MCUXpresso SDK Release Notes Supporting JN5189

Change Logs

Contents

Introduction

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

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

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

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

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

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

Driver Change Log

.1 ADC

The current ADC driver version is 2.3.2.

- 2.3.2

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

Table .2: MCUXpresso SDK Folder Structure

  - Improvements
    - Added delay after enabling using the ADC GPADC_CTRL0 LDO_POWER_EN bit for JN5189/QN9090.
  - New Features
    - Added support for platforms which have only one ADC sequence control/result register.
- 2.3.1
  - Bugfix:
    - Avoid write ADC STARTUP register in ADC_Init().
    - Fix coverity zero divider error in ADC_DoSelfCalibration().
- 2.3.0
  - Updated "ADC_Init()""ADC_GetChannelConversionResult()" API and "adc_resolution_t" structure to match QN9090.
  - Add "ADC_EnableTemperatureSensor" API
- 2.2.1
  - Improvement:
    - Add a brief delay in uSec after ADC calibration start.
- 2.2.0
  - Updated "ADC_DoSelfCalibration" API and "adc_config_t" structure to match LPC845
- 2.1.0
  - Renamed "ADC_EnableShresholdCompareInterrupt" to "ADC_EnableThresholdCompare-Interrupt".
- 2.0.0
  - Initial version.

.2 AES

The current AES driver version is 2.0.1.

- 2.0.1
  - GCM constant time tag comparison.
- 2.0.0
  - Initial version.

.3 COMMON

The current COMMON driver version is 2.2.0.

- 2.2.0
  - New Features
    - Moved SDK_DelayAtLeastUs function from clock driver to common driver.
- 2.1.4
  - New Features
    - Added OTFAD into status group.
- 2.1.3
  - Bug Fixes
    - MISRA C-2012 issue fixed.
      - Fixed the rule: rule-10.3.
- 2.1.2
  - Improvements
    - Add SUPPRESS_FALL_THROUGH_WARNING() macro for the usage of suppressing fallthrough warning.
- 2.1.1
  - Bug Fixes
    - Deleted and optimized repeated macro.
- 2.1.0
  - New Features
    - Added IRQ operation for XCC toolchain.
    - Added group IDs for newly supported drivers.
- 2.0.2
  - Bug Fixes
    - MISRA C-2012 issue fixed.
      - Fixed the rule: rule-10.4.
- 2.0.1
  - Improvements
    - Removed the implementation of LPC8XX Enable/DisableDeepSleepIRQ() function.
    - Added new feature macro switch "FSL_FEATURE_HAS_NO_NONCACHEABLE_SECTION" for specific SoCs which have no noncacheable sections, that helps avoid an unnecessary complex in link file and the startup file.
    - Updated the align(x) to **attribute**(aligned(x)) to support MDK v6 armclang compiler.

- 2.0.0
  - Initial version.

.4 CTIMER

The current CTimer driver version is 2.1.0.

- 2.1.0
  - Improvements
    - Added the CTIMER_GetOutputMatchStatus() API Interface.
    - Added feature macro for FSL_FEATURE_CTIMER_HAS_NO_CCR_CAP2 and FSL_FEATURE_CTIMER_HAS_NO_IR_CR2INT.
- 2.0.3
  - Bug Fixes
    - MISRA C-2012 issue fixed: rule 10.3, 10.4, 10.6, 10.7 and 11.9.
- 2.0.2
  - Add new API "CTIMER_GetTimerCountValue" to get the current timer count value.
  - Add control macro to enable/disable the RESET and CLOCK code in current driver.
  - Add new feature macro to update the API of ctimer driver for lpc8n04.
- 2.0.1
  - API Interface Change Added CTIMER_SetupPwmPeriod and CTIMER_UpdatePwmPulsePeriod API. These two APIs can set up the right PWM with high resolution.
- 2.0.0
  - Initial version.

.5 DMA

The current DMA driver version is 2.4.0.

- 2.4.0
- Improvements:
  - Added new apis DMA_LoadChannelDescriptor/DMA_ChannelIsBusy to support polling transfer case.
- 2.3.0
  - Bug fix:
    - Remove DMA_HandleIRQ prototype definition from header file.
    - Add DMA_IRQHandle prototype definition in header file.
- 2.2.5
  - Improvement:
    - Add new api DMA_SetupChannelDescriptor to support configure wrap descriptor.
    - Add wrap support in function DMA_SubmitChannelTransfer.
- 2.2.4
  - Bug fix:
    - Fix the macro DMA_CHANNEL_CFER use wrong parameter to calculate DSTINC issue.

- 2.2.3
  - Bug fix:
    - Improve DMA driver Deinit function for correct logic order.
  - Improvement:
    - Add api DMA_SubmitChannelTransferParameter to support create head descriptor directly.
    - Add api DMA_SubmitChannelDescriptor to support ping pong transfer.
    - Add macro DMA_ALLOCATE_HEAD_DESCRIPTOR/DMA_ALLOCATE_LINK_DESCRIPTOR to simplify dma descriptor allocation.
- 2.2.2
  - Bug fix:
    - Do not use software trigger when hardware trigger is enabled.
- 2.2.1
  - Bug fix:
    - Fix coverity issue.
- 2.2.0
  - Improvement:
    - Change api DMA_SetupDMADescriptor to non-static.
    - Mark below api as deprecated. DMA_PrepareTransfer. DMA_Submit transfer.
    - Added below new api: DMA_SetChannelConfig. DMA_PrepareChannelTransfer. DMA_InstallDescriptorMemory. DMA_SubmitChannelTransfer. DMA_SetChannelConfigValid. DMA_DoChannelSoftwareTrigger. DMA_LoadChannelTransferConfig.
- 2.0.1
  - Improvement:
    - Add volatile for dma descriptor member xfercfg to avoid optimization.
- 2.0.0
  - Initial version.

.6 DMIC

The current DMIC driver version is 2.1.1.

- 2.1.1
  - Add feature FSL_FEATURE_DMIC_HAS_NO_IOCFG for IOCFG register.
- 2.1.0
  - New feature
    - Add api DMIC_EnbleChannelInterrupt/DMIC_EnbleChannelDma to replace api DMIC_SetOperationMode.
    - Add api DMIC_SetIOCFG and mark DMIC_ConfigIO as deprecated.
    - Add api DMIC_EnableChannelSignExtend to support sign extend feature.
- 2.0.5
  - Improvment
    - Change some parameters value of DMIC_FifoChannel API, such as enable,resetn,trig_level.this is not possible for current code logic.so improve DMIC_FifoChannel logic and

fix incorrect math logic.

- 2.0.4
  - Bugfix
    - Fix DMIC DMA driver(ver2.0.3) not support call DMIC_TransferReceiveDMA in DMA callback which is supported before 2.0.3, but call DMIC_TransferReceiveDMA in callback is not recommanded.
- 2.0.3
  - New feature
    - Support linked transfer in dmic dma driver.
    - Add new api DMIC_EnableChannelFifo/DMIC_DoFifoReset/DMIC_InstallDMADescriptor.
- 2.0.2
  - New feature
    - Support more channel in driver.
- 2.0.1
  - Add control macro to enable/disable the RESET and CLOCK code in current driver.
- 2.0.0
  - Initial version.

.7 FLASH

Current Flash driver version is 2.0.0

- 2.0.0
  - Initial version.

.8 FLEXCOMM

The current FLEXCOMM driver version is 2.0.1.

- 2.0.2
  - Fix typo in FLEXCOMM15_DriverIRQHandler().
  - Add instance calculation in FLEXCOMM16_DriverIRQHandler() to align with Flexcomm 14 and 15.
- 2.0.1
  - Adding more IRQHandler code in drivers to adapt the new devices.
- 2.0.0
  - Initial version.

.9 I2C

The current I2C driver version is 2.0.6.

- 2.0.6
  - New Features
    - Added master timeout self-recovery support for feature FSL_FEATURE_I2C_TIMEOUT_RECOVERY.
- 2.0.5
  - Bug fixes:
    - Fix wrong assignment for datasize in I2C_InitTransferStateMachineDMA.
    - Fix wrong working flow in I2C_RunTransferStateMachineDMA to ensure master can work in no start flag and no stop flag mode.
    - Fix wrong working flow in I2C_RunTransferStateMachine and add kReceiveDataBeginState in _i2c_transfer_states to ensure master can work in no start flag and no stop flag mode.
    - Fix wrong handle state in I2C_MasterTransferDMAHandleIRQ. After all the data has been transfered or nak is returned, handle state should change to idle.
- 2.0.4
  - Improvements:
    - Updated the I2C_WATI_TIMEOUT macro to unified name I2C_RETRY_TIMES
    - Update the "I2C_MasterSetBaudRate" API to support baudrate configuration for feature QN9090.
  - Bug fixes:
    - Fix build warnning caused by uninitialized variable.
    - Fix coverity issue of unchecked return value in I2C_RTOS_Transfer.
- 2.0.3
  - Unify component full name to FLEXCOMM I2C(DMA/FREERTOS) Driver
- 2.0.2
  - Improvements: In slave IRQ:
    1. Changed slave receive process to first set the I2C_SLVCTL_SLVCONTINUE_MASK to ack the received data, then do data receive.
    2. Improved slave transmit process to set the I2C_SLVCTL_SLVCONTINUE_MASK immediately after write the data.
- 2.0.1
  - Improvements:
    - Added I2C_WATI_TIMEOUT macro to allow user to specify the timeout times for waiting flags in functional API and blocking transfer API.
- 2.0.0
  - Initial version.

.10 I2S

The current I2S driver version is 2.1.0.

- 2.1.0
  - Improvement:
    - Add feature for the FLEXCOMM which support I2S and has interconnection with DMIC.

        * Use feature to control PDMDATA instead of I2S_CFG1_PDMDATA.
        * Add member bytesPerFrame in i2s_dma_handle_t, used for dma transfer width configure instead of use sizeof(uint32_t) hardcode.
        * Use the macro provide by DMA driver to define the I2S dma descriptor.
    - Bug Fix:
        * Fix i2s dma driver always generate duplicate callback.
- 2.0.3
    - Add feature to remove configuration for second channel on lpc51u68.
- 2.0.2
    - Add ENABLE_IRQ handle after register i2s interrupt handle
- 2.0.1
    - Unify component full name to FLEXCOMM I2S(DMA) Driver
- 2.0.0
    - Initial version.

.11 SPI

The current SPI driver version is 2.0.5.

- 2.0.5
    - Bug Fixes
        * Fixed Coverity issue of incrementing null pointer in SPI_TransferHandleIRQInternal.
- 2.0.4
    - Bug Fixes
        * Fixed the bug of using read only mode in DMA transfer. In DMA transfer mode, if transfer->txData is NULL, code attempts to read data from the address of 0x0 for configuring the last frame.
        * Fixed wrong assignment of handle->state. During transfer handle->state should be kSPI_Busy rather than kStatus_SPI_Busy.
    - Improvements
        * Rounded up the calculated divider value in SPI_MasterSetBaud.
    - New Features
        * Modified the definition of SPI_SSELPOL_MASK to support the socs that have only 3 SSEL pins.
- 2.0.3
    - Add "SPI_FIFO_DEPTH(base)" more definition.
- 2.0.2
    - Unify component full name to FLEXCOMM SPI(DMA/FREERTOS) Driver
- 2.0.1
    - Changed the data buffer from uint32_t to uint8_t which matches the real applications for SPI DMA driver.
- Added dummy data setup API to allow users to configure the dummy data to be transferred.
    - Added new APIs for half-duplex transfer function, users can send and receive data by one API in polling/interrupt/DMA way, and users can choose either transmit first or receivefirst.

Besides, the PCS pin can be configured as assert status in transimission (between transmit and receive) by setting the isPcsAssertInTransfer to true.

- 2.0.0
  - Initial version.

.12 USART

- The current USART driver version is 2.1.0.
- 2.1.0
  - New feature:
    - Adding features to allow users configure the USART to synchronous transfer(master and slave) mode.
  - Bug fix:
    - Modified USART_SetBaudRate to get more acurate configuration.
- 2.0.3
  - New feature:
    - Add new API to allow users enable the CTS which determines whether CTS is used for flow control.
- 2.0.2
  - Bug Fix:
    - Fix the bug of transfer abort APIs can not disable the interrupts, the FIFOINTENSET register should not be used to disable the insterrupts, using FIFOINTENCLR register instead.
- 2.0.1
  - Unify component full name to FLEXCOMM USART(DMA/FREERTOS) Driver
- 2.0.0
  - Initial version.

.13 FMEAS

The current FMEAS driver version is 2.1.1.

- 2.1.1
  - Add a "FMEAS_StartMeasureWithScale" API to support qn9090.
- 2.1.0
  - Update "FMEAS_GetFrequency","FMEAS_StartMeasure","FMEAS_IsMeasureComplete" API and add definition to match ASYNC_SYSCON.
- 2.0.0
  - Initial version ported from LPCOpen.

.14 GINT

The current GINT driver version is 2.0.01.

- 2.0.1
  - Add control macro to enable/disable the RESET and CLOCK code in current driver.
- 2.0.0
  - Initial version.

.15 GPIO

The current GPIO driver version is 2.1.3.

- 2.1.4
  - Add API GPIO_PortGetInterruptStatus to retrieve interrupt status for whole port
  - Correct typo in header file
- 2.1.3:
  - Update "GPIO_PinInit" API. if it have DIRCLR and DIRSET registers,using them set 1 or clean 0.
- 2.1.2:
  - Remove deprecated APIs.
- 2.1.1:
  - API interface changes:
    - Refined naming of API while keep all original APIs, marking them as deprecated. Original API will be removed in next release. The mainin change is update API with prefix of _-PinXXX() and _PorortXXX
- 2.1.0
  - Added GPIO initialize API.
- 2.0.0
  - Initial version.

.16 INPUTMUX

The current INPUTMUX driver version is 2.0.1.

- 2.0.1
  - Support channel mux setting in INPUTMUX_EnableSignal().
- 2.0.0
  - Initial version.

.17 IOCON

Current IOCON driver version is 2.0.0

- 2.0.0
  - Initial version.

.18 IRM

Current Infra-Red Modulator driver version is 2.0.0

- 2.0.0
  - Initial version.

.19 PINT

The current PINT driver version is 2.1.3.

- 2.1.3
  - Bug fix:
    - Updated PINT_PinInterruptClrStatus to clear PINT interrupt status when the bit is asserted and check whether was triggered by edge-sensitive mode.
    - Write 1 to IST corresponding bit will clear interrupt status only in edge-sensitive mode and will switch the active level for this pin in level-sensitive mode.
    - Fix MISRA c-2012 rule 10.1, rule 10.6, rule 10.7.
    - Add FSL_FEATURE_SECPINT_NUMBER_OF_CONNECTED_OUTPUTS to distinguish IRQ relevant array definitions for SECPINT/PINT on lpc55s69 board.
    - Fix PINT driver c++ build error and remove index offset operation.
- 2.1.2
  - Improvement:
    - Improve the way of initialization for SECPINT/PINT in PINT_Init API.
- 2.1.1
  - Inprovement
    - Enable secure pint interrupt and add secure interrupt handle.
- 2.1.0
  - Add PINT_EnableCallbackByIndex/PINT_DisableCallbackByIndex APIs to enable/disable callback by index.
- 2.0.2
  - Add control macro to enable/disable the RESET and CLOCK code in current driver.
- 2.0.1
  - Bug fix:
    - Updated PINT driver to clear interrupt only in Edge sensitive.
- 2.0.0
  - Initial version.

.20 PWM

Current PWM driver version is 2.0.0

- 2.0.0
  - Initial version.

.21 RNG

Current RNG driver version is 2.0.0

- 2.0.0
  - Initial version.

.22 RTC

The current RTC driver version is 2.0.0.

- 2.0.0
  - Initial version.

.23 SPI Flash Interface

The current SPIFI driver version is 2.0.2.

- 2.0.2
  - Fixed the set command function issue. After the command set, there is no wait for the CMD flag, as it may be cleared by CS deassert.
- 2.0.1
  - Added API to read/write 1/2 Bytes data from/to SPIFI. This interface are useful for flash command, which only needs 1/2 Bytes data. The previous driver needed users to make sure the minimum length should be 4, which may have issues in some flash commands.
- 2.0.0
  - Initial version.

.24 WWDT

The current WWDT driver version is 2.1.6.

- 2.1.6
  - Bug Fixes
    - Fixed the issue that the watchdog reset event affected the system from PMC.

        * Fixed the issue of setting watchdog WDPROTECT field without considering the backwards compatibility.
        * Fixed the issue of clearing bit fields by mistake in the function of WWDT_ClearStatusFlags.
- 2.1.5
  - Bug Fixes
    * deprecated a unusable API in WWWDT driver.
      · WWDT_Disable
- 2.1.4
  - Bug Fixes
    * Fixed violation of the MISRA C-2012 rules Rule 10.1, 10.3, 10.4 and 11.9.
    * Fixed the issue of the inseparable process interrupted by other interrupt source.
      · WWDT_Init
- 2.1.3
  - Bug Fixes
    * Fixed legacy issue when initializing the MOD register.
- 2.1.2
  - Improvements
    * Updated the "WWDT_ClearStatusFlags" API and "WWDT_GetStatusFlags" API to match QN9090. WDTOF is not set in case of WD reset. Get info from PMC instead.
- 2.1.1
  - New Features
    * Added new feature definition macro for devices which have no LCOK control bit in MOD register.
    * Implemented delay/retry in WWDT driver.
- 2.1.0
  - Improvements
    * Added new parameter in configuration when initializing WWDT module. This parameter, which must be set, allows the user to deliver the WWDT clock frequency.
- 2.0.0
  - Initial version.

.25 CMP

Current Cmp driver version is 2.0.1

- 2.0.1
  - Add get INTSTAT register status function.
- 2.0.0
  - Initial version.

.26 SHA

The current SHA driver version is 2.1.0.

- 2.1.0
  - Update "sha_ldm_stm_16_words" "sha_one_block" API to match QN9090. QN9090 have no ALIAS register.
  - Add "SHA_ClkInit" "SHA_ClkInit"
- 2.0.0
  - Initial version.

.27 NTAG

Current NTAG driver version is 2.0.0

- 2.0.0
  - Initial version.

Trademarks

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

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

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

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUI-

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

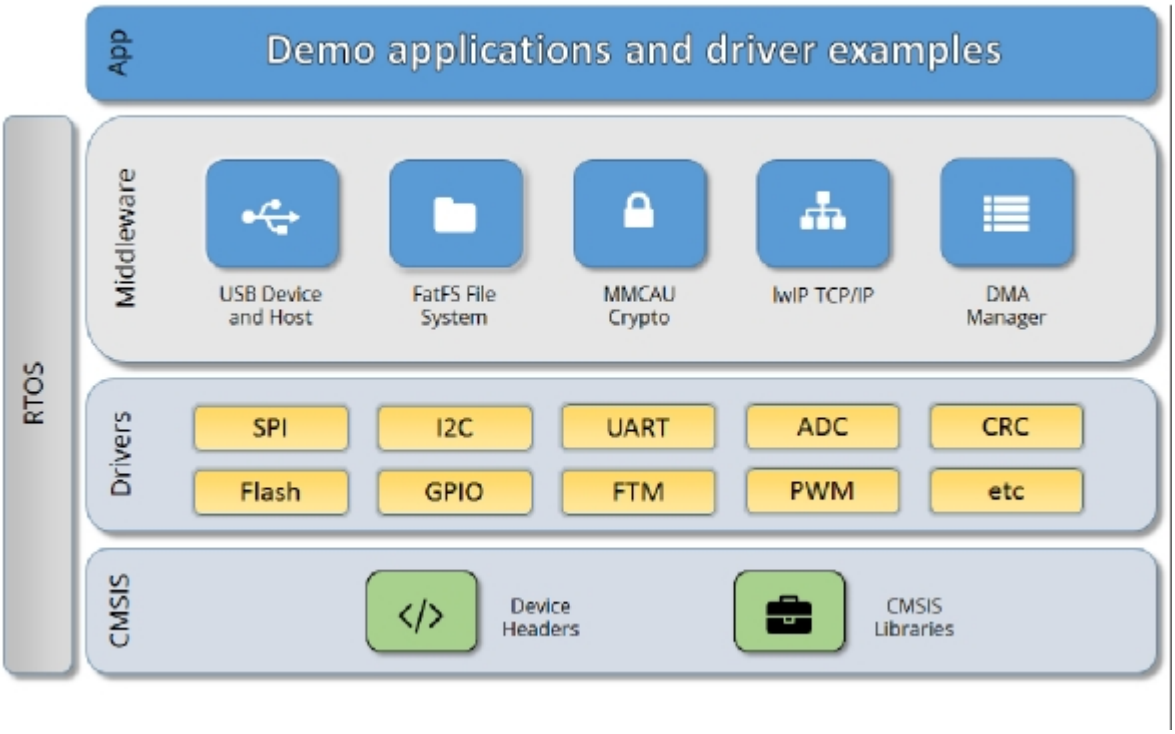

Figure .1: MCUXpresso SDK Block Diagram

MCU header files

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

CMSIS Support

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

MCUXpresso SDK Peripheral Drivers

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

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

Peripheral drivers handle the peripheral clock gating/ungating inside the drivers during initialization and deinitialization respectively.

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

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

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

Interrupt handling for transactional APIs

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

```
        PUBWEAK SPI0_IRQHandler
        PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
        LDR     R0, =SPI0_DriverIRQHandler
        BX      R0
```

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

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

The limitation of the double weak mechanism is that it cannot be used for peripherals that share the same vector entry. For this use case, redefine the first layer weak function to enable the desired peripheral

interrupt functionality. For example, if the MCU's UART0 and UART1 share the same vector entry, redefine the UART0_UART1_IRQHandler according to the use case requirements.

Feature Header Files

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

Application

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

Module Documentation

.1 Clock: Clock driver

The MCUXpresso SDK provides a clock driver for MCUXpresso SDK devices.

.1.1 Function groups

Clock driver provides these functions:

- Functions to obtain frequency of specified clock
- Functions to configure the clock selection muxes.
- Functions to setup peripheral clock dividers
- Functions to enable specific AHB clock channel

SYSCON Clock frequency functions

SYSCON clock module provides clocks, such as ADCCLK, DMICCLK, FXCOMCLK,WDTOSC, RTCOSC and SYSPLL. The functions CLOCK_EnableClock() and CLOCK_DisableClock() enables and disables the various clocks. The SYSCON clock driver provides functions to get the frequency of clocks, such as CLOCK_GetFreq(),

SYSCON clock Selection Muxes

The SYSCON clock driver provides the function to configure the clock selected. The function CLOCK_AttachClk() is implemented for this. The function selects the clock source for a particular peripheral like MAINCLK, DMIC, FLEXCOMM, USB, ADC and PLL.

SYSCON clock dividers

The SYSCON clock module provides the function to setup the peripheral clock dividers. The function CLOCK_SetClkDiv() configures the CLKDIV registers for various periperals like USB, DMIC, I2S, SYSTICK, AHB, ADC and also for CLKOUT and TRACE functions.

.2 Power: Power driver

The MCUXpresso SDK provides a power driver for the MCUXpresso SDK devices.

.2.1 Function description

Power driver and library provides these functions:

- Functions to enable and disable power to different peripherals
- Functions to obtain power down config structure with default parameters
- Functions to determine cause of reset
- Functions to get power Library API to return the library version.

Power enable and disable

Power driver provides two API's POWER_EnablePD() and POWER_DisablePD() to enable or disable the PDRUNCFG bits in SYSCON The PDRUNCFG has an inverted logic i.e. the peripheral is powered on when the bit is cleared and powered off when bit is set. So the API POWER_DisablePD() is used to power on a peripheral and POWER_EnablePD() is used to power off a peripheral. The API's take a parameter of type pd_bit_t which organizes the PDRUNCFG bits. The driver also provides two separate API's to power down and power up Flash, POWER_PowerDownFlash() and POWER_PowerUpFlash()

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

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

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.