U),
kNONE_to_CTIMER1 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER1, 7U),
kFRO12M_to_CTIMER2 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER2, 0U),
kFRO_HF_to_CTIMER2 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER2, 1U),
kCLK_IN_to_CTIMER2 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER2, 3U),
kCLK_16K_to_CTIMER2 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER2, 4U),
kCLK_1M_to_CTIMER2 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER2, 5U),
kNONE_to_CTIMER2 = CLK_ATTACH_MUX(kCLOCK_SelCTIMER2, 7U),
kFRO12M_to_LPI2C0 = CLK_ATTACH_MUX(kCLOCK_SelLPI2C0, 0U),
kFRO_HF_DIV_to_LPI2C0 = CLK_ATTACH_MUX(kCLOCK_SelLPI2C0, 2U),
kCLK_IN_to_LPI2C0 = CLK_ATTACH_MUX(kCLOCK_SelLPI2C0, 3U),
kCLK_1M_to_LPI2C0 = CLK_ATTACH_MUX(kCLOCK_SelLPI2C0, 5U),
kNONE_to_LPI2C0 = CLK_ATTACH_MUX(kCLOCK_SelLPI2C0, 7U),
kFRO12M_to_LPSPI0 = CLK_ATTACH_MUX(kCLOCK_SelLPSPI0, 0U),
kFRO_HF_DIV_to_LPSPI0 = CLK_ATTACH_MUX(kCLOCK_SelLPSPI0, 2U),
kCLK_IN_to_LPSPI0 = CLK_ATTACH_MUX(kCLOCK_SelLPSPI0, 3U),
kCLK_1M_to_LPSPI0 = CLK_ATTACH_MUX(kCLOCK_SelLPSPI0, 5U),
kNONE_to_LPSPI0 = CLK_ATTACH_MUX(kCLOCK_SelLPSPI0, 7U),
kFRO12M_to_LPSPI1 = CLK_ATTACH_MUX(kCLOCK_SelLPSPI1, 0U),
kFRO_HF_DIV_to_LPSPI1 = CLK_ATTACH_MUX(kCLOCK_SelLPSPI1, 2U),
kCLK_IN_to_LPSPI1 = CLK_ATTACH_MUX(kCLOCK_SelLPSPI1, 3U),
kCLK_1M_to_LPSPI1 = CLK_ATTACH_MUX(kCLOCK_SelLPSPI1, 5U),
kNONE_to_LPSPI1 = CLK_ATTACH_MUX(kCLOCK_SelLPSPI1, 7U),
kFRO12M_to_LPUART0 = CLK_ATTACH_MUX(kCLOCK_SelLPUART0, 0U),
kFRO_HF_DIV_to_LPUART0 = CLK_ATTACH_MUX(kCLOCK_SelLPUART0, 2U),
kCLK_IN_to_LPUART0 = CLK_ATTACH_MUX(kCLOCK_SelLPUART0, 3U),
kCLK_16K_to_LPUART0 = CLK_ATTACH_MUX(kCLOCK_SelLPUART0, 4U),
kCLK_1M_to_LPUART0 = CLK_ATTACH_MUX(kCLOCK_SelLPUART0, 5U),
kNONE_to_LPUART0 = CLK_ATTACH_MUX(kCLOCK_SelLPUART0, 7U),

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

- enum clock_div_name_t {
kCLOCK_DivI3C0_FCLK = (0x0A4U),
kCLOCK_DivCTIMER0 = (0x0ACU),
kCLOCK_DivCTIMER1 = (0x0B4U),
kCLOCK_DivCTIMER2 = (0x0BCU),
kCLOCK_DivWWDT0 = (0x0C4U),
kCLOCK_DivLPI2C0 = (0x0CCU),
kCLOCK_DivLPSPI0 = (0x0D4U),
kCLOCK_DivLPSPI1 = (0x0DCU),
kCLOCK_DivLPUART0 = (0x0E4U),
kCLOCK_DivLPUART1 = (0x0ECU),
kCLOCK_DivLPUART2 = (0x0F4U),
kCLOCK_DivLPTMR0 = (0x104U),
kCLOCK_DivADC0 = (0x114U),
kCLOCK_DivCMP0_FUNC = (0x11CU),
kCLOCK_DivCMP0_RR = (0x124U),
kCLOCK_DivCMP1_FUNC = (0x12CU),
kCLOCK_DivCMP1_RR = (0x134U),
kCLOCK_DivTRACE = (0x13CU),
kCLOCK_DivCLKOUT = (0x144U),
kCLOCK_DivSYSTICK = (0x14CU),
kCLOCK_DivFRO_HF_DIV = (0x154U),
kCLOCK_DivSLOWCLK = (0x378U),
kCLOCK_DivAHBCLK = (0x380U),
kCLOCK_DivMax = (0x380U) }
Clock dividers.
- enum firc_trim_mode_t {
kSCG_FircTrimNonUpdate = SCG_FIRCCSR_FIRCTREN_MASK,
kSCG_FircTrimUpdate = SCG_FIRCCSR_FIRCTREN_MASK | SCG_FIRCCSR_FIRCTRUP_-MASK }
firc trim mode.
- enum firc_trim_src_t {
kSCG_FircTrimSrcUsb0 = 0U,
kSCG_FircTrimSrcSysOsc = 2U }
firc trim source.
- enum sirc_trim_mode_t {
kSCG_SircTrimNonUpdate = SCG_SIRCCSR_SIRCTREN_MASK,
kSCG_SircTrimUpdate = SCG_SIRCCSR_SIRCTREN_MASK | SCG_SIRCCSR_SIRCTRUP_-MASK }
sirc trim mode.
- enum sirc_trim_src_t {
kNoTrimSrc = 0,
kSCG_SircTrimSrcSysOsc = 2U }
sirc trim source.

- enum scg_sosc_monitor_mode_t {
  kSCG_SysOscMonitorDisable = 0U,
  kSCG_SysOscMonitorInt = SCG_SOSCCSR_SOSCCM_MASK,
  kSCG_SysOscMonitorReset }
  SCG system OSC monitor mode.
- enum clke_16k_t {
  kCLKE_16K_SYSTEM = VBAT_FROCLKE_CLKE(1U),
  kCLKE_16K_COREMAIN = VBAT_FROCLKE_CLKE(2U) }
  firc trim source.

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.
- void CLOCK_AttachClk (clock_attach_id_t connection)
  Configure the clock selection muxes.
- clock_attach_id_t CLOCK_GetClockAttachId (clock_attach_id_t connection)
  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_SetClockSelect (clock_select_name_t sel_name, uint32_t value)
  Set the clock select value. This fuction set the peripheral clock select value.
- uint32_t CLOCK_GetClockSelect (clock_select_name_t sel_name)
  Get the clock select value. This fuction get the peripheral clock select value.
- void CLOCK_SetClockDiv (clock_div_name_t div_name, uint32_t value)
  Setup peripheral clock dividers.
- uint32_t CLOCK_GetClockDiv (clock_div_name_t div_name)
  Get peripheral clock dividers.
- void CLOCK_HaltClockDiv (clock_div_name_t div_name)
  Halt peripheral clock dividers.
- status_t CLOCK_SetupFROHFClocking (uint32_t iFreq)
  Initialize the FROHF to given frequency (48,64,96,192). This function turns on FIRC and select the given frequency as the source of fro_hf.
- status_t CLOCK_SetupFRO12MClocking (void)
  Initialize the FRO12M. This function turns on FRO12M.
- status_t CLOCK_SetupFRO16KClocking (uint8_t clk_16k_enable_mask)
  Initialize the FRO16K. This function turns on FRO16K.
- status_t CLOCK_SetupExtClocking (uint32_t iFreq)
  Initialize the external osc clock to given frequency.
- status_t CLOCK_SetupExtRefClocking (uint32_t iFreq)
  Initialize the external reference clock to given frequency.
- uint32_t CLOCK_GetFreq (clock_name_t clockName)
  Return Frequency of selected clock.
- uint32_t CLOCK_GetCoreSysClkFreq (void)
  Return Frequency of core.
- uint32_t CLOCK_GetI3CFClkFreq (void)
  Return Frequency of I3C FCLK.
- uint32_t CLOCK_GetCTimerClkFreq (uint32_t id)
  Return Frequency of CTimer functional Clock.

- uint32_t CLOCK_GetLpi2cClkFreq (void)
  Return Frequency of LPI2C0 functional Clock.
- uint32_t CLOCK_GetLpspiClkFreq (uint32_t id)
  Return Frequency of LPSPI functional Clock.
- uint32_t CLOCK_GetLpuartClkFreq (uint32_t id)
  Return Frequency of LPUART functional Clock.
- uint32_t CLOCK_GetLptmrClkFreq (void)
  Return Frequency of LPTMR functional Clock.
- uint32_t CLOCK_GetOstimerClkFreq (void)
  Return Frequency of OSTIMER.
- uint32_t CLOCK_GetAdcClkFreq (void)
  Return Frequency of Adc Clock.
- uint32_t CLOCK_GetCmpFClkFreq (uint32_t id)
  Return Frequency of CMP Function Clock.
- uint32_t CLOCK_GetCmpRRClkFreq (uint32_t id)
  Return Frequency of CMP Round Robin Clock.
- uint32_t CLOCK_GetTraceClkFreq (void)
  Return Frequency of Trace Clock.
- uint32_t CLOCK_GetClkoutClkFreq (void)
  Return Frequency of CLKOUT Clock.
- uint32_t CLOCK_GetSystickClkFreq (void)
  Return Frequency of Systick Clock.
- uint32_t CLOCK_GetWwdtClkFreq (void)
  brief Return Frequency of Systick Clock return Frequency of Systick.
- status_t CLOCK_FROHFTrimConfig (firc_trim_config_t config)
  Setup FROHF trim.
- status_t CLOCK_FRO12MTrimConfig (sirc_trim_config_t config)
  Setup FRO 12M trim.
- void CLOCK_SetSysOscMonitorMode (scg_sosc_monitor_mode_t mode)
  Sets the system OSC monitor mode.
- bool CLOCK_EnableUsbfsClock (void)
  brief Enable USB FS clock.

Driver version

- #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))
  CLOCK driver version 1.0.0.

4.2 Data Structure Documentation

4.2.1 struct firc_trim_config_t

Data Fields

- firc_trim_mode_t trimMode
  Trim mode.
- firc_trim_src_t trimSrc
  Trim source.
- uint16_t trimDiv
  Divider of SOSC.

- uint8_t trimCoar
  Trim coarse value; Irrelevant if trimMode is kSCG_TrimUpdate.
- uint8_t trimFine
  Trim fine value; Irrelevant if trimMode is kSCG_TrimUpdate.

Field Documentation

(1) firc_trim_mode_t firc_trim_config_t::trimMode

(2) firc_trim_src_t firc_trim_config_t::trimSrc

(3) uint16_t firc_trim_config_t::trimDiv

(4) uint8_t firc_trim_config_t::trimCoar

(5) uint8_t firc_trim_config_t::trimFine

4.2.2 struct sirc_trim_config_t

Data Fields

- sirc_trim_mode_t trimMode
  Trim mode.
- sirc_trim_src_t trimSrc
  Trim source.
- uint16_t trimDiv
  Divider of SOSC.
- uint8_t cltrim
  Trim coarse value; Irrelevant if trimMode is kSCG_TrimUpdate.
- uint8_t ccotrim
  Trim fine value; Irrelevant if trimMode is kSCG_TrimUpdate.

Field Documentation

(1) sirc_trim_mode_t sirc_trim_config_t::trimMode

(2) sirc_trim_src_t sirc_trim_config_t::trimSrc

(3) uint16_t sirc_trim_config_t::trimDiv

(4) uint8_t sirc_trim_config_t::cltrim

(5) uint8_t sirc_trim_config_t::ccotrim

4.3 Macro Definition Documentation

4.3.1 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))

4.3.2 #define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0U

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 CLK_GATE_REG_OFFSET(*value*) (((uint32_t)(value)) >> 16U)

4.3.4 #define AOI_CLOCKS

Value:

```
{                   \
        kCLOCK_GateAOI0 \
    }
```

4.3.5 #define CRC_CLOCKS

Value:

```
{                   \
        kCLOCK_GateCRC \
    }
```

4.3.6 #define CTIMER_CLOCKS

Value:

```
{                                                       \
        kCLOCK_GateCTIMER0, kCLOCK_GateCTIMER1,
      kCLOCK_GateCTIMER2 \
    }
```

4.3.7 #define DMA_CLOCKS

Value:

```
{                   \
        kCLOCK_GateDMA \
    }
```

4.3.8 #define EDMA_CLOCKS

Value:

```
{                   \
        kCLOCK_GateDMA \
    }
```

4.3.9 #define ERM_CLOCKS

Value:

```
{                   \
        kCLOCK_GateERM \
    }
```

4.3.10 #define EIM_CLOCKS

Value:

```
{                   \
        kCLOCK_GateEIM \
    }
```

4.3.11 #define FREQME_CLOCKS

Value:

```
{                     \
        kCLOCK_GateFREQME \
    }
```

4.3.12 #define GPIO_CLOCKS

Value:

```
{                                                                       \
        kCLOCK_GateGPIO0, kCLOCK_GateGPIO1,
      kCLOCK_GateGPIO2, kCLOCK_GateGPIO3 \
    }
```

4.3.13 #define INPUTMUX_CLOCKS

Value:

```
{                         \
        kCLOCK_GateINPUTMUX0 \
    }
```

4.3.14 #define LPCMP_CLOCKS

Value:

```
{                                   \
        kCLOCK_GateCMP0, kCLOCK_GateCMP1 \
    }
```

4.3.15 #define LPADC_CLOCKS

Value:

```
{                     \
        kCLOCK_GateADC0 \
    }
```

4.3.16 #define LPUART_CLOCKS

Value:

```
{                                                       \
        kCLOCK_GateLPUART0, kCLOCK_GateLPUART1,
      kCLOCK_GateLPUART2 \
    }
```

4.3.17 #define LPI2C_CLOCKS

Value:

```
{                       \
        kCLOCK_GateLPI2C0 \
    }
```

4.3.18 #define LPSPI_CLOCKS

Value:

```
{                                       \
        kCLOCK_GateLPSPI0, kCLOCK_GateLPSPI1 \
    }
```

4.3.19 #define MTR_CLOCKS

Value:

```
{                   \
        kCLOCK_GateMTR \
    }
```

4.3.20 #define OSTIMER_CLOCKS

Value:

```
{                       \
        kCLOCK_GateOSTIMER0 \
    }
```

4.3.21 #define PWM_CLOCKS

Value:

```
{                                                                   \
        {                                                               \
            kCLOCK_GatePWMSM0, kCLOCK_GatePWMSM1,
      kCLOCK_GatePWMSM2 \
        }                                                               \
    }
```

4.3.22 #define QDC_CLOCKS

Value:

```
{                   \
        kCLOCK_GateQDC0 \
    }
```

4.3.23 #define UTICK_CLOCKS

Value:

```
{                    \
        kCLOCK_GateUTICK0 \
    }
```

4.3.24 #define WWDT_CLOCKS

Value:

```
{                   \
        kCLOCK_GateWWDT0 \
    }
```

4.3.25 #define BUS_CLK kCLOCK_BusClk

4.3.26 #define CLK_ATTACH_REG_OFFSET(*value*) (((uint32_t)(value)) >> 16U)

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

4.4 Enumeration Type Documentation

4.4.1 enum clock_ip_name_t

Enumerator

kCLOCK_GateINPUTMUX0 Clock gate name: INPUTMUX0.
kCLOCK_InputMux Clock gate name: INPUTMUX0.
kCLOCK_GateI3C0 Clock gate name: I3C0.
kCLOCK_GateCTIMER0 Clock gate name: CTIMER0.
kCLOCK_GateCTIMER1 Clock gate name: CTIMER1.
kCLOCK_GateCTIMER2 Clock gate name: CTIMER2.

kCLOCK_GateFREQME Clock gate name: FREQME.
kCLOCK_GateUTICK0 Clock gate name: UTICK0.
kCLOCK_GateWWDT0 Clock gate name: WWDT0.
kCLOCK_GateDMA Clock gate name: DMA.
kCLOCK_GateAOI0 Clock gate name: AOI0.
kCLOCK_GateCRC Clock gate name: CRC.
kCLOCK_Crc0 Clock gate name: CRC.
kCLOCK_GateEIM Clock gate name: EIM.
kCLOCK_GateERM Clock gate name: ERM.
kCLOCK_GateLPI2C0 Clock gate name: LPI2C0.
kCLOCK_GateLPSPI0 Clock gate name: LPSPI0.
kCLOCK_GateLPSPI1 Clock gate name: LPSPI1.
kCLOCK_GateLPUART0 Clock gate name: LPUART0.
kCLOCK_GateLPUART1 Clock gate name: LPUART1.
kCLOCK_GateLPUART2 Clock gate name: LPUART2.
kCLOCK_GateUSB0 Clock gate name: USB0.
kCLOCK_GateQDC0 Clock gate name: QDC0.
kCLOCK_GateFLEXPWM0 Clock gate name: FLEXPWM0.
kCLOCK_GateOSTIMER0 Clock gate name: OSTIMER0.
kCLOCK_GateADC0 Clock gate name: ADC0.
kCLOCK_GateCMP0 Clock gate name: CMP0.
kCLOCK_GateCMP1 Clock gate name: CMP1.
kCLOCK_GatePORT0 Clock gate name: PORT0.
kCLOCK_GatePORT1 Clock gate name: PORT1.
kCLOCK_GatePORT2 Clock gate name: PORT2.
kCLOCK_GatePORT3 Clock gate name: PORT3.
kCLOCK_GateATX0 Clock gate name: ATX0.
kCLOCK_GateMTR Clock gate name: MTR.
kCLOCK_GateTCU Clock gate name: TCU.
kCLOCK_GateEZRAMC_RAMA Clock gate name: EZRAMC_RAMA.
kCLOCK_GateGPIO0 Clock gate name: GPIO0.
kCLOCK_GateGPIO1 Clock gate name: GPIO1.
kCLOCK_GateGPIO2 Clock gate name: GPIO2.
kCLOCK_GateGPIO3 Clock gate name: GPIO3.
kCLOCK_GateROMCP Clock gate name: ROMCP.
kCLOCK_GatePWMSM0 Clock gate name: FlexPWM SM0.
kCLOCK_GatePWMSM1 Clock gate name: FlexPWM SM1.
kCLOCK_GatePWMSM2 Clock gate name: FlexPWM SM2.
kCLOCK_GateNotAvail Clock gate name: None

4.4.2 enum clock_name_t

Enumerator

kCLOCK_MainClk MAIN_CLK.
kCLOCK_CoreSysClk Core/system clock(CPU_CLK)
kCLOCK_SYSTEM_CLK AHB clock.
kCLOCK_BusClk Bus clock (AHB clock)
kCLOCK_ExtClk External Clock.
kCLOCK_FroHf FRO192.
kCLOCK_FroHfDiv Divided by FRO192.
kCLOCK_Clk48M CLK48M.
kCLOCK_Fro12M FRO12M.
kCLOCK_Clk1M CLK1M.
kCLOCK_Fro16K FRO16K.
kCLOCK_Clk16K0 CLK16K[0].
kCLOCK_Clk16K1 CLK16K[1].
kCLOCK_SLOW_CLK SYSTEM_CLK divided by 4.

4.4.3 enum clock_select_name_t

Enumerator

kCLOCK_SelI3C0_FCLK I3C0_FCLK clock selection.
kCLOCK_SelCTIMER0 CTIMER0 clock selection.
kCLOCK_SelCTIMER1 CTIMER1 clock selection.
kCLOCK_SelCTIMER2 CTIMER2 clock selection.
kCLOCK_SelLPI2C0 LPI2C0 clock selection.
kCLOCK_SelLPSPI0 LPSPI0 clock selection.
kCLOCK_SelLPSPI1 LPSPI1 clock selection.
kCLOCK_SelLPUART0 LPUART0 clock selection.
kCLOCK_SelLPUART1 LPUART1 clock selection.
kCLOCK_SelLPUART2 LPUART2 clock selection.
kCLOCK_SelUSB0 USB0 clock selection.
kCLOCK_SelLPTMR0 LPTMR0 clock selection.
kCLOCK_SelOSTIMER0 OSTIMER0 clock selection.
kCLOCK_SelADC0 ADC0 clock selection.
kCLOCK_SelCMP0_RR CMP0_RR clock selection.
kCLOCK_SelCMP1_RR CMP1_RR clock selection.
kCLOCK_SelTRACE TRACE clock selection.
kCLOCK_SelCLKOUT CLKOUT clock selection.
kCLOCK_SelSYSTICK SYSTICK clock selection.
kCLOCK_SelSCGSCS SCG SCS clock selection.
kCLOCK_SelMax MAX clock selection.

4.4.4 enum clock_attach_id_t

Enumerator

kCLK_IN_to_MAIN_CLK Attach clk_in to MAIN_CLK.
kFRO12M_to_MAIN_CLK Attach FRO_12M to MAIN_CLK.
kFRO_HF_to_MAIN_CLK Attach FRO_HF to MAIN_CLK.
kCLK_16K_to_MAIN_CLK Attach CLK_16K[1] to MAIN_CLK.
kNONE_to_MAIN_CLK Attach NONE to MAIN_CLK.
kFRO12M_to_I3C0FCLK Attach FRO12M to I3C0FCLK.
kFRO_HF_DIV_to_I3C0FCLK Attach FRO_HF_DIV to I3C0FCLK.
kCLK_IN_to_I3C0FCLK Attach CLK_IN to I3C0FCLK.
kCLK_1M_to_I3C0FCLK Attach CLK_1M to I3C0FCLK.
kNONE_to_I3C0FCLK Attach NONE to I3C0FCLK.
kFRO12M_to_CTIMER0 Attach FRO12M to CTIMER0.
kFRO_HF_to_CTIMER0 Attach FRO_HF to CTIMER0.
kCLK_IN_to_CTIMER0 Attach CLK_IN to CTIMER0.
kCLK_16K_to_CTIMER0 Attach CLK_16K to CTIMER0.
kCLK_1M_to_CTIMER0 Attach CLK_1M to CTIMER0.
kNONE_to_CTIMER0 Attach NONE to CTIMER0.
kFRO12M_to_CTIMER1 Attach FRO12M to CTIMER1.
kFRO_HF_to_CTIMER1 Attach FRO_HF to CTIMER1.
kCLK_IN_to_CTIMER1 Attach CLK_IN to CTIMER1.
kCLK_16K_to_CTIMER1 Attach CLK_16K to CTIMER1.
kCLK_1M_to_CTIMER1 Attach CLK_1M to CTIMER1.
kNONE_to_CTIMER1 Attach NONE to CTIMER1.
kFRO12M_to_CTIMER2 Attach FRO12M to CTIMER2.
kFRO_HF_to_CTIMER2 Attach FRO_HF to CTIMER2.
kCLK_IN_to_CTIMER2 Attach CLK_IN to CTIMER2.
kCLK_16K_to_CTIMER2 Attach CLK_16K to CTIMER2.
kCLK_1M_to_CTIMER2 Attach CLK_1M to CTIMER2.
kNONE_to_CTIMER2 Attach NONE to CTIMER2.
kFRO12M_to_LPI2C0 Attach FRO12M to LPI2C0.
kFRO_HF_DIV_to_LPI2C0 Attach FRO_HF_DIV to LPI2C0.
kCLK_IN_to_LPI2C0 Attach CLK_IN to LPI2C0.
kCLK_1M_to_LPI2C0 Attach CLK_1M to LPI2C0.
kNONE_to_LPI2C0 Attach NONE to LPI2C0.
kFRO12M_to_LPSPI0 Attach FRO12M to LPSPI0.
kFRO_HF_DIV_to_LPSPI0 Attach FRO_HF_DIV to LPSPI0.
kCLK_IN_to_LPSPI0 Attach CLK_IN to LPSPI0.
kCLK_1M_to_LPSPI0 Attach CLK_1M to LPSPI0.
kNONE_to_LPSPI0 Attach NONE to LPSPI0.
kFRO12M_to_LPSPI1 Attach FRO12M to LPSPI1.
kFRO_HF_DIV_to_LPSPI1 Attach FRO_HF_DIV to LPSPI1.
kCLK_IN_to_LPSPI1 Attach CLK_IN to LPSPI1.

kCLK_1M_to_LPSPI1 Attach CLK_1M to LPSPI1.
kNONE_to_LPSPI1 Attach NONE to LPSPI1.
kFRO12M_to_LPUART0 Attach FRO12M to LPUART0.
kFRO_HF_DIV_to_LPUART0 Attach FRO_HF_DIV to LPUART0.
kCLK_IN_to_LPUART0 Attach CLK_IN to LPUART0.
kCLK_16K_to_LPUART0 Attach CLK_16K to LPUART0.
kCLK_1M_to_LPUART0 Attach CLK_1M to LPUART0.
kNONE_to_LPUART0 Attach NONE to LPUART0.
kFRO12M_to_LPUART1 Attach FRO12M to LPUART1.
kFRO_HF_DIV_to_LPUART1 Attach FRO_HF_DIV to LPUART1.
kCLK_IN_to_LPUART1 Attach CLK_IN to LPUART1.
kCLK_16K_to_LPUART1 Attach CLK_16K to LPUART1.
kCLK_1M_to_LPUART1 Attach CLK_1M to LPUART1.
kNONE_to_LPUART1 Attach NONE to LPUART1.
kFRO12M_to_LPUART2 Attach FRO12M to LPUART2.
kFRO_HF_DIV_to_LPUART2 Attach FRO_HF_DIV to LPUART2.
kCLK_IN_to_LPUART2 Attach CLK_IN to LPUART2.
kCLK_16K_to_LPUART2 Attach CLK_16K to LPUART2.
kCLK_1M_to_LPUART2 Attach CLK_1M to LPUART2.
kNONE_to_LPUART2 Attach NONE to LPUART2.
kCLK_48M_to_USB0 Attach FRO12M to USB0.
kCLK_IN_to_USB0 Attach CLK_IN to USB0.
kNONE_to_USB0 Attach NONE to USB0.
kFRO12M_to_LPTMR0 Attach FRO12M to LPTMR0.
kFRO_HF_DIV_to_LPTMR0 Attach FRO_HF_DIV to LPTMR0.
kCLK_IN_to_LPTMR0 Attach CLK_IN to LPTMR0.
kCLK_1M_to_LPTMR0 Attach CLK_1M to LPTMR0.
kNONE_to_LPTMR0 Attach NONE to LPTMR0.
kCLK_16K_to_OSTIMER Attach FRO16K to OSTIMER0.
kCLK_1M_to_OSTIMER Attach CLK_1M to OSTIMER0.
kNONE_to_OSTIMER Attach NONE to OSTIMER0.
kFRO12M_to_ADC0 Attach FRO12M to ADC0.
kFRO_HF_to_ADC0 Attach FRO_HF to ADC0.
kCLK_IN_to_ADC0 Attach CLK_IN to ADC0.
kCLK_1M_to_ADC0 Attach CLK_1M to ADC0.
kNONE_to_ADC0 Attach NONE to ADC0.
kFRO12M_to_CMP0 Attach FRO12M to CMP0.
kFRO_HF_DIV_to_CMP0 Attach FRO_HF_DIV to CMP0.
kCLK_IN_to_CMP0 Attach CLK_IN to CMP0.
kCLK_1M_to_CMP0 Attach CLK_1M to CMP0.
kNONE_to_CMP0 Attach NONE to CMP0.
kFRO12M_to_CMP1 Attach FRO12M to CMP1.
kFRO_HF_DIV_to_CMP1 Attach FRO_HF_DIV to CMP1.
kCLK_IN_to_CMP1 Attach CLK_IN to CMP1.
kCLK_1M_to_CMP1 Attach CLK_1M to CMP1.

kNONE_to_CMP1 Attach NONE to CMP1.
kCPU_CLK_to_TRACE Attach CPU_CLK to TRACE.
kCLK_1M_to_TRACE Attach CLK_1M to TRACE.
kCLK_16K_to_TRACE Attach CLK_16K to TRACE.
kNONE_to_TRACE Attach NONE to TRACE.
kFRO12M_to_CLKOUT Attach FRO12M to CLKOUT.
kFRO_HF_DIV_to_CLKOUT Attach FRO_HF_DIV to CLKOUT.
kCLK_IN_to_CLKOUT Attach CLK_IN to CLKOUT.
kCLK_16K_to_CLKOUT Attach CLK_16K to CLKOUT.
kSLOW_CLK_to_CLKOUT Attach SLOW_CLK to CLKOUT.
kNONE_to_CLKOUT Attach NONE to CLKOUT.
kCPU_CLK_to_SYSTICK Attach CPU_CLK to SYSTICK.
kCLK_1M_to_SYSTICK Attach CLK_1M to SYSTICK.
kCLK_16K_to_SYSTICK Attach CLK_16K to SYSTICK.
kNONE_to_SYSTICK Attach NONE to SYSTICK.
kNONE_to_NONE Attach NONE to NONE.

4.4.5 enum clock_div_name_t

Enumerator

kCLOCK_DivI3C0_FCLK I3C0_FCLK clock divider.
kCLOCK_DivCTIMER0 CTIMER0 clock divider.
kCLOCK_DivCTIMER1 CTIMER1 clock divider.
kCLOCK_DivCTIMER2 CTIMER2 clock divider.
kCLOCK_DivWWDT0 WWDT0 clock divider.
kCLOCK_DivLPI2C0 LPI2C0 clock divider.
kCLOCK_DivLPSPI0 LPSPI0 clock divider.
kCLOCK_DivLPSPI1 LPSPI1 clock divider.
kCLOCK_DivLPUART0 LPUART0 clock divider.
kCLOCK_DivLPUART1 LPUART1 clock divider.
kCLOCK_DivLPUART2 LPUART2 clock divider.
kCLOCK_DivLPTMR0 LPTMR0 clock divider.
kCLOCK_DivADC0 ADC0 clock divider.
kCLOCK_DivCMP0_FUNC CMP0_FUNC clock divider.
kCLOCK_DivCMP0_RR CMP0_RR clock divider.
kCLOCK_DivCMP1_FUNC CMP1_FUNC clock divider.
kCLOCK_DivCMP1_RR CMP1_RR clock divider.
kCLOCK_DivTRACE TRACE clock divider.
kCLOCK_DivCLKOUT CLKOUT clock divider.
kCLOCK_DivSYSTICK SYSTICK clock divider.
kCLOCK_DivFRO_HF_DIV FRO_HF_DIV clock divider.
kCLOCK_DivSLOWCLK SLOWCLK clock divider.
kCLOCK_DivAHBCLK System clock divider.

kCLOCK_DivMax MAX clock divider.

4.4.6 enum firc_trim_mode_t

Enumerator

kSCG_FircTrimNonUpdate Trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure firc_trim_config_t.

kSCG_FircTrimUpdate Trim enable and trim value update enable. In this mode, the trim value is auto update.

4.4.7 enum firc_trim_src_t

Enumerator

kSCG_FircTrimSrcUsb0 USB0 start of frame (1kHz).

kSCG_FircTrimSrcSysOsc System OSC.

4.4.8 enum sirc_trim_mode_t

Enumerator

kSCG_SircTrimNonUpdate Trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure sirc_trim_config_t.

kSCG_SircTrimUpdate Trim enable and trim value update enable. In this mode, the trim value is auto update.

4.4.9 enum sirc_trim_src_t

Enumerator

kNoTrimSrc No external tirm source.

kSCG_SircTrimSrcSysOsc System OSC.

4.4.10 enum scg_sosc_monitor_mode_t

Enumerator

kSCG_SysOscMonitorDisable Monitor disabled.
kSCG_SysOscMonitorInt Interrupt when the SOSC error is detected.
kSCG_SysOscMonitorReset Reset when the SOSC error is detected.

4.4.11 enum clke_16k_t

Enumerator

kCLKE_16K_SYSTEM To VSYS domain.
kCLKE_16K_COREMAIN To VDD_CORE domain.

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 void CLOCK_AttachClk (clock_attach_id_t *connection*)

Parameters

connection	: Clock to be configured.

Returns

Nothing

4.5.4 clock_attach_id_t CLOCK_GetClockAttachId (clock_attach_id_t *connection*)

Parameters

connection	: Clock attach id to get.

Returns

Clock source value.

4.5.5 void CLOCK_SetClockSelect (clock_select_name_t *sel_name,* uint32_t *value*)

Parameters

sel_name	: Clock select.
value	: value to be set.

4.5.6 uint32_t CLOCK_GetClockSelect (clock_select_name_t *sel_name*)

Parameters

sel_name	: Clock select.

Returns

Clock source value.

4.5.7 void CLOCK_SetClockDiv (clock_div_name_t *div_name,* uint32_t *value*)

Parameters

div_name	: Clock divider name
value	: Value to be divided

Returns

Nothing

4.5.8 uint32_t CLOCK_GetClockDiv (clock_div_name_t *div_name*)

Parameters

div_name	: Clock divider name

Returns

peripheral clock dividers

4.5.9 void CLOCK_HaltClockDiv (clock_div_name_t *div_name*)

Parameters

div_name	: Clock divider name

Returns

Nothing

4.5.10 status_t CLOCK_SetupFROHFClocking (uint32_t *iFreq*)

Parameters

iFreq	: Desired frequency.

Returns

returns success or fail status.

4.5.11 status_t CLOCK_SetupFRO12MClocking (void)

Returns

returns success or fail status.

4.5.12 status_t CLOCK_SetupFRO16KClocking (uint8_t *clk_16k_enable_mask*)

Parameters

clk_16k_enable_mask,:	0-3 0b00: disable both clk_16k0 and clk_16k1 0b01: only enable clk_16k0 0b10: only enable clk_16k1 0b11: enable both clk_16k0 and clk_16k1

Returns

returns success or fail status.

4.5.13 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.

4.5.14 status_t CLOCK_SetupExtRefClocking (uint32_t *iFreq*)

Parameters

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

Returns

returns success or fail status.

4.5.15 uint32_t CLOCK_GetFreq (clock_name_t *clockName*)

Returns

Frequency of selected clock

4.5.16 uint32_t CLOCK_GetCoreSysClkFreq (void)

Returns

Frequency of the core

4.5.17 uint32_t CLOCK_GetI3CFClkFreq (void)

Returns

Frequency of I3C FCLK.

4.5.18 uint32_t CLOCK_GetCTimerClkFreq (uint32_t *id*)

Returns

Frequency of CTimer functional Clock

4.5.19 uint32_t CLOCK_GetLpi2cClkFreq (void)

Returns

Frequency of LPI2C0 functional Clock

4.5.20 uint32_t CLOCK_GetLpspiClkFreq (uint32_t *id*)

Returns

Frequency of LPSPI functional Clock

4.5.21 uint32_t CLOCK_GetLpuartClkFreq (uint32_t *id*)

Returns

Frequency of LPUART functional Clock

4.5.22 uint32_t CLOCK_GetLptmrClkFreq (void)

Returns

Frequency of LPTMR functional Clock

4.5.23 uint32_t CLOCK_GetOstimerClkFreq (void)

Returns

Frequency of OSTIMER Clock

4.5.24 uint32_t CLOCK_GetAdcClkFreq (void)

Returns

Frequency of Adc.

4.5.25 uint32_t CLOCK_GetCmpFClkFreq (uint32_t *id*)

Returns

Frequency of CMP Function.

4.5.26 uint32_t CLOCK_GetCmpRRClkFreq (uint32_t *id*)

Returns

Frequency of CMP Round Robin.

4.5.27 uint32_t CLOCK_GetTraceClkFreq (void)

Returns

Frequency of Trace.

4.5.28 uint32_t CLOCK_GetClkoutClkFreq (void)

Returns

Frequency of CLKOUT.

4.5.29 uint32_t CLOCK_GetSystickClkFreq (void)

Returns

Frequency of Systick.

4.5.30 status_t CLOCK_FROHFTrimConfig (firc_trim_config_t *config*)

Parameters

config	: FROHF trim value

Returns

returns success or fail status.

4.5.31 status_t CLOCK_FRO12MTrimConfig (sirc_trim_config_t *config*)

Parameters

config	: FRO 12M trim value

Returns

returns success or fail status.

4.5.32 void CLOCK_SetSysOscMonitorMode (scg_sosc_monitor_mode_t *mode*)

This function sets the system OSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

mode	Monitor mode to set.

4.5.33 bool CLOCK_EnableUsbfsClock (void)

Enable USB Full Speed clock.

Chapter 5
Reset Driver

5.1 Overview

Reset driver supports peripheral reset and system reset.

Macros

- #define AOI_RSTS

Enumerations

- enum SYSCON_RSTn_t {
  kINPUTMUX0_RST_SHIFT_RSTn = (0U | (0U)),
  kI3C0_RST_SHIFT_RSTn = (0U | (1U)),
  kCTIMER0_RST_SHIFT_RSTn = (0U | (2U)),
  kCTIMER1_RST_SHIFT_RSTn = (0U | (3U)),
  kCTIMER2_RST_SHIFT_RSTn = (0U | (4U)),
  kFREQME_RST_SHIFT_RSTn = (0U | (5U)),
  kUTICK0_RST_SHIFT_RSTn = (0U | (6U)),
  kDMA_RST_SHIFT_RSTn = (0U | (8U)),
  kAOI0_RST_SHIFT_RSTn = (0U | (9U)),
  kCRC_RST_SHIFT_RSTn = (0U | (10U)),
  kEIM_RST_SHIFT_RSTn = (0U | (11U)),
  kERM_RST_SHIFT_RSTn = (0U | (12U)),
  kLPI2C0_RST_SHIFT_RSTn = (0U | (16U)),
  kLPSPI0_RST_SHIFT_RSTn = (0U | (17U)),
  kLPSPI1_RST_SHIFT_RSTn = (0U | (18U)),
  kLPUART0_RST_SHIFT_RSTn = (0U | (19U)),
  kLPUART1_RST_SHIFT_RSTn = (0U | (20U)),
  kLPUART2_RST_SHIFT_RSTn = (0U | (21U)),
  kUSB0_RST_SHIFT_RSTn = (0U | (22U)),
  kQDC0_RST_SHIFT_RSTn = (0U | (23U)),
  kFLEXPWM0_RST_SHIFT_RSTn = (0U | (24U)),
  kOSTIMER0_RST_SHIFT_RSTn = (0U | (25U)),
  kADC0_RST_SHIFT_RSTn = (0U | (26U)),
  kCMP1_RST_SHIFT_RSTn = (0U | (28U)),
  kPORT0_RST_SHIFT_RSTn = (0U | (29U)),
  kPORT1_RST_SHIFT_RSTn = (0U | (30U)),
  kPORT2_RST_SHIFT_RSTn = (0U | (31U)),
  kPORT3_RST_SHIFT_RSTn = ((1U << 8U) | (0U)),
  kATX0_RST_SHIFT_RSTn = ((1U << 8U) | (1U)),
  kGPIO0_RST_SHIFT_RSTn = ((1U << 8U) | (5U)),
  kGPIO1_RST_SHIFT_RSTn = ((1U << 8U) | (6U)),
  kGPIO2_RST_SHIFT_RSTn = ((1U << 8U) | (7U)),
  kGPIO3_RST_SHIFT_RSTn = ((1U << 8U) | (8U)),
  NotAvail_RSTn = (0xFFFFU) }
  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.

- static void RESET_ReleasePeripheralReset (reset_ip_name_t peripheral)
  Release peripheral module.

Driver version

- #define FSL_RESET_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
  reset driver version 2.4.0

5.2 Macro Definition Documentation

5.2.1 #define AOI_RSTS

Value:

```
{                          \
        kAOI0_RST_SHIFT_RSTn \
    } /* Reset bits for ADC peripheral */
```

Array initializers with peripheral reset bits

5.3 Enumeration Type Documentation

5.3.1 enum SYSCON_RSTn_t

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

Enumerator

kINPUTMUX0_RST_SHIFT_RSTn INPUTMUX0 reset control.
kI3C0_RST_SHIFT_RSTn I3C0 reset control.
kCTIMER0_RST_SHIFT_RSTn CTIMER0 reset control.
kCTIMER1_RST_SHIFT_RSTn CTIMER1 reset control.
kCTIMER2_RST_SHIFT_RSTn CTIMER2 reset control.
kFREQME_RST_SHIFT_RSTn FREQME reset control.
kUTICK0_RST_SHIFT_RSTn UTICK0 reset control.
kDMA_RST_SHIFT_RSTn DMA reset control.
kAOI0_RST_SHIFT_RSTn AOI0 reset control.
kCRC_RST_SHIFT_RSTn CRC reset control.
kEIM_RST_SHIFT_RSTn EIM reset control.
kERM_RST_SHIFT_RSTn ERM reset control.
kLPI2C0_RST_SHIFT_RSTn LPI2C0 reset control.
kLPSPI0_RST_SHIFT_RSTn LPSPI0 reset control.
kLPSPI1_RST_SHIFT_RSTn LPSPI1 reset control.
kLPUART0_RST_SHIFT_RSTn LPUART0 reset control.
kLPUART1_RST_SHIFT_RSTn LPUART1 reset control.
kLPUART2_RST_SHIFT_RSTn LPUART2 reset control.

kUSB0_RST_SHIFT_RSTn USB0 reset control.
kQDC0_RST_SHIFT_RSTn QDC0 reset control.
kFLEXPWM0_RST_SHIFT_RSTn FLEXPWM0 reset control.
kOSTIMER0_RST_SHIFT_RSTn OSTIMER0 reset control.
kADC0_RST_SHIFT_RSTn ADC0 reset control.
kCMP1_RST_SHIFT_RSTn CMP1 reset control.
kPORT0_RST_SHIFT_RSTn PORT0 reset control.
kPORT1_RST_SHIFT_RSTn PORT1 reset control.
kPORT2_RST_SHIFT_RSTn PORT2 reset control.
kPORT3_RST_SHIFT_RSTn PORT3 reset control.
kATX0_RST_SHIFT_RSTn ATX0 reset control.
kGPIO0_RST_SHIFT_RSTn GPIO0 reset control.
kGPIO1_RST_SHIFT_RSTn GPIO1 reset control.
kGPIO2_RST_SHIFT_RSTn GPIO2 reset control.
kGPIO3_RST_SHIFT_RSTn GPIO3 reset control.
NotAvail_RSTn No reset control.

5.4 Function Documentation

5.4.1 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.

5.4.2 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.

5.4.3 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.

5.4.4 static void RESET_ReleasePeripheralReset (reset_ip_name_t *peripheral*) [inline], [static]

Release peripheral module.

Parameters

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

Chapter 6
ROMAPI Driver

The ROMAPI driver provides the functionalities to operate the internal Flash.

Chapter 7
Common Driver

7.1 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_GLIKEY = 168 }
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.
- static status_t EnableIRQ (IRQn_Type interrupt)
Enable specific interrupt.
- static status_t DisableIRQ (IRQn_Type interrupt)
Disable specific interrupt.
- static status_t EnableIRQWithPriority (IRQn_Type interrupt, uint8_t priNum)
Enable the IRQ, and also set the interrupt priority.
- static status_t IRQ_SetPriority (IRQn_Type interrupt, uint8_t priNum)
Set the IRQ priority.
- static status_t IRQ_ClearPendingIRQ (IRQn_Type interrupt)
Clear the pending IRQ flag.
- static uint32_t DisableGlobalIRQ (void)
Disable the global IRQ.
- static void EnableGlobalIRQ (uint32_t primask)
Enable the global IRQ.

Driver version

- #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
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**()

Atomic modification

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

- SDK_ATOMIC_LOCAL_ADD
- SDK_ATOMIC_LOCAL_SET
- SDK_ATOMIC_LOCAL_CLEAR
- SDK_ATOMIC_LOCAL_TOGGLE
- SDK_ATOMIC_LOCAL_CLEAR_AND_SET

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

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

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

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

Note

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

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

Timer utilities

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

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

Alignment variable definition macros

- #define SDK_SIZEALIGN(var, alignbytes) ((unsigned int)((var) + ((alignbytes)-1U)) & (unsigned int)(∼(unsigned int)((alignbytes)-1U)))
  Macro to define a variable with L1 d-cache line size alignment.

Non-cacheable region definition macros

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

7.2 Macro Definition Documentation

7.2.1 #define FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ 1

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

7.2.3 #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
```

7.2.4 #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

7.2.5 #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U

7.2.6 #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U

7.2.7 #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U

7.2.8 #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U

7.2.9 #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U

7.2.10 #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U

7.2.11 #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U

7.2.12 #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U

7.2.13 #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U

7.2.14 #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U

7.2.15 #define DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U

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

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

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value

7.3 Typedef Documentation

7.3.1 typedef int32_t status_t

7.4 Enumeration Type Documentation

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

7.4.2 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.

7.5 Function Documentation

7.5.1 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.

7.5.2 void SDK_Free (void * *ptr*)

Parameters

ptr	The memory to be release.

7.5.3 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.

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

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

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

Parameters

interrupt	The IRQ number.

Return values

kStatus_Success	Interrupt enabled successfully
kStatus_Fail	Failed to enable the interrupt

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

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

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

Parameters

interrupt	The IRQ number.

Return values

kStatus_Success	Interrupt disabled successfully
kStatus_Fail	Failed to disable the interrupt

7.5.6 static status_t EnableIRQWithPriority (IRQn_Type *interrupt,* uint8_t *priNum*) [inline], [static]

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

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

Parameters

interrupt	The IRQ to Enable.
priNum	Priority number set to interrupt controller register.

Return values

kStatus_Success	Interrupt priority set successfully
kStatus_Fail	Failed to set the interrupt priority.

7.5.7 static status_t IRQ_SetPriority (IRQn_Type *interrupt,* uint8_t *priNum*) [inline], [static]

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

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

Parameters

interrupt	The IRQ to set.
priNum	Priority number set to interrupt controller register.

Return values

kStatus_Success	Interrupt priority set successfully
kStatus_Fail	Failed to set the interrupt priority.

7.5.8 static status_t IRQ_ClearPendingIRQ (IRQn_Type *interrupt*) [inline], [static]

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

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

Parameters

interrupt	The flag which IRQ to clear.

Return values

kStatus_Success	Interrupt priority set successfully
kStatus_Fail	Failed to set the interrupt priority.

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

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

Returns

Current primask value.

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

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management mechanism of primask.

User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

Parameters

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

Chapter 8
GPIO: General-Purpose Input/Output Driver

8.1 Overview

Modules

- FGPIO Driver
- GPIO Driver

Data Structures

- struct gpio_pin_config_t
  The GPIO pin configuration structure. *More...*
- struct gpio_version_info_t
  GPIO version information. *More...*

Enumerations

- enum gpio_pin_direction_t {
  kGPIO_DigitalInput = 0U,
  kGPIO_DigitalOutput = 1U }
  GPIO direction definition.
- enum gpio_interrupt_config_t {
  kGPIO_InterruptStatusFlagDisabled = 0x0U,
  kGPIO_DMARisingEdge = 0x1U,
  kGPIO_DMAFallingEdge = 0x2U,
  kGPIO_DMAEitherEdge = 0x3U,
  kGPIO_FlagRisingEdge = 0x05U,
  kGPIO_FlagFallingEdge = 0x06U,
  kGPIO_FlagEitherEdge = 0x07U,
  kGPIO_InterruptLogicZero = 0x8U,
  kGPIO_InterruptRisingEdge = 0x9U,
  kGPIO_InterruptFallingEdge = 0xAU,
  kGPIO_InterruptEitherEdge = 0xBU,
  kGPIO_InterruptLogicOne = 0xCU,
  kGPIO_ActiveHighTriggerOutputEnable = 0xDU,
  kGPIO_ActiveLowTriggerOutputEnable = 0xEU }
  Configures the interrupt generation condition.

Driver version

- #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 7, 3))
  GPIO driver version.

8.2 Data Structure Documentation

8.2.1 struct gpio_pin_config_t

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

Data Fields

- gpio_pin_direction_t pinDirection
  GPIO direction, input or output.
- uint8_t outputLogic
  Set a default output logic, which has no use in input.

8.2.2 struct gpio_version_info_t

Data Fields

- uint16_t feature
  Feature Specification Number.
- uint8_t minor
  Minor Version Number.
- uint8_t major
  Major Version Number.

Field Documentation

(1) uint16_t gpio_version_info_t::feature

(2) uint8_t gpio_version_info_t::minor

(3) uint8_t gpio_version_info_t::major

8.3 Macro Definition Documentation

8.3.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 7, 3))

8.4 Enumeration Type Documentation

8.4.1 enum gpio_pin_direction_t

Enumerator

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

8.4.2 enum gpio_interrupt_config_t

Enumerator

kGPIO_InterruptStatusFlagDisabled Interrupt status flag is disabled.
kGPIO_DMARisingEdge ISF flag and DMA request on rising edge.
kGPIO_DMAFallingEdge ISF flag and DMA request on falling edge.
kGPIO_DMAEitherEdge ISF flag and DMA request on either edge.
kGPIO_FlagRisingEdge Flag sets on rising edge.
kGPIO_FlagFallingEdge Flag sets on falling edge.
kGPIO_FlagEitherEdge Flag sets on either edge.
kGPIO_InterruptLogicZero Interrupt when logic zero.
kGPIO_InterruptRisingEdge Interrupt on rising edge.
kGPIO_InterruptFallingEdge Interrupt on falling edge.
kGPIO_InterruptEitherEdge Interrupt on either edge.
kGPIO_InterruptLogicOne Interrupt when logic one.
kGPIO_ActiveHighTriggerOutputEnable Enable active high-trigger output.
kGPIO_ActiveLowTriggerOutputEnable Enable active low-trigger output.

8.5 GPIO Driver

8.5.1 Overview

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

8.5.2 Typical use case

8.5.2.1 Output Operation

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

8.5.2.2 Input Operation

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

GPIO Configuration

- void GPIO_PinInit (GPIO_Type *base, uint32_t pin, const gpio_pin_config_t *config)
  Initializes a GPIO pin used by the board.
- void GPIO_GetVersionInfo (GPIO_Type *base, gpio_version_info_t *info)
  Get GPIO version information.
- static void GPIO_PortInputEnable (GPIO_Type *base, uint32_t mask)
  Enable port input.
- static void GPIO_PortInputDisable (GPIO_Type *base, uint32_t mask)
  Disable port input.

GPIO Output Operations

- static void GPIO_PinWrite (GPIO_Type *base, uint32_t pin, uint8_t output)
  Sets the output level of the multiple GPIO pins to the logic 1 or 0.
- static void GPIO_PortSet (GPIO_Type *base, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 1.
- static void GPIO_PortClear (GPIO_Type *base, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 0.
- static void GPIO_PortToggle (GPIO_Type *base, uint32_t mask)
  Reverses the current output logic of the multiple GPIO pins.

GPIO Input Operations

- static uint32_t GPIO_PinRead (GPIO_Type *base, uint32_t pin)
  Reads the current input value of the GPIO port.

GPIO Interrupt

- static void GPIO_SetPinInterruptConfig (GPIO_Type *base, uint32_t pin, gpio_interrupt_config_t config)
  Configures the gpio pin interrupt/DMA request.
- uint32_t GPIO_GpioGetInterruptFlags (GPIO_Type *base)
  Read the GPIO interrupt status flags.
- uint8_t GPIO_PinGetInterruptFlag (GPIO_Type *base, uint32_t pin)
  Read individual pin's interrupt status flag.
- void GPIO_GpioClearInterruptFlags (GPIO_Type *base, uint32_t mask)
  Clears GPIO pin interrupt status flags.
- void GPIO_PinClearInterruptFlag (GPIO_Type *base, uint32_t pin)
  Clear GPIO individual pin's interrupt status flag.
- static uint32_t GPIO_GetPinsDMARequestFlags (GPIO_Type *base)
  Reads the GPIO DMA request flags.
- static void GPIO_SetMultipleInterruptPinsConfig (GPIO_Type *base, uint32_t mask, gpio_interrupt_config_t config)
  Sets the GPIO interrupt configuration in PCR register for multiple pins.

8.5.3 Function Documentation

8.5.3.1 void GPIO_PinInit (GPIO_Type * *base*, uint32_t *pin*, const gpio_pin_config_t * *config*)

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration.

```
* Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalInput,
*   0,
* }
* Define a digital output pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalOutput,
*   0,
* }
*
```

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
pin	GPIO port pin number
config	GPIO pin configuration pointer

8.5.3.2 void GPIO_GetVersionInfo (GPIO_Type * *base,* gpio_version_info_t * *info*)

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
info	GPIO version information

8.5.3.3 static void GPIO_PortInputEnable (GPIO_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

8.5.3.4 static void GPIO_PortInputDisable (GPIO_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

8.5.3.5 static void GPIO_PinWrite (GPIO_Type * *base,* uint32_t *pin,* uint8_t *output*) [inline], [static]

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
pin	GPIO pin number
output	GPIO pin output logic level. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

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

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

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

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

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

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

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

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
pin	GPIO pin number

Return values

GPIO	port input value • 0: corresponding pin input low-logic level. • 1: corresponding pin input high-logic level.

8.5.3.10 static void GPIO_SetPinInterruptConfig (GPIO_Type * *base*, uint32_t *pin*, gpio_interrupt_config_t *config*) [inline], [static]

Parameters

base	GPIO peripheral base pointer.
pin	GPIO pin number.
config	GPIO pin interrupt configuration. • kGPIO_InterruptStatusFlagDisabled: Interrupt/DMA request disabled. • kGPIO_DMARisingEdge : DMA request on rising edge(if the DMA requests exit). • kGPIO_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit). • kGPIO_DMAEitherEdge : DMA request on either edge(if the DMA requests exit). • kGPIO_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit). • kGPIO_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit). • kGPIO_FlagEitherEdge : Flag sets on either edge(if the Flag states exit). • kGPIO_InterruptLogicZero : Interrupt when logic zero. • kGPIO_InterruptRisingEdge : Interrupt on rising edge. • kGPIO_InterruptFallingEdge: Interrupt on falling edge. • kGPIO_InterruptEitherEdge : Interrupt on either edge. • kGPIO_InterruptLogicOne : Interrupt when logic one. • kGPIO_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit). • kGPIO_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

8.5.3.11 uint32_t GPIO_GpioGetInterruptFlags (GPIO_Type * *base*)

Parameters

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

Returns

The current GPIO's interrupt status flag. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

8.5.3.12 uint8_t GPIO_PinGetInterruptFlag (GPIO_Type * *base*, uint32_t *pin*)

Parameters

base	GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on)
pin	GPIO specific pin number.

Returns

The current selected pin's interrupt status flag.

8.5.3.13 void GPIO_GpioClearInterruptFlags (GPIO_Type * *base*, uint32_t *mask*)

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

8.5.3.14 void GPIO_PinClearInterruptFlag (GPIO_Type * *base*, uint32_t *pin*)

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on).
pin	GPIO specific pin number.

8.5.3.15 static uint32_t GPIO_GetPinsDMARequestFlags (GPIO_Type * *base*) [inline], [static]

The corresponding flag will be cleared automatically at the completion of the requested DMA transfer

8.5.3.16 static void GPIO_SetMultipleInterruptPinsConfig (GPIO_Type * *base,* uint32_t *mask,* gpio_interrupt_config_t *config*) [inline], [static]

Parameters

base	GPIO peripheral base pointer.
mask	GPIO pin number macro.
config	GPIO pin interrupt configuration. • kGPIO_InterruptStatusFlagDisabled: Interrupt disabled. • kGPIO_DMARisingEdge : DMA request on rising edge(if the DMA requests exit). • kGPIO_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit). • kGPIO_DMAEitherEdge : DMA request on either edge(if the DMA requests exit). • kGPIO_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit). • kGPIO_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit). • kGPIO_FlagEitherEdge : Flag sets on either edge(if the Flag states exit). • kGPIO_InterruptLogicZero : Interrupt when logic zero. • kGPIO_InterruptRisingEdge : Interrupt on rising edge. • kGPIO_InterruptFallingEdge: Interrupt on falling edge. • kGPIO_InterruptEitherEdge : Interrupt on either edge. • kGPIO_InterruptLogicOne : Interrupt when logic one. • kGPIO_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit). • kGPIO_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit)..

8.6 FGPIO Driver

This section describes the programming interface of the FGPIO driver. The FGPIO driver configures the FGPIO module and provides a functional interface to build the GPIO application.

Note

FGPIO (Fast GPIO) is only available in a few MCUs. FGPIO and GPIO share the same peripheral but use different registers. FGPIO is closer to the core than the regular GPIO and it's faster to read and write.

8.6.1 Typical use case

8.6.1.1 Output Operation

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

8.6.1.2 Input Operation

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

Chapter 9
LPTMR: Low-Power Timer

9.1 Overview

The MCUXpresso SDK provides a driver for the Low-Power Timer (LPTMR) of MCUXpresso SDK devices.

9.2 Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

9.2.1 Initialization and deinitialization

The function LPTMR_Init() initializes the LPTMR with specified configurations. The function LPTMR_GetDefaultConfig() gets the default configurations. The initialization function configures the LPTMR for a timer or a pulse counter mode mode. It also sets up the LPTMR's free running mode operation and a clock source.

The function LPTMR_DeInit() disables the LPTMR module and gates the module clock.

9.2.2 Timer period Operations

The function LPTMR_SetTimerPeriod() sets the timer period in units of count. Timers counts from 0 to the count value set here.

The function LPTMR_GetCurrentTimerCount() reads the current timer counting value. This function returns the real-time timer counting value ranging from 0 to a timer period.

The timer period operation function takes the count value in ticks. Call the utility macros provided in the fsl_common.h file to convert to microseconds or milliseconds.

9.2.3 Start and Stop timer operations

The function LPTMR_StartTimer() starts the timer counting. After calling this function, the timer counts up to the counter value set earlier by using the LPTMR_SetPeriod() function. Each time the timer reaches the count value and increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

The function LPTMR_StopTimer() stops the timer counting and resets the timer's counter register.

9.2.4 Status

Provides functions to get and clear the LPTMR status.

9.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get the currently enabled interrupts.

9.3 Typical use case

9.3.1 LPTMR tick example

Updates the LPTMR period and toggles an LED periodically. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/lptmr

Data Structures

- struct lptmr_config_t
  LPTMR config structure. *More...*

Enumerations

- enum lptmr_pin_select_t {
  kLPTMR_PinSelectInput_0 = 0x0U,
  kLPTMR_PinSelectInput_1 = 0x1U,
  kLPTMR_PinSelectInput_2 = 0x2U,
  kLPTMR_PinSelectInput_3 = 0x3U }
  LPTMR pin selection used in pulse counter mode.
- enum lptmr_pin_polarity_t {
  kLPTMR_PinPolarityActiveHigh = 0x0U,
  kLPTMR_PinPolarityActiveLow = 0x1U }
  LPTMR pin polarity used in pulse counter mode.
- enum lptmr_timer_mode_t {
  kLPTMR_TimerModeTimeCounter = 0x0U,
  kLPTMR_TimerModePulseCounter = 0x1U }
  LPTMR timer mode selection.
- enum lptmr_prescaler_glitch_value_t {

kLPTMR_Prescale_Glitch_0 = 0x0U,
kLPTMR_Prescale_Glitch_1 = 0x1U,
kLPTMR_Prescale_Glitch_2 = 0x2U,
kLPTMR_Prescale_Glitch_3 = 0x3U,
kLPTMR_Prescale_Glitch_4 = 0x4U,
kLPTMR_Prescale_Glitch_5 = 0x5U,
kLPTMR_Prescale_Glitch_6 = 0x6U,
kLPTMR_Prescale_Glitch_7 = 0x7U,
kLPTMR_Prescale_Glitch_8 = 0x8U,
kLPTMR_Prescale_Glitch_9 = 0x9U,
kLPTMR_Prescale_Glitch_10 = 0xAU,
kLPTMR_Prescale_Glitch_11 = 0xBU,
kLPTMR_Prescale_Glitch_12 = 0xCU,
kLPTMR_Prescale_Glitch_13 = 0xDU,
kLPTMR_Prescale_Glitch_14 = 0xEU,
kLPTMR_Prescale_Glitch_15 = 0xFU }
LPTMR prescaler/glitch filter values.

- enum lptmr_prescaler_clock_select_t {
  kLPTMR_PrescalerClock_0 = 0x0U,
  kLPTMR_PrescalerClock_1 = 0x1U,
  kLPTMR_PrescalerClock_2 = 0x2U,
  kLPTMR_PrescalerClock_3 = 0x3U }
  LPTMR prescaler/glitch filter clock select.
- enum lptmr_interrupt_enable_t { kLPTMR_TimerInterruptEnable = LPTMR_CSR_TIE_MASK }
  List of the LPTMR interrupts.
- enum lptmr_status_flags_t { kLPTMR_TimerCompareFlag = LPTMR_CSR_TCF_MASK }
  List of the LPTMR status flags.

Functions

- static void LPTMR_EnableTimerDMA (LPTMR_Type *base, bool enable)
  Enable or disable timer DMA request.

Driver version

- #define FSL_LPTMR_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  Version 2.1.1.

Initialization and deinitialization

- void LPTMR_Init (LPTMR_Type *base, const lptmr_config_t *config)
  Ungates the LPTMR clock and configures the peripheral for a basic operation.
- void LPTMR_Deinit (LPTMR_Type *base)
  Gates the LPTMR clock.
- void LPTMR_GetDefaultConfig (lptmr_config_t *config)
  Fills in the LPTMR configuration structure with default settings.

Interrupt Interface

- static void LPTMR_EnableInterrupts (LPTMR_Type *base, uint32_t mask)
  Enables the selected LPTMR interrupts.
- static void LPTMR_DisableInterrupts (LPTMR_Type *base, uint32_t mask)
  Disables the selected LPTMR interrupts.
- static uint32_t LPTMR_GetEnabledInterrupts (LPTMR_Type *base)
  Gets the enabled LPTMR interrupts.

Status Interface

- static uint32_t LPTMR_GetStatusFlags (LPTMR_Type *base)
  Gets the LPTMR status flags.
- static void LPTMR_ClearStatusFlags (LPTMR_Type *base, uint32_t mask)
  Clears the LPTMR status flags.

Read and write the timer period

- static void LPTMR_SetTimerPeriod (LPTMR_Type *base, uint32_t ticks)
  Sets the timer period in units of count.
- static uint32_t LPTMR_GetCurrentTimerCount (LPTMR_Type *base)
  Reads the current timer counting value.

Timer Start and Stop

- static void LPTMR_StartTimer (LPTMR_Type *base)
  Starts the timer.
- static void LPTMR_StopTimer (LPTMR_Type *base)
  Stops the timer.

9.4 Data Structure Documentation

9.4.1 struct lptmr_config_t

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

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

Data Fields

- lptmr_timer_mode_t timerMode
  Time counter mode or pulse counter mode.
- lptmr_pin_select_t pinSelect
  LPTMR pulse input pin select; used only in pulse counter mode.
- lptmr_pin_polarity_t pinPolarity
  LPTMR pulse input pin polarity; used only in pulse counter mode.
- bool enableFreeRunning

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set.

- bool bypassPrescaler
  True: bypass prescaler; false: use clock from prescaler.
- lptmr_prescaler_clock_select_t prescalerClockSource
  LPTMR clock source.
- lptmr_prescaler_glitch_value_t value
  Prescaler or glitch filter value.

9.5 Enumeration Type Documentation

9.5.1 enum lptmr_pin_select_t

Enumerator

kLPTMR_PinSelectInput_0 Pulse counter input 0 is selected.
kLPTMR_PinSelectInput_1 Pulse counter input 1 is selected.
kLPTMR_PinSelectInput_2 Pulse counter input 2 is selected.
kLPTMR_PinSelectInput_3 Pulse counter input 3 is selected.

9.5.2 enum lptmr_pin_polarity_t

Enumerator

kLPTMR_PinPolarityActiveHigh Pulse Counter input source is active-high.
kLPTMR_PinPolarityActiveLow Pulse Counter input source is active-low.

9.5.3 enum lptmr_timer_mode_t

Enumerator

kLPTMR_TimerModeTimeCounter Time Counter mode.
kLPTMR_TimerModePulseCounter Pulse Counter mode.

9.5.4 enum lptmr_prescaler_glitch_value_t

Enumerator

kLPTMR_Prescale_Glitch_0 Prescaler divide 2, glitch filter does not support this setting.
kLPTMR_Prescale_Glitch_1 Prescaler divide 4, glitch filter 2.
kLPTMR_Prescale_Glitch_2 Prescaler divide 8, glitch filter 4.
kLPTMR_Prescale_Glitch_3 Prescaler divide 16, glitch filter 8.
kLPTMR_Prescale_Glitch_4 Prescaler divide 32, glitch filter 16.

kLPTMR_Prescale_Glitch_5 Prescaler divide 64, glitch filter 32.
kLPTMR_Prescale_Glitch_6 Prescaler divide 128, glitch filter 64.
kLPTMR_Prescale_Glitch_7 Prescaler divide 256, glitch filter 128.
kLPTMR_Prescale_Glitch_8 Prescaler divide 512, glitch filter 256.
kLPTMR_Prescale_Glitch_9 Prescaler divide 1024, glitch filter 512.
kLPTMR_Prescale_Glitch_10 Prescaler divide 2048 glitch filter 1024.
kLPTMR_Prescale_Glitch_11 Prescaler divide 4096, glitch filter 2048.
kLPTMR_Prescale_Glitch_12 Prescaler divide 8192, glitch filter 4096.
kLPTMR_Prescale_Glitch_13 Prescaler divide 16384, glitch filter 8192.
kLPTMR_Prescale_Glitch_14 Prescaler divide 32768, glitch filter 16384.
kLPTMR_Prescale_Glitch_15 Prescaler divide 65536, glitch filter 32768.

9.5.5 enum lptmr_prescaler_clock_select_t

Note

Clock connections are SoC-specific

Enumerator

kLPTMR_PrescalerClock_0 Prescaler/glitch filter clock 0 selected.
kLPTMR_PrescalerClock_1 Prescaler/glitch filter clock 1 selected.
kLPTMR_PrescalerClock_2 Prescaler/glitch filter clock 2 selected.
kLPTMR_PrescalerClock_3 Prescaler/glitch filter clock 3 selected.

9.5.6 enum lptmr_interrupt_enable_t

Enumerator

kLPTMR_TimerInterruptEnable Timer interrupt enable.

9.5.7 enum lptmr_status_flags_t

Enumerator

kLPTMR_TimerCompareFlag Timer compare flag.

9.6 Function Documentation

9.6.1 void LPTMR_Init (LPTMR_Type * *base*, const lptmr_config_t * *config*)

Note

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

Parameters

base	LPTMR peripheral base address
config	A pointer to the LPTMR configuration structure.

9.6.2 void LPTMR_Deinit (LPTMR_Type * *base*)

Parameters

base	LPTMR peripheral base address

9.6.3 void LPTMR_GetDefaultConfig (lptmr_config_t * *config*)

The default values are as follows.

```
*     config->timerMode = kLPTMR_TimerModeTimeCounter;
*     config->pinSelect = kLPTMR_PinSelectInput_0;
*     config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
*     config->enableFreeRunning = false;
*     config->bypassPrescaler = true;
*     config->prescalerClockSource = kLPTMR_PrescalerClock_1;
*     config->value = kLPTMR_Prescale_Glitch_0;
*
```

Parameters

config	A pointer to the LPTMR configuration structure.

9.6.4 static void LPTMR_EnableInterrupts (LPTMR_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

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

9.6.5 static void LPTMR_DisableInterrupts (LPTMR_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

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

9.6.6 static uint32_t LPTMR_GetEnabledInterrupts (LPTMR_Type * *base*) [inline], [static]

Parameters

base	LPTMR peripheral base address

Returns

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

9.6.7 static void LPTMR_EnableTimerDMA (LPTMR_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	base LPTMR peripheral base address
enable	Switcher of timer DMA feature. "true" means to enable, "false" means to disable.

9.6.8 static uint32_t LPTMR_GetStatusFlags (LPTMR_Type * *base*) [inline], [static]

Parameters

base	LPTMR peripheral base address

Returns

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

9.6.9 static void LPTMR_ClearStatusFlags (LPTMR_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

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

9.6.10 static void LPTMR_SetTimerPeriod (LPTMR_Type * *base*, uint32_t *ticks*) [inline], [static]

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note

1. The TCF flag is set with the CNR equals the count provided here and then increments.
2. Call the utility macros provided in the fsl_common.h to convert to ticks.

Parameters

base	LPTMR peripheral base address
ticks	A timer period in units of ticks, which should be equal or greater than 1.

9.6.11 static uint32_t LPTMR_GetCurrentTimerCount (LPTMR_Type * *base*) [inline], [static]

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

base	LPTMR peripheral base address

Returns

The current counter value in ticks

9.6.12 static void LPTMR_StartTimer (LPTMR_Type * *base*) [inline], [static]

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

base	LPTMR peripheral base address

9.6.13 static void LPTMR_StopTimer (LPTMR_Type * *base*) [inline], [static]

This function stops the timer and resets the timer's counter register.

Parameters

base	LPTMR peripheral base address

Chapter 10
LPUART: Low Power Universal Asynchronous Receiver/-Transmitter Driver

10.1 Overview

Modules

- LPUART CMSIS Driver
- LPUART DMA Driver
- LPUART Driver
- LPUART FreeRTOS Driver

10.2 LPUART Driver

10.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

10.2.2 Typical use case

10.2.2.1 LPUART Operation

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

Data Structures

- struct lpuart_config_t
  LPUART configuration structure. *More...*
- struct lpuart_transfer_t
  LPUART transfer structure. *More...*
- struct lpuart_handle_t
  LPUART handle structure. *More...*

Macros

- #define UART_RETRY_TIMES 0U /∗ Defining to zero means to keep waiting for the flag until it is assert/deassert. ∗/
  Retry times for waiting flag.

Typedefs

- typedef void(∗ lpuart_transfer_callback_t)(LPUART_Type ∗base, lpuart_handle_t ∗handle, status_t status, void ∗userData)
  LPUART transfer callback function.

Enumerations

- enum {
  kStatus_LPUART_TxBusy = MAKE_STATUS(kStatusGroup_LPUART, 0),
  kStatus_LPUART_RxBusy = MAKE_STATUS(kStatusGroup_LPUART, 1),
  kStatus_LPUART_TxIdle = MAKE_STATUS(kStatusGroup_LPUART, 2),
  kStatus_LPUART_RxIdle = MAKE_STATUS(kStatusGroup_LPUART, 3),
  kStatus_LPUART_TxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_LPUART, 4),
  kStatus_LPUART_RxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_LPUART, 5),
  kStatus_LPUART_FlagCannotClearManually = MAKE_STATUS(kStatusGroup_LPUART, 6),
  kStatus_LPUART_Error = MAKE_STATUS(kStatusGroup_LPUART, 7),
  kStatus_LPUART_RxRingBufferOverrun,
  kStatus_LPUART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_LPUART, 9),
  kStatus_LPUART_NoiseError = MAKE_STATUS(kStatusGroup_LPUART, 10),
  kStatus_LPUART_FramingError = MAKE_STATUS(kStatusGroup_LPUART, 11),
  kStatus_LPUART_ParityError = MAKE_STATUS(kStatusGroup_LPUART, 12),
  kStatus_LPUART_BaudrateNotSupport,
  kStatus_LPUART_IdleLineDetected = MAKE_STATUS(kStatusGroup_LPUART, 14),
  kStatus_LPUART_Timeout = MAKE_STATUS(kStatusGroup_LPUART, 15) }
  Error codes for the LPUART driver.
- enum lpuart_parity_mode_t {
  kLPUART_ParityDisabled = 0x0U,
  kLPUART_ParityEven = 0x2U,
  kLPUART_ParityOdd = 0x3U }
  LPUART parity mode.
- enum lpuart_data_bits_t {
  kLPUART_EightDataBits = 0x0U,
  kLPUART_SevenDataBits = 0x1U }
  LPUART data bits count.
- enum lpuart_stop_bit_count_t {
  kLPUART_OneStopBit = 0U,
  kLPUART_TwoStopBit = 1U }
  LPUART stop bit count.
- enum lpuart_transmit_cts_source_t {
  kLPUART_CtsSourcePin = 0U,
  kLPUART_CtsSourceMatchResult = 1U }
  LPUART transmit CTS source.
- enum lpuart_transmit_cts_config_t {
  kLPUART_CtsSampleAtStart = 0U,
  kLPUART_CtsSampleAtIdle = 1U }
  LPUART transmit CTS configure.
- enum lpuart_idle_type_select_t {
  kLPUART_IdleTypeStartBit = 0U,
  kLPUART_IdleTypeStopBit = 1U }
  LPUART idle flag type defines when the receiver starts counting.
- enum lpuart_idle_config_t {

kLPUART_IdleCharacter1 = 0U,
kLPUART_IdleCharacter2 = 1U,
kLPUART_IdleCharacter4 = 2U,
kLPUART_IdleCharacter8 = 3U,
kLPUART_IdleCharacter16 = 4U,
kLPUART_IdleCharacter32 = 5U,
kLPUART_IdleCharacter64 = 6U,
kLPUART_IdleCharacter128 = 7U }
LPUART idle detected configuration.

- enum _lpuart_interrupt_enable {
kLPUART_LinBreakInterruptEnable = (LPUART_BAUD_LBKDIE_MASK >> 8U),
kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8U),
kLPUART_TxDataRegEmptyInterruptEnable = (LPUART_CTRL_TIE_MASK),
kLPUART_TransmissionCompleteInterruptEnable = (LPUART_CTRL_TCIE_MASK),
kLPUART_RxDataRegFullInterruptEnable = (LPUART_CTRL_RIE_MASK),
kLPUART_IdleLineInterruptEnable = (LPUART_CTRL_ILIE_MASK),
kLPUART_RxOverrunInterruptEnable = (LPUART_CTRL_ORIE_MASK),
kLPUART_NoiseErrorInterruptEnable = (LPUART_CTRL_NEIE_MASK),
kLPUART_FramingErrorInterruptEnable = (LPUART_CTRL_FEIE_MASK),
kLPUART_ParityErrorInterruptEnable = (LPUART_CTRL_PEIE_MASK),
kLPUART_Match1InterruptEnable = (LPUART_CTRL_MA1IE_MASK),
kLPUART_Match2InterruptEnable = (LPUART_CTRL_MA2IE_MASK),
kLPUART_TxFifoOverflowInterruptEnable = (LPUART_FIFO_TXOFE_MASK),
kLPUART_RxFifoUnderflowInterruptEnable = (LPUART_FIFO_RXUFE_MASK) }
LPUART interrupt configuration structure, default settings all disabled.
- enum _lpuart_flags {
kLPUART_TxDataRegEmptyFlag,
kLPUART_TransmissionCompleteFlag,
kLPUART_RxDataRegFullFlag = (LPUART_STAT_RDRF_MASK),
kLPUART_IdleLineFlag = (LPUART_STAT_IDLE_MASK),
kLPUART_RxOverrunFlag = (LPUART_STAT_OR_MASK),
kLPUART_NoiseErrorFlag = (LPUART_STAT_NF_MASK),
kLPUART_FramingErrorFlag,
kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),
kLPUART_LinBreakFlag = (LPUART_STAT_LBKDIF_MASK),
kLPUART_RxActiveEdgeFlag = (LPUART_STAT_RXEDGIF_MASK),
kLPUART_RxActiveFlag,
kLPUART_DataMatch1Flag,
kLPUART_DataMatch2Flag,
kLPUART_TxFifoEmptyFlag,
kLPUART_RxFifoEmptyFlag,
kLPUART_TxFifoOverflowFlag,
kLPUART_RxFifoUnderflowFlag }
LPUART status flags.

Driver version

- #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 7, 4))
  LPUART driver version.

Software Reset

- static void LPUART_SoftwareReset (LPUART_Type *base)
  Resets the LPUART using software.

Initialization and deinitialization

- status_t LPUART_Init (LPUART_Type *base, const lpuart_config_t *config, uint32_t srcClock_-Hz)
  Initializes an LPUART instance with the user configuration structure and the peripheral clock.
- void LPUART_Deinit (LPUART_Type *base)
  Deinitializes a LPUART instance.
- void LPUART_GetDefaultConfig (lpuart_config_t *config)
  Gets the default configuration structure.

Module configuration

- status_t LPUART_SetBaudRate (LPUART_Type *base, uint32_t baudRate_Bps, uint32_t src-Clock_Hz)
  Sets the LPUART instance baudrate.
- void LPUART_Enable9bitMode (LPUART_Type *base, bool enable)
  Enable 9-bit data mode for LPUART.
- static void LPUART_SetMatchAddress (LPUART_Type *base, uint16_t address1, uint16_t address2)
  Set the LPUART address.
- static void LPUART_EnableMatchAddress (LPUART_Type *base, bool match1, bool match2)
  Enable the LPUART match address feature.
- static void LPUART_SetRxFifoWatermark (LPUART_Type *base, uint8_t water)
  Sets the rx FIFO watermark.
- static void LPUART_SetTxFifoWatermark (LPUART_Type *base, uint8_t water)
  Sets the tx FIFO watermark.

Status

- uint32_t LPUART_GetStatusFlags (LPUART_Type *base)
  Gets LPUART status flags.
- status_t LPUART_ClearStatusFlags (LPUART_Type *base, uint32_t mask)
  Clears status flags with a provided mask.

Interrupts

- void LPUART_EnableInterrupts (LPUART_Type *base, uint32_t mask)
  Enables LPUART interrupts according to a provided mask.
- void LPUART_DisableInterrupts (LPUART_Type *base, uint32_t mask)
  Disables LPUART interrupts according to a provided mask.
- uint32_t LPUART_GetEnabledInterrupts (LPUART_Type *base)
  Gets enabled LPUART interrupts.

DMA Configuration

- static uintptr_t LPUART_GetDataRegisterAddress (LPUART_Type *base)
  Gets the LPUART data register address.
- static void LPUART_EnableTxDMA (LPUART_Type *base, bool enable)
  Enables or disables the LPUART transmitter DMA request.
- static void LPUART_EnableRxDMA (LPUART_Type *base, bool enable)
  Enables or disables the LPUART receiver DMA.

Bus Operations

- uint32_t LPUART_GetInstance (LPUART_Type *base)
  Get the LPUART instance from peripheral base address.
- static void LPUART_EnableTx (LPUART_Type *base, bool enable)
  Enables or disables the LPUART transmitter.
- static void LPUART_EnableRx (LPUART_Type *base, bool enable)
  Enables or disables the LPUART receiver.
- static void LPUART_WriteByte (LPUART_Type *base, uint8_t data)
  Writes to the transmitter register.
- static uint8_t LPUART_ReadByte (LPUART_Type *base)
  Reads the receiver register.
- static uint8_t LPUART_GetRxFifoCount (LPUART_Type *base)
  Gets the rx FIFO data count.
- static uint8_t LPUART_GetTxFifoCount (LPUART_Type *base)
  Gets the tx FIFO data count.
- void LPUART_SendAddress (LPUART_Type *base, uint8_t address)
  Transmit an address frame in 9-bit data mode.
- status_t LPUART_WriteBlocking (LPUART_Type *base, const uint8_t *data, size_t length)
  Writes to the transmitter register using a blocking method.
- status_t LPUART_ReadBlocking (LPUART_Type *base, uint8_t *data, size_t length)
  Reads the receiver data register using a blocking method.

Transactional

- void LPUART_TransferCreateHandle (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_callback_t callback, void *userData)
  Initializes the LPUART handle.

- status_t LPUART_TransferSendNonBlocking (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer)
  Transmits a buffer of data using the interrupt method.
- void LPUART_TransferStartRingBuffer (LPUART_Type *base, lpuart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)
  Sets up the RX ring buffer.
- void LPUART_TransferStopRingBuffer (LPUART_Type *base, lpuart_handle_t *handle)
  Aborts the background transfer and uninstalls the ring buffer.
- size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type *base, lpuart_handle_t *handle)
  Get the length of received data in RX ring buffer.
- void LPUART_TransferAbortSend (LPUART_Type *base, lpuart_handle_t *handle)
  Aborts the interrupt-driven data transmit.
- status_t LPUART_TransferGetSendCount (LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)
  Gets the number of bytes that have been sent out to bus.
- status_t LPUART_TransferReceiveNonBlocking (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer, size_t *receivedBytes)
  Receives a buffer of data using the interrupt method.
- void LPUART_TransferAbortReceive (LPUART_Type *base, lpuart_handle_t *handle)
  Aborts the interrupt-driven data receiving.
- status_t LPUART_TransferGetReceiveCount (LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)
  Gets the number of bytes that have been received.
- void LPUART_TransferHandleIRQ (LPUART_Type *base, void *irqHandle)
  LPUART IRQ handle function.
- void LPUART_TransferHandleErrorIRQ (LPUART_Type *base, void *irqHandle)
  LPUART Error IRQ handle function.

10.2.3 Data Structure Documentation

10.2.3.1 struct lpuart_config_t

Data Fields

- uint32_t baudRate_Bps
  LPUART baud rate.
- lpuart_parity_mode_t parityMode
  Parity mode, disabled (default), even, odd.
- lpuart_data_bits_t dataBitsCount
  Data bits count, eight (default), seven.
- bool isMsb
  Data bits order, LSB (default), MSB.
- lpuart_stop_bit_count_t stopBitCount
  Number of stop bits, 1 stop bit (default) or 2 stop bits.
- uint8_t txFifoWatermark
  TX FIFO watermark.
- uint8_t rxFifoWatermark

RX FIFO watermark.
- bool enableRxRTS
  RX RTS enable.
- bool enableTxCTS
  TX CTS enable.
- lpuart_transmit_cts_source_t txCtsSource
  TX CTS source.
- lpuart_transmit_cts_config_t txCtsConfig
  TX CTS configure.
- lpuart_idle_type_select_t rxIdleType
  RX IDLE type.
- lpuart_idle_config_t rxIdleConfig
  RX IDLE configuration.
- bool enableTx
  Enable TX.
- bool enableRx
  Enable RX.

Field Documentation

(1) lpuart_idle_type_select_t lpuart_config_t::rxIdleType

(2) lpuart_idle_config_t lpuart_config_t::rxIdleConfig

10.2.3.2 struct lpuart_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* lpuart_transfer_t::data

(2) uint8_t* lpuart_transfer_t::rxData

(3) const uint8_t* lpuart_transfer_t::txData

(4) size_t lpuart_transfer_t::dataSize

10.2.3.3 struct _lpuart_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.
- lpuart_transfer_callback_t callback
  Callback function.
- void * userData
  LPUART callback function parameter.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.
- bool isSevenDataBits
  Seven data bits flag.

Field Documentation

(1) const uint8_t∗ volatile lpuart_handle_t::txData

(2) volatile size_t lpuart_handle_t::txDataSize

(3) size_t lpuart_handle_t::txDataSizeAll

(4) uint8_t∗ volatile lpuart_handle_t::rxData

(5) volatile size_t lpuart_handle_t::rxDataSize

(6) size_t lpuart_handle_t::rxDataSizeAll

(7) uint8_t∗ lpuart_handle_t::rxRingBuffer

(8) size_t lpuart_handle_t::rxRingBufferSize

(9) volatile uint16_t lpuart_handle_t::rxRingBufferHead

(10) volatile uint16_t lpuart_handle_t::rxRingBufferTail

(11) lpuart_transfer_callback_t lpuart_handle_t::callback

(12) void∗ lpuart_handle_t::userData

(13) volatile uint8_t lpuart_handle_t::txState

(14) volatile uint8_t lpuart_handle_t::rxState

(15) bool lpuart_handle_t::isSevenDataBits

10.2.4 Macro Definition Documentation

10.2.4.1 #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 7, 4))

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

10.2.5 Typedef Documentation

10.2.5.1 typedef void(∗ lpuart_transfer_callback_t)(LPUART_Type ∗base, lpuart_handle_t ∗handle, status_t status, void ∗userData)

10.2.6 Enumeration Type Documentation

10.2.6.1 anonymous enum

Enumerator

kStatus_LPUART_TxBusy

kStatus_LPUART_RxBusy RX busy.
kStatus_LPUART_TxIdle LPUART transmitter is idle.
kStatus_LPUART_RxIdle LPUART receiver is idle.
kStatus_LPUART_TxWatermarkTooLarge TX FIFO watermark too large.
kStatus_LPUART_RxWatermarkTooLarge RX FIFO watermark too large.
kStatus_LPUART_FlagCannotClearManually Some flag can't manually clear.
kStatus_LPUART_Error Error happens on LPUART.
kStatus_LPUART_RxRingBufferOverrun LPUART RX software ring buffer overrun.
kStatus_LPUART_RxHardwareOverrun LPUART RX receiver overrun.
kStatus_LPUART_NoiseError LPUART noise error.
kStatus_LPUART_FramingError LPUART framing error.
kStatus_LPUART_ParityError LPUART parity error.
kStatus_LPUART_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_LPUART_IdleLineDetected IDLE flag.
kStatus_LPUART_Timeout LPUART times out.

10.2.6.2 enum lpuart_parity_mode_t

Enumerator

kLPUART_ParityDisabled Parity disabled.
kLPUART_ParityEven Parity enabled, type even, bit setting: PE|PT = 10.
kLPUART_ParityOdd Parity enabled, type odd, bit setting: PE|PT = 11.

10.2.6.3 enum lpuart_data_bits_t

Enumerator

kLPUART_EightDataBits Eight data bit.
kLPUART_SevenDataBits Seven data bit.

10.2.6.4 enum lpuart_stop_bit_count_t

Enumerator

kLPUART_OneStopBit One stop bit.
kLPUART_TwoStopBit Two stop bits.

10.2.6.5 enum lpuart_transmit_cts_source_t

Enumerator

kLPUART_CtsSourcePin CTS resource is the LPUART_CTS pin.

kLPUART_CtsSourceMatchResult CTS resource is the match result.

10.2.6.6 enum lpuart_transmit_cts_config_t

Enumerator

kLPUART_CtsSampleAtStart CTS input is sampled at the start of each character.
kLPUART_CtsSampleAtIdle CTS input is sampled when the transmitter is idle.

10.2.6.7 enum lpuart_idle_type_select_t

Enumerator

kLPUART_IdleTypeStartBit Start counting after a valid start bit.
kLPUART_IdleTypeStopBit Start counting after a stop bit.

10.2.6.8 enum lpuart_idle_config_t

This structure defines the number of idle characters that must be received before the IDLE flag is set.

Enumerator

kLPUART_IdleCharacter1 the number of idle characters.
kLPUART_IdleCharacter2 the number of idle characters.
kLPUART_IdleCharacter4 the number of idle characters.
kLPUART_IdleCharacter8 the number of idle characters.
kLPUART_IdleCharacter16 the number of idle characters.
kLPUART_IdleCharacter32 the number of idle characters.
kLPUART_IdleCharacter64 the number of idle characters.
kLPUART_IdleCharacter128 the number of idle characters.

10.2.6.9 enum _lpuart_interrupt_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

kLPUART_LinBreakInterruptEnable LIN break detect. bit 7
kLPUART_RxActiveEdgeInterruptEnable Receive Active Edge. bit 6
kLPUART_TxDataRegEmptyInterruptEnable Transmit data register empty. bit 23
kLPUART_TransmissionCompleteInterruptEnable Transmission complete. bit 22
kLPUART_RxDataRegFullInterruptEnable Receiver data register full. bit 21

kLPUART_IdleLineInterruptEnable Idle line. bit 20
kLPUART_RxOverrunInterruptEnable Receiver Overrun. bit 27
kLPUART_NoiseErrorInterruptEnable Noise error flag. bit 26
kLPUART_FramingErrorInterruptEnable Framing error flag. bit 25
kLPUART_ParityErrorInterruptEnable Parity error flag. bit 24
kLPUART_Match1InterruptEnable Parity error flag. bit 15
kLPUART_Match2InterruptEnable Parity error flag. bit 14
kLPUART_TxFifoOverflowInterruptEnable Transmit FIFO Overflow. bit 9
kLPUART_RxFifoUnderflowInterruptEnable Receive FIFO Underflow. bit 8

10.2.6.10 enum _lpuart_flags

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

Enumerator

kLPUART_TxDataRegEmptyFlag Transmit data register empty flag, sets when transmit buffer is empty. bit 23
kLPUART_TransmissionCompleteFlag Transmission complete flag, sets when transmission activity complete. bit 22
kLPUART_RxDataRegFullFlag Receive data register full flag, sets when the receive data buffer is full. bit 21
kLPUART_IdleLineFlag Idle line detect flag, sets when idle line detected. bit 20
kLPUART_RxOverrunFlag Receive Overrun, sets when new data is received before data is read from receive register. bit 19
kLPUART_NoiseErrorFlag Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18
kLPUART_FramingErrorFlag Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17
kLPUART_ParityErrorFlag If parity enabled, sets upon parity error detection. bit 16
kLPUART_LinBreakFlag LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31
kLPUART_RxActiveEdgeFlag Receive pin active edge interrupt flag, sets when active edge detected. bit 30
kLPUART_RxActiveFlag Receiver Active Flag (RAF), sets at beginning of valid start. bit 24
kLPUART_DataMatch1Flag The next character to be read from LPUART_DATA matches MA1. bit 15
kLPUART_DataMatch2Flag The next character to be read from LPUART_DATA matches MA2. bit 14
kLPUART_TxFifoEmptyFlag TXEMPT bit, sets if transmit buffer is empty. bit 7
kLPUART_RxFifoEmptyFlag RXEMPT bit, sets if receive buffer is empty. bit 6
kLPUART_TxFifoOverflowFlag TXOF bit, sets if transmit buffer overflow occurred. bit 1
kLPUART_RxFifoUnderflowFlag RXUF bit, sets if receive buffer underflow occurred. bit 0

10.2.7 Function Documentation

10.2.7.1 static void LPUART_SoftwareReset (LPUART_Type * *base*) [inline], [static]

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

base	LPUART peripheral base address.

10.2.7.2 status_t LPUART_Init (LPUART_Type * *base*, const lpuart_config_t * *config*, uint32_t *srcClock_Hz*)

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

```
*  lpuart_config_t lpuartConfig;
*  lpuartConfig.baudRate_Bps = 115200U;
*  lpuartConfig.parityMode = kLPUART_ParityDisabled;
*  lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
*  lpuartConfig.isMsb = false;
*  lpuartConfig.stopBitCount = kLPUART_OneStopBit;
*  lpuartConfig.txFifoWatermark = 0;
*  lpuartConfig.rxFifoWatermark = 1;
*  LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

Parameters

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

Return values

kStatus_LPUART_-BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_Success	LPUART initialize succeed

10.2.7.3 void LPUART_Deinit (LPUART_Type * *base*)

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

Parameters

base	LPUART peripheral base address.

10.2.7.4 void LPUART_GetDefaultConfig (lpuart_config_t * *config*)

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

config	Pointer to a configuration structure.

10.2.7.5 status_t LPUART_SetBaudRate (LPUART_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
*  LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
*
```

Parameters

base	LPUART peripheral base address.
baudRate_Bps	LPUART baudrate to be set.
srcClock_Hz	LPUART clock source frequency in HZ.

Return values

kStatus_LPUART_BaudrateNotSupport	Baudrate is not supported in the current clock source.
kStatus_Success	Set baudrate succeeded.

10.2.7.6 void LPUART_Enable9bitMode (LPUART_Type * *base*, bool *enable*)

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

base	LPUART peripheral base address.
enable	true to enable, flase to disable.

10.2.7.7 static void LPUART_SetMatchAddress (LPUART_Type * *base,* uint16_t *address1,* uint16_t *address2*) [inline], [static]

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note

Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

base	LPUART peripheral base address.
address1	LPUART slave address1.
address2	LPUART slave address2.

10.2.7.8 static void LPUART_EnableMatchAddress (LPUART_Type * *base,* bool *match1,* bool *match2*) [inline], [static]

Parameters

base	LPUART peripheral base address.
match1	true to enable match address1, false to disable.
match2	true to enable match address2, false to disable.

10.2.7.9 static void LPUART_SetRxFifoWatermark (LPUART_Type * *base,* uint8_t *water*) [inline], [static]

Parameters

base	LPUART peripheral base address.
water	Rx FIFO watermark.

10.2.7.10 static void LPUART_SetTxFifoWatermark (LPUART_Type * *base*, uint8_t *water*) [inline], [static]

Parameters

base	LPUART peripheral base address.
water	Tx FIFO watermark.

10.2.7.11 uint32_t LPUART_GetStatusFlags (LPUART_Type * *base*)

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators _lpuart_flags. To check for a specific status, compare the return value with enumerators in the _lpuart_flags. For example, to check whether the TX is empty:

```
*     if (kLPUART_TxDataRegEmptyFlag &
      LPUART_GetStatusFlags(LPUART1))
*     {
*         ...
*     }
*
```

Parameters

base	LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

10.2.7.12 status_t LPUART_ClearStatusFlags (LPUART_Type * *base*, uint32_t *mask*)

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorFlag, kLPUART_ParityErrorFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

base	LPUART peripheral base address.
mask	the status flags to be cleared. The user can use the enumerators in the _lpuart_status-_flag_t to do the OR operation and get the mask.

Returns

0 succeed, others failed.

Return values

kStatus_LPUART_Flag-CannotClearManually	The flag can't be cleared by this function but it is cleared automatically by hardware.
kStatus_Success	Status in the mask are cleared.

10.2.7.13 void LPUART_EnableInterrupts (LPUART_Type * *base*, uint32_t *mask*)

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the _lpuart_interrupt_enable. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
*     LPUART_EnableInterrupts(LPUART1,
      kLPUART_TxDataRegEmptyInterruptEnable |
      kLPUART_RxDataRegFullInterruptEnable);
*
```

Parameters

base	LPUART peripheral base address.
mask	The interrupts to enable. Logical OR of _lpuart_interrupt_enable.

10.2.7.14 void LPUART_DisableInterrupts (LPUART_Type * *base*, uint32_t *mask*)

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _lpuart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
*     LPUART_DisableInterrupts(LPUART1,
      kLPUART_TxDataRegEmptyInterruptEnable |
      kLPUART_RxDataRegFullInterruptEnable);
*
```

Parameters

base	LPUART peripheral base address.
mask	The interrupts to disable. Logical OR of _lpuart_interrupt_enable.

10.2.7.15 uint32_t LPUART_GetEnabledInterrupts (LPUART_Type * *base*)

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _lpuart_interrupt_enable. To check a specific interrupt enable status, compare the return value with enumerators in _lpuart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
*     uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);
*
*     if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
*     {
*         ...
*     }
*
```

Parameters

base	LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in _lpuart_interrupt_enable.

10.2.7.16 static uintptr_t LPUART_GetDataRegisterAddress (LPUART_Type * *base*) [inline], [static]

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

base	LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

10.2.7.17 static void LPUART_EnableTxDMA (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

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

10.2.7.18 static void LPUART_EnableRxDMA (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

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

10.2.7.19 uint32_t LPUART_GetInstance (LPUART_Type * *base*)

Parameters

base	LPUART peripheral base address.

Returns

LPUART instance.

10.2.7.20 static void LPUART_EnableTx (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the LPUART transmitter.

Parameters

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

10.2.7.21 static void LPUART_EnableRx (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the LPUART receiver.

Parameters

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

10.2.7.22 static void LPUART_WriteByte (LPUART_Type * *base,* uint8_t *data*) [inline], [static]

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

Parameters

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

10.2.7.23 static uint8_t LPUART_ReadByte (LPUART_Type * *base*) [inline], [static]

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

Parameters

base	LPUART peripheral base address.

Returns

Data read from data register.

10.2.7.24 static uint8_t LPUART_GetRxFifoCount (LPUART_Type * *base*) [inline], [static]

Parameters

base	LPUART peripheral base address.

Returns

rx FIFO data count.

10.2.7.25 static uint8_t LPUART_GetTxFifoCount (LPUART_Type * *base*) [inline], [static]

Parameters

base	LPUART peripheral base address.

Returns

tx FIFO data count.

10.2.7.26 void LPUART_SendAddress (LPUART_Type * *base,* uint8_t *address*)

Parameters

base	LPUART peripheral base address.
address	LPUART slave address.

10.2.7.27 status_t LPUART_WriteBlocking (LPUART_Type * *base,* const uint8_t * *data,* size_t *length*)

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

base	LPUART peripheral base address.
data	Start address of the data to write.

length	Size of the data to write.

Return values

kStatus_LPUART_-Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully wrote all data.

10.2.7.28 status_t LPUART_ReadBlocking (LPUART_Type * *base,* uint8_t * *data,* size_t *length*)

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

base	LPUART peripheral base address.
data	Start address of the buffer to store the received data.
length	Size of the buffer.

Return values

kStatus_LPUART_Rx-HardwareOverrun	Receiver overrun happened while receiving data.
kStatus_LPUART_Noise-Error	Noise error happened while receiving data.
kStatus_LPUART_-FramingError	Framing error happened while receiving data.
kStatus_LPUART_Parity-Error	Parity error happened while receiving data.
kStatus_LPUART_-Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully received all data.

10.2.7.29 void LPUART_TransferCreateHandle (LPUART_Type * *base,* lpuart_handle_t * *handle,* lpuart_transfer_callback_t *callback,* void * *userData*)

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

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the LPUART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

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

10.2.7.30 status_t LPUART_TransferSendNonBlocking (LPUART_Type * *base*, lpuart_handle_t * *handle*, lpuart_transfer_t * *xfer*)

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus_LPUART_TxIdle as status parameter.

Note

The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART transfer structure, see lpuart_transfer_t.

Return values

kStatus_Success	Successfully start the data transmission.
kStatus_LPUART_TxBusy	Previous transmission still not finished, data not all written to the TX register.

kStatus_InvalidArgument	Invalid argument.

10.2.7.31 void LPUART_TransferStartRingBuffer (LPUART_Type * *base,* lpuart_handle_t * *handle,* uint8_t * *ringBuffer,* size_t *ringBufferSize*)

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note

When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBuffer-Size` is 32, then only 31 bytes are used for saving data.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
ringBuffer	Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
ringBufferSize	size of the ring buffer.

10.2.7.32 void LPUART_TransferStopRingBuffer (LPUART_Type * *base,* lpuart_handle_t * *handle*)

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

10.2.7.33 size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type * *base,* lpuart_handle_t * *handle*)

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

10.2.7.34 void LPUART_TransferAbortSend (LPUART_Type * *base*, lpuart_handle_t * *handle*)

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

10.2.7.35 status_t LPUART_TransferGetSendCount (LPUART_Type * *base*, lpuart_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferInProgress	No send in progress.

kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

10.2.7.36 status_t LPUART_TransferReceiveNonBlocking (LPUART_Type * *base,* lpuart_handle_t * *handle,* lpuart_transfer_t * *xfer,* size_t * *receivedBytes*)

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter `receivedBytes` set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART transfer structure, see uart_transfer_t.
receivedBytes	Bytes received from the ring buffer directly.

Return values

kStatus_Success	Successfully queue the transfer into the transmit queue.
kStatus_LPUART_RxBusy	Previous receive request is not finished.
kStatus_InvalidArgument	Invalid argument.

10.2.7.37 void LPUART_TransferAbortReceive (LPUART_Type * *base,* lpuart_handle_t * *handle*)

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

10.2.7.38 status_t LPUART_TransferGetReceiveCount (LPUART_Type * *base*, lpuart_handle_t * *handle*, uint32_t * *count*)

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

Parameters

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

Return values

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

10.2.7.39 void LPUART_TransferHandleIRQ (LPUART_Type * *base*, void * *irqHandle*)

This function handles the LPUART transmit and receive IRQ request.

Parameters

base	LPUART peripheral base address.
irqHandle	LPUART handle pointer.

10.2.7.40 void LPUART_TransferHandleErrorIRQ (LPUART_Type * *base*, void * *irqHandle*)

This function handles the LPUART error IRQ request.

Parameters

base	LPUART peripheral base address.
irqHandle	LPUART handle pointer.

10.3 LPUART DMA Driver

10.3.1 Overview

Data Structures

- struct lpuart_dma_handle_t
  LPUART DMA handle. More...

Typedefs

- typedef void(∗ lpuart_dma_transfer_callback_t)(LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, status_t status, void ∗userData)
  LPUART transfer callback function.

Driver version

- #define FSL_LPUART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  LPUART DMA driver version.

EDMA transactional

- void LPUART_TransferCreateHandleDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, lpuart_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  Initializes the LPUART handle which is used in transactional functions.
- status_t LPUART_TransferSendDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  Sends data using DMA.
- status_t LPUART_TransferReceiveDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  Receives data using DMA.
- void LPUART_TransferAbortSendDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle)
  Aborts the sent data using DMA.
- void LPUART_TransferAbortReceiveDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle)
  Aborts the received data using DMA.
- status_t LPUART_TransferGetSendCountDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, uint32_t ∗count)
  Gets the number of bytes written to the LPUART TX register.
- status_t LPUART_TransferGetReceiveCountDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, uint32_t ∗count)
  Gets the number of received bytes.
- void LPUART_TransferDMAHandleIRQ (LPUART_Type ∗base, void ∗lpuartDmaHandle)
  LPUART DMA IRQ handle function.

10.3.2 Data Structure Documentation

10.3.2.1 struct _lpuart_dma_handle

Data Fields

- lpuart_dma_transfer_callback_t callback
  Callback function.
- void * userData
  LPUART callback function parameter.
- size_t rxDataSizeAll
  Size of the data to receive.
- size_t txDataSizeAll
  Size of the data to send out.
- 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) lpuart_dma_transfer_callback_t lpuart_dma_handle_t::callback

(2) void* lpuart_dma_handle_t::userData

(3) size_t lpuart_dma_handle_t::rxDataSizeAll

(4) size_t lpuart_dma_handle_t::txDataSizeAll

(5) dma_handle_t* lpuart_dma_handle_t::txDmaHandle

(6) dma_handle_t* lpuart_dma_handle_t::rxDmaHandle

(7) volatile uint8_t lpuart_dma_handle_t::txState

10.3.3 Macro Definition Documentation

10.3.3.1 #define FSL_LPUART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

10.3.4 Typedef Documentation

10.3.4.1 typedef void(* lpuart_dma_transfer_callback_t)(LPUART_Type *base, lpuart_dma_handle_t *handle, status_t status, void *userData)

10.3.5 Function Documentation

10.3.5.1 void LPUART_TransferCreateHandleDMA (LPUART_Type * *base,* lpuart_dma_handle_t * *handle,* lpuart_dma_transfer_callback_t *callback,* void * *userData,* dma_handle_t * *txDmaHandle,* dma_handle_t * *rxDmaHandle*)

Note

This function disables all LPUART interrupts.

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_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.

10.3.5.2 status_t LPUART_TransferSendDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, lpuart_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	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART DMA transfer structure. See lpuart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_LPUART_TxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

10.3.5.3 status_t LPUART_TransferReceiveDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, lpuart_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	LPUART peripheral base address.
handle	Pointer to lpuart_dma_handle_t structure.
xfer	LPUART DMA transfer structure. See lpuart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_LPUART_Rx-Busy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

10.3.5.4 void LPUART_TransferAbortSendDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*)

This function aborts send data using DMA.

Parameters

base	LPUART peripheral base address
handle	Pointer to lpuart_dma_handle_t structure

10.3.5.5 void LPUART_TransferAbortReceiveDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*)

This function aborts the received data using DMA.

Parameters

base	LPUART peripheral base address
handle	Pointer to lpuart_dma_handle_t structure

10.3.5.6 status_t LPUART_TransferGetSendCountDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes that have been written to LPUART TX register by DMA.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

count	Send bytes count.

Return values

kStatus_NoTransferInProgress	No send in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

10.3.5.7 status_t LPUART_TransferGetReceiveCountDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, uint32_t * *count*)

This function gets the number of received bytes.

Parameters

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

Return values

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

10.3.5.8 void LPUART_TransferDMAHandleIRQ (LPUART_Type * *base*, void * *lpuartDmaHandle*)

This function handles the LPUART tx complete IRQ request and invoke user callback.

Note

This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

base	LPUART peripheral base address.
lpuartDma-Handle	LPUART handle pointer.

10.4 LPUART FreeRTOS Driver

10.4.1 Overview

Data Structures

- struct lpuart_rtos_config_t
  LPUART RTOS configuration structure. More...

Driver version

- #define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  LPUART FreeRTOS driver version.

LPUART RTOS Operation

- int LPUART_RTOS_Init (lpuart_rtos_handle_t *handle, lpuart_handle_t *t_handle, const lpuart_rtos_config_t *cfg)
  Initializes an LPUART instance for operation in RTOS.
- int LPUART_RTOS_Deinit (lpuart_rtos_handle_t *handle)
  Deinitializes an LPUART instance for operation.

LPUART transactional Operation

- int LPUART_RTOS_Send (lpuart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length)
  Sends data in the background.
- int LPUART_RTOS_Receive (lpuart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length, size_t *received)
  Receives data.
- int LPUART_RTOS_SetRxTimeout (lpuart_rtos_handle_t *handle, uint32_t rx_timeout_constant_ms, uint32_t rx_timeout_multiplier_ms)
  Set RX timeout in runtime.
- int LPUART_RTOS_SetTxTimeout (lpuart_rtos_handle_t *handle, uint32_t tx_timeout_constant_ms, uint32_t tx_timeout_multiplier_ms)
  Set TX timeout in runtime.

10.4.2 Data Structure Documentation

10.4.2.1 struct lpuart_rtos_config_t

Data Fields

- LPUART_Type * base
  UART base address.

- uint32_t srcclk
  UART source clock in Hz.
- uint32_t baudrate
  Desired communication speed.
- lpuart_parity_mode_t parity
  Parity setting.
- lpuart_stop_bit_count_t stopbits
  Number of stop bits to use.
- uint8_t * buffer
  Buffer for background reception.
- uint32_t buffer_size
  Size of buffer for background reception.
- uint32_t rx_timeout_constant_ms
  RX timeout applied per receive.
- uint32_t rx_timeout_multiplier_ms
  RX timeout added for each byte of the receive.
- uint32_t tx_timeout_constant_ms
  TX timeout applied per transmition.
- uint32_t tx_timeout_multiplier_ms
  TX timeout added for each byte of the transmition.
- bool enableRxRTS
  RX RTS enable.
- bool enableTxCTS
  TX CTS enable.
- lpuart_transmit_cts_source_t txCtsSource
  TX CTS source.
- lpuart_transmit_cts_config_t txCtsConfig
  TX CTS configure.

Field Documentation

(1) uint32_t lpuart_rtos_config_t::rx_timeout_multiplier_ms

(2) uint32_t lpuart_rtos_config_t::tx_timeout_multiplier_ms

10.4.3 Macro Definition Documentation

10.4.3.1 #define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

10.4.4 Function Documentation

10.4.4.1 int LPUART_RTOS_Init (lpuart_rtos_handle_t * *handle,* lpuart_handle_t * *t_handle,* const lpuart_rtos_config_t * *cfg*)

Parameters

handle	The RTOS LPUART handle, the pointer to an allocated space for RTOS context.
t_handle	The pointer to an allocated space to store the transactional layer internal state.
cfg	The pointer to the parameters required to configure the LPUART after initialization.

Returns

0 succeed, others failed

10.4.4.2 int LPUART_RTOS_Deinit (lpuart_rtos_handle_t * *handle*)

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

Parameters

handle	The RTOS LPUART handle.

10.4.4.3 int LPUART_RTOS_Send (lpuart_rtos_handle_t * *handle,* uint8_t * *buffer,* uint32_t *length*)

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

Parameters

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

10.4.4.4 int LPUART_RTOS_Receive (lpuart_rtos_handle_t * *handle,* uint8_t * *buffer,* uint32_t *length,* size_t * *received*)

This function receives data from LPUART. It is an synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

Parameters

handle	The RTOS LPUART handle.
buffer	The pointer to buffer where to write received data.
length	The number of bytes to receive.
received	The pointer to a variable of size_t where the number of received data is filled.

10.4.4.5 int LPUART_RTOS_SetRxTimeout (lpuart_rtos_handle_t * *handle*, uint32_t *rx_timeout_constant_ms*, uint32_t *rx_timeout_multiplier_ms*)

This function can modify RX timeout between initialization and receive.

param handle The RTOS LPUART handle. param rx_timeout_constant_ms RX timeout applied per receive. param rx_timeout_multiplier_ms RX timeout added for each byte of the receive.

10.4.4.6 int LPUART_RTOS_SetTxTimeout (lpuart_rtos_handle_t * *handle*, uint32_t *tx_timeout_constant_ms*, uint32_t *tx_timeout_multiplier_ms*)

This function can modify TX timeout between initialization and send.

param handle The RTOS LPUART handle. param tx_timeout_constant_ms TX timeout applied per transmition. param tx_timeout_multiplier_ms TX timeout added for each byte of the transmition.

10.5 LPUART CMSIS Driver

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

The LPUART driver includes transactional APIs.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements please write custom code.

10.5.1 Function groups

10.5.1.1 LPUART CMSIS GetVersion Operation

This function group will return the LPUART CMSIS Driver version to user.

10.5.1.2 LPUART CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

10.5.1.3 LPUART CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize the lpuart instance . And this API must be called before you configure a lpuart instance or after you Deinit a lpuart instance.The right steps to start an instance is that you must initialize the instance which been slected firstly,then you can power on the instance.After these all have been done,you can configure the instance by using control operation.If you want to Uninitialize the instance, you must power off the instance first.

10.5.1.4 LPUART CMSIS Transfer Operation

This function group controls the transfer, send/receive data.

10.5.1.5 LPUART CMSIS Status Operation

This function group gets the LPUART transfer status.

10.5.1.6 LPUART CMSIS Control Operation

This function can configure an instance ,set baudrate for lpuart, get current baudrate ,set transfer data bits and other control command.

Chapter 11
PORT: Port Control and Interrupts

11.1 Overview

The MCUXpresso SDK provides a driver for the Port Control and Interrupts (PORT) module of MCUXpresso SDK devices.

Data Structures

- struct port_pin_config_t
  PORT pin configuration structure. More...
- struct port_version_info_t
  PORT version information. More...

Enumerations

- enum _port_pull {
  kPORT_PullDisable = 0U,
  kPORT_PullDown = 2U,
  kPORT_PullUp = 3U }
  Internal resistor pull feature selection.
- enum _port_pull_value {
  kPORT_LowPullResistor = 0U,
  kPORT_HighPullResistor = 1U }
  Internal resistor pull value selection.
- enum _port_slew_rate {
  kPORT_FastSlewRate = 0U,
  kPORT_SlowSlewRate = 1U }
  Slew rate selection.
- enum _port_open_drain_enable {
  kPORT_OpenDrainDisable = 0U,
  kPORT_OpenDrainEnable = 1U }
  Open Drain feature enable/disable.
- enum _port_passive_filter_enable {
  kPORT_PassiveFilterDisable = 0U,
  kPORT_PassiveFilterEnable = 1U }
  Passive filter feature enable/disable.
- enum _port_drive_strength {
  kPORT_LowDriveStrength = 0U,
  kPORT_HighDriveStrength = 1U }
  Configures the drive strength.
- enum _port_drive_strength1 {
  kPORT_NormalDriveStrength = 0U,
  kPORT_DoubleDriveStrength = 1U }

Configures the drive strength1.

- enum _port_input_buffer {
  kPORT_InputBufferDisable = 0U,
  kPORT_InputBufferEnable = 1U }
  input buffer disable/enable.
- enum _port_invet_input {
  kPORT_InputNormal = 0U,
  kPORT_InputInvert = 1U }
  Digital input is not inverted or it is inverted.
- enum _port_lock_register {
  kPORT_UnlockRegister = 0U,
  kPORT_LockRegister = 1U }
  Unlock/lock the pin control register field[15:0].
- enum port_mux_t {
  kPORT_PinDisabledOrAnalog = 0U,
  kPORT_MuxAsGpio = 1U,
  kPORT_MuxAlt0 = 0U,
  kPORT_MuxAlt1 = 1U,
  kPORT_MuxAlt2 = 2U,
  kPORT_MuxAlt3 = 3U,
  kPORT_MuxAlt4 = 4U,
  kPORT_MuxAlt5 = 5U,
  kPORT_MuxAlt6 = 6U,
  kPORT_MuxAlt7 = 7U,
  kPORT_MuxAlt8 = 8U,
  kPORT_MuxAlt9 = 9U,
  kPORT_MuxAlt10 = 10U,
  kPORT_MuxAlt11 = 11U,
  kPORT_MuxAlt12 = 12U,
  kPORT_MuxAlt13 = 13U,
  kPORT_MuxAlt14 = 14U,
  kPORT_MuxAlt15 = 15U }
  Pin mux selection.
- enum port_voltage_range_t {
  kPORT_VoltageRange1Dot71V_3Dot6V = 0x0U,
  kPORT_VoltageRange2Dot70V_3Dot6V = 0x1U }
  PORT voltage range.

Driver version

- #define FSL_PORT_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))
  PORT driver version.

Configuration

- static void PORT_GetVersionInfo (PORT_Type *base, port_version_info_t *info)
  Get PORT version information.

- static void PORT_SecletPortVoltageRange (PORT_Type *base, port_voltage_range_t range)
  Get PORT version information.
- static void PORT_SetPinConfig (PORT_Type *base, uint32_t pin, const port_pin_config_t *config)
  Sets the port PCR register.
- static void PORT_SetMultiplePinsConfig (PORT_Type *base, uint32_t mask, const port_pin_config_t *config)
  Sets the port PCR register for multiple pins.
- static void PORT_SetPinMux (PORT_Type *base, uint32_t pin, port_mux_t mux)
  Configures the pin muxing.

Interrupt

- static void PORT_SetPinDriveStrength (PORT_Type *base, uint32_t pin, uint8_t strength)
  Configures the port pin drive strength.
- static void PORT_EnablePinDoubleDriveStrength (PORT_Type *base, uint32_t pin, bool enable)
  Enables the port pin double drive strength.
- static void PORT_SetPinPullValue (PORT_Type *base, uint32_t pin, uint8_t value)
  Configures the port pin pull value.

11.2 Data Structure Documentation

11.2.1 struct port_pin_config_t

Data Fields

- uint16_t pullSelect: 2
  No-pull/pull-down/pull-up select.
- uint16_t pullValueSelect: 1
  Pull value select.
- uint16_t slewRate: 1
  Fast/slow slew rate Configure.
- uint16_t passiveFilterEnable: 1
  Passive filter enable/disable.
- uint16_t openDrainEnable: 1
  Open drain enable/disable.
- uint16_t driveStrength: 1
  Fast/slow drive strength configure.
- uint16_t driveStrength1: 1
  Normal/Double drive strength enable/disable.
- uint16_t mux: 4
  Pin mux Configure.
- uint16_t inputBuffer: 1
  Input Buffer Configure.
- uint16_t invertInput: 1
  Invert Input Configure.
- uint16_t lockRegister: 1
  Lock/unlock the PCR field[15:0].

11.2.2 struct port_version_info_t

Data Fields

- uint16_t feature
  Feature Specification Number.
- uint8_t minor
  Minor Version Number.
- uint8_t major
  Major Version Number.

Field Documentation

(1) uint16_t port_version_info_t::feature

(2) uint8_t port_version_info_t::minor

(3) uint8_t port_version_info_t::major

11.3 Macro Definition Documentation

11.3.1 #define FSL_PORT_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))

11.4 Enumeration Type Documentation

11.4.1 enum _port_pull

Enumerator

kPORT_PullDisable Internal pull-up/down resistor is disabled.
kPORT_PullDown Internal pull-down resistor is enabled.
kPORT_PullUp Internal pull-up resistor is enabled.

11.4.2 enum _port_pull_value

Enumerator

kPORT_LowPullResistor Low internal pull resistor value is selected.
kPORT_HighPullResistor High internal pull resistor value is selected.

11.4.3 enum _port_slew_rate

Enumerator

kPORT_FastSlewRate Fast slew rate is configured.
kPORT_SlowSlewRate Slow slew rate is configured.

11.4.4 enum _port_open_drain_enable

Enumerator

kPORT_OpenDrainDisable Open drain output is disabled.
kPORT_OpenDrainEnable Open drain output is enabled.

11.4.5 enum _port_passive_filter_enable

Enumerator

kPORT_PassiveFilterDisable Passive input filter is disabled.
kPORT_PassiveFilterEnable Passive input filter is enabled.

11.4.6 enum _port_drive_strength

Enumerator

kPORT_LowDriveStrength Low-drive strength is configured.
kPORT_HighDriveStrength High-drive strength is configured.

11.4.7 enum _port_drive_strength1

Enumerator

kPORT_NormalDriveStrength Normal drive strength.
kPORT_DoubleDriveStrength Double drive strength.

11.4.8 enum _port_input_buffer

Enumerator

kPORT_InputBufferDisable Digital input is disabled.
kPORT_InputBufferEnable Digital input is enabled.

11.4.9 enum _port_invet_input

Enumerator

kPORT_InputNormal Digital input is not inverted.
kPORT_InputInvert Digital input is inverted.

11.4.10 enum _port_lock_register

Enumerator

kPORT_UnlockRegister Pin Control Register fields [15:0] are not locked.
kPORT_LockRegister Pin Control Register fields [15:0] are locked.

11.4.11 enum port_mux_t

Enumerator

kPORT_PinDisabledOrAnalog Corresponding pin is disabled, but is used as an analog pin.
kPORT_MuxAsGpio Corresponding pin is configured as GPIO.
kPORT_MuxAlt0 Chip-specific.
kPORT_MuxAlt1 Chip-specific.
kPORT_MuxAlt2 Chip-specific.
kPORT_MuxAlt3 Chip-specific.
kPORT_MuxAlt4 Chip-specific.
kPORT_MuxAlt5 Chip-specific.
kPORT_MuxAlt6 Chip-specific.
kPORT_MuxAlt7 Chip-specific.
kPORT_MuxAlt8 Chip-specific.
kPORT_MuxAlt9 Chip-specific.
kPORT_MuxAlt10 Chip-specific.
kPORT_MuxAlt11 Chip-specific.
kPORT_MuxAlt12 Chip-specific.
kPORT_MuxAlt13 Chip-specific.
kPORT_MuxAlt14 Chip-specific.
kPORT_MuxAlt15 Chip-specific.

11.4.12 enum port_voltage_range_t

Enumerator

kPORT_VoltageRange1Dot71V_3Dot6V Port voltage range is 1.71 V - 3.6 V.
kPORT_VoltageRange2Dot70V_3Dot6V Port voltage range is 2.70 V - 3.6 V.

11.5 Function Documentation

11.5.1 static void PORT_GetVersionInfo (PORT_Type * *base,* port_version_info_t * *info*) [inline], [static]

Parameters

base	PORT peripheral base pointer
info	PORT version information

11.5.2 static void PORT_SecletPortVoltageRange (PORT_Type * *base*, port_voltage_range_t *range*) [inline], [static]

Note

: PORTA_CONFIG[RANGE] controls the voltage ranges of Port A, B, and C. Read or write PORTB_CONFIG[RANGE] and PORTC_CONFIG[RANGE] does not take effect.

Parameters

base	PORT peripheral base pointer
range	port voltage range

11.5.3 static void PORT_SetPinConfig (PORT_Type * *base*, uint32_t *pin*, const port_pin_config_t * *config*) [inline], [static]

This is an example to define an input pin or output pin PCR configuration.

```
* // Define a digital input pin PCR configuration
* port_pin_config_t config = {
*      kPORT_PullUp,
*      kPORT_FastSlewRate,
*      kPORT_PassiveFilterDisable,
*      kPORT_OpenDrainDisable,
*      kPORT_LowDriveStrength,
*      kPORT_MuxAsGpio,
*      kPORT_UnLockRegister,
* };
*
```

Parameters

base	PORT peripheral base pointer.

pin	PORT pin number.
config	PORT PCR register configuration structure.

11.5.4 static void PORT_SetMultiplePinsConfig (PORT_Type * *base*, uint32_t *mask*, const port_pin_config_t * *config*) [inline], [static]

This is an example to define input pins or output pins PCR configuration.

```
* Define a digital input pin PCR configuration
* port_pin_config_t config = {
*      kPORT_PullUp ,
*      kPORT_PullEnable,
*      kPORT_FastSlewRate,
*      kPORT_PassiveFilterDisable,
*      kPORT_OpenDrainDisable,
*      kPORT_LowDriveStrength,
*      kPORT_MuxAsGpio,
*      kPORT_UnlockRegister,
* };
*
```

Parameters

base	PORT peripheral base pointer.
mask	PORT pin number macro.
config	PORT PCR register configuration structure.

11.5.5 static void PORT_SetPinMux (PORT_Type * *base*, uint32_t *pin*, port_mux_t *mux*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.
mux	pin muxing slot selection. • kPORT_PinDisabledOrAnalog: Pin disabled or work in analog function. • kPORT_MuxAsGpio : Set as GPIO. • kPORT_MuxAlt2 : chip-specific. • kPORT_MuxAlt3 : chip-specific. • kPORT_MuxAlt4 : chip-specific. • kPORT_MuxAlt5 : chip-specific. • kPORT_MuxAlt6 : chip-specific. • kPORT_MuxAlt7 : chip-specific.

Note

: This function is NOT recommended to use together with the PORT_SetPinsConfig, because the PORT_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

11.5.6 static void PORT_SetPinDriveStrength (PORT_Type * *base*, uint32_t *pin*, uint8_t *strength*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.
strength	PORT pin drive strength • kPORT_LowDriveStrength = 0U - Low-drive strength is configured. • kPORT_HighDriveStrength = 1U - High-drive strength is configured.

11.5.7 static void PORT_EnablePinDoubleDriveStrength (PORT_Type * *base*, uint32_t *pin*, bool *enable*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.
enable	PORT pin drive strength configuration.

11.5.8 static void PORT_SetPinPullValue (PORT_Type * *base*, uint32_t *pin*, uint8_t *value*) [inline], [static]

Parameters

base	PORT peripheral base pointer.

pin	PORT pin number.
value	PORT pin pull value • kPORT_LowPullResistor = 0U - Low internal pull resistor value is selected. • kPORT_HighPullResistor = 1U - High internal pull resistor value is selected.

Chapter 12
Debug Console Lite

12.1 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.

12.2 Function groups

12.2.1 Initialization

To initialize the debug console, call the DbgConsole_Init() function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate, serial_port_type_t
      device, uint32_t clkSrcFreq);
```

Selects the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
    kSerialPort_None = 0U,
    kSerialPort_Uart = 1U,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral. The debug console state is stored in the debug_console_state_t structure, such as shown here.

```
typedef struct DebugConsoleState
{
    uint8_t uartHandleBuffer[HAL_UART_HANDLE_SIZE];
    hal_uart_status_t (*putChar)(hal_uart_handle_t handle, const uint8_t
      *data, size_t length);
    hal_uart_status_t (*getChar)(hal_uart_handle_t handle, uint8_t *data,
       size_t length);
    serial_port_type_t type;
} debug_console_state_t;
```

This example shows how to call the DbgConsole_Init() given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_USART_INSTANCE, BOARD_DEBUG_USART_BAUDRATE,
      BOARD_DEBUG_USART_TYPE,
                            BOARD_DEBUG_USART_CLK_FREQ);
```

12.2.2 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

specifier	Description

- 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 */
```

12.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART

There are two macros SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART added to configure PRINTF and low level output perihperal.

- The macro SDK_DEBUGCONSOLE is used for forntend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The fucntion can be set by the macro SDK_DEBUGCONSOLE.
- The macro SDK_DEBUGCONSOLE_UART is used for backend. It is use to decide whether provide low level IO implementation to toolchain printf and scanf. For example, within MCUXpresso, if the macro SDK_DEBUGCONSOLE_UART is defined, __sys_write and __sys_readc will be used when __REDLIB__ is defined; _write and _read will be used in other cases.The macro does not specifically refer to the perihpheral "UART". It refers to the external perihperal UART. 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	UART	UART
DEBUGCONSOLE_REDIRECT_TO_SDK	undefined	UART	semihost
DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	defined	UART	UART
DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	undefined	semihost	semihost
DEBUGCONSOLE_DISABLE	defined	No ouput	UART
DEBUGCONSOLE_DISABLE	undefined	No ouput	semihost

SDK_DEBUGCONSOLE	SDK_DEBUGCONSOLE_UART	PRINTF	printf

12.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

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

Some examples use the PRINTF function

Statement prints the string format.

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

Statement prints the hexadecimal format/

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

Statement prints the decimal floating point and unsigned decimal.

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

Some examples use the SCANF function

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

Print out failure messages using MCUXpresso SDK __assert_func:

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

Note:

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

Modules

- Semihosting

Macros

- #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U
  Definition select redirect toolchain printf, scanf to uart or not.
- #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U
  Select SDK version printf, scanf.
- #define DEBUGCONSOLE_DISABLE 2U
  Disable debugconsole function.
- #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK
  Definition to select sdk or toolchain printf, scanf.
- #define PRINTF_FLOAT_ENABLE 0U
  Definition to printf the float number.
- #define SCANF_FLOAT_ENABLE 0U
  Definition to scanf the float number.
- #define PRINTF_ADVANCED_ENABLE 0U
  Definition to support advanced format specifier for printf.
- #define SCANF_ADVANCED_ENABLE 0U
  Definition to support advanced format specifier for scanf.
- #define PRINTF DbgConsole_Printf
  Definition to select redirect toolchain printf, scanf to uart or not.

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.

12.4 Macro Definition Documentation

12.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

12.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

12.4.3 #define DEBUGCONSOLE_DISABLE 2U

12.4.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

12.4.5 #define PRINTF_FLOAT_ENABLE 0U

12.4.6 #define SCANF_FLOAT_ENABLE 0U

12.4.7 #define PRINTF_ADVANCED_ENABLE 0U

12.4.8 #define SCANF_ADVANCED_ENABLE 0U

12.4.9 #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.

12.5 Function Documentation

12.5.1 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, frequency of peripheral source clock, and base address at the specified baud rate. 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.
clkSrcFreq	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

kStatus_Success	Execution successfully
kStatus_Fail	Execution failure

12.5.2 status_t DbgConsole_Deinit (void)

Call this function to disable debug log messages to be output via the specified peripheral base address and at the specified baud rate.

Returns

Indicates whether de-initialization was successful or not.

12.5.3 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.

12.5.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

12.5.5 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.

12.5.6 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.

12.5.7 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.

12.5.8 int DbgConsole_Scanf (char * *fmt_s, ...*)

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

Parameters

fmt_s	Format control string.

Returns

Returns the number of fields successfully converted and assigned.

12.5.9 int DbgConsole_Getchar (void)

Call this function to read a character from standard input.

Returns

Returns the character read.

12.6 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.

12.6.1 Guide Semihosting for IAR

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

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.

12.6.2 Guide Semihosting for Keil µVision

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

12.6.3 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.

12.6.4 Guide Semihosting for ARMGCC

Step 1: Setting up the environment

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

Add to "CMakeLists.txt"

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

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

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

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

Step 2: Building the project

1. Change "CMakeLists.txt":
   Change "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=nano.specs")"
   to "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=rdimon.specs")"
   Replace paragraph
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")
   To
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --specs=rdimon.specs ")
   Remove
   target_link_libraries(semihosting_ARMGCC.elf debug nosys)
2. Run "build_debug.bat" to build project

Step 3: Starting semihosting

1. Download the image and set as follows.

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

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

Chapter 13
GenericList

13.1 Overview

Data Structures

- struct list_handle_t
  The list structure. More...
- struct list_element_handle_t
  The list element. More...

Macros

- #define GENERIC_LIST_LIGHT (1)
  Definition to determine whether use list light.
- #define GENERIC_LIST_DUPLICATED_CHECKING (0)
  Definition to determine whether enable list duplicated checking.

Enumerations

- enum list_status_t {
  kLIST_Ok = kStatus_Success,
  kLIST_DuplicateError = MAKE_STATUS(kStatusGroup_LIST, 1),
  kLIST_Full = MAKE_STATUS(kStatusGroup_LIST, 2),
  kLIST_Empty = MAKE_STATUS(kStatusGroup_LIST, 3),
  kLIST_OrphanElement = MAKE_STATUS(kStatusGroup_LIST, 4),
  kLIST_NotSupport = MAKE_STATUS(kStatusGroup_LIST, 5) }
  The list status.

Functions

- void LIST_Init (list_handle_t list, uint32_t max)
  Initialize the list.
- list_handle_t LIST_GetList (list_element_handle_t listElement)
  Gets the list that contains the given element.
- list_status_t LIST_AddHead (list_handle_t list, list_element_handle_t listElement)
  Links element to the head of the list.
- list_status_t LIST_AddTail (list_handle_t list, list_element_handle_t listElement)
  Links element to the tail of the list.
- list_element_handle_t LIST_RemoveHead (list_handle_t list)
  Unlinks element from the head of the list.
- list_element_handle_t LIST_GetHead (list_handle_t list)
  Gets head element handle.
- list_element_handle_t LIST_GetNext (list_element_handle_t listElement)
  Gets next element handle for given element handle.

- list_element_handle_t LIST_GetPrev (list_element_handle_t listElement)
  Gets previous element handle for given element handle.
- list_status_t LIST_RemoveElement (list_element_handle_t listElement)
  Unlinks an element from its list.
- list_status_t LIST_AddPrevElement (list_element_handle_t listElement, list_element_handle_t newElement)
  Links an element in the previous position relative to a given member of a list.
- uint32_t LIST_GetSize (list_handle_t list)
  Gets the current size of a list.
- uint32_t LIST_GetAvailableSize (list_handle_t list)
  Gets the number of free places in the list.

13.2 Data Structure Documentation

13.2.1 struct list_label_t

Data Fields

- struct list_element_tag * head
  list head
- struct list_element_tag * tail
  list tail
- uint32_t size
  list size
- uint32_t max
  list max number of elements

13.2.2 struct list_element_t

Data Fields

- struct list_element_tag * next
  next list element
- struct list_label * list
  pointer to the list

13.3 Macro Definition Documentation

13.3.1 #define GENERIC_LIST_LIGHT (1)

13.3.2 #define GENERIC_LIST_DUPLICATED_CHECKING (0)

13.4 Enumeration Type Documentation

13.4.1 enum list_status_t

Enumerator

kLIST_Ok Success.
kLIST_DuplicateError Duplicate Error.
kLIST_Full FULL.
kLIST_Empty Empty.
kLIST_OrphanElement Orphan Element.
kLIST_NotSupport Not Support.

13.5 Function Documentation

13.5.1 void LIST_Init (list_handle_t *list,* uint32_t *max*)

This function initialize the list.

Parameters

list	- List handle to initialize.
max	- Maximum number of elements in list. 0 for unlimited.

13.5.2 list_handle_t LIST_GetList (list_element_handle_t *listElement*)

Parameters

element	- Handle of the element.

Return values

NULL	if element is orphan, Handle of the list the element is inserted into.

13.5.3 list_status_t LIST_AddHead (list_handle_t *list,* list_element_handle_t *listElement*)

Parameters

list	- Handle of the list.
element	- Handle of the element.

Return values

kLIST_Full	if list is full, kLIST_Ok if insertion was successful.

13.5.4 list_status_t LIST_AddTail (list_handle_t *list,* list_element_handle_t *listElement*)

Parameters

list	- Handle of the list.
element	- Handle of the element.

Return values

kLIST_Full	if list is full, kLIST_Ok if insertion was successful.

13.5.5 list_element_handle_t LIST_RemoveHead (list_handle_t *list*)

Parameters

list	- Handle of the list.

Return values

NULL	if list is empty, handle of removed element(pointer) if removal was successful.

13.5.6 list_element_handle_t LIST_GetHead (list_handle_t *list*)

Parameters

list	- Handle of the list.

Return values

NULL	if list is empty, handle of removed element(pointer) if removal was successful.

13.5.7 list_element_handle_t LIST_GetNext (list_element_handle_t *listElement*)

Parameters

element	- Handle of the element.

Return values

NULL	if list is empty, handle of removed element(pointer) if removal was successful.

13.5.8 list_element_handle_t LIST_GetPrev (list_element_handle_t *listElement*)

Parameters

element	- Handle of the element.

Return values

NULL	if list is empty, handle of removed element(pointer) if removal was successful.

13.5.9 list_status_t LIST_RemoveElement (list_element_handle_t *listElement*)

Parameters

element	- Handle of the element.

Return values

kLIST_OrphanElement	if element is not part of any list.
kLIST_Ok	if removal was successful.

13.5.10 list_status_t LIST_AddPrevElement (list_element_handle_t *listElement,* list_element_handle_t *newElement*)

Parameters

element	- Handle of the element.
newElement	- New element to insert before the given member.

Return values

kLIST_OrphanElement	if element is not part of any list.
kLIST_Ok	if removal was successful.

13.5.11 uint32_t LIST_GetSize (list_handle_t *list*)

Parameters

list	- Handle of the list.

Return values

Current	size of the list.

13.5.12 uint32_t LIST_GetAvailableSize (list_handle_t *list*)

Parameters

list	- Handle of the list.

Return values

Available	size of the list.

Chapter 14
UART_Adapter

14.1 Overview

Data Structures

- struct hal_uart_config_t
  UART configuration structure. More...
- struct hal_uart_transfer_t
  UART transfer structure. More...

Macros

- #define UART_ADAPTER_NON_BLOCKING_MODE (1U)
  Enable or disable UART adapter non-blocking mode (1 - enable, 0 - disable)
- #define HAL_UART_ADAPTER_FIFO (1U)
  Enable or disable uart hardware FIFO mode (1 - enable, 0 - disable)
- #define HAL_UART_DMA_INIT_ENABLE (1U)
  Enable or disable uart DMA adapter int mode (1 - enable, 0 - disable)
- #define HAL_UART_DMA_IDLELINE_TIMEOUT (1U)
  Definition of uart dma adapter software idleline detection timeout value in ms.
- #define HAL_UART_HANDLE_SIZE (92U + HAL_UART_ADAPTER_LOWPOWER * 16U + HAL_UART_DMA_ENABLE * 4U)
  Definition of uart adapter handle size.
- #define UART_HANDLE_DEFINE(name) uint32_t name[((HAL_UART_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Definition of uart dma adapter handle size.
- #define HAL_UART_TRANSFER_MODE (0U)
  Whether enable transactional function of the UART.

Typedefs

- typedef void * hal_uart_handle_t
  The handle of uart adapter.
- typedef void * hal_uart_dma_handle_t
  The handle of uart dma adapter.
- typedef void(* hal_uart_transfer_callback_t)(hal_uart_handle_t handle, hal_uart_status_t status, void *callbackParam)
  UART transfer callback function.

Enumerations

- enum hal_uart_status_t {
  kStatus_HAL_UartSuccess = kStatus_Success,
  kStatus_HAL_UartTxBusy = MAKE_STATUS(kStatusGroup_HAL_UART, 1),
  kStatus_HAL_UartRxBusy = MAKE_STATUS(kStatusGroup_HAL_UART, 2),
  kStatus_HAL_UartTxIdle = MAKE_STATUS(kStatusGroup_HAL_UART, 3),
  kStatus_HAL_UartRxIdle = MAKE_STATUS(kStatusGroup_HAL_UART, 4),
  kStatus_HAL_UartBaudrateNotSupport,
  kStatus_HAL_UartProtocolError,
  kStatus_HAL_UartError = MAKE_STATUS(kStatusGroup_HAL_UART, 7) }
  UART status.
- enum hal_uart_parity_mode_t {
  kHAL_UartParityDisabled = 0x0U,
  kHAL_UartParityEven = 0x2U,
  kHAL_UartParityOdd = 0x3U }
  UART parity mode.
- enum hal_uart_stop_bit_count_t {
  kHAL_UartOneStopBit = 0U,
  kHAL_UartTwoStopBit = 1U }
  UART stop bit count.

Functions

- hal_uart_status_t HAL_UartEnterLowpower (hal_uart_handle_t handle)
  Prepares to enter low power consumption.
- hal_uart_status_t HAL_UartExitLowpower (hal_uart_handle_t handle)
  Restores from low power consumption.
- void HAL_UartIsrFunction (hal_uart_handle_t handle)
  UART IRQ handle function.

Initialization and deinitialization

- hal_uart_status_t HAL_UartInit (hal_uart_handle_t handle, const hal_uart_config_t *uart_config)
  Initializes a UART instance with the UART handle and the user configuration structure.
- hal_uart_status_t HAL_UartDeinit (hal_uart_handle_t handle)
  Deinitializes a UART instance.

Blocking bus Operations

- hal_uart_status_t HAL_UartReceiveBlocking (hal_uart_handle_t handle, uint8_t *data, size_t length)
  Reads RX data register using a blocking method.
- hal_uart_status_t HAL_UartSendBlocking (hal_uart_handle_t handle, const uint8_t *data, size_t length)
  Writes to the TX register using a blocking method.

Functional API with non-blocking mode.

Note

The functional API and the transactional API cannot be used at the same time. The macro HAL_UART_TRANSFER_MODE is used to set which one will be used. If HAL_UART_TRANSFER_MODE is zero, the functional API with non-blocking mode will be used. Otherwise, transactional API will be used.

- hal_uart_status_t HAL_UartInstallCallback (hal_uart_handle_t handle, hal_uart_transfer_callback_t callback, void ∗callbackParam)
  Installs a callback and callback parameter.
- hal_uart_status_t HAL_UartReceiveNonBlocking (hal_uart_handle_t handle, uint8_t ∗data, size_t length)
  Receives a buffer of data using an interrupt method.
- hal_uart_status_t HAL_UartSendNonBlocking (hal_uart_handle_t handle, uint8_t ∗data, size_t length)
  Transmits a buffer of data using the interrupt method.
- hal_uart_status_t HAL_UartGetReceiveCount (hal_uart_handle_t handle, uint32_t ∗reCount)
  Gets the number of bytes that have been received.
- hal_uart_status_t HAL_UartGetSendCount (hal_uart_handle_t handle, uint32_t ∗seCount)
  Gets the number of bytes written to the UART TX register.
- hal_uart_status_t HAL_UartAbortReceive (hal_uart_handle_t handle)
  Aborts the interrupt-driven data receiving.
- hal_uart_status_t HAL_UartAbortSend (hal_uart_handle_t handle)
  Aborts the interrupt-driven data sending.

14.2 Data Structure Documentation

14.2.1 struct hal_uart_config_t

Data Fields

- uint32_t srcClock_Hz
  Source clock.
- uint32_t baudRate_Bps
  Baud rate.
- hal_uart_parity_mode_t parityMode
  Parity mode, disabled (default), even, odd.
- hal_uart_stop_bit_count_t stopBitCount
  Number of stop bits, 1 stop bit (default) or 2 stop bits.
- uint8_t enableRx
  Enable RX.
- uint8_t enableTx
  Enable TX.
- uint8_t enableRxRTS
  Enable RX RTS.
- uint8_t enableTxCTS
  Enable TX CTS.
- uint8_t instance
  Instance (0 - UART0, 1 - UART1, ...), detail information please refer to the SOC corresponding RM.

Field Documentation

(1) uint8_t hal_uart_config_t::instance

Invalid instance value will cause initialization failure.

14.2.2 struct hal_uart_transfer_t

Data Fields

- uint8_t * data
  The buffer of data to be transfer.
- size_t dataSize
  The byte count to be transfer.

Field Documentation

(1) uint8_t* hal_uart_transfer_t::data

(2) size_t hal_uart_transfer_t::dataSize

14.3 Macro Definition Documentation

14.3.1 #define HAL_UART_DMA_IDLELINE_TIMEOUT (1U)

14.3.2 #define HAL_UART_HANDLE_SIZE (92U + HAL_UART_ADAPTER_LOWPOWER * 16U + HAL_UART_DMA_ENABLE * 4U)

14.3.3 #define UART_HANDLE_DEFINE(*name*) uint32_t name[((HAL_UART_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

Defines the uart handle

This macro is used to define a 4 byte aligned uart handle. Then use "(hal_uart_handle_t)name" to get the uart handle.

The macro should be global and could be optional. You could also define uart handle by yourself.

This is an example,

```
* UART_HANDLE_DEFINE(uartHandle);
*
```

Parameters

name	The name string of the uart handle.

14.3.4 #define HAL_UART_TRANSFER_MODE (0U)

(0 - disable, 1 - enable)

14.4 Typedef Documentation

14.4.1 typedef void* hal_uart_handle_t

14.4.2 typedef void* hal_uart_dma_handle_t

14.4.3 typedef void(* hal_uart_transfer_callback_t)(hal_uart_handle_t handle, hal_uart_status_t status, void *callbackParam)

14.5 Enumeration Type Documentation

14.5.1 enum hal_uart_status_t

Enumerator

kStatus_HAL_UartSuccess Successfully.
kStatus_HAL_UartTxBusy TX busy.
kStatus_HAL_UartRxBusy RX busy.
kStatus_HAL_UartTxIdle HAL UART transmitter is idle.
kStatus_HAL_UartRxIdle HAL UART receiver is idle.
kStatus_HAL_UartBaudrateNotSupport Baudrate is not support in current clock source.
kStatus_HAL_UartProtocolError Error occurs for Noise, Framing, Parity, etc. For transactional transfer, The up layer needs to abort the transfer and then starts again
kStatus_HAL_UartError Error occurs on HAL UART.

14.5.2 enum hal_uart_parity_mode_t

Enumerator

kHAL_UartParityDisabled Parity disabled.
kHAL_UartParityEven Parity even enabled.
kHAL_UartParityOdd Parity odd enabled.

14.5.3 enum hal_uart_stop_bit_count_t

Enumerator

kHAL_UartOneStopBit One stop bit.
kHAL_UartTwoStopBit Two stop bits.

14.6 Function Documentation

14.6.1 hal_uart_status_t HAL_UartInit (hal_uart_handle_t *handle,* const hal_uart_config_t * *uart_config*)

This function configures the UART module with user-defined settings. The user can configure the configuration structure. The parameter handle is a pointer to point to a memory space of size HAL_UART_HANDLE_SIZE allocated by the caller. Example below shows how to use this API to configure the UART.

```
*   UART_HANDLE_DEFINE(g_UartHandle);
*   hal_uart_config_t config;
*   config.srcClock_Hz = 48000000;
*   config.baudRate_Bps = 115200U;
*   config.parityMode = kHAL_UartParityDisabled;
*   config.stopBitCount = kHAL_UartOneStopBit;
*   config.enableRx = 1;
*   config.enableTx = 1;
*   config.enableRxRTS = 0;
*   config.enableTxCTS = 0;
*   config.instance = 0;
*   HAL_UartInit((hal_uart_handle_t)g_UartHandle, &config);
*
```

Parameters

handle	Pointer to point to a memory space of size HAL_UART_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: UART_HANDLE_DEFINE(handle); or uint32_t handle[((HAL_UART_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];

config	Pointer to user-defined configuration structure.

Return values

kStatus_HAL_Uart-BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_HAL_Uart-Success	UART initialization succeed

14.6.2 hal_uart_status_t HAL_UartDeinit (hal_uart_handle_t *handle*)

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

Parameters

handle	UART handle pointer.

Return values

kStatus_HAL_Uart-Success	UART de-initialization succeed

14.6.3 hal_uart_status_t HAL_UartReceiveBlocking (hal_uart_handle_t *handle,* uint8_t * *data,* size_t *length*)

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data, and reads data from the RX register.

Note

The function HAL_UartReceiveBlocking and the function HAL_UartTransferReceiveNonBlocking cannot be used at the same time. And, the function HAL_UartTransferAbortReceive cannot be used to abort the transmission of this function.

Parameters

handle	UART handle pointer.
data	Start address of the buffer to store the received data.
length	Size of the buffer.

Return values

kStatus_HAL_UartError	An error occurred while receiving data.
kStatus_HAL_UartParity-Error	A parity error occurred while receiving data.
kStatus_HAL_Uart-Success	Successfully received all data.

14.6.4 hal_uart_status_t HAL_UartSendBlocking (hal_uart_handle_t *handle*, const uint8_t * *data*, size_t *length*)

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Note

The function HAL_UartSendBlocking and the function HAL_UartTransferSendNonBlocking cannot be used at the same time. And, the function HAL_UartTransferAbortSend cannot be used to abort the transmission of this function.

Parameters

handle	UART handle pointer.
data	Start address of the data to write.
length	Size of the data to write.

Return values

kStatus_HAL_Uart-Success	Successfully sent all data.

14.6.5 hal_uart_status_t HAL_UartInstallCallback (hal_uart_handle_t *handle*, hal_uart_transfer_callback_t *callback*, void * *callbackParam*)

This function is used to install the callback and callback parameter for UART module. When non-blocking sending or receiving finished, the adapter will notify the upper layer by the installed callback function. And

the status is also passed as status parameter when the callback is called.

Parameters

handle	UART handle pointer.
callback	The callback function.
callbackParam	The parameter of the callback function.

Return values

kStatus_HAL_Uart-Success	Successfully install the callback.

14.6.6 hal_uart_status_t HAL_UartReceiveNonBlocking (hal_uart_handle_t *handle*, uint8_t * *data*, size_t *length*)

This function receives data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be received. The receive request is saved by the UART adapter. When the new data arrives, the receive request is serviced first. When all data is received, the UART adapter notifies the upper layer through a callback function and passes the status parameter kStatus_UART_RxIdle.

Note

The function HAL_UartReceiveBlocking and the function HAL_UartReceiveNonBlocking cannot be used at the same time.

Parameters

handle	UART handle pointer.
data	Start address of the data to write.
length	Size of the data to write.

Return values

kStatus_HAL_Uart-Success	Successfully queue the transfer into transmit queue.

kStatus_HAL_UartRx-Busy	Previous receive request is not finished.
kStatus_HAL_UartError	An error occurred.

14.6.7 hal_uart_status_t HAL_UartSendNonBlocking (hal_uart_handle_t *handle*, uint8_t * *data*, size_t *length*)

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the ISR, the UART driver calls the callback function and passes the kStatus_UART_TxIdle as status parameter.

Note

The function HAL_UartSendBlocking and the function HAL_UartSendNonBlocking cannot be used at the same time.

Parameters

handle	UART handle pointer.
data	Start address of the data to write.
length	Size of the data to write.

Return values

kStatus_HAL_Uart-Success	Successfully start the data transmission.
kStatus_HAL_UartTx-Busy	Previous transmission still not finished; data not all written to TX register yet.
kStatus_HAL_UartError	An error occurred.

14.6.8 hal_uart_status_t HAL_UartGetReceiveCount (hal_uart_handle_t *handle*, uint32_t * *reCount*)

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

Parameters

handle	UART handle pointer.
count	Receive bytes count.

Return values

kStatus_HAL_UartError	An error occurred.
kStatus_Success	Get successfully through the parameter `count`.

14.6.9 hal_uart_status_t HAL_UartGetSendCount (hal_uart_handle_t *handle*, uint32_t * *seCount*)

This function gets the number of bytes written to the UART TX register by using the interrupt method.

Parameters

handle	UART handle pointer.
count	Send bytes count.

Return values

kStatus_HAL_UartError	An error occurred.
kStatus_Success	Get successfully through the parameter `count`.

14.6.10 hal_uart_status_t HAL_UartAbortReceive (hal_uart_handle_t *handle*)

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to know how many bytes are not received yet.

Note

The function HAL_UartAbortReceive cannot be used to abort the transmission of the function HAL_UartReceiveBlocking.

Parameters

handle	UART handle pointer.

Return values

kStatus_Success	Get successfully abort the receiving.

14.6.11 hal_uart_status_t HAL_UartAbortSend (hal_uart_handle_t *handle*)

This function aborts the interrupt-driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

Note

The function HAL_UartAbortSend cannot be used to abort the transmission of the function HAL_UartSendBlocking.

Parameters

handle	UART handle pointer.

Return values

kStatus_Success	Get successfully abort the sending.

14.6.12 hal_uart_status_t HAL_UartEnterLowpower (hal_uart_handle_t *handle*)

This function is used to prepare to enter low power consumption.

Parameters

handle	UART handle pointer.

Return values

kStatus_HAL_Uart-Success	Successful operation.

kStatus_HAL_UartError	An error occurred.

14.6.13 hal_uart_status_t HAL_UartExitLowpower (hal_uart_handle_t *handle*)

This function is used to restore from low power consumption.

Parameters

handle	UART handle pointer.

Return values

kStatus_HAL_Uart-Success	Successful operation.
kStatus_HAL_UartError	An error occurred.

14.6.14 void HAL_UartIsrFunction (hal_uart_handle_t *handle*)

This function handles the UART transmit and receive IRQ request.

Parameters

handle	UART handle pointer.

Chapter 15
Lpuart_edma_driver

15.1 Overview

Data Structures

- struct lpuart_edma_handle_t
  LPUART eDMA handle. More...

Typedefs

- typedef void(∗ lpuart_edma_transfer_callback_t)(LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, status_t status, void ∗userData)
  LPUART transfer callback function.

Driver version

- #define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  LPUART EDMA driver version.

eDMA transactional

- void LPUART_TransferCreateHandleEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, lpuart_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗txEdmaHandle, edma_handle_t ∗rxEdmaHandle)
  Initializes the LPUART handle which is used in transactional functions.
- status_t LPUART_SendEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  Sends data using eDMA.
- status_t LPUART_ReceiveEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  Receives data using eDMA.
- void LPUART_TransferAbortSendEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle)
  Aborts the sent data using eDMA.
- void LPUART_TransferAbortReceiveEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle)
  Aborts the received data using eDMA.
- status_t LPUART_TransferGetSendCountEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, uint32_t ∗count)
  Gets the number of bytes written to the LPUART TX register.
- status_t LPUART_TransferGetReceiveCountEDMA (LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, uint32_t ∗count)
  Gets the number of received bytes.
- void LPUART_TransferEdmaHandleIRQ (LPUART_Type ∗base, void ∗lpuartEdmaHandle)
  LPUART eDMA IRQ handle function.

15.2 Data Structure Documentation

15.2.1 struct _lpuart_edma_handle

Data Fields

- lpuart_edma_transfer_callback_t callback
  Callback function.
- void * userData
  LPUART callback function parameter.
- size_t rxDataSizeAll
  Size of the data to receive.
- size_t txDataSizeAll
  Size of the data to send out.
- edma_handle_t * txEdmaHandle
  The eDMA TX channel used.
- edma_handle_t * rxEdmaHandle
  The eDMA RX channel used.
- uint8_t nbytes
  eDMA minor byte transfer count initially configured.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.

Field Documentation

(1) lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback

(2) void* lpuart_edma_handle_t::userData

(3) size_t lpuart_edma_handle_t::rxDataSizeAll

(4) size_t lpuart_edma_handle_t::txDataSizeAll

(5) edma_handle_t* lpuart_edma_handle_t::txEdmaHandle

(6) edma_handle_t* lpuart_edma_handle_t::rxEdmaHandle

(7) uint8_t lpuart_edma_handle_t::nbytes

(8) volatile uint8_t lpuart_edma_handle_t::txState

15.3 Macro Definition Documentation

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

15.4 Typedef Documentation

15.4.1 typedef void(∗ lpuart_edma_transfer_callback_t)(LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, status_t status, void ∗userData)

15.5 Function Documentation

15.5.1 void LPUART_TransferCreateHandleEDMA (LPUART_Type ∗ *base,* lpuart_edma_handle_t ∗ *handle,* lpuart_edma_transfer_callback_t *callback,* void ∗ *userData,* edma_handle_t ∗ *txEdmaHandle,* edma_handle_t ∗ *rxEdmaHandle*)

Note

This function disables all LPUART interrupts.

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.
callback	Callback function.
userData	User data.
txEdmaHandle	User requested DMA handle for TX DMA transfer.
rxEdmaHandle	User requested DMA handle for RX DMA transfer.

15.5.2 status_t LPUART_SendEDMA (LPUART_Type ∗ *base,* lpuart_edma_handle_t ∗ *handle,* lpuart_transfer_t ∗ *xfer*)

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

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART eDMA transfer structure. See lpuart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_LPUART_TxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

15.5.3 status_t LPUART_ReceiveEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, lpuart_transfer_t * *xfer*)

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

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.
xfer	LPUART eDMA transfer structure, see lpuart_transfer_t.

Return values

kStatus_Success	if succeed, others fail.
kStatus_LPUART_Rx-Busy	Previous transfer ongoing.
kStatus_InvalidArgument	Invalid argument.

15.5.4 void LPUART_TransferAbortSendEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*)

This function aborts the sent data using eDMA.

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.

15.5.5 void LPUART_TransferAbortReceiveEDMA (LPUART_Type * *base,* lpuart_edma_handle_t * *handle*)

This function aborts the received data using eDMA.

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.

15.5.6 status_t LPUART_TransferGetSendCountEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferInProgress	No send in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

15.5.7 status_t LPUART_TransferGetReceiveCountEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, uint32_t * *count*)

This function gets the number of received bytes.

Parameters

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

Return values

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

15.5.8 void LPUART_TransferEdmaHandleIRQ (LPUART_Type * *base,* void * *lpuartEdmaHandle*)

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note

This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

base	LPUART peripheral base address.
lpuartEdmaHandle	LPUART handle pointer.

Chapter 16
Debugconsole

16.1 Overview

Macros

- #define PRINTF_FLOAT_ENABLE 0U
  Definition to printf the float number.
- #define SCANF_FLOAT_ENABLE 0U
  Definition to scanf the float number.
- #define PRINTF_ADVANCED_ENABLE 0U
  Definition to support advanced format specifier for printf.
- #define SCANF_ADVANCED_ENABLE 0U
  Definition to support advanced format specifier for scanf.

Typedefs

- typedef void(∗ printfCb)(char ∗buf, int32_t ∗indicator, char val, int len)
  A function pointer which is used when format printf log.

Enumerations

- enum _debugconsole_scanf_flag {
  kSCANF_Suppress = 0x2U,
  kSCANF_DestMask = 0x7cU,
  kSCANF_DestChar = 0x4U,
  kSCANF_DestString = 0x8U,
  kSCANF_DestSet = 0x10U,
  kSCANF_DestInt = 0x20U,
  kSCANF_DestFloat = 0x30U,
  kSCANF_LengthMask = 0x1f00U,
  kSCANF_TypeSinged = 0x2000U }
  Specification modifier flags for scanf.

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.

16.2 Macro Definition Documentation

16.2.1 #define PRINTF_FLOAT_ENABLE 0U

16.2.2 #define SCANF_FLOAT_ENABLE 0U

16.2.3 #define PRINTF_ADVANCED_ENABLE 0U

16.2.4 #define SCANF_ADVANCED_ENABLE 0U

16.3 Enumeration Type Documentation

16.3.1 enum _debugconsole_scanf_flag

Enumerator

kSCANF_Suppress Suppress Flag.
kSCANF_DestMask Destination Mask.
kSCANF_DestChar Destination Char Flag.
kSCANF_DestString Destination String FLag.
kSCANF_DestSet Destination Set Flag.
kSCANF_DestInt Destination Int Flag.
kSCANF_DestFloat Destination Float Flag.
kSCANF_LengthMask Length Mask Flag.
kSCANF_TypeSinged TypeSinged Flag.

16.4 Function Documentation

16.4.1 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

16.4.2 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 "".

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support