Kinetis SDK API Reference Manual

Freescale Semiconductor, Inc.

KSDKAPIRM
1.0.0-beta
Mar 2014

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

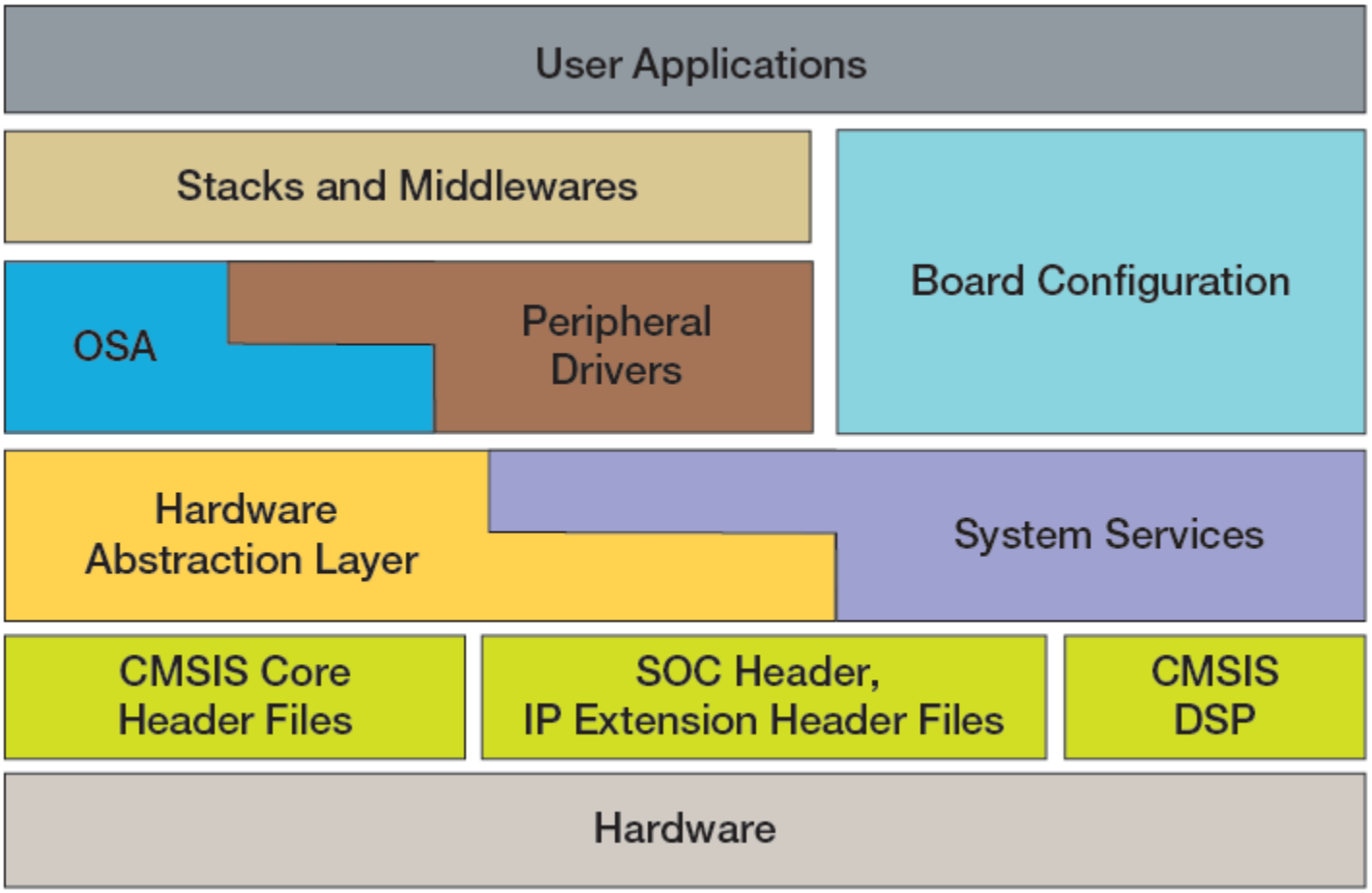
Contents

Contents

Chapter 1
Introduction

The Kinetis software development kit (SDK) is an extensive suite of robust peripheral drivers, stacks and middleware designed to simplify and accelerate application development on any Freescale Kinetis MC-U. The addition of Processor Expert technology for software and board support configuration provides unmatched ease of use and flexibility. The Kinetis SDK is complimentary and includes full source code under a permissive open-source license for all hardware abstraction and peripheral driver software.

The Kinetis SDK consists of the following runtime software components written in C:



- ARM CMSIS Core and DSP standard libraries and CMSIS-compliant device header files which provide direct access to the peripheral registers and bits
- An open-source hardware abstraction layer (HAL) that provides a simple, stateless driver with an API encapsulating the low-level functions of the peripheral
- ∗System services for centralized resources including a clock manager, interrupt manager, low-power manager, and a hardware timer

- Open-source, high-level peripheral drivers that build upon the HAL layer and may utilize one or more of the system services; drivers may be used as-is or as a reference for creating custom drivers
- An operating system abstraction (OSA) layer for adapting applications for use with a real time operating system (RTOS) or bare metal (no RTOS) applications. OSAs are provided for:
  - Freescale MQX™ RTOS
  - FreeRTOS
  - Micrium uC/OS-II
  - Micrium uC/OS-III
  - CMSIS-RTOS API compliant RTOS
  - bare-metal (no RTOS)
- Stacks and middleware in source or object formats including:
  - a comprehensive device and host USB stack with comprehensive USB class support
  - CMSIS DSP, a suite of common signal processing functions
  - ∗Freescale MQX™ Real-Time TCP/IP Communication Suite (RTCS)
  - ∗Freescale MQX™ File System (MFS)
  - ∗FatFs, a FAT file system for small embedded systems
  - Encryption software utilizing the mmCAU hardware acceleration unit

The Kinetis SDK comes complete with software examples demonstrating the usage of the HAL, peripheral drivers, middleware, and RTOSes. All examples are provided with projects for the following toolchains:

- ∗Atollic TrueSTUDIO
- IAR Embedded Workbench
- ∗Keil MDK
- ∗Kinetis Design Studio
- GNU toolchain for ARM® Cortex®-M with makefile system

The HAL, peripheral drivers and system services can be used across multiple devices within the Kinetis product family without code modification. The configurable items for each driver, at all levels, are encapsulated into C language data structures. Kinetis devices specific configuration information is provided as part of the SDK and need not be modified by the user. HAL, peripheral driver, and system services configuration is not fixed and can be changed at runtime. Optionally, the entire driver set can be pre-built into a library.

The example applications demonstrate how to configure the drivers by passing configuration data to the APIs. In addition to the software source, Processor Expert is provided as a time-saving option for software configuration. Processor Expert is a complimentary PC-hosted software configuration tool (Eclipse plug-in) with complete knowledge of all Kinetis MCUs. It provides a graphical user interface to handle MCU-specific board configuration and driver tuning tasks including:

- Optional generation of low-level device initialization code for post-reset configuration
- Package I/O allocation and pin initialization source code generation
- Creation and management of HAL and peripheral driver C source configuration data structures

Note – items indicated with as asterisk above are not completed and/or not included in this 1.0.0 Beta release of the Kinetis SDK. See the release notes for the set of features supported. The indicated items are planned for a 1.0.0 General Availability (GA) release in Summer 2014. Please contact your local Freescale Representative for additional future release information.

The organization of files in the Kinetis SDK release package is focused on ease-of-use. The Kinetis SDK folder hierarchy is organized at the top level with these folders:

<table>
<tr><td>platform</td><td>Platform folder contains code for SDK HAL and Peripheral drivers. The folder contain following sub-folders

<table>
<tr><td>drivers</td><td>Peripheral drivers for every peripheral supported
For each peripheral there is a sub-folder under it containing following files/folders:
<table>
<tr><td>Source</td><td>Source folder containing .c files for the driver code</td></tr>
<tr><td>fsl_ipname_driver.h</td><td>Peripheral driver API header file</td></tr>
</table></td></tr>
<tr><td>hal</td><td>HAL drivers for every peripheral supported
For each peripheral there is a sub-folder under it containing following files:
<table>
<tr><td>fsl_ipname_hal.c</td><td>HAL driver source code file</td></tr>
<tr><td>fsl_ipname_hal.h</td><td>HAL driver API header file</td></tr>
<tr><td>fsl_ipname_features.h</td><td>Features header file for the IP defining features supported for specific chipset</td></tr>
</table></td></tr>
<tr><td>CMSIS</td><td>Platform specific header files, CMSIS header files and CMSIS DSP library</td></tr>
<tr><td>startup</td><td>Chip specific CMSIS compliant startup code</td></tr>
<tr><td>utilities</td><td>Tools</td></tr>
<tr><td>linker</td><td>Board specific linker files</td></tr>
</table></td></tr>
<tr><td>apps</td><td>Contain demo applications code</td></tr>
<tr><td>doc</td><td>User Guide and Quick start guides for each chip supported</td></tr>
<tr><td>boards</td><td>Board specific code</td></tr>
<tr><td>lib</td><td>Prebuilt libraries for each tool chain and boards supported</td></tr>
<tr><td>mk</td><td>Common make files used for compiling with GCC</td></tr>
<tr><td>rtos</td><td>RTOS abstraction layer code for supported RTOSes</td></tr>
<tr><td>usb</td><td>Freescale's USB stack code</td></tr>
</table>

The rest of the document describes the API references in detail for HAL and Peripheral drivers.

Chapter 2
Analog-to-Digital Converter (ADC)

The Kinetis SDK provides both HAL and Peripheral drivers for the Analog-to-Digital Converter (ADC) block of Kinetis devices.

Modules

- ADC HAL Driver
  The part describes the programming interface of the ADC HAL driver.
- ADC Peripheral Driver
  The part describes the programming interface of the ADC Peripheral driver.

2.1 ADC HAL Driver

The chapter describes the programming interface of the ADC HAL driver.

Enumerations

- enum adc_clock_source_mode_t {
  kAdcClockSourceBusClk = 0U,
  kAdcClockSourceBusClk2 = 1U,
  kAdcClockSourceAlternate = 2U,
  kAdcClockSourceAsynchrounous = 3U }
  Defines the selection of the clock source that ADC module uses.
- enum adc_clock_divider_mode_t {
  kAdcClockDivider1 = 0U,
  kAdcClockDivider2 = 1U,
  kAdcClockDivider4 = 2U,
  kAdcClockDivider8 = 3U }
  Defines the selection of the clock divider.
- enum adc_reference_voltage_mode_t {
  kAdcVoltageVref = 0U,
  kAdcVoltageValt = 1U }
  Defines the selection of the voltage source that ADC module uses.
- enum adc_long_sample_mode_t {
  kAdcLongSampleExtra20 = 0U,
  kAdcLongSampleExtra12 = 1U,
  kAdcLongSampleExtra6 = 2U,
  kAdcLongSampleExtra2 = 3U }
  Defines the selection of the long sample extra cycle configuration.
- enum adc_resolution_mode_t {
  kAdcSingleDiff8or9 = 0U,
  kAdcSingleDiff12or13 = 1U,
  kAdcSingleDiff10or11 = 2U,
  kAdcSingleDiff16 = 3U }
  Defines the selection of the sample resolution.
- enum adc_group_mux_mode_t {
  kAdcChannelMuxA = 0U,
  kAdcChannelMuxB = 1U }
  Defines the selection of the A/B group mux.
- enum adc_hw_average_mode_t {
  kAdcHwAverageCount4 = 0U,
  kAdcHwAverageCount8 = 1U,
  kAdcHwAverageCount16 = 2U,
  kAdcHwAverageCount32 = 3U }
  Defines the selection of the time in a hard average mode.
- enum adc_channel_mode_t {

kAdcChannel0 = 0U,
kAdcChannelI = 1U,
kAdcChannel2 = 2U,
kAdcChannel3 = 3U,
kAdcChannel4 = 4U,
kAdcChannel5 = 5U,
kAdcChannel6 = 6U,
kAdcChannel7 = 7U,
kAdcChannel8 = 8U,
kAdcChannel9 = 9U,
kAdcChannel10 = 10U,
kAdcChannel11 = 11U,
kAdcChannel12 = 12U,
kAdcChannel13 = 13U,
kAdcChannel14 = 14U,
kAdcChannel15 = 15U,
kAdcChannel16 = 16U,
kAdcChannel17 = 17U,
kAdcChannel18 = 18U,
kAdcChannel19 = 19U,
kAdcChannel20 = 20U,
kAdcChannel21 = 21U,
kAdcChannel22 = 22U,
kAdcChannel23 = 23U,
kAdcChannelTemperature = 26U,
kAdcChannelBandgap = 27U,
kAdcChannelReferenceVoltageHigh = 29U,
kAdcChannelReferenceVoltageLow = 30U,
kAdcChannelDisable = 31U }
Defines the selection of the channel inside the ADC module.

- enum adc_status_t { ,
kStatus_ADC_InvalidArgument = 1U,
kStatus_ADC_Failed = 2U }
Defines the status returned from the ADC API.

Functions

- adc_status_t adc_hal_start_calibration (uint32_t instance)
Starts the calibration process.
- static void adc_hal_end_calibration (uint32_t instance)
Ends the calibration process.
- uint32_t adc_hal_get_calibration_PG (uint32_t instance)
Gets and calculates the plus-side calibration parameter.
- static void adc_hal_set_calibration_PG (uint32_t instance, uint32_t val)
Sets the plus-side calibration parameter to the ADC instance.

- uint32_t adc_hal_get_calibration_MG (uint32_t instance)
  Gets and calculates the minus-side calibration parameter.
- static void adc_hal_set_calibration_MG (uint32_t instance, uint32_t val)
  Sets the minus-side calibration parameter to the ADC instance.
- static uint32_t adc_hal_get_calibration_offset (uint32_t instance)
  Gets the offset value after the auto-calibration.
- static void adc_hal_set_calibration_offset (uint32_t instance, uint32_t value)
  Sets the offset value for manual calibration.
- static void adc_hal_set_clock_source_mode (uint32_t instance, adc_clock_source_mode_t mode)
  Sets the selection of the clock source.
- static void adc_hal_configure_asynchronous_clock (uint32_t instance, bool isEnabled)
  Switches the asynchronous clock on/off.
- static void adc_hal_set_clock_divider_mode (uint32_t instance, adc_clock_divider_mode_t mode)
  Sets the selection of the clock divider.
- static void adc_hal_set_reference_voltage_mode (uint32_t instance, adc_reference_voltage_mode_t mode)
  Sets the selection of the reference voltage source.
- static void adc_hal_configure_high_speed (uint32_t instance, bool isEnabled)
  Switches the high speed mode on/off .
- static void adc_hal_configure_long_sample (uint32_t instance, bool isEnabled)
  Switch the long sample mode on/off.
- static void adc_hal_set_long_sample_mode (uint32_t instance, adc_long_sample_mode_t mode)
  Sets the selection of the long sample mode.
- static void adc_hal_configure_low_power (uint32_t instance, bool isEnabled)
  Switches the low power mode on/off.
- static void adc_hal_set_resolution_mode (uint32_t instance, adc_resolution_mode_t mode)
  Sets the selection of the resolution mode.
- static void adc_hal_configure_continuous_conversion (uint32_t instance, bool isEnabled)
  Switches the continuous conversion mode on/off.
- static void adc_hal_configure_hw_trigger (uint32_t instance, bool isEnabled)
  Switches the hardware trigger mode on/off .
- static void adc_hal_configure_hw_average (uint32_t instance, bool isEnabled)
  Switches the hardware average mode on/off.
- static void adc_hal_set_hw_average_mode (uint32_t instance, adc_hw_average_mode_t mode)
  Sets the selection of the hardware average mode.
- static void adc_hal_configure_hw_compare (uint32_t instance, bool isEnabled)
  Switches the hardware compare mode on/off.
- static void adc_hal_configure_hw_compare_greater (uint32_t instance, bool isEnabled)
  Switches the hardware compare greater configuration on/off .
- static void adc_hal_configure_hw_compare_in_range (uint32_t instance, bool isEnabled)
  Switches the hardware compare range configuration on/off .
- static void adc_hal_set_hw_compare_value1 (uint32_t instance, uint32_t value)
  Sets the value1 in the hardware compare.
- static void adc_hal_set_hw_compare_value2 (uint32_t instance, uint32_t value)
  Sets the value2 in the hardware compare.
- static void adc_hal_configure_dma (uint32_t instance, bool isEnabled)
  Switches the ADC DMA trigger on/off.
- static void adc_hal_disable (uint32_t instance, uint32_t group)
  Switches off the ADC channel conversion.
- static void adc_hal_enable (uint32_t instance, uint32_t group, adc_channel_mode_t mode, bool isDifferential)

Sets the channel number and switches on the conversion.
- static void adc_hal_configure_interrupt (uint32_t instance, uint32_t group, bool isEnabled)
  Switches the ADC interrupt trigger on/off .
- static bool adc_hal_is_in_process (uint32_t instance)
  Checks whether the ADC is in process.
- static bool adc_hal_is_conversion_completed (uint32_t instance, uint32_t group)
  Checks whether the channel conversion is complete.
- static bool adc_hal_is_calibration_fail (uint32_t instance)
  Checks whether the calibration failed.
- static uint32_t adc_hal_get_conversion_value (uint32_t instance, uint32_t group)
  Gets the conversion value.
- static void adc_hal_set_group_mux (uint32_t instance, adc_group_mux_mode_t group)
  Sets the current group mux that executes the conversion.

2.1.0.1 ADC hal driver

Overview

The ADC HAL driver is used to mask the hardware and provide user a comprehensible way to use ADC hardware.

2.1.1 Enumeration Type Documentation

2.1.1.1 enum adc_clock_source_mode_t

Enumerator

kAdcClockSourceBusClk Use bus clock.
kAdcClockSourceBusClk2 Use bus clock / 2.
kAdcClockSourceAlternate Use the optional external clock.
kAdcClockSourceAsynchrounous Use ADC's internal asynchronous clock.

2.1.1.2 enum adc_clock_divider_mode_t

Enumerator

kAdcClockDivider1 Divide 1.
kAdcClockDivider2 Divide 2.
kAdcClockDivider4 Divide 4.
kAdcClockDivider8 Divide 8.

2.1.1.3 enum adc_reference_voltage_mode_t

Enumerator

kAdcVoltageVref Use V_REFH & V_REFL as ref source pin.
kAdcVoltageValt Use V_ALTH & V_REFL as ref source pin.

2.1.1.4 enum adc_long_sample_mode_t

Enumerator

kAdcLongSampleExtra20 Extra 20 cycles, total 24 cycles, default.
kAdcLongSampleExtra12 Extra 12 cycles.
kAdcLongSampleExtra6 Extra 6 cycles.
kAdcLongSampleExtra2 Extra 2 cycles.

2.1.1.5 enum adc_resolution_mode_t

Enumerator

kAdcSingleDiff8or9 8-bits in single-end or 9-bits in differential.
kAdcSingleDiff12or13 12-bits in single-end or 13-bits in differential.
kAdcSingleDiff10or11 10-bits in single-end or 11-bits in differential.
kAdcSingleDiff16 16-bits both in single-end and differential.

2.1.1.6 enum adc_group_mux_mode_t

Enumerator

kAdcChannelMuxA Mux A group is active.
kAdcChannelMuxB Mux B group is active.

2.1.1.7 enum adc_hw_average_mode_t

Enumerator

kAdcHwAverageCount4 Average the result after accumulating 4 conversion.
kAdcHwAverageCount8 Average the result after accumulating 8 conversion.
kAdcHwAverageCount16 Average the result after accumulating 16 conversion.
kAdcHwAverageCount32 Average the result after accumulating 32 conversion.

2.1.1.8 enum adc_channel_mode_t

Enumerator

kAdcChannel0 ADC channel 0.
kAdcChannell ADC channel 1.
kAdcChannel2 ADC channel 2.
kAdcChannel3 ADC channel 3.
kAdcChannel4 ADC channel 4.
kAdcChannel5 ADC channel 5.
kAdcChannel6 ADC channel 6.
kAdcChannel7 ADC channel 7.
kAdcChannel8 ADC channel 8.
kAdcChannel9 ADC channel 9.
kAdcChannel10 ADC channel 10.
kAdcChannel11 ADC channel 11.
kAdcChannel12 ADC channel 12.
kAdcChannel13 ADC channel 13.
kAdcChannel14 ADC channel 14.
kAdcChannel15 ADC channel 15.
kAdcChannel16 ADC channel 16.
kAdcChannel17 ADC channel 17.
kAdcChannel18 ADC channel 18.
kAdcChannel19 ADC channel 19.
kAdcChannel20 ADC channel 20.
kAdcChannel21 ADC channel 21.
kAdcChannel22 ADC channel 22.
kAdcChannel23 ADC channel 23.
kAdcChannelTemperature Internal temperature sensor.
kAdcChannelBandgap Internal band gap.
kAdcChannelReferenceVoltageHigh Internal ref voltage High.
kAdcChannelReferenceVoltageLow Internal ref voltage L.
kAdcChannelDisable Disable the sample process.

2.1.1.9 enum adc_status_t

Enumerator

kStatus_ADC_InvalidArgument Parameter is not available for the current configuration.
kStatus_ADC_Failed Function operation failed.

2.1.2 Function Documentation

2.1.2.1 adc_status_t adc_hal_start_calibration (uint32_t *instance*)

This function clears the calibration flag bit and sets the enable bit to start the calibration.

Parameters

instance	ADC instance ID.

2.1.2.2 static void adc_hal_end_calibration (uint32_t *instance*) [inline], [static]

This function clears the calibration enable bit to end the calibration.

Parameters

instance	ADC instance ID.

2.1.2.3 uint32_t adc_hal_get_calibration_PG (uint32_t *instance*)

This function gets the CLP0 - CLP4 and CLPS, accumulates them, and returns the value that can be set to the PG directly.

Parameters

instance	ADC instance ID.

Returns

the value that can be set to PG directly.

2.1.2.4 static void adc_hal_set_calibration_PG (uint32_t *instance,* uint32_t *val*) [inline], [static]

This function sets the PG register directly.

Parameters

instance	ADC instance ID.
val	the value that can be set to PG directly.

2.1.2.5 uint32_t adc_hal_get_calibration_MG (uint32_t *instance*)

This function gets the CLM0 - CLM4 and CLMS, accumulates them, and returns the value that can be set to the MG directly.

Parameters

instance	ADC instance ID.

Returns

the value that can be set to MG directly.

2.1.2.6 static void adc_hal_set_calibration_MG (uint32_t *instance,* uint32_t *val*) [inline], [static]

This function sets the MG register directly.

Parameters

instance	ADC instance ID.
val	the value that can be set to MG directly.

2.1.2.7 static uint32_t adc_hal_get_calibration_offset (uint32_t *instance*) [inline], [static]

If the user wants to adjust the offset value according to the application, the origin offset value will be a reference.

Parameters

instance	ADC instance ID.

Returns

The offset value created by auto-calibration.

2.1.2.8 static void adc_hal_set_calibration_offset (uint32_t *instance,* uint32_t *value*) [inline], [static]

This function is to set the user selected or calibration generated offset error correction value. The value set here is subtracted from the conversion and the result is transferred into the result registers (Rn). If the result is above the maximum or below the minimum result value, it is forced to the appropriate limit for the current mode of operation.

Parameters

instance	ADC instance ID.
value	The manual offset value.

2.1.2.9 static void adc_hal_set_clock_source_mode (uint32_t *instance,* adc_clock_source_mode_t *mode*) [inline], [static]

The selection of ADC clock source can see to the type definition of adc_clock_source_mode_t. This function selects the input clock source to generate the internal clock, ADCK. Note that when the ADACK clock source is selected, it does not have to be activated prior to the start of the conversion. When it is selected and it is not activated prior to start a conversion , the asynchronous clock will be activated at the start of a conversion and shuts off when conversions are terminated. In this case, there is an associated clock startup delay each time the clock source is re-activated.

Parameters

instance	ADC instance ID.
mode	The indicated clock source mode.

2.1.2.10 static void adc_hal_configure_asynchronous_clock (uint32_t *instance,* bool *isEnabled*) [inline], [static]

When enables the ADC's asynchronous clock source and the clock source output regardless of the conversion and input clock select status of the ADC. Based on MCU configuration, the asynchronous clock may be used by other modules. Setting this mode allows the clock to be used even while the ADC is idle or operating from a different clock source. Also, latency of initiating a single or first-continuous conversion with the asynchronous clock selected is reduced since the ADACK clock is already operational.

Parameters

instance	ADC instance ID.
isEnabled	The switcher.

2.1.2.11 static void adc_hal_set_clock_divider_mode (uint32_t *instance,* adc_clock_divider_mode_t *mode*) [inline], [static]

The selection of ADC's clock divider can see to the type definition of the adc_clock_divider_mode_t. This function selects the divide ratio used by the ADC to generate the internal clock ADCK.

Parameters

instance	ADC instance ID.
mode	The selection of the divider.

2.1.2.12 static void adc_hal_set_reference_voltage_mode (uint32_t *instance,* adc_reference_voltage_mode_t *mode*) [inline], [static]

The selection of ADC's reference voltage can see to the type definition of adc_reference_voltage_mode_t. This function selects the voltage reference source used for conversions.

Parameters

instance	ADC instance ID.
mode	The selection of the reference voltage source.

2.1.2.13 static void adc_hal_configure_high_speed (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This function configures the ADC for high speed operations. The conversion sequence is altered (2 ADCK cycles added to the conversion time) to allow higher speed conversion clocks.

Parameters

instance	ADC instance ID.
isEnabled	The switcher.

2.1.2.14 static void adc_hal_configure_long_sample (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This function selects between the different sample times based on the conversion mode selected. It adjusts the sample period to allow higher impedance inputs to be accurately sampled or to maximize conversion speed for lower impedance inputs. Longer sample times can also be used to lower overall power consumption if the continuous conversions are enabled and the high conversion rates are not required. In fact this will be able to charge the SAR in a timely manner way without affecting the SAR configuration.

Parameters

instance	ADC instance ID.
isEnabled	The switcher.

2.1.2.15 static void adc_hal_set_long_sample_mode (uint32_t *instance,* adc_long_sample_mode_t *mode*) [inline], [static]

The selection of ADC long sample mode can see to the type definition of the adc_long_sample_mode_t. This function selects the long sample mode that indicating the different count of extra ADCK cycles are needed.

Parameters

instance	ADC instance ID.
mode	The selection of long sample mode.

2.1.2.16 static void adc_hal_configure_low_power (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This function controls the power configuration of the successive approximation converter. This optimizes power consumption when higher sample rates are not required.

Parameters

instance	ADC instance ID.
isEnabled	The switcher.

2.1.2.17 static void adc_hal_set_resolution_mode (uint32_t *instance,* adc_resolution_mode_t *mode*) [inline], [static]

The selection of ADC resolution mode can see to the type definition of the adc_resolution_mode_t. This function selects the ADC resolution mode. Note that the differential conversion is different to single-end conversion.

Parameters

instance	ADC instance ID.
mode	The selection of resolution mode.

2.1.2.18 static void adc_hal_configure_continuous_conversion (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This function configures the continuous conversions or sets of conversions if the hardware average function is enabled after initiating a conversion.

Parameters

instance	ADC instance ID.
isEnabled	The switcher.

2.1.2.19 static void adc_hal_configure_hw_trigger (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This function selects the type of trigger used for initiating a conversion. Two types of triggers can be selected: software trigger and hardware trigger. When software trigger is selected, a conversion is initiated following a write to SC1A. When hardware trigger is selected, a conversion is initiated following the assertion of the external events. The event will come through the signal on the line of ADHWT input after a pulse of ADHWTSn input inside SOC.

Parameters

instance	ADC instance ID.
isEnabled	The switcher.

2.1.2.20 static void adc_hal_configure_hw_average (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This function enables the hardware average function of the ADC.

Parameters

instance	ADC instance ID.

isEnabled	The switcher.

2.1.2.21 static void adc_hal_set_hw_average_mode (uint32_t *instance,* adc_hw_average_mode_t *mode*) [inline], [static]

The selection of ADC hardware average mode can see to the type definition of the adc_hw_average_-mode_t. This function determines how many ADC conversions are averaged to create the ADC average result.

Parameters

instance	ADC instance ID.
mode	The selection of hardware average mode.

2.1.2.22 static void adc_hal_configure_hw_compare (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This function enables the compare function.

Parameters

instance	ADC instance ID.
isEnabled	The switcher.

2.1.2.23 static void adc_hal_configure_hw_compare_greater (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This function configures the compare function to check the conversion result relative to the compare value register(s) (CV1 and CV2). To enable will configure greater than or equal to threshold, outside range inclusive and inside range inclusive functionality based on the values placed in the CV1 and CV2 registers. Otherwise, it will configure less than threshold, outside range not inclusive and inside range not inclusive functionality based on the values placed in the CV1 and CV2 registers.

Parameters

instance	ADC instance ID.

isEnabled	The switcher.

2.1.2.24 static void adc_hal_configure_hw_compare_in_range (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This function configures the compare function to check if the conversion result of the input being monitored is either inside or outside the range formed by the compare value registers (CV1 and CV2). However, the actual compare range should be determined alone with the function of adc_hal_configure_hw_compare_greater() as well.

Parameters

instance	ADC instance ID.
isEnabled	The switcher.

2.1.2.25 static void adc_hal_set_hw_compare_value1 (uint32_t *instance,* uint32_t *value*) [inline], [static]

This function sets the value of the CV1 register.

Parameters

instance	ADC instance ID.
value	The setting value.

2.1.2.26 static void adc_hal_set_hw_compare_value2 (uint32_t *instance,* uint32_t *value*) [inline], [static]

This function sets the value of the CV2 register.

Parameters

instance	ADC instance ID.
value	The setting value.

2.1.2.27 static void adc_hal_configure_dma (uint32_t *instance,* bool *isEnabled*) [inline], [static]

When DMA is enabled, it asserts the ADC DMA request during the ADC conversion complete event noted by the assertion of any of the ADC COCO flags.

Parameters

instance	ADC instance ID.
isEnabled	The switcher.

2.1.2.28 static void adc_hal_disable (uint32_t *instance,* uint32_t *group*) [inline], [static]

Here the "NUll" channel is set to the conversion channel.

Parameters

instance	ADC instance ID.
group	The group mux index.

2.1.2.29 static void adc_hal_enable (uint32_t *instance,* uint32_t *group,* adc_channel_mode_t *mode,* bool *isDifferential*) [inline], [static]

When the available channel is set, the conversion begins to execute.

Parameters

instance	ADC instance ID.
group	The group mux index.
mode	The selection of channel number.
isDifferential	the selection of differential input.

2.1.2.30 static void adc_hal_configure_interrupt (uint32_t *instance,* uint32_t *group,* bool *isEnabled*) [inline], [static]

This function enables conversion complete interrupts. When COCO is set while the respective AIEN is high, an interrupt is asserted.

Parameters

instance	ADC instance ID.

group	The group mux index.
inEnable	The switcher.

2.1.2.31 static bool adc_hal_is_in_process (uint32_t *instance*) [inline], [static]

This function indicates that a conversion or hardware averaging is in progress. ADACT is set when a conversion is initiated and cleared when a conversion is completed or aborted. Note that if the continuous conversion is been use, this function will always return true.

Parameters

instance	ADC instance ID.

Returns

true if it is.

2.1.2.32 static bool adc_hal_is_conversion_completed (uint32_t *instance,* uint32_t *group*) [inline], [static]

This function indicates whether each conversion is completed.

Parameters

instance	ADC instance ID.
group	The grout mux index.

Returns

true if it is.

2.1.2.33 static bool adc_hal_is_calibration_fail (uint32_t *instance*) [inline], [static]

This function displays the result of the calibration sequence.

Parameters

instance	ADC instance ID.

Returns

true if it is.

2.1.2.34 static uint32_t adc_hal_get_conversion_value (uint32_t *instance,* uint32_t *group*) [inline], [static]

This function returns the conversion value kept in the Rn Register. Unused bits in the Rn register are cleared in unsigned right justified modes and carry the sign bit (MSB) in sign extended 2's complement modes.

Parameters

instance	ADC instance ID.
group	The group mux index.

2.1.2.35 static void adc_hal_set_group_mux (uint32_t *instance,* adc_group_mux_mode_t *group*) [inline], [static]

ADC Mux select bit changes the ADC group setting to select between alternate sets of ADC channels. It will activate group A or group B.

Parameters

instance	ADC instance ID.
group	The group mux index.

2.2 ADC Peripheral Driver

The chapter describes the programming interface of the ADC Peripheral driver.

Data Structures

- struct adc_calibration_param_t
  Defines the calibration parameter structure. More...
- struct adc_user_config_t
  Defines the ADC basic configuration structure. More...
- struct adc_extend_config_t
  Defines the ADC extended configuration structure. More...
- struct adc_channel_config_t
  Defines the channel configuration structure. More...

Typedefs

- typedef void(∗ adc_isr_callback_t)(void)
  Defines the ADC ISR callback function.

Functions

- adc_status_t adc_get_calibration_param (uint32_t instance, adc_calibration_param_t ∗paramPtr)
  Gets the parameters for calibration.
- adc_status_t adc_set_calibration_param (uint32_t instance, adc_calibration_param_t ∗paramPtr)
  Sets the parameters for calibration.
- adc_status_t adc_auto_calibration (uint32_t instance, adc_calibration_param_t ∗paramPtr)
  Executes the auto calibration.
- adc_status_t adc_init (uint32_t instance, adc_user_config_t ∗cfgPtr)
  Initializes the ADC with the basic configuration.
- adc_status_t adc_init_extend (uint32_t instance, adc_extend_config_t ∗extendCfgPtr)
  Initializes the ADC with the extended configurations when necessary.
- void adc_shutdown (uint32_t instance)
  Shuts down the ADC.
- adc_status_t adc_start_conversion (uint32_t instance, adc_channel_config_t ∗channelCfgPtr)
  Starts the conversion from the indicated channel.
- adc_status_t adc_stop_conversion (uint32_t instance, adc_channel_config_t ∗channelCfgPtr)
  Stops the conversion.
- bool adc_is_conversion_completed (uint32_t instance, adc_channel_config_t ∗channelCfgPtr)
  Checks whether the conversion is completed.
- uint32_t adc_get_conversion_value (uint32_t instance, adc_channel_config_t ∗channelCfgPtr)
  Gets the value after the conversion.
- void adc_register_user_callback_isr (uint32_t instance, adc_isr_callback_t func)
  Registers the custom callback function of the ADC ISR.

2.2.0.36 ADC peripheral driver

Overview

The ADC peripheral driver configures the ADC (Analog-to-Digital Converter). It handles calibration, initialization and configuration of ADC module.

Initialization

To initialize the ADC module, call adc_init() and pass the configuration data structure. This function automatically enables the ADC module and clock. When working with advanced features, call adc_-init_extend() additionally and pass the extended configuration data structure. This function enables the extended features, such as the hardware trigger, hardware average, hardware compare, low power mode, and high speed mode.

Model building

Multiple channels share the converter, which can only serve one channel at a time. As a result the ADC channel is considered as an attribute when configuring the ADC converter instance. Therefore, when you obtain the ADC value, the value comes with the converter, after the channel is set as expected. ADC can work in three modes: Interrupt mode, Polling mode, and DMA mode. Interrupt is the recommended mode where the conversion value buffer is updated by ADC ISR. The polling mode polls the complete flag before obtaining the conversion value. The DMA mode needs the DMA support.

Call diagram

To use the ADC driver, the user should follow these steps:

1. Execute adc_auto_calibration() to calibrate the ADC device if no other indicated offset values are used.
2. Initialize the ADC converter by calling the adc_init().
3. Initialize the advanced feature for ADC converter by calling the adc_init_extend() as needed.
4. Register the user-defined callback to ADC ISR by calling adc_register_user_callback_isr() as needed. The callback function is executed when the ADC interrupt occurs.
5. Set or reset the expected channel to a converter and execute the conversion by calling the adc_start-_convertion().
6. Obtain the ADC value by calling the adc_get_conversion_value().
7. Pause the converter by calling the adc_stop_conversion() as needed.
8. Shut down the converter by calling the adc_shutdown().

This is an example code to initialize and configure the ADC driver in interrupt mode:

```
uint32_t val;
```

```
// Define the ADC calibration parameter variable.
adc_calibration_param_t myAdcCalParam;
// Define the ADC converter basically.
adc_user_config_t myAdcUserCfg =
{
    .clockSourceMode        = kAdcClockSourceAsynchrounous,  //
       Use the ADC asynchronous clock when converting.
    .clockSourceDividerMode = kAdcClockDivider8,             // Set the clock divider from
       the bus clock.
    .resolutionMode         = kAdcSingleDiff16,              // Conversion resolution.
    .referenceVoltageMode   = kAdcVoltageVref,               // Reference voltage.
    .isContinuousEnabled    = true                           // Enable continuous work.
};
// Define the ADC channel.
adc_channel_config_t myAdcChannelCfg =
{
    .channelId              = kAdcChannelTemperature,        // Set the
       channel ID.
    .isDifferentialEnabled  = false,                         // Use single end channel.
    .isInterruptEnabled     = true,                          // Use interrupt mode.
    .groupMux               = kAdcChannelMuxA                // Set the group Mux.
};
// Define the ADC converter for advanced features.
adc_extend_config_t myAdcExtendCfg =
{
    .isLowPowerEnabled      = false,                         // Disable the low power
       mode.
    .isLongSampleEnabled    = true,                          // Enable the long sample.
    .hwLongSampleMode       = kAdcLongSampleDefault,         // Set the long sample mode.
    .isHighSpeedEnabled     = false,                         // Disable the high speed mode.
    .isAsynClockEnabled     = true,                          // Enable the ADC asynchronous clock when
       initialized.
    .isHwTriggerEnabled     = false,                         // Disable the hardware trigger.
    .isHwCompareEnabled     = false,                         // Disable the hardware compare.
    .isHwCompareGreaterEnabled = false,                      // Hardware compare related.
    .isHwCompareRangeEnabled= false,                         // Hardware compare related.
    .hwCompareValue1        = 0,                             // Hardware compare related.
    .hwCompareValue2        = 0,                             // Hardware compare related.
    .isHwAverageEnabled     = true,                          // Enable the hardware average.
    .hwAverageSampleMode    = kAdcHwAverageCount32,          // Hardware average
       related.
    .isDmaEnabled           = false                          // Disable the DMA switcher.
};

// Execute the auto calibration before use ADC.
adc_auto_calibration(0U, &myAdcCalParam)

// Initialize the ADC converter basically.
adc_init(0U, &myAdcUserCfg);

// Initialize the ADC additionally for extend features.
adc_init_extend(0U, &myAdcExtendCfg);

// Set the ADC channel and start conversion.
adc_start_conversion(0U, &myAdcChannelCfg);

// Get the ADC value.
val = adc_get_conversion_value(0U, &myAdcChannelCfg);

// ...
// Pause the converter.
adc_stop_conversion(0U, &myAdcChannelCfg);

// ...
// Shut down the ADC converter.
adc_shutdown(0U);
```

2.2.1 Data Structure Documentation

2.2.1.1 struct adc_calibration_param_t

This structure keeps the calibration parameter after executing the auto-calibration or filled by indicated ones.

Data Fields

- uint32_t PG
  The value for PG register.
- uint32_t MG
  The value for MG register.

2.2.1.2 struct adc_user_config_t

This structure is used when initializing the ADC device associated with adc_init(). It contains the basic feature configuration which are necessary.

Data Fields

- adc_clock_source_mode_t clockSourceMode
  Selection of ADC clock source.
- adc_clock_divider_mode_t clockSourceDividerMode
  Selection of ADC clock divider.
- adc_resolution_mode_t resolutionMode
  Selection of ADC resolution.
- adc_reference_voltage_mode_t referenceVoltageMode
  Selection of ref voltage source.
- bool isContinuousEnabled
  Switcher to enable continuous conversion.

2.2.1.3 struct adc_extend_config_t

This structure is used when initializing the ADC device associated with adc_init_extend(). It contains the advanced feature configuration when necessary.

Data Fields

- bool isLowPowerEnabled
  Switcher to enable the low power mode.
- bool isLongSampleEnabled
  Switcher to enable the long sample mode.
- adc_long_sample_mode_t hwLongSampleMode
  Selection of long sample mode.

- bool isHighSpeedEnabled
  Switcher to enable high speed sample mode.
- bool isAsynClockEnabled
  Switcher to enable internal asynchronous clock at initialization.
- bool isHwTriggerEnabled
  Switcher to enable hardware trigger.
- bool isHwCompareEnabled
  Switcher to enable hardware compare.
- bool isHwCompareGreaterEnabled
  Switcher to enable greater compare.
- bool isHwCompareRangeEnabled
  Switcher to enable range compare.
- uint32_t hwCompareValue1
  Low limit in hardware compare.
- uint32_t hwCompareValue2
  High limit in hardware compare.
- bool isHwAverageEnabled
  Switcher to enable hardware average.
- adc_hw_average_mode_t hwAverageSampleMode
  Selection of hardware average time.

2.2.1.4 struct adc_channel_config_t

This structure is used when setting the conversion channel associated with adc_start_conversion(), adc_stop_conversion(), adc_is_conversion_completed() and adc_get_conversion_value(). It contains all the information that can identify an ADC channel.

Data Fields

- adc_channel_mode_t channelId
  Channel number.
- bool isDifferentialEnabled
  The switcher to enable differential channel.
- bool isInterruptEnabled
  The switcher to enable interrupt when conversion is completed.
- adc_group_mux_mode_t muxSelect
  Selection mux to group A(0) or group B(1)

2.2.2 Typedef Documentation

2.2.2.1 typedef void(* adc_isr_callback_t)(void)

This type defines the prototype of ADC ISR callback function that can be registered inside the ISR.

2.2.3 Function Documentation

2.2.3.1 adc_status_t adc_get_calibration_param (uint32_t *instance,* adc_calibration_param_t * *paramPtr*)

This function is used to get the calibration parameters in auto-calibrate mode. Execute this function to obtain the parameter for the calibration during the initialization. This process may be time consuming.

Parameters

instance	ADC instance ID.
paramPtr	The pointer to a empty calibration parameter structure.

Returns

The execution status.

2.2.3.2 adc_status_t adc_set_calibration_param (uint32_t *instance*, adc_calibration_param_t * *paramPtr*)

This function is used to set the calibration parameters. The parameters can be generated from the auto-calibration by the adc_get_calibration_param() or created by manually indicated parameters.

Parameters

instance	ADC instance ID.
paramPtr	The pointer to a filled calibration parameter structure.

Returns

The execution status.

2.2.3.3 adc_status_t adc_auto_calibration (uint32_t *instance*, adc_calibration_param_t * *paramPtr*)

This function is used to execute the auto calibration. Recommended configuration has been accepted to fetch calibration parameters for highest accuracy. The calibration offset is returned to the application for further use. After the auto calibration process, the initialization function should be called explicitly to update the configuration according to the application.

Parameters

instance	ADC instance ID.
paramPtr	The pointer to an empty calibration parameter structure. It is filled with the calibration offset value after the function is called.

Returns

The execution status.

2.2.3.4 adc_status_t adc_init (uint32_t *instance,* adc_user_config_t * *cfgPtr*)

This function ensures that the basic operations of ADC function correctly. This function should be called when an application does not require complex features.

Parameters

instance	ADC instance ID.
cfgPtr	The pointer to basic configuration structure.

Returns

The execution status.

2.2.3.5 adc_status_t adc_init_extend (uint32_t *instance,* adc_extend_config_t * *extendCfgPtr*)

This function provides advanced features according when an application requires complex configurations. They are: low power mode, long sample mode, high speed mode, asynchronous work mode, hardware trigger, hardware compare, and hardware average.

Parameters

instance	ADC instance ID.
extendCfgPtr	The pointer to extend configuration structure.

Returns

The execution status.

2.2.3.6 void adc_shutdown (uint32_t *instance*)

Shutting down the ADC cuts off the clock to the indicated ADC device.

Parameters

instance	ADC instance ID.

2.2.3.7 adc_status_t adc_start_conversion (uint32_t *instance,* adc_channel_config_t * *channelCfgPtr*)

Triggers the indicated channel conversion in a single conversion mode. This function should be called when each time the conversion is triggered. In a continuous conversion mode, this function can be called only once at the beginning of conversion. The ADC executes the conversion periodically and automatically.

Parameters

instance	ADC instance ID.
channelCfgPtr	The pointer to channel configuration structure.

Returns

The execution status.

2.2.3.8 adc_status_t adc_stop_conversion (uint32_t *instance,* adc_channel_config_t * *channelCfgPtr*)

Stops the ADC conversion. This function sets ADC to a "NULL" channel, which stops ADC conversion from any channel. It is a different function than the adc_shutdown().

Parameters

instance	ADC instance ID.
channelCfgPtr	The pointer to channel configuration structure.

Returns

The execution status.

2.2.3.9 bool adc_is_conversion_completed (uint32_t *instance,* adc_channel_config_t * *channelCfgPtr*)

Checks whether the current conversion is completed. Because there are multiple channels sharing the same converter, the status is used to indicate the converter status.

Parameters

instance	ADC instance ID.
channelCfgPtr	The pointer to channel configuration structure.

Returns

True if the event is asserted.

2.2.3.10 uint32_t adc_get_conversion_value (uint32_t *instance,* adc_channel_config_t * *channelCfgPtr*)

The value comes from the value register that may be eventually processed according to the application. When using polling mode, the value is obtained after the conversion is completed. When using the interrupt mode, the value comes from the buffer that is updated by the ADC ISR.

Parameters

instance	ADC instance ID.
channelCfgPtr	The pointer to channel configuration structure.

Returns

the value of conversion.

2.2.3.11 void adc_register_user_callback_isr (uint32_t *instance,* adc_isr_callback_t *func*)

Callback provides a friendly API for application to program the ISR. A special function needs to be executed at the moment conversion is completed and can be inserted to the ISR by calling the function registered by the user.

Parameters

instance	ADC instance ID.
func	The pointer to user indicating callback function.

Chapter 3
Controller Area Network (FlexCAN)

The Kinetis SDK provides both HAL and Peripheral drivers for the FlexCAN Controller Area Network (FlexCAN) block of Kinetis devices.

Modules

- FlexCAN Driver
  The part describes the programming interface of the FlexCAN Peripheral driver.
- FlexCAN HAL driver
  The part describes the programming interface of the FlexCAN HAL driver.

3.1 FlexCAN HAL driver

The chapter describes the programming interface of the FlexCAN HAL driver.

Data Structures

- struct flexcan_id_table_t
  FlexCAN RX FIFO ID filter table structure. More...
- struct flexcan_berr_counter_t
  FlexCAN bus error counters. More...
- struct flexcan_mb_code_status_tx_t
  FlexCAN MB code and status for transmitting. More...
- struct flexcan_mb_code_status_rx_t
  FlexCAN MB code and status for receiving. More...
- struct flexcan_rx_fifo_config_t
  FlexCAN Rx FIFO configuration. More...
- struct flexcan_mb_t
  FlexCAN message buffer structure. More...
- struct flexcan_user_config_t
  FlexCAN configuration. More...
- struct flexcan_time_segment_t
  FlexCAN timing related structures. More...

Enumerations

- enum _flexcan_constants { kFlexCanMessageSize = 8 }
  FlexCAN constants.
- enum _flexcan_err_status {
  kFlexCan_RxWrn = 0x0080,
  kFlexCan_TxWrn = 0x0100,
  kFlexCan_StfErr = 0x0200,
  kFlexCan_FrmErr = 0x0400,
  kFlexCan_CrcErr = 0x0800,
  kFlexCan_AckErr = 0x1000,
  kFlexCan_Bit0Err = 0x2000,
  kFlexCan_Bit1Err = 0x4000 }
  The Status enum is used to report current status of the FlexCAN interface.
- enum flexcan_status_t
  FlexCAN status return codes.
- enum flexcan_operation_modes_t {
  kFlexCanNormalMode,
  kFlexCanListenOnlyMode,
  kFlexCanLoopBackMode,
  kFlexCanFreezeMode,
  kFlexCanDisableMode }
  FlexCAN operation modes.

- enum flexcan_mb_code_rx_t {
  kFlexCanRX_Inactive = 0x0,
  kFlexCanRX_Full = 0x2,
  kFlexCanRX_Empty = 0x4,
  kFlexCanRX_Overrun = 0x6,
  kFlexCanRX_Busy = 0x8,
  kFlexCanRX_Ranswer = 0xA,
  kFlexCanRX_NotUsed = 0xF }
  FlexCAN message buffer CODE for Rx buffers.
- enum flexcan_mb_code_tx_t {
  kFlexCanTX_Inactive = 0x08,
  kFlexCanTX_Abort = 0x09,
  kFlexCanTX_Data = 0x0C,
  kFlexCanTX_Remote = 0x1C,
  kFlexCanTX_Tanswer = 0x0E,
  kFlexCanTX_NotUsed = 0xF }
  FlexCAN message buffer CODE FOR Tx buffers.
- enum flexcan_mb_transmission_type_t {
  kFlexCanMBStatusType_TX,
  kFlexCanMBStatusType_TXRemote,
  kFlexCanMBStatusType_RX,
  kFlexCanMBStatusType_RXRemote,
  kFlexCanMBStatusType_RXTXRemote }
  FlexCAN message buffer transmission types.
- enum flexcan_rx_fifo_id_element_format_t {
  kFlexCanRxFifoIdElementFormat_A,
  kFlexCanRxFifoIdElementFormat_B,
  kFlexCanRxFifoIdElementFormat_C,
  kFlexCanRxFifoIdElementFormat_D }
- enum flexcan_rx_fifo_id_filter_num_t {
  kFlexCanRxFifoIDFilters_8 = 0x0,
  kFlexCanRxFifoIDFilters_16 = 0x1,
  kFlexCanRxFifoIDFilters_24 = 0x2,
  kFlexCanRxFifoIDFilters_32 = 0x3,
  kFlexCanRxFifoIDFilters_40 = 0x4,
  kFlexCanRxFifoIDFilters_48 = 0x5,
  kFlexCanRxFifoIDFilters_56 = 0x6,
  kFlexCanRxFifoIDFilters_64 = 0x7,
  kFlexCanRxFifoIDFilters_72 = 0x8,
  kFlexCanRxFifoIDFilters_80 = 0x9,
  kFlexCanRxFifoIDFilters_88 = 0xA,
  kFlexCanRxFifoIDFilters_96 = 0xB,
  kFlexCanRxFifoIDFilters_104 = 0xC,
  kFlexCanRxFifoIDFilters_112 = 0xD,
  kFlexCanRxFifoIDFilters_120 = 0xE,
  kFlexCanRxFifoIDFilters_128 = 0xF }

FlexCAN Rx FIFO filters number.

- enum flexcan_rx_mask_type_t { kFlexCanRxMask_Global, kFlexCanRxMask_Individual }
  FlexCAN RX mask type.
- enum flexcan_mb_id_type_t { kFlexCanMbId_Std, kFlexCanMbId_Ext }
  FlexCAN MB ID type.
- enum flexcan_clk_source_t { kFlexCanClkSource_Osc, kFlexCanClkSource_Ipbus }
  FlexCAN clock source.
- enum flexcan_int_type_t { kFlexCanInt_Buf, kFlexCanInt_Err, kFlexCanInt_Boff, kFlexCanInt_Wakeup, kFlexCanInt_Txwarning, kFlexCanInt_Rxwarning }
  FlexCAN error interrupt types.

Configuration

- flexcan_status_t flexcan_hal_enable (uint8_t instance)
  Enables FlexCAN controller.
- flexcan_status_t flexcan_hal_disable (uint8_t instance)
  Disables FlexCAN controller.
- static bool flexcan_hal_is_enabled (uint8_t instance)
  Checks whether the FlexCAN is enabled or disabled.
- flexcan_status_t flexcan_hal_sw_reset (uint8_t instance)
  Resets the FlexCAN controller.
- flexcan_status_t flexcan_hal_select_clk (uint8_t instance, flexcan_clk_source_t clk)
  Selects the clock source for FlexCAN.
- flexcan_status_t flexcan_hal_init (uint8_t instance, const flexcan_user_config_t *data)
  Initializes the FlexCAN controller.
- void flexcan_hal_set_time_segments (uint8_t instance, flexcan_time_segment_t *time_seg)
  Sets the FlexCAN time segments for setting up bit rate.
- void flexcan_hal_get_time_segments (uint8_t instance, flexcan_time_segment_t *time_seg)
  Gets the FlexCAN time segments to calculate the bit rate.
- void flexcan_hal_exit_freeze_mode (uint8_t instance)
  Un freezes the FlexCAN module.
- void flexcan_hal_enter_freeze_mode (uint8_t instance)
  Freezes the FlexCAN module.
- flexcan_status_t flexcan_hal_enable_operation_mode (uint8_t instance, flexcan_operation_modes_t mode)
  Enables operation mode.
- flexcan_status_t flexcan_hal_disable_operation_mode (uint8_t instance, flexcan_operation_modes-

_t mode)
Disables operation mode.

Data transfer

- flexcan_status_t flexcan_hal_set_mb_tx (uint8_t instance, const flexcan_user_config_t *data, uint32_t mb_idx, flexcan_mb_code_status_tx_t *cs, uint32_t msg_id, uint8_t *mb_data)
  Sets the FlexCAN message buffer fields for transmitting.
- flexcan_status_t flexcan_hal_set_mb_rx (uint8_t instance, const flexcan_user_config_t *data, uint32_t mb_idx, flexcan_mb_code_status_rx_t *cs, uint32_t msg_id)
  Set the FlexCAN message buffer fields for receiving.
- flexcan_status_t flexcan_hal_get_mb (uint8_t instance, const flexcan_user_config_t *data, uint32_t mb_idx, flexcan_mb_t *mb)
  Gets the FlexCAN message buffer fields.
- flexcan_status_t flexcan_hal_lock_rx_mb (uint8_t instance, const flexcan_user_config_t *data, uint32_t mb_idx)
  Locks the FlexCAN Rx message buffer.
- static void flexcan_hal_unlock_rx_mb (uint8_t instance)
  Unlocks the FlexCAN Rx message buffer.
- void flexcan_hal_enable_rx_fifo (uint8_t instance)
  Enables the Rx FIFO.
- void flexcan_hal_disable_rx_fifo (uint8_t instance)
  Disables the Rx FIFO.
- void flexcan_hal_set_rx_fifo_filters_number (uint8_t instance, uint32_t number)
  Sets the number of the Rx FIFO filters.
- void flexcan_hal_set_max_mb_number (uint8_t instance, const flexcan_user_config_t *data)
  Sets the maximum number of Message Buffers.
- flexcan_status_t flexcan_hal_set_id_filter_table_elements (uint8_t instance, const flexcan_user_config_t *data, flexcan_rx_fifo_id_element_format_t id_format, flexcan_id_table_t *id_filter_table)
  Sets the Rx FIFO ID filter table elements.
- flexcan_status_t flexcan_hal_set_rx_fifo (uint8_t instance, const flexcan_user_config_t *data, flexcan_rx_fifo_id_element_format_t id_format, flexcan_id_table_t *id_filter_table)
  Sets the FlexCAN Rx FIFO fields.
- flexcan_status_t flexcan_hal_read_fifo (uint8_t instance, flexcan_mb_t *rx_fifo)
  Gets the FlexCAN Rx FIFO data.

Interrupts

- flexcan_status_t flexcan_hal_enable_mb_interrupt (uint8_t instance, const flexcan_user_config_t *data, uint32_t mb_idx)
  Enables the FlexCAN Message Buffer interrupt.
- flexcan_status_t flexcan_hal_disable_mb_interrupt (uint8_t instance, const flexcan_user_config_t *data, uint32_t mb_idx)
  Disables the FlexCAN Message Buffer interrupt.
- void flexcan_hal_enable_error_interrupt (uint8_t instance)
  Enables error interrupt of the FlexCAN module.
- void flexcan_hal_disable_error_interrupt (uint8_t instance)

Disables error interrupt of the FlexCAN module.
- void flexcan_hal_enable_bus_off_interrupt (uint8_t instance)
  Enables Bus off interrupt of the FlexCAN module.
- void flexcan_hal_disable_bus_off_interrupt (uint8_t instance)
  Disables Bus off interrupt of the FlexCAN module.
- void flexcan_hal_enable_wakeup_interrupt (uint8_t instance)
  Enables Wakeup interrupt of the FlexCAN module.
- void flexcan_hal_disable_wakeup_interrupt (uint8_t instance)
  Disables Wakeup interrupt of the FlexCAN module.
- void flexcan_hal_enable_tx_warning_interrupt (uint8_t instance)
  Enables TX warning interrupt of the FlexCAN module.
- void flexcan_hal_disable_tx_warning_interrupt (uint8_t instance)
  Disables TX warning interrupt of the FlexCAN module.
- void flexcan_hal_enable_rx_warning_interrupt (uint8_t instance)
  Enables RX warning interrupt of the FlexCAN module.
- void flexcan_hal_disable_rx_warning_interrupt (uint8_t instance)
  Disables RX warning interrupt of the FlexCAN module.

Status

- static uint32_t flexcan_hal_get_freeze_ack (uint8_t instance)
  Gets the value of FlexCAN freeze ACK.
- uint8_t flexcan_hal_get_mb_int_flag (uint8_t instance, const flexcan_user_config_t *data, uint32_t mb_idx)
  Gets the individual FlexCAN MB interrupt flag.
- static uint32_t flexcan_hal_get_all_mb_int_flags (uint8_t instance)
  Gets all FlexCAN MB interrupt flags.
- static void flexcan_hal_clear_mb_int_flag (uint8_t instance, uint32_t reg_val)
  Clears the interrupt flag of the message buffers.
- void flexcan_hal_get_err_counter (uint8_t instance, flexcan_berr_counter_t *err_cnt)
  Gets the transmit error counter and receives the error counter.
- static uint32_t flexcan_hal_get_err_status (uint8_t instance)
  Gets error and status.
- void flexcan_hal_clear_err_interrupt_status (uint8_t instance)
  Clears all other interrupts in ERRSTAT register (Error, Busoff, Wakeup).

Mask

- void flexcan_hal_set_mask_type (uint8_t instance, flexcan_rx_mask_type_t type)
  Sets the Rx masking type.
- void flexcan_hal_set_rx_fifo_global_std_mask (uint8_t instance, uint32_t std_mask)
  Sets the FlexCAN RX FIFO global standard mask.
- void flexcan_hal_set_rx_fifo_global_ext_mask (uint8_t instance, uint32_t ext_mask)
  Sets the FlexCAN Rx FIFO global extended mask.
- flexcan_status_t flexcan_hal_set_rx_individual_std_mask (uint8_t instance, const flexcan_user_config_t *data, uint32_t mb_idx, uint32_t std_mask)
  Sets the FlexCAN Rx individual standard mask for ID filtering in the Rx MBs and the Rx FIFO.

- flexcan_status_t flexcan_hal_set_rx_individual_ext_mask (uint8_t instance, const flexcan_user_config_t *data, uint32_t mb_idx, uint32_t ext_mask)
  Sets the FlexCAN Rx individual extended mask for ID filtering in the Rx MBs and the Rx FIFO.
- void flexcan_hal_set_rx_mb_global_std_mask (uint8_t instance, uint32_t std_mask)
  Sets the FlexCAN Rx MB global standard mask.
- void flexcan_hal_set_rx_mb_buf14_std_mask (uint8_t instance, uint32_t std_mask)
  Sets the FlexCAN RX MB BUF14 standard mask.
- void flexcan_hal_set_rx_mb_buf15_std_mask (uint8_t instance, uint32_t std_mask)
  Sets the FlexCAN Rx MB BUF15 standard mask.
- void flexcan_hal_set_rx_mb_global_ext_mask (uint8_t instance, uint32_t ext_mask)
  Sets the FlexCAN RX MB global extended mask.
- void flexcan_hal_set_rx_mb_buf14_ext_mask (uint8_t instance, uint32_t ext_mask)
  Sets the FlexCAN RX MB BUF14 extended mask.
- void flexcan_hal_set_rx_mb_buf15_ext_mask (uint8_t instance, uint32_t ext_mask)
  Sets the FlexCAN RX MB BUF15 extended mask.
- static uint32_t flexcan_hal_get_rx_fifo_id_acceptance_filter (uint8_t instance)
  Gets the FlexCAN ID acceptance filter hit indicator on Rx FIFO.

3.1.1 Data Structure Documentation

3.1.1.1 struct flexcan_id_table_t

Data Fields

- bool is_remote_mb
  Remote frame.
- bool is_extended_mb
  Extended frame.
- uint32_t * id_filter
  Rx FIFO ID filter elements.

3.1.1.2 struct flexcan_berr_counter_t

Data Fields

- uint16_t txerr
  Transmit error counter.
- uint16_t rxerr
  Receive error counter.

3.1.1.3 struct flexcan_mb_code_status_tx_t

Data Fields

- flexcan_mb_code_tx_t code
  MB code for Tx buffers.
- flexcan_mb_id_type_t msg_id_type

Type of message ID (standard or extended)
- uint32_t data_length
  Length of Data in Bytes.
- uint32_t substitute_remote
  Substitute remote request (used only in.
- uint32_t remote_transmission
  extended format)
- bool local_priority_enable
  1 if enable it; 0 if disable it
- uint32_t local_priority_val
  Local priority value [0..2].

3.1.1.3.0.1 Field Documentation

3.1.1.3.0.1.1 uint32_t flexcan_mb_code_status_tx_t::remote_transmission

Remote transmission request

3.1.1.4 struct flexcan_mb_code_status_rx_t

Data Fields

- flexcan_mb_code_rx_t code
  MB code for Rx buffers.
- flexcan_mb_id_type_t msg_id_type
  Type of message ID (standard or extended)
- uint32_t data_length
  Length of Data in Bytes.
- uint32_t substitute_remote
  Substitute remote request (used only in.
- uint32_t remote_transmission
  extended format)
- bool local_priority_enable
  1 if enable it; 0 if disable it
- uint32_t local_priority_val
  Local priority value [0..2].

3.1.1.4.0.2 Field Documentation

3.1.1.4.0.2.1 uint32_t flexcan_mb_code_status_rx_t::remote_transmission

Remote transmission request

3.1.1.5 struct flexcan_rx_fifo_config_t

Data Fields

- flexcan_mb_id_type_t msg_id_type
  Type of message ID.

- uint32_t data_length
  (standard or extended)
- uint32_t substitute_remote
  Substitute remote request (used.
- uint32_t remote_transmission
  only in extended format)
- flexcan_rx_fifo_id_element_format_t id_filter_number
  The number of Rx FIFO ID filters.

3.1.1.5.0.3 Field Documentation

3.1.1.5.0.3.1 uint32_t flexcan_rx_fifo_config_t::data_length

Length of Data in Bytes

3.1.1.5.0.3.2 uint32_t flexcan_rx_fifo_config_t::remote_transmission

Remote transmission request

3.1.1.6 struct flexcan_mb_t

Data Fields

- uint32_t cs
  Code and Status.
- uint32_t msg_id
  Message Buffer ID.
- uint8_t data [kFlexCanMessageSize]
  Bytes of the FlexCAN message.

3.1.1.7 struct flexcan_user_config_t

Data Fields

- uint32_t num_mb
  The number of Message Buffers needed.
- uint32_t max_num_mb
  The maximum number of Message Buffers.
- flexcan_rx_fifo_id_filter_num_t num_id_filters
  The number of Rx FIFO ID filters needed.
- bool is_rx_fifo_needed
  1 if needed; 0 if not
- bool is_rx_mb_needed
  1 if needed; 0 if not

3.1.1.8 struct flexcan_time_segment_t

Data Fields

- uint32_t propseg
  Propagation segment.
- uint32_t pseg1
  Phase segment 1.
- uint32_t pseg2
  Phase segment 2.
- uint32_t pre_divider
  Clock pre divider.
- uint32_t rjw
  Resync jump width.

3.1.2 Enumeration Type Documentation

3.1.2.1 enum _flexcan_constants

Enumerator

kFlexCanMessageSize FlexCAN message buffer data size in bytes.

3.1.2.2 enum _flexcan_err_status

Enumerator

kFlexCan_RxWrn Reached warning level for RX errors.
kFlexCan_TxWrn Reached warning level for TX errors.
kFlexCan_StfErr Stuffing Error.
kFlexCan_FrmErr Form Error.
kFlexCan_CrcErr Cyclic Redundancy Check Error.
kFlexCan_AckErr Received no ACK on transmission.
kFlexCan_Bit0Err Unable to send dominant bit.
kFlexCan_Bit1Err Unable to send recessive bit.

3.1.2.3 enum flexcan_operation_modes_t

Enumerator

kFlexCanNormalMode Normal mode or user mode.
kFlexCanListenOnlyMode Listen-only mode.
kFlexCanLoopBackMode Loop-back mode.
kFlexCanFreezeMode Freeze mode.
kFlexCanDisableMode Module disable mode.

3.1.2.4 enum flexcan_mb_code_rx_t

Enumerator

kFlexCanRX_Inactive MB is not active.
kFlexCanRX_Full MB is full.
kFlexCanRX_Empty MB is active and empty.
kFlexCanRX_Overrun MB is overwritten into a full buffer.
kFlexCanRX_Busy FlexCAN is updating the contents of the MB.
kFlexCanRX_Ranswer The CPU must not access the MB. A frame was configured to recognize a Remote Request Frame
kFlexCanRX_NotUsed and transmit a Response Frame in return. Not used

3.1.2.5 enum flexcan_mb_code_tx_t

Enumerator

kFlexCanTX_Inactive MB is not active.
kFlexCanTX_Abort MB is aborted.
kFlexCanTX_Data MB is a TX Data Frame(MB RTR must be 0).
kFlexCanTX_Remote MB is a TX Remote Request Frame (MB RTR must be 1).
kFlexCanTX_Tanswer MB is a TX Response Request Frame from.
kFlexCanTX_NotUsed an incoming Remote Request Frame. Not used

3.1.2.6 enum flexcan_mb_transmission_type_t

Enumerator

kFlexCanMBStatusType_TX Transmit MB.
kFlexCanMBStatusType_TXRemote Transmit remote request MB.
kFlexCanMBStatusType_RX Receive MB.
kFlexCanMBStatusType_RXRemote Receive remote request MB.
kFlexCanMBStatusType_RXTXRemote FlexCAN remote frame receives remote request and.

3.1.2.7 enum flexcan_rx_fifo_id_element_format_t

Enumerator

kFlexCanRxFifoIdElementFormat_A One full ID (standard and extended) per ID Filter Table.
kFlexCanRxFifoIdElementFormat_B element. Two full standard IDs or two partial 14-bit (standard and
kFlexCanRxFifoIdElementFormat_C extended) IDs per ID Filter Table element. Four partial 8-bit Standard IDs per ID Filter Table
kFlexCanRxFifoIdElementFormat_D element. All frames rejected.

3.1.2.8 enum flexcan_rx_fifo_id_filter_num_t

Enumerator

kFlexCanRxFifoIDFilters_8 8 Rx FIFO Filters
kFlexCanRxFifoIDFilters_16 16 Rx FIFO Filters
kFlexCanRxFifoIDFilters_24 24 Rx FIFO Filters
kFlexCanRxFifoIDFilters_32 32 Rx FIFO Filters
kFlexCanRxFifoIDFilters_40 40 Rx FIFO Filters
kFlexCanRxFifoIDFilters_48 48 Rx FIFO Filters
kFlexCanRxFifoIDFilters_56 56 Rx FIFO Filters
kFlexCanRxFifoIDFilters_64 64 Rx FIFO Filters
kFlexCanRxFifoIDFilters_72 72 Rx FIFO Filters
kFlexCanRxFifoIDFilters_80 80 Rx FIFO Filters
kFlexCanRxFifoIDFilters_88 88 Rx FIFO Filters
kFlexCanRxFifoIDFilters_96 96 Rx FIFO Filters
kFlexCanRxFifoIDFilters_104 104 Rx FIFO Filters
kFlexCanRxFifoIDFilters_112 112 Rx FIFO Filters
kFlexCanRxFifoIDFilters_120 120 Rx FIFO Filters
kFlexCanRxFifoIDFilters_128 128 Rx FIFO Filters

3.1.2.9 enum flexcan_rx_mask_type_t

Enumerator

kFlexCanRxMask_Global Rx global mask.
kFlexCanRxMask_Individual Rx individual mask.

3.1.2.10 enum flexcan_mb_id_type_t

Enumerator

kFlexCanMbId_Std Standard ID.
kFlexCanMbId_Ext Extended ID.

3.1.2.11 enum flexcan_clk_source_t

Enumerator

kFlexCanClkSource_Osc Oscillator clock.
kFlexCanClkSource_Ipbus Peripheral clock.

3.1.2.12 enum flexcan_int_type_t

Enumerator

kFlexCanInt_Buf OR'd message buffers interrupt.
kFlexCanInt_Err Error interrupt.
kFlexCanInt_Boff Bus off interrupt.
kFlexCanInt_Wakeup Wakeup interrupt.
kFlexCanInt_Txwarning TX warning interrupt.
kFlexCanInt_Rxwarning RX warning interrupt.

3.1.3 Function Documentation

3.1.3.1 flexcan_status_t flexcan_hal_enable (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

Returns

0 if successful; non-zero failed

3.1.3.2 flexcan_status_t flexcan_hal_disable (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

Returns

0 if successful; non-zero failed

3.1.3.3 static bool flexcan_hal_is_enabled (uint8_t *instance*) [inline], [static]

Parameters

instance	The FlexCAN instance number

Returns

State of FlexCAN enable(0)/disable(1)

3.1.3.4 flexcan_status_t flexcan_hal_sw_reset (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

Returns

0 if successful; non-zero failed

3.1.3.5 flexcan_status_t flexcan_hal_select_clk (uint8_t *instance,* flexcan_clk_source_t *clk*)

Parameters

instance	The FlexCAN instance number
clk	The FlexCAN clock source

Returns

0 if successful; non-zero failed

3.1.3.6 flexcan_status_t flexcan_hal_init (uint8_t *instance,* const flexcan_user_config_t * *data*)

Parameters

instance	The FlexCAN instance number

data	The FlexCAN platform data.

Returns

0 if successful; non-zero failed

3.1.3.7 void flexcan_hal_set_time_segments (uint8_t *instance,* flexcan_time_segment_t * *time_seg*)

Parameters

instance	The FlexCAN instance number
time_seg	FlexCAN time segments, which need to be set for the bit rate.

Returns

0 if successful; non-zero failed

3.1.3.8 void flexcan_hal_get_time_segments (uint8_t *instance,* flexcan_time_segment_t * *time_seg*)

Parameters

instance	The FlexCAN instance number
time_seg	FlexCAN time segments read for bit rate

Returns

0 if successful; non-zero failed

3.1.3.9 void flexcan_hal_exit_freeze_mode (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

Returns

0 if successful; non-zero failed.

3.1.3.10 void flexcan_hal_enter_freeze_mode (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.11 flexcan_status_t flexcan_hal_enable_operation_mode (uint8_t *instance,* flexcan_operation_modes_t *mode*)

Parameters

instance	The FlexCAN instance number
mode	An operation mode to be enabled

Returns

0 if successful; non-zero failed.

3.1.3.12 flexcan_status_t flexcan_hal_disable_operation_mode (uint8_t *instance,* flexcan_operation_modes_t *mode*)

Parameters

instance	The FlexCAN instance number
mode	An operation mode to be disabled

Returns

0 if successful; non-zero failed.

3.1.3.13 flexcan_status_t flexcan_hal_set_mb_tx (uint8_t *instance,* const flexcan_user_config_t * *data,* uint32_t *mb_idx,* flexcan_mb_code_status_tx_t * *cs,* uint32_t *msg_id,* uint8_t * *mb_data*)

Parameters

instance	The FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer
cs	CODE/status values (TX)
msg_id	ID of the message to transmit
mb_data	Bytes of the FlexCAN message

Returns

0 if successful; non-zero failed

3.1.3.14 flexcan_status_t flexcan_hal_set_mb_rx (uint8_t *instance*, const flexcan_user_config_t * *data*, uint32_t *mb_idx*, flexcan_mb_code_status_rx_t * *cs*, uint32_t *msg_id*)

Parameters

instance	The FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer
cs	CODE/status values (RX)
msg_id	ID of the message to receive

Returns

0 if successful; non-zero failed

3.1.3.15 flexcan_status_t flexcan_hal_get_mb (uint8_t *instance*, const flexcan_user_config_t * *data*, uint32_t *mb_idx*, flexcan_mb_t * *mb*)

Parameters

instance	The FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer
mb	The fields of the message buffer

Returns

0 if successful; non-zero failed

3.1.3.16 flexcan_status_t flexcan_hal_lock_rx_mb (uint8_t *instance,* const flexcan_user_config_t * *data,* uint32_t *mb_idx*)

Parameters

instance	The FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer

Returns

0 if successful; non-zero failed

3.1.3.17 static void flexcan_hal_unlock_rx_mb (uint8_t *instance*) [inline], [static]

Parameters

instance	The FlexCAN instance number

Returns

0 if successful; non-zero failed

3.1.3.18 void flexcan_hal_enable_rx_fifo (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.19 void flexcan_hal_disable_rx_fifo (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.20 void flexcan_hal_set_rx_fifo_filters_number (uint8_t *instance,* uint32_t *number*)

Parameters

instance	The FlexCAN instance number
number	The number of Rx FIFO filters

3.1.3.21 void flexcan_hal_set_max_mb_number (uint8_t *instance,* const flexcan_user_config_t * *data*)

Parameters

instance	The FlexCAN instance number
data	The FlexCAN platform data

3.1.3.22 flexcan_status_t flexcan_hal_set_id_filter_table_elements (uint8_t *instance,* const flexcan_user_config_t * *data,* flexcan_rx_fifo_id_element_format_t *id_format,* flexcan_id_table_t * *id_filter_table*)

Parameters

instance	The FlexCAN instance number

data	The FlexCAN platform data
id_format	The format of the Rx FIFO ID Filter Table Elements
id_filter_table	The ID filter table elements which contain if RTR bit, IDE bit and RX message ID need to be set.

Returns

0 if successful; non-zero failed.

3.1.3.23 flexcan_status_t flexcan_hal_set_rx_fifo (uint8_t *instance*, const flexcan_user_config_t * *data*, flexcan_rx_fifo_id_element_format_t *id_format*, flexcan_id_table_t * *id_filter_table*)

Parameters

instance	The FlexCAN instance number
data	The FlexCAN platform data
id_format	The format of the Rx FIFO ID Filter Table Elements
id_filter_table	The ID filter table elements which contain RTR bit, IDE bit, and RX message ID.

Returns

0 if successful; non-zero failed.

3.1.3.24 flexcan_status_t flexcan_hal_read_fifo (uint8_t *instance*, flexcan_mb_t * *rx_fifo*)

Parameters

instance	The FlexCAN instance number
rx_fifo	The FlexCAN receive FIFO data

Returns

0 if successful; non-zero failed.

3.1.3.25 flexcan_status_t flexcan_hal_enable_mb_interrupt (uint8_t *instance*, const flexcan_user_config_t * *data*, uint32_t *mb_idx*)

Parameters

instance	The FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer

Returns

0 if successful; non-zero failed

3.1.3.26 flexcan_status_t flexcan_hal_disable_mb_interrupt (uint8_t *instance,* const flexcan_user_config_t * *data,* uint32_t *mb_idx*)

Parameters

instance	The FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer

Returns

0 if successful; non-zero failed

3.1.3.27 void flexcan_hal_enable_error_interrupt (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.28 void flexcan_hal_disable_error_interrupt (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.29 void flexcan_hal_enable_bus_off_interrupt (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.30 void flexcan_hal_disable_bus_off_interrupt (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.31 void flexcan_hal_enable_wakeup_interrupt (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.32 void flexcan_hal_disable_wakeup_interrupt (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.33 void flexcan_hal_enable_tx_warning_interrupt (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.34 void flexcan_hal_disable_tx_warning_interrupt (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.35 void flexcan_hal_enable_rx_warning_interrupt (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.36 void flexcan_hal_disable_rx_warning_interrupt (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.37 static uint32_t flexcan_hal_get_freeze_ack (uint8_t *instance*) [inline], [static]

Parameters

instance	The FlexCAN instance number

Returns

freeze ACK state (1-freeze mode, 0-not in freeze mode).

3.1.3.38 uint8_t flexcan_hal_get_mb_int_flag (uint8_t *instance,* const flexcan_user_config_t * *data,* uint32_t *mb_idx*)

Parameters

instance	The FlexCAN instance number

data	The FlexCAN platform data
mb_idx	Index of the message buffer

Returns

the individual MB interrupt flag (0 and 1 are the flag value)

3.1.3.39 static uint32_t flexcan_hal_get_all_mb_int_flags (uint8_t *instance*) [inline], [static]

Parameters

instance	The FlexCAN instance number

Returns

all MB interrupt flags

3.1.3.40 static void flexcan_hal_clear_mb_int_flag (uint8_t *instance,* uint32_t *reg_val*) [inline], [static]

Parameters

instance	The FlexCAN instance number
reg_val	The value to be written to the interrupt flag1 register.

3.1.3.41 void flexcan_hal_get_err_counter (uint8_t *instance,* flexcan_berr_counter_t * *err_cnt*)

Parameters

instance	The FlexCAN instance number
err_cnt	Transmit error counter and receive error counter

3.1.3.42 static uint32_t flexcan_hal_get_err_status (uint8_t *instance*) [inline], [static]

Parameters

instance	The FlexCAN instance number

Returns

The current error and status

3.1.3.43 void flexcan_hal_clear_err_interrupt_status (uint8_t *instance*)

Parameters

instance	The FlexCAN instance number

3.1.3.44 void flexcan_hal_set_mask_type (uint8_t *instance,* flexcan_rx_mask_type_t *type*)

Parameters

instance	The FlexCAN instance number
type	The FlexCAN Rx mask type

3.1.3.45 void flexcan_hal_set_rx_fifo_global_std_mask (uint8_t *instance,* uint32_t *std_mask*)

Parameters

instance	The FlexCAN instance number
std_mask	Standard mask

3.1.3.46 void flexcan_hal_set_rx_fifo_global_ext_mask (uint8_t *instance,* uint32_t *ext_mask*)

Parameters

instance	The FlexCAN instance number
ext_mask	Extended mask

3.1.3.47 flexcan_status_t flexcan_hal_set_rx_individual_std_mask (uint8_t *instance,* const flexcan_user_config_t * *data,* uint32_t *mb_idx,* uint32_t *std_mask*)

Parameters

instance	The FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer
std_mask	Individual standard mask

Returns

0 if successful; non-zero failed

3.1.3.48 flexcan_status_t flexcan_hal_set_rx_individual_ext_mask (uint8_t *instance,* const flexcan_user_config_t * *data,* uint32_t *mb_idx,* uint32_t *ext_mask*)

Parameters

instance	The FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer
ext_mask	Individual extended mask

Returns

0 if successful; non-zero failed

3.1.3.49 void flexcan_hal_set_rx_mb_global_std_mask (uint8_t *instance,* uint32_t *std_mask*)

Parameters

instance	The FlexCAN instance number
std_mask	Standard mask

3.1.3.50 void flexcan_hal_set_rx_mb_buf14_std_mask (uint8_t *instance,* uint32_t *std_mask*)

Parameters

instance	The FlexCAN instance number
std_mask	Standard mask

3.1.3.51 void flexcan_hal_set_rx_mb_buf15_std_mask (uint8_t *instance,* uint32_t *std_mask*)

Parameters

instance	The FlexCAN instance number
std_mask	Standard mask

Returns

0 if successful; non-zero failed

3.1.3.52 void flexcan_hal_set_rx_mb_global_ext_mask (uint8_t *instance,* uint32_t *ext_mask*)

Parameters

instance	The FlexCAN instance number
ext_mask	Extended mask

3.1.3.53 void flexcan_hal_set_rx_mb_buf14_ext_mask (uint8_t *instance,* uint32_t *ext_mask*)

Parameters

instance	The FlexCAN instance number
ext_mask	Extended mask

3.1.3.54 void flexcan_hal_set_rx_mb_buf15_ext_mask (uint8_t *instance,* uint32_t *ext_mask*)

Parameters

instance	The FlexCAN instance number
ext_mask	Extended mask

3.1.3.55 static uint32_t flexcan_hal_get_rx_fifo_id_acceptance_filter (uint8_t *instance*) [inline], [static]

Parameters

instance	The FlexCAN instance number

Returns

RX FIFO information

3.2 FlexCAN Driver

The chapter describes the programming interface of the FlexCAN Peripheral driver.

Data Structures

- struct flexcan_bitrate_table_t
  FlexCAN bit rate and the related timing segments structure. More...
- struct flexcan_data_info_t
  FlexCAN data info from user. More...

Enumerations

- enum flexcan_bitrate_t {
  kFlexCanBitrate_125k = 125000,
  kFlexCanBitrate_250k = 250000,
  kFlexCanBitrate_500k = 500000,
  kFlexCanBitrate_750k = 750000,
  kFlexCanBitrate_1M = 1000000 }
  FlexCAN bitrates supported.

Bit rate

- flexcan_status_t flexcan_set_bitrate (uint8_t instance, flexcan_bitrate_t bitrate)
  Sets FlexCAN bit rate.
- flexcan_status_t flexcan_get_bitrate (uint8_t instance, flexcan_bitrate_t *bitrate)
  Gets FlexCAN bit rate.

Global mask

- void flexcan_set_mask_type (uint8_t instance, flexcan_rx_mask_type_t type)
  Sets the RX masking type.
- flexcan_status_t flexcan_set_rx_fifo_global_mask (uint8_t instance, flexcan_mb_id_type_t id_type, uint32_t mask)
  Sets the FlexCAN RX FIFO global standard or extended mask.
- flexcan_status_t flexcan_set_rx_mb_global_mask (uint8_t instance, flexcan_mb_id_type_t id_type, uint32_t mask)
  Sets the FlexCAN RX MB global standard or extended mask.
- flexcan_status_t flexcan_set_rx_individual_mask (uint8_t instance, const flexcan_user_config_t *data, flexcan_mb_id_type_t id_type, uint32_t mb_idx, uint32_t mask)
  Sets the FlexCAN RX individual standard or extended mask.

Init and Shutdown

- flexcan_status_t flexcan_init (uint8_t instance, const flexcan_user_config_t *data, bool enable_err-_interrupts)
  Initializes the FlexCAN peripheral.
- uint32_t flexcan_shutdown (uint8_t instance)
  Shut down a FlexCAN instance.

Send configuration

- flexcan_status_t flexcan_tx_mb_config (uint8_t instance, const flexcan_user_config_t *data, uint32-_t mb_idx, flexcan_data_info_t *tx_info, uint32_t msg_id)
  FlexCAN transmit message buffer field configuration.
- flexcan_status_t flexcan_send (uint8_t instance, const flexcan_user_config_t *data, uint32_t mb_-idx, flexcan_data_info_t *tx_info, uint32_t msg_id, uint32_t num_bytes, uint8_t *mb_data)
  Sends FlexCAN messages.

Receive configuration

- flexcan_status_t flexcan_rx_mb_config (uint8_t instance, const flexcan_user_config_t *data, uint32-_t mb_idx, flexcan_data_info_t *rx_info, uint32_t msg_id)
  FlexCAN receive message buffer field configuration.
- flexcan_status_t flexcan_rx_fifo_config (uint8_t instance, const flexcan_user_config_t *data, flexcan_rx_fifo_id_element_format_t id_format, flexcan_id_table_t *id_filter_table)
  FlexCAN RX FIFO field configuration.
- flexcan_status_t flexcan_start_receive (uint8_t instance, const flexcan_user_config_t *data, uint32-_t mb_idx, uint32_t receiveDataCount, bool *is_rx_mb_data, bool *is_rx_fifo_data, flexcan_mb_t *rx_mb, flexcan_mb_t *rx_fifo)
  FlexCAN is waiting to receive data.
- flexcan_status_t flexcan_receive (uint8_t instance, const flexcan_user_config_t *data, uint32_-t mb_idx, flexcan_data_info_t *rx_info, uint32_t msg_id, flexcan_rx_fifo_id_element_format_-t id_format, flexcan_id_table_t *id_filter_table, uint32_t receiveDataCount, flexcan_mb_t *rx_mb, flexcan_mb_t *rx_fifo)
  FlexCAN is preparing to receive data.

3.2.0.56 FlexCAN Driver

Overview

The FlexCAN (flexible controller area network) module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. The FlexCAN module supports both standard and extended message frames. The Message Buffers are stored in an embedded RAM dedicated to the FlexCAN module. The CAN Protocol Engine (PE) sub-module manages the serial communication

on the CAN bus by requesting RAM access for receiving and transmitting message frames, validating received messages, and performing error handling.

Initialization

To initialize the FlexCAN driver, call the flexcan_init() function and pass the instance number of the FlexCAN you want to use. For instance, to use FlexCAN0, pass a value of 0 to the flexcan_init function. In addition, you should also pass a user configuration structure #flexcan_config_t, as shown here:

```
// FlexCAN configuration structure for user
typedef struct FLEXCANConfig {
    uint32_t num_mb;
    uint32_t max_num_mb;
    flexcan_rx_fifo_id_filter_num_t num_id_filters;
    bool is_rx_fifo_needed;
    bool is_rx_mb_needed;
} flexcan_config_t;
```

Typically, the user configures the #flexcan_config_t instantiation as 16 message buffers needed, 16 message buffers available, 8 RX FIFO ID filters, set to true when RX FIFO is needed and when RX Message Buffers are needed. The user can easily modify the #flexcan_config_t instantiation to configure the FlexCAN peripheral to a different number of message buffers, which are used with/without RX FIFO and RX MB. This is a code example to set up a FlexCAN configuration instantiation:

```
flexcan_config_t flexcan1_data;
flexcan1_data.num_mb = 16;
flexcan1_data.max_num_mb = 16;
flexcan1_data.num_id_filters = kFlexCanRxFifoIDFilters_8;
flexcan1_data.is_rx_fifo_needed = true;
flexcan1_data.is_rx_mb_needed = true;
```

Module timing

FlexCAN bit rate is derived from the serial clock, which is generated by dividing the PE clock by the programmed PRESDIV value. Each serial clock period is also called a time quantum. The FlexCAN bit-rate is defined as the Sclock divided by the number of time quanta, where time quanta are further broken down segments within the bit time (time to transmit and sample a bit). The following list shows the CAN bit-rates that are supported in the FlexCAN driver.

- 1 Mbytes/s
- 750 Kbytes/s
- 500 Kbytes/s
- 250 Kbytes/s
- 125 Kbytes/s

The FlexCAN module supports several different ways to set up the bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2, and RJW.

To calculate the CAN bit timing parameters, use the method outlined in AN1798, chapter 4.1. A maximum time for PROP_SEG is used, the remaining TQ is split equally between PSEG1 and PSEG2, provided P-SEG2 >=2. RJW is set to the minimum of 4 or to the PSEG1.

Transfers

To transmit a FlexCAN frame, the CPU must prepare a message buffer for transmission by calling the flexcan_send() function. To receive the FlexCAN frames into a message buffer, the CPU must also prepare it for reception by calling the flexcan_receive() function. The FlexCAN driver implements send and receive configurations and prepares to send and receive data from TX MBs to RX MBs or/and RX FIFO. The FlexCAN uses only the interrupt-driven process to transfer data.

3.2.1 Data Structure Documentation

3.2.1.1 struct flexcan_bitrate_table_t

Data Fields

- flexcan_bitrate_t bit_rate
  bit rate
- uint32_t propseg
  Propagation segment.
- uint32_t pseg1
  Phase segment 1.
- uint32_t pseg2
  Phase segment 2.
- uint32_t pre_divider
  Clock pre divider.
- uint32_t rjw
  Re-sync jump width.

3.2.1.2 struct flexcan_data_info_t

Data Fields

- flexcan_mb_id_type_t msg_id_type
  Type of message ID (standard or extended)
- uint32_t data_length
  Length of Data in Bytes.

3.2.2 Enumeration Type Documentation

3.2.2.1 enum flexcan_bitrate_t

Enumerator

kFlexCanBitrate_125k 125 kHz
kFlexCanBitrate_250k 250 kHz
kFlexCanBitrate_500k 500 kHz
kFlexCanBitrate_750k 750 kHz
kFlexCanBitrate_1M 1 MHz

3.2.3 Function Documentation

3.2.3.1 flexcan_status_t flexcan_set_bitrate (uint8_t *instance,* flexcan_bitrate_t *bitrate*)

Parameters

instance	A FlexCAN instance number
bit	rate Selects a FlexCAN bit rate in the bit_rate_table.

Returns

0 if successful; non-zero failed

3.2.3.2 flexcan_status_t flexcan_get_bitrate (uint8_t *instance,* flexcan_bitrate_t * *bitrate*)

Parameters

instance	A FlexCAN instance number
bit	rate A pointer to a variable for returning the FlexCAN bit rate in the bit_rate_table.

Returns

0 if successful; non-zero failed

3.2.3.3 void flexcan_set_mask_type (uint8_t *instance,* flexcan_rx_mask_type_t *type*)

Parameters

instance	A FlexCAN instance number
type	The FlexCAN RX mask type

3.2.3.4 flexcan_status_t flexcan_set_rx_fifo_global_mask (uint8_t *instance,* flexcan_mb_id_type_t *id_type,* uint32_t *mask*)

Parameters

instance	A FlexCAN instance number
id_type	Standard ID or extended ID
mask	Mask value

Returns

0 if successful; non-zero failed

3.2.3.5 flexcan_status_t flexcan_set_rx_mb_global_mask (uint8_t *instance,* flexcan_mb_id_type_t *id_type,* uint32_t *mask*)

Parameters

instance	A FlexCAN instance number
id_type	Standard ID or extended ID
mask	Mask value

Returns

0 if successful; non-zero failed

3.2.3.6 flexcan_status_t flexcan_set_rx_individual_mask (uint8_t *instance,* const flexcan_user_config_t * *data,* flexcan_mb_id_type_t *id_type,* uint32_t *mb_idx,* uint32_t *mask*)

Parameters

instance	A FlexCAN instance number
id_type	A standard ID or an extended ID
mb_idx	Index of the message buffer
mask	Mask value

Returns

0 if successful; non-zero failed.

3.2.3.7 flexcan_status_t flexcan_init (uint8_t *instance,* const flexcan_user_config_t * *data,* bool *enable_err_interrupts*)

This function initializes

Parameters

instance	A FlexCAN instance number
data	The FlexCAN platform data
enable_err_interrupts	1 if enabled, 0 if not

Returns

0 if successful; non-zero failed

3.2.3.8 uint32_t flexcan_shutdown (uint8_t *instance*)

Parameters

instance	A FlexCAN instance number

Returns

0 if successful; non-zero failed

3.2.3.9 flexcan_status_t flexcan_tx_mb_config (uint8_t *instance,* const flexcan_user_config_t * *data,* uint32_t *mb_idx,* flexcan_data_info_t * *tx_info,* uint32_t *msg_id*)

Parameters

instance	A FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer
tx_info	Data info
msg_id	ID of the message to transmit

Returns

0 if successful; non-zero failed

3.2.3.10 flexcan_status_t flexcan_send (uint8_t *instance*, const flexcan_user_config_t * *data*, uint32_t *mb_idx*, flexcan_data_info_t * *tx_info*, uint32_t *msg_id*, uint32_t *num_bytes*, uint8_t * *mb_data*)

Parameters

instance	A FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer
tx_info	Data info
msg_id	ID of the message to transmit
num_bytes	The number of bytes in mb_data
mb_data	Bytes of the FlexCAN message

Returns

0 if successful; non-zero failed

3.2.3.11 flexcan_status_t flexcan_rx_mb_config (uint8_t *instance*, const flexcan_user_config_t * *data*, uint32_t *mb_idx*, flexcan_data_info_t * *rx_info*, uint32_t *msg_id*)

Parameters

instance	A FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer
rx_info	Data info
msg_id	ID of the message to transmit

Returns

0 if successful; non-zero failed

3.2.3.12 flexcan_status_t flexcan_rx_fifo_config (uint8_t *instance,* const flexcan_user_config_t * *data,* flexcan_rx_fifo_id_element_format_t *id_format,* flexcan_id_table_t * *id_filter_table*)

Parameters

instance	A FlexCAN instance number
data	The FlexCAN platform data
id_format	The format of the Rx FIFO ID Filter Table Elements
id_filter_table	The ID filter table elements which contain RTR bit, IDE bit, and RX message ID

Returns

0 if successful; non-zero failed.

3.2.3.13 flexcan_status_t flexcan_start_receive (uint8_t *instance,* const flexcan_user_config_t * *data,* uint32_t *mb_idx,* uint32_t *receiveDataCount,* bool * *is_rx_mb_data,* bool * *is_rx_fifo_data,* flexcan_mb_t * *rx_mb,* flexcan_mb_t * *rx_fifo*)

Parameters

instance	A FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer
msg_id	ID of the message to transmit
receiveData-Count	The number of data to be received
rx_mb	The FlexCAN receive message buffer data.
rx_fifo	The FlexCAN receive FIFO data.

Returns

0 if successful; non-zero failed

3.2.3.14 flexcan_status_t flexcan_receive (uint8_t *instance,* const flexcan_user_config_t * *data,* uint32_t *mb_idx,* flexcan_data_info_t * *rx_info,* uint32_t *msg_id,* flexcan_rx_fifo_id_element_format_t *id_format,* flexcan_id_table_t * *id_filter_table,* uint32_t *receiveDataCount,* flexcan_mb_t * *rx_mb,* flexcan_mb_t * *rx_fifo*)

Parameters

instance	A FlexCAN instance number
data	The FlexCAN platform data
mb_idx	Index of the message buffer
cs	CODE/status values (RX)
msg_id	ID of the message to transmit
id_format	The format of the Rx FIFO ID Filter Table Elements
id_filter_table	The ID filter table elements which contain RTR bit, IDE bit, and RX message ID
receiveData-Count	The number of data to be received

rx_mb	The FlexCAN receive message buffer data
rx_fifo	The FlexCAN receive FIFO data

Returns

0 if successful; non-zero failed

Chapter 4
Clock Manager (Clock)

The Kinetis SDK Clock Manager provides a set of API/services to configure the clock-related IPs, such as MCG, SIM, etc.

Enumerations

- enum clock_names_t
  Clock names.
- enum clock_gate_module_names_t
  Clock gate module names.
- enum clock_source_names_t
  Clock source and SEL names.
- enum clock_manager_error_code_t {
  kClockManagerSuccess,
  kClockManagerNoSuchClockName,
  kClockManagerNoSuchClockModule,
  kClockManagerNoSuchClockSource,
  kClockManagerNoSuchDivider,
  kClockManagerUnknown }
  Error code definition for the clock manager APIs.

Variables

- sim_clock_names_t kClockNameSimMap [kClockNameCount]
  Clock manager clock names mapping into the SIM clock name.
- sim_clock_source_names_t kClockSourceNameSimMap [kClockSourceMax]
  Clock manager clock source names mapping into the SIM clock source name.
- sim_clock_gate_module_names_t kClockModuleNameSimMap [kClockModuleMax]
  Clock manager clock module names mapping into the SIM clock module name.

Clock Gating

- clock_manager_error_code_t clock_manager_set_gate (clock_gate_module_names_t moduleName, uint8_t instance, bool enable)
  Enables or disables the clock for a specific clock module.
- clock_manager_error_code_t clock_manager_get_gate (clock_gate_module_names_t moduleName, uint8_t instance, bool *isEnabled)
  Gets the current clock gate status for a specific clock module.

Clock Frequencies

- clock_manager_error_code_t clock_manager_get_frequency (clock_names_t clockName, uint32_t *frequency)

Gets the clock frequency for a specific clock name.

- clock_manager_error_code_t clock_manager_get_frequency_by_source (clock_source_names_t clockSource, uint32_t *frequency)
  Gets the clock frequency for a specified clock source.

System out clock access API

- uint32_t clock_hal_get_outclk (void)
  Gets the current out clock.
- uint32_t clock_hal_get_fllclk (void)
  Gets the current FLL clock.
- uint32_t clock_hal_get_pll0clk (void)
  Gets the current PLL0 clock.
- uint32_t clock_hal_get_pll1clk (void)
  Gets the current PLL1 clock.
- uint32_t clock_hal_get_irclk (void)
  Get the current IR (internal reference) clock.

4.0.4 Clock Manager

Overview

The Clock Manager is configures, accesses core, platform, system and bus clock setting. It provides a set of APIs to get and set functions of the clock gate control.

It includes manager-level, SIM HAL-level, and MCG HAL-level access APIs.

Clock names, Clock source names and Clock module names

There are three sets of clock related names: clock names, clock source names, and clock module names. Each CPU only supports a subset of clock related names. If there is a new clock or a clock module that does exist in the current name set, it has to be added in the definition.

Clock names get a system clock frequency. The clock module names control and access the clock module gate status for a specific module.

Example to get a system clock frequency:

```
#include "clock/fsl_clock_manager.h"
uint32_t uFrequency = 0;

// get the system clock frequency based on current mode configuration
if (kClockManagerSuccess == clock_manager_get_frequency(
      kSystemClock, &frequency))
   // check the frequency value for system clock in frequency
```

Example to get a clock frequency based on the clock source:

```
#include "fsl_clock_manager.h"
uint32_t uFrequency = 0;
```

```
// get the USBSRC clock frequency based on current mode configuration
if (kClockManagerSuccess ==
      clock_manager_get_frequency_by_source(kClockUsbSrc, &frequency))
   // check the frequency value for USBSRC clock in frequency
```

Example to enable a clock module:

```
#include "fsl_clock_manager.h"
clock_manager_error_code_t retCode = 0;

// enable the DMA clock
retCode = clock_manager_set_gate(kClockModuleDMA, 0, true);
// check the error to confirm if the enabling is done successfully
```

4.1 Enumeration Type Documentation

4.1.1 enum clock_manager_error_code_t

Enumerator

kClockManagerSuccess success
kClockManagerNoSuchClockName cannot find the clock name
kClockManagerNoSuchClockModule cannot find the clock module name
kClockManagerNoSuchClockSource cannot find the clock source name
kClockManagerNoSuchDivider cannot find the divider name
kClockManagerUnknown unknown error

4.2 Function Documentation

4.2.1 clock_manager_error_code_t clock_manager_set_gate (clock_gate_module_names_t *moduleName*, uint8_t *instance*, bool *enable*)

This function enables/disables the clock for a specified clock module and instance. See the clock_gate_module_names_t for supported clock module names for a specific function and see the Reference Manual for supported clock module name for a specific chip family. Most module drivers call this function to gate(disable)/ungate(enable) the clock for a module. However, the application can also call this function as needed. Disabling the clock causes the module to stop working. See the Reference Manual to properly enable and disable the clock for a device module.

Parameters

moduleName	Gate control module name defined in clock_gate_module_names_t

instance	Instance of the module
enable	Enable or disable the clock for the module • true: Enable • false: Disable

Returns

status Error code defined in clock_manager_error_code_t

4.2.2 clock_manager_error_code_t clock_manager_get_gate (clock_gate_module_names_t *moduleName,* uint8_t *instance,* bool * *isEnabled*)

This function returns the current clock gate status for a specific clock module. See clock_gate_module_names_t for supported clock module name.

Parameters

moduleName	Gate control module name defined in clock_gate_module_names_t
instance	Instance of the module
isEnabled	Status of the module clock • true: Enabled • false: Disabled

Returns

status Error code defined in clock_manager_error_code_t

4.2.3 clock_manager_error_code_t clock_manager_get_frequency (clock_names_t *clockName,* uint32_t * *frequency*)

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock_names_t. The MCG must be properly configured before using this function. See the Reference Manual for supported clock names for different chip families. The returned value is in Hertz. If it cannot find the clock name or the name is not supported for a specific chip family, it returns an error.

Parameters

clockName	Clock names defined in clock_names_t
frequency	Returned clock frequency value in Hertz

Returns

status Error code defined in clock_manager_error_code_t

4.2.4 clock_manager_error_code_t clock_manager_get_frequency_by_source (clock_source_names_t *clockSource,* uint32_t * *frequency*)

This function gets the specified clock source setting and converts it into a clock name. It calls the internal function to get the value for that clock name. The returned value is in Hertz. If it cannot find the clock source or the source is not supported for the specific chip family, it returns an error.

Parameters

clockSource	Clock source names defined in clock_source_names_t
frequency	Returned clock frequency value in Hertz

Returns

status Error code defined in clock_manager_error_code_t

4.2.5 uint32_t clock_hal_get_outclk (void)

Parameters

none	

Returns

frequency Out clock frequency for the clock system

4.2.6 uint32_t clock_hal_get_fllclk (void)

Parameters

none	

Returns

frequency FLL clock frequency for the clock system

4.2.7 uint32_t clock_hal_get_pll0clk (void)

Parameters

none	

Returns

frequency PLL0 clock frequency for the clock system

4.2.8 uint32_t clock_hal_get_pll1clk (void)

Parameters

none	

Returns

frequency PLL1 clock frequency for the clock system

4.2.9 uint32_t clock_hal_get_irclk (void)

Parameters

none	

Returns

frequency IR clock frequency for the clock system

Chapter 5
Direct Memory Access (DMA)

The Kinetis SDK provides both HAL and Peripheral drivers for the Direct Memory Access block of Kinetis devices.

Modules

- DMA Driver
  The part describes the programming interface of the DMA Peripheral driver.
- DMA HAL driver
  The part describes the programming interface of the DMA HAL driver.
- DMA request
  DMA request resource.
- DMAMUX HAL driver
  The part describes the programming interface of the DMAMUX HAL module.

5.1 DMA HAL driver

The chapter describes the programming interface of the DMA HAL driver.

Data Structures

- struct dma_channel_link_config_t
  Data structure for data structure configuration. More...
- union dma_error_status_t
  Data structure to get status of the DMA channel status. More...

Enumerations

- enum dma_status_t { ,
  kStatus_DMA_InvalidArgument = 1U,
  kStatus_DMA_Fail = 2U }
  DMA status.
- enum dma_transfer_size_t {
  kDmaTransfersize32bits = 0x0U,
  kDmaTransfersize8bits = 0x1U,
  kDmaTransfersize16bits = 0x2U }
  DMA transfer size type.
- enum dma_modulo_t
  Configuration type for the DMA modulo.
- enum dma_channel_link_type_t {
  kDmaChannelLinkDisable = 0x0U,
  kDmaChannelLinkChan1AndChan2 = 0x1U,
  kDmaChannelLinkChan1 = 0x2U,
  kDmaChannelLinkChan1AfterBCR0 = 0x3 }
  DMA channel link type.

DMA HAL channel configuration

- static void dma_hal_configure_source_address (uint32_t instance, uint32_t channel, uint32_t address)
  Configures the source address.
- static void dma_hal_configure_dest_address (uint32_t instance, uint32_t channel, uint32_t address)
  Configures the source address.
- static void dma_hal_configure_count (uint32_t instance, uint32_t channel, uint32_t count)
  Configures the bytes to be transferred.
- static uint32_t dma_hal_get_unfinished_bytes (uint32_t instance, uint32_t channel)
  Gets the left bytes not to be transferred.
- static void dma_hal_enable_interrupt (uint32_t instance, uint8_t channel)
  Enables the interrupt for the DMA channel after the work is done.
- static void dma_hal_disable_interrupt (uint32_t instance, uint8_t channel)
  Disables the interrupt for the DMA channel after the work is done.

- static void dma_hal_set_cycle_steal (uint32_t instance, uint8_t channel, bool isCycleSteal)
  Configure the DMA transfer mode to cycle steal or continuous modes.
- static void dma_hal_set_autoalign_ability (uint32_t instance, uint8_t channel, bool isEnabled)
  Configures the auto-align feature.
- static void dma_hal_set_async_dma_request_ability (uint32_t instance, uint8_t channel, bool isEnabled)
  Configures the a-sync DMA request feature.
- static void dma_hal_set_source_increment (uint32_t instance, uint32_t channel, bool isEnabled)
  Enables/Disables the source increment.
- static void dma_hal_set_dest_increment (uint32_t instance, uint32_t channel, bool isEnabled)
  Enables/Disables destination increment.
- static void dma_hal_configure_source_transfersize (uint32_t instance, uint32_t channel, dma_transfer_size_t transfersize)
  Configures the source transfer size.
- static void dma_hal_configure_dest_transfersize (uint32_t instance, uint32_t channel, dma_transfer_size_t transfersize)
  Configures the destination transfer size.
- static void dma_hal_trigger_start (uint32_t instance, uint32_t channel)
  Triggers the start.
- static void dma_hal_configure_source_modulo (uint32_t instance, uint32_t channel, dma_modulo_t modulo)
  Configures the modulo for source address.
- static void dma_hal_configure_dest_modulo (uint32_t instance, uint32_t channel, dma_modulo_t modulo)
  Configures the modulo for destination address.
- static void dma_hal_set_dma_request (uint32_t instance, uint32_t channel, bool isEnabled)
  Enables/Disables the DMA request.
- static void dma_hal_set_disable_dma_request_after_done (uint32_t instance, uint32_t channel, bool isDisabled)
  Configures the DMA request state after the work is done.
- void dma_hal_set_channel_link (uint32_t instance, uint8_t channel, dma_channel_link_config_t *mode)
  Configures the channel link feature.
- static void dma_hal_clear_status (uint32_t instance, uint8_t channel)
  Clears the status of DMA channel.
- static dma_error_status_t dma_hal_get_status (uint32_t instance, uint8_t channel)
  Gets the DMA controller channel status.

5.1.0.1 DMA HAL Driver

Overview

The DMA HAL driver is used to mask the hardware and provide user a comprehensible way to use dma hardware.

5.1.1 Data Structure Documentation

5.1.1.1 struct dma_channel_link_config_t

Data Fields

- dma_channel_link_type_t linkType
  Channel link type.
- uint32_t channel1
  Channel 1 configuration.
- uint32_t channel2
  Channel 2 configuration.

5.1.1.2 union dma_error_status_t

5.1.1.2.0.4 Field Documentation

5.1.1.2.0.4.1 uint32_t dma_error_status_t::dmaTransDone

5.1.1.2.0.4.2 uint32_t dma_error_status_t::dmaBusy

5.1.1.2.0.4.3 uint32_t dma_error_status_t::dmaPendingRequest

5.1.2 Enumeration Type Documentation

5.1.2.1 enum dma_status_t

Enumerator

kStatus_DMA_InvalidArgument Parameter is not available for the current configuration.
kStatus_DMA_Fail Function operation failed.

5.1.2.2 enum dma_transfer_size_t

Enumerator

kDmaTransfersize32bits 32 bits are transferred for every read/write
kDmaTransfersize8bits 8 bits are transferred for every read/write
kDmaTransfersize16bits 16b its are transferred for every read/write

5.1.2.3 enum dma_channel_link_type_t

Enumerator

kDmaChannelLinkDisable No channel link.

kDmaChannelLinkChan1AndChan2 Perform a link to channel 1 after each cycle-steal transfer followed by a link and to channel 2 after the BCR decrements to zeros.
kDmaChannelLinkChan1 Perform a link to channel 1 after each cycle-steal transfer.
kDmaChannelLinkChan1AfterBCR0 Perform a link to channel1 after the BCR decrements to zero.

5.1.3 Function Documentation

5.1.3.1 static void dma_hal_configure_source_address (uint32_t *instance,* uint32_t *channel,* uint32_t *address*) [inline], [static]

Each SAR contains the byte address used by the DMA to read data. The SARn is typically aligned on a 0-modulo-size boundary-that is on the natural alignment of the source data. Bits 31-20 of this register must be written with one of the only four allowed values. Each of these allowed values corresponds to a valid region of the devices' memory map. The allowed values are: 0x000x_xxxx 0x1FFx_xxxx 0x200x_xxxx 0x400x_xxxx After they are written with one of the allowed values, bits 31-20 read back as the written value. After they are written with any other value, bits 31-20 read back as an indeterminate value.

This function enables the request for a specified channel.

Parameters

instance	DMA instance.
channel	DMA channel.
address	memory address pointing to the source address.

5.1.3.2 static void dma_hal_configure_dest_address (uint32_t *instance,* uint32_t *channel,* uint32_t *address*) [inline], [static]

Each DAR contains the byte address used by the DMA to read data. The DARn is typically aligned on a 0-modulo-size boundary-that is on the natural alignment of the source data. Bits 31-20 of this register must be written with one of the only four allowed values. Each of these allowed values corresponds to a valid region of the devices' memory map. The allowed values are: 0x000x_xxxx 0x1FFx_xxxx 0x200x_xxxx 0x400x_xxxx After they are written with one of the allowed values, bits 31-20 read back as the written value. After they are written with any other value, bits 31-20 read back as an indeterminate value.

This function enables the request for specified channel.

Parameters

instance	DMA instance.
channel	DMA channel.

5.1.3.3 static void dma_hal_configure_count (uint32_t *instance,* uint32_t *channel,* uint32_t *count*) [inline], [static]

Transfer bytes must be written with a value equal to or less than 0F_FFFFh. After being written with a value in this range, bits 23-20 of the BCR read back as 1110b. A write to the BCR with a value greater than 0F_FFFFh causes a configuration error when the channel starts to execute. After they are written with a value in this range, bits 23-20 of BCR read back as 1111b.

Parameters

instance	DMA instance.
channel	DMA channel.
count	bytes to be transferred.

5.1.3.4 static uint32_t dma_hal_get_unfinished_bytes (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	DMA instance.
channel	DMA channel.

Returns

unfinished bytes.

5.1.3.5 static void dma_hal_enable_interrupt (uint32_t *instance,* uint8_t *channel*) [inline], [static]

This function enables the request for specified channel.

Parameters

instance	DMA instance.
channel	DMA channel.

5.1.3.6 static void dma_hal_disable_interrupt (uint32_t *instance,* uint8_t *channel*) [inline], [static]

This function disables the request for a specified channel.

Parameters

instance	DMA instance.
channel	DMA channel.

5.1.3.7 static void dma_hal_set_cycle_steal (uint32_t *instance,* uint8_t *channel,* bool *isCycleSteal*) [inline], [static]

If continuous mode is enabled, DMA continuously makes write/read transfers until BCR decrement to 0. If continuous mode is disabled, DMA write/read is only triggered on every request. s

Parameters

instance	DMA instance.
channel	DMA channel.
isContinue	0 means cycle-steal mode, 1 means continuous mode.

5.1.3.8 static void dma_hal_set_autoalign_ability (uint32_t *instance,* uint8_t *channel,* bool *isEnabled*) [inline], [static]

If auto-align is enabled, the appropriate address register increments, regardless of whether it is a source increment or a destination increment.

Parameters

instance	DMA instance.
channel	DMA channel.

isEnable	0 means disable auto-align. 1 means enable auto-align.

5.1.3.9 static void dma_hal_set_async_dma_request_ability (uint32_t *instance,* uint8_t *channel,* bool *isEnabled*) [inline], [static]

Enables/Disables the a-synchronization mode in a STOP mode for each DMA channel.

Parameters

instance	DMA instance.
channel	DMA channel.
isEnable	0 means disable DMA request a-sync. 1 means enable DMA request -.

5.1.3.10 static void dma_hal_set_source_increment (uint32_t *instance,* uint32_t *channel,* bool *isEnabled*) [inline], [static]

Controls whether the source address increments after each successful transfer. If enabled, the SAR increments by 1,2,4 as determined by the transfer size.

Parameters

instance	DMA instance.
channel	DMA channel.
isEnabled	Enabled/Disable increment.

5.1.3.11 static void dma_hal_set_dest_increment (uint32_t *instance,* uint32_t *channel,* bool *isEnabled*) [inline], [static]

Controls whether the destination address increments after each successful transfer. If enabled, the DAR increments by 1,2,4 as determined by the transfer size.

Parameters

instance	DMA instance.

channel	DMA channel.
isEnabled	Enabled/Disable increment.

5.1.3.12 static void dma_hal_configure_source_transfersize (uint32_t *instance,* uint32_t *channel,* dma_transfer_size_t *transfersize*) [inline], [static]

Parameters

instance	DMA instance.
channel	DMA channel.
transfersize	enum type for transfer size.

5.1.3.13 static void dma_hal_configure_dest_transfersize (uint32_t *instance,* uint32_t *channel,* dma_transfer_size_t *transfersize*) [inline], [static]

Parameters

instance	DMA instance.
channel	DMA channel.
transfersize	enum type for transfer size.

5.1.3.14 static void dma_hal_trigger_start (uint32_t *instance,* uint32_t *channel*) [inline], [static]

When the DMA begins the transfer, the START bit is cleared automatically after one module clock and always reads as logic 0.

Parameters

instance	DMA instance.
channel	DMA channel.

5.1.3.15 static void dma_hal_configure_source_modulo (uint32_t *instance,* uint32_t *channel,* dma_modulo_t *modulo*) [inline], [static]

Parameters

instance	DMA instance.
channel	DMA channel.
modulo	enum data type for source modulo.

5.1.3.16 static void dma_hal_configure_dest_modulo (uint32_t *instance,* uint32_t *channel,* dma_modulo_t *modulo*) [inline], [static]

Parameters

instance	DMA instance.
channel	DMA channel.
modulo	enum data type for dest modulo.

5.1.3.17 static void dma_hal_set_dma_request (uint32_t *instance,* uint32_t *channel,* bool *isEnabled*) [inline], [static]

Parameters

instance	DMA instance.
channel	DMA channel.
isEnabled	

5.1.3.18 static void dma_hal_set_disable_dma_request_after_done (uint32_t *instance,* uint32_t *channel,* bool *isDisabled*) [inline], [static]

Disables/Enables the DMA request after a DMA DONE is generated. If it works in the loop mode, this bit should not be set.

Parameters

channel	DMA channel.

isDisabled	0 means DMA request would not be disabled after work done. 1 means disable.

5.1.3.19 void dma_hal_set_channel_link (uint32_t *instance,* uint8_t *channel,* dma_channel_link_config_t * *mode*)

Parameters

channel	DMA channel.
mode	Mode of channel link in DMA.

5.1.3.20 static void dma_hal_clear_status (uint32_t *instance,* uint8_t *channel*) [inline], [static]

This function clears all status for specified DMA channel. The error status and DONE status would all be cleared.

Parameters

channel	DMA channel.

5.1.3.21 static dma_error_status_t dma_hal_get_status (uint32_t *instance,* uint8_t *channel*) [inline], [static]

Gets the status of the DMA channel. The user can get the error status, as to whether the descriptor is finished or there are bytes left.

Parameters

channel	DMA channel.
mode	Mode of channel link in DMA.

Returns

Status of the DMA channel.

5.2 DMAMUX HAL driver

The chapter describes the programming interface of the DMAMUX HAL module.

Enumerations

- enum dmamux_dma_request_source { kDmamuxDmaRequestSource = 64U }
  A constant for the length of the DMA hardware source.

DMAMUX HAL function

- void dmamux_hal_init (uint8_t module)
  Initializes the DMAMUX module to the reset state.
- static void dmamux_hal_enable_channel (uint8_t module, uint8_t channel)
  Enables the DMAMUX channel.
- static void dmamux_hal_disable_channel (uint8_t module, uint8_t channel)
  Disables the DMAMUX channel.
- static void dmamux_hal_enable_period_trigger (uint8_t module, uint8_t channel)
  Enables the period trigger.
- static void dmamux_hal_disable_period_trigger (uint8_t module, uint8_t channel)
  Disables the period trigger.
- static void dmamux_hal_set_trigger_source (uint8_t module, uint8_t channel, uint8_t source)
  Configures the DMA request for the DMAMUX channel.

5.3 Enumeration Type Documentation

5.3.1 enum dmamux_dma_request_source

This structure is used inside the DMA driver.

Enumerator

kDmamuxDmaRequestSource Maximum number of the DMA requests allowed for the DMA mux.

5.4 Function Documentation

5.4.1 void dmamux_hal_init (uint8_t *module*)

Initializes the DMAMUX module to the reset state.

Parameters

module	DMAMUX module index

5.4.2 static void dmamux_hal_enable_channel (uint8_t *module,* uint8_t *channel*) [inline], [static]

Enables the hardware request. If enabled, the hardware request is sent to the corresponding DMA channel.

Parameters

module	DMAMUX module.
channel	DMAMUX channel.

5.4.3 static void dmamux_hal_disable_channel (uint8_t *module,* uint8_t *channel*) [inline], [static]

Disable hardware request. If disabled, the hardware request is not sent to the corresponding DMA channel.

Parameters

module	DMAMUX module.
channel	DMAMUX channel.

5.4.4 static void dmamux_hal_enable_period_trigger (uint8_t *module,* uint8_t *channel*) [inline], [static]

Parameters

module	DMAMUX module.
channel	DMAMUX channel.

5.4.5 static void dmamux_hal_disable_period_trigger (uint8_t *module,* uint8_t *channel*) [inline], [static]

Parameters

module	DMAMUX module.
channel	DMAMUX channel.

5.4.6 static void dmamux_hal_set_trigger_source (uint8_t *module,* uint8_t *channel,* uint8_t *source*) [inline], [static]

Sets the trigger source for the DMA channel. The trigger source is in the file fsl_dma_request.h.

Parameters

module	DMAMUX module.
channel	DMAMUX channel.
source	DMA request source.

5.5 DMA Driver

The chapter describes the programming interface of the DMA Peripheral driver.

Data Structures

- struct dma_channel_t
  Data structure for the DMA channel management. More...

Typedefs

- typedef void(∗ dma_callback_t)(void ∗parameter, dma_channel_status_t status)
  A definition for the DMA channel callback function.

Enumerations

- enum dma_channel_status_t {
  kDmaIdle,
  kDmaNormal,
  kDmaError }
  Channel status for DMA channel.
- enum dma_transfer_type_t {
  kDmaPeripheralToMemory,
  kDmaMemoryToPeripheral,
  kDmaMemoryToMemory,
  kDmaPeripheralToPeripheral }
  Type for DMA transfer.
- enum dma_channel_type_t {
  kDmaInvalidChannel = 0xFFU,
  kDmaAnyChannel = 0xFEU }
  Type for the DMA channel, which is used for the DMA channel allocation.

DMA Driver

- dma_status_t dma_init (void)
  Initializes the DMA.
- dma_status_t dma_deinit (void)
  De-initializes the DMA.
- dma_status_t dma_register_callback (dma_channel_t ∗chn, dma_callback_t callback, void ∗para)
  Registers the callback function and a parameter.
- uint32_t dma_get_descriptor_status (dma_channel_t ∗chn)
  Gets the status of the EDMA channel descriptor chain.
- uint32_t dma_request_channel (uint32_t channel, dma_request_source_t source, dma_channel_-t ∗chn)
  Requests a DMA channel.

- dma_status_t dma_free_channel (dma_channel_t *chn)
  Frees DMA channel hardware and software resource.
- dma_status_t dma_start_channel (dma_channel_t *chn)
  Starts a DMA channel.
- dma_status_t dma_stop_channel (dma_channel_t *chn)
  Stops a DMA channel.
- dma_status_t dma_config_transfer (dma_channel_t *chn, dma_transfer_type_t type, uint32_t size, uint32_t sourceAddr, uint32_t destAddr, uint32_t length)
  Configures a transfer for the DMA.
- void dma_IRQhandler (uint32_t channel)
  DMA IRQ handler for both an interrupt and an error.

5.5.0.1 DMA PERIPHERAL Driver

Overview

The DMA driver requests, configures, and uses the DMA hardware. It supports module initializations and DMA channel configurations.

Initialization

To initialize the DMA module, call the dma_init() function. Configuration data structure does not need to be passed. This function enables the DMA module and clock automatically.

Channel concept

DMA module consists of many channels. The DMA peripheral driver is designed based on the channel concept. All tasks should start by requesting a DMA channel and end by freeing a DMA channel. By getting a channel allocated, the user can configure and run operations on the DMA module. If a channel is not allocated, a system error may occur.

DMA request concept

DMA request triggers a DMA transfer. The DMA request table is available in chip configuration chapters in a Reference Manual.

Memory allocation

DMA peripheral driver does not allocate memory dynamically. The user must provide the allocated memory pointer for the driver and ensure that the memory is valid. Otherwise, a system error occurs. The

user needs to provide the memory for the [dma_channel_t]. The driver must store the status data for every channel and the dma_channel_t is designed for this purpose.

Call diagram

To use the DMA driver, follow these steps:

1. Initialize the DMA module: dma_init().
2. Request a DMA channel: dma_request_channel().
3. Configure the TCD: dma_config_transfer().
4. Register callback function: dma_register_callback().
5. Start the DMA channel: dma_start_channel().
6. [OPTION] Stop the DMA channel: dma_stop_channel().
7. Free the DMA channel: dma_free_channel().

This is an example code to initialize and configure a memory-to-memory transfer:

```
uint32_t j, temp;
dma_channel_t chan_handler;
uint8_t *srcAddr, *destAddr;
fsl_rtos_status syncStatus;

srcAddr = malloc(kDmaTestBufferSize);
destAddr = malloc(kDmaTestBufferSize);
if (((uint32_t)srcAddr == 0x0U) & ((uint32_t)destAddr == 0x0U))
{
    printf("Fali to allocate memory for test! \r\n");
    goto error;
}

/* Init the memory buffer. */
for (j = 0; j < kDmaTestBufferSize; j++)
{
    srcAddr[j] = j;
    destAddr[j] = 0;
}

temp = dma_request_channel(channel, kDmaRequestMux0AlwaysOn62, &chan_handler);
if (temp != channel)
{
    printf("Failed to request channel %d !\r\n", channel);
    goto error;
}

dma_config_transfer(&chan_handler, kDmaMemoryToMemory,
                                0x1U,
                                (uint32_t)srcAddr, (uint32_t)destAddr,
                                kDmaTestBufferSize);

dma_register_callback(&chan_handler, test_callback, &chan_handler);

dma_start_channel(&chan_handler);
//Wait until channel complete...
dma_stop_channel(&chan_handler);
dma_free_channel(&chan_handler);
```

5.5.1 Data Structure Documentation

5.5.1.1 struct dma_channel_t

Data Fields

- uint8_t channel
  Channel number.
- uint8_t dmamuxModule
  Dmamux module index.
- uint8_t dmamuxChannel
  Dmamux module channel.
- dma_callback_t callback
  Callback function for this channel.
- void * parameter
  Parameter for the callback function.
- volatile dma_channel_status_t status
  Channel status.

5.5.2 Typedef Documentation

5.5.2.1 typedef void(* dma_callback_t)(void *parameter, dma_channel_status_t status)

A prototype for the callback function registered into the DMA driver.

5.5.3 Enumeration Type Documentation

5.5.3.1 enum dma_channel_status_t

A structure describing the status of the DMA channel. The user can get the status from the channel callback function.

Enumerator

kDmaIdle DMA channel is idle.
kDmaNormal DMA channel is occupied.
kDmaError Error occurs in the DMA channel.

5.5.3.2 enum dma_transfer_type_t

Enumerator

kDmaPeripheralToMemory Transfer from the peripheral to memory.
kDmaMemoryToPeripheral Transfer from the memory to peripheral.
kDmaMemoryToMemory Transfer from the memory to memory.

kDmaPeripheralToPeripheral Transfer from the peripheral to peripheral.

5.5.3.3 enum dma_channel_type_t

Enumerator

kDmaInvalidChannel Macros indicating the failure of the channel request.
kDmaAnyChannel Macros used when requesting a channel. kEdmaAnyChannel means a request of dynamic channel allocation.

5.5.4 Function Documentation

5.5.4.1 dma_status_t dma_register_callback (dma_channel_t * *chn*, dma_callback_t *callback*, void * *para*)

The user registers the callback function and a parameter for a specified DMA channel. When the channel interrupt or a channel error happens, the callback and the parameter are called. The user parameter is also provided to give a channel status.

Parameters

chn	A handler for the DMA channel
callback	Callback function
para	A parameter for callback functions

5.5.4.2 uint32_t dma_get_descriptor_status (dma_channel_t * *chn*)

Gets the left bytes to be transferred.

Parameters

chn	A handler for the DMA channel

5.5.4.3 uint32_t dma_request_channel (uint32_t *channel*, dma_request_source_t *source*, dma_channel_t * *chn*)

This function provides two ways to allocate a DMA channel. The first way is a static allocation. The second way is a dynamic allocation. To allocate a channel dynamically, the user needs to set the channel parameter with the value of kDmaAnyChannel. The driver searches into all available free channels and assigns the first channel to the user. To allocate the channel statically, the user needs to set the channel parameter with the value of a specified channel. If the channel is available, the driver assigns the channel

to the user. Notes: The user must provide a handler memory for the DMA channel. The driver initializes the handler and configures the handler memory.

Parameters

channel	A DMA channel number. If a channel is assigned with a valid channel number, the DMA driver tries to assign a specified channel to the user. If a channel is assigned with kDmaAnyChannel, the DMA driver searches all available channels and assigns the first channel to the user.
source	A DMA hardware request.
chan	Memory pointing to DMA channel. The user must ensure that the handler memory is valid and that it will not be released or changed by any other codes before the channel dma_free_channel() operation.

Returns

If the channel allocation is successful, the return value indicates the requested channel. If not, the driver returns a kDmaInvalidChannel value to indicate that the request operation has failed.

5.5.4.4 dma_status_t dma_free_channel (dma_channel_t * *chn*)

This function frees the relevant software and hardware resources. Both the request and the free operations need to be called in a pair.

Parameters

chn	Memory pointing to DMA channel.

5.5.4.5 dma_status_t dma_start_channel (dma_channel_t * *chn*)

Starts a DMA channel. The driver starts a DMA channel by enabling the DMA request. A software start bit is not used in the DMA Peripheral driver.

Parameters

chn	Memory pointing to the DMA channel.

5.5.4.6 dma_status_t dma_stop_channel (dma_channel_t * *chn*)

Parameters

chn	Memory pointing to the DMA channel.

5.5.4.7 dma_status_t dma_config_transfer (dma_channel_t * *chn*, dma_transfer_type_t *type*, uint32_t *size*, uint32_t *sourceAddr*, uint32_t *destAddr*, uint32_t *length*)

Configures a transfer for the DMA.

Parameters

chn	Memory pointing to the DMA channel.
type	Transfer type.
size	Size to be transferred on each DMA write/read. Source/Dest share the same write/read size.
sourceAddr	Source address.
destAddr	Destination address.
length	Bytes to be transferred.

5.5.4.8 void dma_IRQhandler (uint32_t *channel*)

Parameters

channel	DMA channel number.

5.6 DMA request

DMA request resource.

Chapter 6
Serial Peripheral Interface (DSPI)

The Kinetis SDK provides both HAL and Peripheral drivers for the Serial Peripheral Interface (DSPI) block of Kinetis devices.

Modules

- DSPI HAL driver
  The part describes the programming interface of the DSPI HAL driver.
- DSPI Master Driver
  The part describes the programming interface of the DSPI master mode Peripheral driver.
- DSPI Shared IRQ Driver
  The part describes the programming interface of the DSPI shared IRQ driver for master and slave Peripheral drivers.
- DSPI Slave Driver
  The part describes the programming interface of the DSPI slave mode Peripheral driver.

6.1 DSPI HAL driver

The chapter describes the programming interface of the DSPI HAL driver.

Data Structures

- struct dspi_data_format_config_t
  DSPI data format settings configuration structure. More...
- struct dspi_master_config_t
  DSPI hardware configuration settings for master mode. More...
- struct dspi_slave_config_t
  DSPI hardware configuration settings for slave mode. More...
- struct dspi_baud_rate_divisors_t
  DSPI baud rate divisors settings configuration structure. More...
- struct dspi_delay_settings_config_t
  DSPI delay settings configuration structure. More...
- struct dspi_command_config_t
  DSPI command and data configuration structure. More...

Enumerations

- enum dspi_status_t { ,
  kStatus_DSPI_SlaveTxUnderrun,
  kStatus_DSPI_SlaveRxOverrun,
  kStatus_DSPI_Timeout,
  kStatus_DSPI_Busy,
  kStatus_DSPI_NoTransferInProgress,
  kStatus_DSPI_InvalidBitCount,
  kStatus_DSPI_InvalidInstanceNumber }
  Error codes for the DSPI driver.
- enum dspi_master_slave_mode_t {
  kDspiMaster = 1,
  kDspiSlave = 0 }
  DSPI master or slave configuration.
- enum dspi_clock_polarity_t {
  kDspiClockPolarity_ActiveHigh = 0,
  kDspiClockPolarity_ActiveLow = 1 }
  DSPI clock polarity configuration for a given CTAR.
- enum dspi_clock_phase_t {
  kDspiClockPhase_FirstEdge = 0,
  kDspiClockPhase_SecondEdge = 1 }
  DSPI clock phase configuration for a given CTAR.
- enum dspi_shift_direction_t {
  kDspiMsbFirst = 0,
  kDspiLsbFirst = 1 }
  DSPI data shifter direction options for a given CTAR.

- enum dspi_ctar_selection_t {
  kDspiCtar0 = 0,
  kDspiCtar1 = 1 }
  DSPI Clock and Transfer Attributes Register (CTAR) selection.
- enum dspi_pcs_polarity_config_t {
  kDspiPcs_ActiveHigh = 0,
  kDspiPcs_ActiveLow = 1 }
  DSPI Peripheral Chip Select (PCS) Polarity configuration.
- enum dspi_which_pcs_config_t {
  kDspiPcs0 = 1 << 0,
  kDspiPcs1 = 1 << 1,
  kDspiPcs2 = 1 << 2,
  kDspiPcs3 = 1 << 3,
  kDspiPcs4 = 1 << 4,
  kDspiPcs5 = 1 << 5 }
  DSPI Peripheral Chip Select (PCS) configuration (which PCS to configure)
- enum dspi_master_sample_point_t {
  kDspiSckToSin_0Clock = 0,
  kDspiSckToSin_1Clock = 1,
  kDspiSckToSin_2Clock = 2 }
  DSPI Sample Point: Controls when the DSPI master samples SIN in Modified Transfer Format.
- enum dspi_fifo_t {
  kDspiTxFifo = 0,
  kDspiRxFifo = 1 }
  DSPI FIFO selects.
- enum dspi_status_and_interrupt_request_t {
  kDspiTxComplete = BP_SPI_RSER_TCF_RE,
  kDspiTxAndRxStatus = BP_SPI_SR_TXRXS,
  kDspiEndOfQueue = BP_SPI_RSER_EOQF_RE,
  kDspiTxFifoUnderflow = BP_SPI_RSER_TFUF_RE,
  kDspiTxFifoFillRequest = BP_SPI_RSER_TFFF_RE,
  kDspiRxFifoOverflow = BP_SPI_RSER_RFOF_RE,
  kDspiRxFifoDrainRequest = BP_SPI_RSER_RFDF_RE }
  DSPI status flags and interrupt request enable.
- enum dspi_fifo_counter_pointer_t {
  kDspiRxFifoPointer = BP_SPI_SR_POPNXTPTR,
  kDspiRxFifoCounter = BP_SPI_SR_RXCTR,
  kDspiTxFifoPointer = BP_SPI_SR_TXNXTPTR,
  kDspiTxFifoCounter = BP_SPI_SR_TXCTR }
  DSPI FIFO counter or pointer defines based on bit positions.

Variables

- uint32_t dspi_data_format_config_t::bitsPerFrame
  Bits per frame, minimum 4, maximum 16 (master), 32 (slave)
- dspi_clock_polarity_t dspi_data_format_config_t::clkPolarity

Active high or low clock polarity.
- dspi_clock_phase_t dspi_data_format_config_t::clkPhase
  Clock phase setting to change and capture data.
- dspi_shift_direction_t dspi_data_format_config_t::direction
  MSB or LSB data shift direction This setting relevant only in master mode and can be ignored in slave mode.
- bool dspi_master_config_t::isEnabled
  Set to true to enable the DSPI peripheral.
- dspi_ctar_selection_t dspi_master_config_t::whichCtar
  Desired Clock and Transfer Attributes Register (CTAR)
- uint32_t dspi_master_config_t::bitsPerSec
  Baud rate in bits per second.
- uint32_t dspi_master_config_t::sourceClockInHz
  Module source clock.
- dspi_data_format_config_t dspi_master_config_t::dataConfig
  Data format configuration structure.
- bool dspi_master_config_t::isSckContinuous
  Disable(0) or Enable(1) continuous SCK operation.
- dspi_which_pcs_config_t dspi_master_config_t::whichPcs
  Desired Peripheral Chip Select (PCS)
- dspi_pcs_polarity_config_t dspi_master_config_t::pcsPolarity
  Peripheral Chip Select (PCS) polarity setting.
- dspi_master_sample_point_t dspi_master_config_t::masterInSample
  Master data-in (SIN) sample point setting.
- bool dspi_master_config_t::isModifiedTimingFormatEnabled
  Disable(0) or Enable(1) modified timing format.
- bool dspi_master_config_t::isTxFifoDisabled
  Disable(1) or Enable(0) Tx FIFO.
- bool dspi_master_config_t::isRxFifoDisabled
  Disable(1) or Enable(0) Rx FIFO.
- bool dspi_slave_config_t::isEnabled
  Set to true to enable the DSPI peripheral.
- dspi_data_format_config_t dspi_slave_config_t::dataConfig
  Data format configuration structure.
- bool dspi_slave_config_t::isTxFifoDisabled
  Disable(1) or Enable(0) Tx FIFO.
- bool dspi_slave_config_t::isRxFifoDisabled
  Disable(1) or Enable(0) Rx FIFO.
- bool dspi_baud_rate_divisors_t::doubleBaudRate
  Double Baud rate parameter setting.
- uint32_t dspi_baud_rate_divisors_t::prescaleDivisor
  Baud Rate Pre-scalar parameter setting.
- uint32_t dspi_baud_rate_divisors_t::baudRateDivisor
  Baud Rate scaler parameter setting.
- uint32_t dspi_delay_settings_config_t::pcsToSckPre
  PCS to SCK delay pre-scalar (PCSSCK)
- uint32_t dspi_delay_settings_config_t::pcsToSck
  PCS to SCK Delay scalar (CSSCK)
- uint32_t dspi_delay_settings_config_t::afterSckPre
  After SCK delay pre-scalar (PASC)
- uint32_t dspi_delay_settings_config_t::afterSck

After SCK delay scalar (ASC)
- uint32_t dspi_delay_settings_config_t::afterTransferPre
  Delay after transfer pre-scalar (PDT)
- uint32_t dspi_delay_settings_config_t::afterTransfer
  Delay after transfer scalar (DT)
- bool dspi_command_config_t::isChipSelectContinuous
  Option to enable the continuous assertion of chip select between transfers.
- dspi_ctar_selection_t dspi_command_config_t::whichCtar
  The desired Clock and Transfer Attributes Register (CTAR) to use for CTAS.
- dspi_which_pcs_config_t dspi_command_config_t::whichPcs
  The desired PCS signal to use for the data transfer.
- bool dspi_command_config_t::isEndOfQueue
  Signals that the current transfer is the last in the queue.
- bool dspi_command_config_t::clearTransferCount
  Clears SPI_TCNT field; cleared before transmission starts.

Configuration

- dspi_status_t dspi_hal_master_init (uint32_t instance, const dspi_master_config_t *config, uint32_t *calculatedBaudRate)
  Configure the DSPI peripheral in master mode.
- dspi_status_t dspi_hal_slave_init (uint32_t instance, const dspi_slave_config_t *config)
  Configures the DSPI peripheral in slave mode.
- void dspi_hal_reset (uint32_t instance)
  Restores the DSPI to reset the configuration.
- static void dspi_hal_enable (uint32_t instance)
  Enable the DSPI peripheral, set MCR MDIS to 0.
- static void dspi_hal_disable (uint32_t instance)
  Disables the DSPI peripheral, sets MCR MDIS to 1.
- uint32_t dspi_hal_set_baud (uint32_t instance, dspi_ctar_selection_t whichCtar, uint32_t bitsPerSec, uint32_t sourceClockInHz)
  Sets the DSPI baud rate in bits per second.
- void dspi_hal_set_baud_divisors (uint32_t instance, dspi_ctar_selection_t whichCtar, const dspi_baud_rate_divisors_t *divisors)
  Configures the baud rate divisors manually.
- static void dspi_hal_set_master_slave (uint32_t instance, dspi_master_slave_mode_t mode)
  Configures the DSPI for master or slave.
- static void dspi_hal_configure_continuous_sck (uint32_t instance, bool enable)
  Configures the DSPI for the continuous SCK operation.
- static void dspi_hal_configure_modified_timing_format (uint32_t instance, bool enable)
  Configures the DSPI to enable modified timing format.
- static void dspi_hal_configure_pcs_strobe (uint32_t instance, bool enable)
  Configures the DSPI peripheral chip select strobe enable.
- static void dspi_hal_configure_rx_fifo_overwrite (uint32_t instance, bool enable)
  Configures the DSPI received FIFO overflow overwrite enable.
- void dspi_hal_configure_pcs_polarity (uint32_t instance, dspi_which_pcs_config_t pcs, dspi_pcs_polarity_config_t activeLowOrHigh)
  Configures the DSPI peripheral chip select polarity.
- void dspi_hal_configure_fifos (uint32_t instance, bool disableTxFifo, bool disableRxFifo)

Configures the DSPI FIFOs.

- void dspi_hal_flush_fifos (uint32_t instance, bool enableFlushTxFifo, bool enableFlushRxFifo)
  Flushes the DSPI FIFOs.
- static void dspi_hal_set_datain_samplepoint (uint32_t instance, dspi_master_sample_point_t samplePnt)
  Configures when the DSPI master samples SIN in the Modified Transfer Format.
- static void dspi_hal_start_transfer (uint32_t instance)
  Starts the DSPI transfers, clears HALT bit in MCR.
- static void dspi_hal_stop_transfer (uint32_t instance)
  Stops (halts) DSPI transfers, sets HALT bit in MCR.
- dspi_status_t dspi_hal_configure_data_format (uint32_t instance, dspi_ctar_selection_t whichCtar, const dspi_data_format_config_t *config)
  Configures the data format for a particular CTAR.
- void dspi_hal_configure_delays (uint32_t instance, dspi_ctar_selection_t whichCtar, const dspi_delay_settings_config_t *config)
  Configures the delays for a particular CTAR, master mode only.

DMA

- void dspi_hal_configure_dma (uint32_t instance, bool enableTransmit, bool enableReceive)
  Configures transmit and receive DMA requests.

Low power

- static void dspi_hal_configure_doze_mode (uint32_t instance, bool enable)
  Configures the DSPI operation during doze mode.

Interrupts

- void dspi_hal_configure_interrupt (uint32_t instance, dspi_status_and_interrupt_request_t interruptSrc, bool enable)
  Configures the DSPI interrupts.
- static bool dspi_hal_get_interrupt_config (uint32_t instance, dspi_status_and_interrupt_request_t interruptSrc)
  Gets the DSPI interrupt configuration, returns if interrupt request is enabled or disabled.

Status

- static bool dspi_hal_get_status_flag (uint32_t instance, dspi_status_and_interrupt_request_t statusFlag)
  Gets the DSPI status flag state.
- static void dspi_hal_clear_status_flag (uint32_t instance, dspi_status_and_interrupt_request_t statusFlag)
  Clears the DSPI status flag.

- static uint32_t dspi_hal_get_fifo_counter_or_pointer (uint32_t instance, dspi_fifo_counter_pointer_t desiredParameter)
  Gets the DSPI FIFO counter or pointer.

Data transfer

- static uint32_t dspi_hal_read_data (uint32_t instance)
  Reads data from the data buffer.
- static void dspi_hal_write_data_slave_mode (uint32_t instance, uint32_t data)
  Writes data into the data buffer, slave mode.

6.1.1 Data Structure Documentation

6.1.1.1 struct dspi_data_format_config_t

This structure contains the data format settings. These settings apply to a specific CTARn register, which the user must provide in this structure.

Data Fields

- uint32_t bitsPerFrame
  Bits per frame, minimum 4, maximum 16 (master), 32 (slave)
- dspi_clock_polarity_t clkPolarity
  Active high or low clock polarity.
- dspi_clock_phase_t clkPhase
  Clock phase setting to change and capture data.
- dspi_shift_direction_t direction
  MSB or LSB data shift direction This setting relevant only in master mode and can be ignored in slave mode.

6.1.1.2 struct dspi_master_config_t

Use an instance of this structure with the dspi_hal_master_init() to configure the most common settings of the DSPI peripheral in master mode with a single function call.

The `bitsPerSec` member is handled in a special way. If this value is set to 0, then the baud is not set by the dspi_hal_master_init(), and must be set with a separate call to either the dspi_hal_set_baud() or the dspi_hal_set_baud_divisors(). This can be useful when you know the divisors in advance and don't want to spend the time to compute them for the provided rate in bits/sec.

This structure also contains another structure template as a member: `dspi_data_format_config_t dataConfig`. An example usage for this is assuming declaration `dspi_master_config_t dspiConfig`:

```
dspiConfig.dataConfig.bitsPerFrame = 16;
```

```
dspiConfig.dataConfig.clkPolarity = kDspiClockPolarity_ActiveHigh;
dspiConfig.dataConfig.clkPhase = kDspiClockPhase_FirstEdge;
dspiConfig.dataConfig.direction = kDspiMsbFirst;
```

Data Fields

- bool isEnabled
  Set to true to enable the DSPI peripheral.
- dspi_ctar_selection_t whichCtar
  Desired Clock and Transfer Attributes Register (CTAR)
- uint32_t bitsPerSec
  Baud rate in bits per second.
- uint32_t sourceClockInHz
  Module source clock.
- dspi_data_format_config_t dataConfig
  Data format configuration structure.
- bool isSckContinuous
  Disable(0) or Enable(1) continuous SCK operation.
- dspi_which_pcs_config_t whichPcs
  Desired Peripheral Chip Select (PCS)
- dspi_pcs_polarity_config_t pcsPolarity
  Peripheral Chip Select (PCS) polarity setting.
- dspi_master_sample_point_t masterInSample
  Master data-in (SIN) sample point setting.
- bool isModifiedTimingFormatEnabled
  Disable(0) or Enable(1) modified timing format.
- bool isTxFifoDisabled
  Disable(1) or Enable(0) Tx FIFO.
- bool isRxFifoDisabled
  Disable(1) or Enable(0) Rx FIFO.

6.1.1.3 struct dspi_slave_config_t

Use an instance of this structure with the dspi_hal_slave_init() to configure the most common settings of the DSPI peripheral in slave mode with a single function call.

Data Fields

- bool isEnabled
  Set to true to enable the DSPI peripheral.
- dspi_data_format_config_t dataConfig
  Data format configuration structure.
- bool isTxFifoDisabled
  Disable(1) or Enable(0) Tx FIFO.
- bool isRxFifoDisabled
  Disable(1) or Enable(0) Rx FIFO.

6.1.1.4 struct dspi_baud_rate_divisors_t

Note: These settings are relevant only in master mode. This structure contains the baud rate divisor settings, which provides the user with the option to explicitly set these baud rate divisors. In addition, the user must also set the CTARn register with the divisor settings.

Data Fields

- bool doubleBaudRate
  Double Baud rate parameter setting.
- uint32_t prescaleDivisor
  Baud Rate Pre-scalar parameter setting.
- uint32_t baudRateDivisor
  Baud Rate scaler parameter setting.

6.1.1.5 struct dspi_delay_settings_config_t

Note: These settings are relevant only in master mode. This structure contains the various delay settings. These settings apply to a specific CTARn register, which the user must provide in this structure.

Data Fields

- uint32_t pcsToSckPre
  PCS to SCK delay pre-scalar (PCSSCK)
- uint32_t pcsToSck
  PCS to SCK Delay scalar (CSSCK)
- uint32_t afterSckPre
  After SCK delay pre-scalar (PASC)
- uint32_t afterSck
  After SCK delay scalar (ASC)
- uint32_t afterTransferPre
  Delay after transfer pre-scalar (PDT)
- uint32_t afterTransfer
  Delay after transfer scalar (DT)

6.1.1.6 struct dspi_command_config_t

Note: This structure is used with the PUSHR register, which provides the means to write to the Tx FIFO. Data written to this register is transferred to the Tx FIFO. Eight or sixteen-bit write accesses to the PUSHR transfer all 32 register bits to the Tx FIFO. The register structure is different in master and slave modes. In master mode, the register provides 16-bit command and 16-bit data to the Tx FIFO. In slave mode all 32 register bits can be used as data, supporting up to 32-bit SPI frame operation.

Data Fields

- bool isChipSelectContinuous
  Option to enable the continuous assertion of chip select between transfers.
- dspi_ctar_selection_t whichCtar
  The desired Clock and Transfer Attributes Register (CTAR) to use for CTAS.
- dspi_which_pcs_config_t whichPcs
  The desired PCS signal to use for the data transfer.
- bool isEndOfQueue
  Signals that the current transfer is the last in the queue.
- bool clearTransferCount
  Clears SPI_TCNT field; cleared before transmission starts.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum dspi_status_t

Enumerator

kStatus_DSPI_SlaveTxUnderrun DSPI Slave Tx Under run error.
kStatus_DSPI_SlaveRxOverrun DSPI Slave Rx Overrun error.
kStatus_DSPI_Timeout DSPI transfer timed out.
kStatus_DSPI_Busy DSPI instance is already busy performing a transfer.
kStatus_DSPI_NoTransferInProgress Attempt to abort a transfer when no transfer was in progress.

kStatus_DSPI_InvalidBitCount bits-per-frame value not valid
kStatus_DSPI_InvalidInstanceNumber DSPI instance number does not match current count.

6.1.2.2 enum dspi_master_slave_mode_t

Enumerator

kDspiMaster DSPI peripheral operates in master mode.
kDspiSlave DSPI peripheral operates in slave mode.

6.1.2.3 enum dspi_clock_polarity_t

Enumerator

kDspiClockPolarity_ActiveHigh Active-high DSPI clock (idles low)
kDspiClockPolarity_ActiveLow Active-low DSPI clock (idles high)

6.1.2.4 enum dspi_clock_phase_t

Enumerator

kDspiClockPhase_FirstEdge Data is captured on the leading edge of the SCK and changed on the following edge.

kDspiClockPhase_SecondEdge Data is changed on the leading edge of the SCK and captured on the following edge.

6.1.2.5 enum dspi_shift_direction_t

Enumerator

kDspiMsbFirst Data transfers start with most significant bit.

kDspiLsbFirst Data transfers start with least significant bit.

6.1.2.6 enum dspi_ctar_selection_t

Enumerator

kDspiCtar0 CTAR0 selection option for master or slave mode.

kDspiCtar1 CTAR1 selection option for master mode only.

6.1.2.7 enum dspi_pcs_polarity_config_t

Enumerator

kDspiPcs_ActiveHigh PCS Active High (idles low)

kDspiPcs_ActiveLow PCS Active Low (idles high)

6.1.2.8 enum dspi_which_pcs_config_t

Enumerator

kDspiPcs0 PCS[0].

kDspiPcs1 PCS[1].

kDspiPcs2 PCS[2].

kDspiPcs3 PCS[3].

kDspiPcs4 PCS[4].

kDspiPcs5 PCS[5].

6.1.2.9 enum dspi_master_sample_point_t

This field is valid only when CPHA bit in CTAR register is 0.

Enumerator

kDspiSckToSin_0Clock 0 system clocks between SCK edge and SIN sample
kDspiSckToSin_1Clock 1 system clock between SCK edge and SIN sample
kDspiSckToSin_2Clock 2 system clocks between SCK edge and SIN sample

6.1.2.10 enum dspi_fifo_t

Enumerator

kDspiTxFifo DSPI Tx FIFO.
kDspiRxFifo DSPI Rx FIFO.

6.1.2.11 enum dspi_status_and_interrupt_request_t

Enumerator

kDspiTxComplete TCF status/interrupt enable.
kDspiTxAndRxStatus TXRXS status only, no interrupt.
kDspiEndOfQueue EOQF status/interrupt enable.
kDspiTxFifoUnderflow TFUF status/interrupt enable.
kDspiTxFifoFillRequest TFFF status/interrupt enable.
kDspiRxFifoOverflow RFOF status/interrupt enable.
kDspiRxFifoDrainRequest RFDF status/interrupt enable.

6.1.2.12 enum dspi_fifo_counter_pointer_t

Enumerator

kDspiRxFifoPointer Rx FIFO pointer.
kDspiRxFifoCounter Rx FIFO counter.
kDspiTxFifoPointer Tx FIFO pointer.
kDspiTxFifoCounter Tx FIFO counter.

6.1.3 Function Documentation

6.1.3.1 dspi_status_t dspi_hal_master_init (uint32_t *instance,* const dspi_master_config_t * *config,* uint32_t * *calculatedBaudRate*)

This function initializes the module to the user defined settings and default settings in master mode. This is an example demonstrating how to define the dspi_master_config_t structure and call the dspi_hal_master-_init function:

```
dspi_master_config_t dspiConfig;
dspiConfig.isEnabled = false;
dspiConfig.whichCtar = kDspiCtar0;
dspiConfig.bitsPerSec = 0;
dspiConfig.sourceClockInHz = dspiSourceClock;
dspiConfig.isSckContinuous = false;
dspiConfig.whichPcs = kDspiPcs0;
dspiConfig.pcsPolarity = kDspiPcs_ActiveLow;
dspiConfig.masterInSample = kDspiSckToSin_0Clock;
dspiConfig.isModifiedTimingFormatEnabled = false;
dspiConfig.isTxFifoDisabled = false;
dspiConfig.isRxFifoDisabled = false;
dspiConfig.dataConfig.bitsPerFrame = 16;
dspiConfig.dataConfig.clkPolarity =
      kDspiClockPolarity_ActiveHigh;
dspiConfig.dataConfig.clkPhase = kDspiClockPhase_FirstEdge;
dspiConfig.dataConfig.direction = kDspiMsbFirst;
dspi_hal_master_init(instance, &dspiConfig, calculatedBaudRate);
```

Parameters

instance	Module instance number
config	Pointer to the master mode configuration data structure
calculated-BaudRate	The calculated baud rate passed back to the user for them to determine if the calculated baud rate is close enough to meet their needs.

Returns

An error code or kStatus_DSPI_Success.

6.1.3.2 dspi_status_t dspi_hal_slave_init (uint32_t *instance,* const dspi_slave_config_t * *config*)

This function initializes the DSPI module for slave mode. This is an example demonstrating how to define the dspi_slave_config_t structure and call the dspi_hal_slave_init function:

```
dspi_slave_config_t dspiConfig;
dspiConfig.isEnabled = false;
dspiConfig.isTxFifoDisabled = false;
dspiConfig.isRxFifoDisabled = false;
dspiConfig.dataConfig.bitsPerFrame = 16;
```

```
dspiConfig.dataConfig.clkPolarity =
      kDspiClockPolarity_ActiveHigh;
dspiConfig.dataConfig.clkPhase = kDspiClockPhase_FirstEdge;
dspi_hal_slave_init(instance, &dspiConfig);
```

Parameters

instance	Module instance number
config	Pointer to the slave mode configuration data structure

Returns

An error code or kStatus_DSPI_Success.

6.1.3.3 void dspi_hal_reset (uint32_t *instance*)

This function basically resets all of the DSPI registers to their default setting including disabling the module.

Parameters

instance	Module instance number

6.1.3.4 static void dspi_hal_enable (uint32_t *instance*) [inline], [static]

Parameters

instance	Module instance number

6.1.3.5 static void dspi_hal_disable (uint32_t *instance*) [inline], [static]

Parameters

instance	Module instance number

6.1.3.6 uint32_t dspi_hal_set_baud (uint32_t *instance*, dspi_ctar_selection_t *whichCtar*, uint32_t *bitsPerSec*, uint32_t *sourceClockInHz*)

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate, and returns the calculated baud rate in bits-per-second. It requires that the caller also provide the frequency of the module source clock (in Hertz).

Parameters

instance	Module instance number
whichCtar	The desired Clock and Transfer Attributes Register (CTAR) of the type dspi_ctar_selection_t
bitsPerSec	The desired baud rate in bits per second
sourceClockInHz	Module source input clock in Hertz

Returns

The actual calculated baud rate

6.1.3.7 void dspi_hal_set_baud_divisors (uint32_t *instance,* dspi_ctar_selection_t *whichCtar,* const dspi_baud_rate_divisors_t * *divisors*)

This function allows the caller to manually set the baud rate divisors in the event that these dividers are known and the caller does not wish to call the dspi_hal_set_baud function.

Parameters

instance	Module instance number
whichCtar	The desired Clock and Transfer Attributes Register (CTAR) of type dspi_ctar_selection_t
divisors	Pointer to a structure containing the user defined baud rate divisor settings

6.1.3.8 static void dspi_hal_set_master_slave (uint32_t *instance,* dspi_master_slave_mode_t *mode*) [inline], [static]

Parameters

instance	Module instance number
mode	Mode setting (master or slave) of type dspi_master_slave_mode_t

6.1.3.9 static void dspi_hal_configure_continuous_sck (uint32_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	Module instance number
enable	Enables (true) or disables(false) continuous SCK operation.

6.1.3.10 static void dspi_hal_configure_modified_timing_format (uint32_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	Module instance number
enable	Enables (true) or disables(false) modified timing format.

6.1.3.11 static void dspi_hal_configure_pcs_strobe (uint32_t *instance,* bool *enable*) [inline], [static]

Configures the PCS[5] to be the active-low PCS Strobe output.

PCS[5] is a special case that can be configured as an active low PCS strobe or as a Peripheral Chip Select in master mode. When configured as a strobe, it provides a signal to an external demultiplexer to decode PCS[0] to PCS[4] signals into as many as 128 glitch-free PCS signals.

Parameters

instance	Module instance number
enable	Enable (true) PCS[5] to operate as the peripheral chip select (PCS) strobe If disable (false), PCS[5] operates as a peripheral chip select

6.1.3.12 static void dspi_hal_configure_rx_fifo_overwrite (uint32_t *instance,* bool *enable*) [inline], [static]

When enabled, this function allows incoming receive data to overwrite the existing data in the receive shift register when the Rx FIFO is full. Otherwise when disabled, the incoming data is ignored when the RX FIFO is full.

Parameters

instance	Module instance number.
enable	If enabled (true), allows incoming data to overwrite Rx FIFO contents when full, else incoming data is ignored.

6.1.3.13 void dspi_hal_configure_pcs_polarity (uint32_t *instance,* dspi_which_pcs_config_t *pcs,* dspi_pcs_polarity_config_t *activeLowOrHigh*)

This function takes in the desired peripheral chip select (PCS) and it's corresponding desired polarity and configures the PCS signal to operate with the desired characteristic.

Parameters

instance	Module instance number
pcs	The particular peripheral chip select (parameter value is of type dspi_which_pcs_config_t) for which we wish to apply the active high or active low characteristic.
activeLowOrHigh	The setting for either "active high, inactive low (0)" or "active low, inactive high(1)" of type dspi_pcs_polarity_config_t.

6.1.3.14 void dspi_hal_configure_fifos (uint32_t *instance,* bool *disableTxFifo,* bool *disableRxFifo*)

This function allows the caller to disable/enable the Tx and Rx FIFOs (independently). Note that to disable, the caller must pass in a logic 1 (true) for the particular FIFO configuration. To enable, the caller must pass in a logic 0 (false). For example, to enable both the Tx and Rx FIFOs, the caller makes this function call (where instance is the desired module instance number):

```
dspi_hal_configure_fifos(instance, false, false);
```

Parameters

instance	Module instance number
disableTxFifo	Disables (false) the TX FIFO, else enables (true) the TX FIFO
disableRxFifo	Disables (false) the RX FIFO, else enables (true) the RX FIFO

6.1.3.15 void dspi_hal_flush_fifos (uint32_t *instance,* bool *enableFlushTxFifo,* bool *enableFlushRxFifo*)

Parameters

instance	Module instance number
enableFlushTx-Fifo	Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO
enableFlush-RxFifo	Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO

6.1.3.16 static void dspi_hal_set_datain_samplepoint (uint32_t *instance,* dspi_master_sample_point_t *samplePnt*) [inline], [static]

This function controls when the DSPI master samples SIN (data in) in the Modified Transfer Format. Note that this is valid only when the CPHA bit in the CTAR register is 0.

Parameters

instance	Module instance number
samplePnt	selects when the data in (SIN) is sampled, of type dspi_master_sample_point_t. This value selects either 0, 1, or 2 system clocks between the SCK edge and the SIN (data in) sample.

6.1.3.17 static void dspi_hal_start_transfer (uint32_t *instance*) [inline], [static]

This function call called whenever the module is ready to begin data transfers in either master or slave mode.

Parameters

instance	Module instance number

6.1.3.18 static void dspi_hal_stop_transfer (uint32_t *instance*) [inline], [static]

This function call stops data transfers in either master or slave mode.

Parameters

instance	Module instance number

6.1.3.19 dspi_status_t dspi_hal_configure_data_format (uint32_t *instance,* dspi_ctar_selection_t *whichCtar,* const dspi_data_format_config_t * *config*)

This function configures the bits-per-frame, polarity, phase, and shift direction for a particular CTAR. An example use case is as follows:

```
dspi_data_format_config_t dataFormat;
dataFormat.bitsPerFrame = 16;
dataFormat.clkPolarity = kDspiClockPolarity_ActiveLow;
dataFormat.clkPhase = kDspiClockPhase_FirstEdge;
dataFormat.direction = kDspiMsbFirst;
dspi_hal_configure_data_format(instance,
      kDspiCtar0, &dataFormat);
```

Parameters

instance	Module instance number
whichCtar	The desired Clock and Transfer Attributes Register (CTAR) of type dspi_ctar_selection_t.
config	Pointer to a structure containing the user defined data format configuration settings.

Returns

An error code or kStatus_DSPI_Success

6.1.3.20 void dspi_hal_configure_delays (uint32_t *instance,* dspi_ctar_selection_t *whichCtar,* const dspi_delay_settings_config_t * *config*)

This function configures the PCS to SCK delay pre-scalar (PCSSCK), the PCS to SCK Delay scalar (CSSCK), the After SCK delay pre-scalar (PASC), the After SCK delay scalar (ASC), the Delay after transfer pre-scalar (PDT), and the Delay after transfer scalar (DT). The following is an example use case of this function:

```
   dspi_delay_settings_config_t delayConfig;
   delayConfig.pcsToSckPre = 0x3;
   delayConfig.pcsToSck = 0xF;
   delayConfig.afterSckPre = 0x2;
   delayConfig.afterSck = 0xA;
   delayConfig.afterTransferPre = 0x1;
   delayConfig.afterTransfer = 0x5;
   dspi_hal_configure_delays(instance, kDspiCtar0, &delayConfig);
*
```

Parameters

instance	Module instance number
whichCtar	The desired Clock and Transfer Attributes Register (CTAR) of type dspi_ctar_selection_t.
config	Pointer to a structure containing the user defined delay configuration settings.

6.1.3.21 void dspi_hal_configure_dma (uint32_t *instance,* bool *enableTransmit,* bool *enableReceive*)

This function configures the FIFOs to generate a DMA or an interrupt request. Note that the corresponding request enable must also be set. For the Transmit FIFO Fill, in order to generate a DMA request, the Transmit FIFO Fill Request Enable (TFFF_RE) must also be set. Similarly for the Receive FIFO Drain Request, to generate a DMA request, the Receive FIFO Drain Request Enable (RFDF_RE) must also be set. These requests can be configured with the function dspi_hal_configure_interrupt(). To enable DMA operation, first enable the desired request enable by using the dspi_hal_configure_interrupt() function and then use the dspi_hal_configure_dma() to configure the request and generate a DMA request.

Parameters

enableTransmit	Configures Tx FIFO fill request to generate a DMA or interrupt request
enableReceive	Configures Rx FIFO fill request to generate a DMA or interrupt request

6.1.3.22 static void dspi_hal_configure_doze_mode (uint32_t *instance,* bool *enable*) [inline], [static]

This function provides support for an externally controlled doze mode, power-saving, mechanism. When disabled, the doze mode has no effect on the DSPI, and when enabled, the Doze mode disables the DSPI.

Parameters

instance	Module instance number
enable	If disabled (false), the doze mode has no effect on the DSPI, if enabled (true), the doze mode disables the DSPI.

6.1.3.23 void dspi_hal_configure_interrupt (uint32_t *instance,* dspi_status_and_interrupt_request_t *interruptSrc,* bool *enable*)

This function configures the various interrupt sources of the DSPI. The parameters are instance, interrupt source, and enable/disable setting. The interrupt source is a typedef enum whose value is the bit position

of the interrupt source setting within the RSER register. In the DSPI, all interrupt configuration settings are in one register. The typedef enum equates each interrupt source to the bit position defined in the device header file. The function uses these bit positions in its algorithm to enable/disable the interrupt source, where interrupt source is the dspi_status_and_interrupt_request_t type.

```
   temp = (HW_SPI_RSER_RD(instance) & ~interruptSrc) | (enable << interruptSrc);
   HW_SPI_RSER_WR(instance, temp);

   dspi_hal_configure_interrupt(instance,
      kDspiTxComplete, true); <- example use-case
*
```

Parameters

instance	Module instance number
interruptSrc	The interrupt source, of type dspi_status_and_interrupt_request_t
enable	Enable (true) or disable (false) the interrupt source to generate requests

6.1.3.24 static bool dspi_hal_get_interrupt_config (uint32_t *instance,* dspi_status_and_interrupt_request_t *interruptSrc*) [inline], [static]

This function returns the requested interrupt source setting (enabled or disabled, of type bool). The parameters to pass in are instance and interrupt source. It utilizes the same enum definitions for the interrupt sources as described in the "interrupt configuration" function. The function uses these bit positions in its algorithm to obtain the desired interrupt source setting.

```
   return ((HW_SPI_RSER_RD(instance) & interruptSrc) >> interruptSrc);

  getInterruptSetting = dspi_hal_get_interrupt_config(instance,
      kDspiTxComplete);
*
```

Parameters

instance	Module instance number
interruptSrc	The interrupt source, of type dspi_status_and_interrupt_request_t

Returns

Configuration of interrupt request: enable (true) or disable (false).

6.1.3.25 static bool dspi_hal_get_status_flag (uint32_t *instance,* dspi_status_and_interrupt_request_t *statusFlag*) [inline], [static]

The status flag is defined in the same enum as the interrupt source enable because the bit position of the interrupt source and corresponding status flag are the same in the RSER and SR registers. The function

uses these bit positions in its algorithm to obtain the desired flag state, similar to the dspi_get_interrupt_-config function.

```
   return ((HW_SPI_SR_RD(instance) & statusFlag) >> statusFlag);

   getStatus = dspi_hal_get_status_flag(instance,
      kDspiTxComplete);
*
```

Parameters

instance	Module instance number
statusFlag	The status flag, of type dspi_status_and_interrupt_request_t

Returns

State of the status flag: asserted (true) or not-asserted (false)

6.1.3.26 static void dspi_hal_clear_status_flag (uint32_t *instance,* dspi_status_and_interrupt_request_t *statusFlag*) [inline], [static]

This function clears the desired status bit by using a write-1-to-clear. The user passes in the instance and the desired status bit to clear. The list of status bits is defined in the dspi_status_and_interrupt_request_t. The function uses these bit positions in its algorithm to clear the desired flag state. It uses this macro:

```
   HW_SPI_SR_WR(instance, statusFlag);

   dspi_hal_clear_status_flag(instance,
      kDspiTxComplete);
*
```

Parameters

instance	Module instance number
statusFlag	The status flag, of type dspi_status_and_interrupt_request_t

6.1.3.27 static uint32_t dspi_hal_get_fifo_counter_or_pointer (uint32_t *instance,* dspi_fifo_counter_pointer_t *desiredParameter*) [inline], [static]

This function returns the number of entries or the next pointer in the Tx or Rx FIFO. The parameters to pass in are the instance and either the Tx or Rx FIFO counter or a pointer. The latter is an enum type defined as the bitmask of those particular bit fields found in the device header file. For example:

```
   return ((HW_SPI_SR_RD(instance) >> desiredParamter) & 0xF);

   dspi_hal_get_fifo_counter_or_pointer(instance,
      kDspiRxFifoCounter);
*
```

Parameters

instance	Module instance number
desired-Parameter	Desired parameter to obtain, of type dspi_fifo_counter_pointer_t

6.1.3.28 static uint32_t dspi_hal_read_data (uint32_t *instance*) [inline], [static]

Parameters

instance	Module instance number

6.1.3.29 static void dspi_hal_write_data_slave_mode (uint32_t *instance,* uint32_t *data*) [inline], [static]

In slave mode, up to 32-bit words may be written.

Parameters

instance	Module instance number
data	The data to send

6.1.4 Variable Documentation

6.1.4.1 bool dspi_master_config_t::isEnabled

6.1.4.2 dspi_pcs_polarity_config_t dspi_master_config_t::pcsPolarity

6.1.4.3 dspi_master_sample_point_t dspi_master_config_t::masterInSample

6.1.4.4 bool dspi_master_config_t::isModifiedTimingFormatEnabled

6.1.4.5 bool dspi_slave_config_t::isEnabled

6.2 DSPI Master Driver

The chapter describes the programming interface of the DSPI master mode Peripheral driver.

Data Structures

- struct dspi_device_t
  Data structure containing information about a device on the SPI bus. More...
- struct dspi_master_state_t
  Runtime state structure for the DSPI master driver. More...
- struct dspi_master_user_config_t
  The user configuration structure for the DSPI master driver. More...

Init and shutdown

- dspi_status_t dspi_master_init (uint32_t instance, dspi_master_state_t *dspiState, const dspi_master_user_config_t *userConfig, uint32_t *calculatedBaudRate)
  Initialize a DSPI instance for master mode operation.
- void dspi_master_shutdown (dspi_master_state_t *dspiState)
  Shuts down a DSPI instance.
- void dspi_master_configure_modified_transfer_format (dspi_master_state_t *dspiState, bool enableOrDisable, dspi_master_sample_point_t samplePnt)
  Configures the DSPI modified transfer format in master mode.

Bus configuration

- dspi_status_t dspi_master_configure_bus (dspi_master_state_t *dspiState, const dspi_device_t *device, uint32_t *calculatedBaudRate)
  Configures the DSPI port physical parameters to access a device on the bus.

Blocking transfers

- dspi_status_t dspi_master_transfer (dspi_master_state_t *dspiState, const dspi_device_t *restrict device, const uint8_t *sendBuffer, uint8_t *receiveBuffer, size_t transferByteCount, uint32_t timeout)
  Performs a blocking SPI master mode transfer.

Non-blocking transfers

- dspi_status_t dspi_master_transfer_async (dspi_master_state_t *dspiState, const dspi_device_t *restrict device, const uint8_t *sendBuffer, uint8_t *receiveBuffer, size_t transferByteCount)
  Performs a non-blocking SPI master mode transfer.
- dspi_status_t dspi_master_get_transfer_status (dspi_master_state_t *dspiState, uint32_t *framesTransferred)

Returns whether the previous transfer is finished.

- dspi_status_t dspi_master_abort_transfer (dspi_master_state_t *dspiState)
  Terminates an asynchronous transfer early.

6.2.0.6 DSPI Master Driver

Overview

The DSPI master mode peripheral driver transfers data to and from external devices on the DSPI bus in master mode. It supports transferring buffers of data with a single function call.

Run-time state structs

The DSPI master driver uses a run-time state struct dspi_master_state_t to track of the ongoing data transfer. The struct holds data that the DSPI master peripheral driver uses to communicate between the transfer function and the interrupt handler. The interrupt handler also uses this information to keep track of its progress. The user is only responsible to pass the memory for this run-time state structure and the DSPI master driver will fill out the members.

User config structs

The DSPI master driver uses instances of the user configuration structure dspi_master_user_config_t for the DSPI master driver. For this reason, the user can configure the most common settings of the DSPI peripheral with a single function call.

Device structs

The DSPI master driver uses instances of the dspi_device_t structure to represent the SPI bus configuration required to communicate to an external device that is connected to the bus.

The device struct can be passed to the data transfer functions, and the bus will be reconfigured before the transfer is started. Alternatively, the user can configure the SPI bus for a device manually. For instance, if there is only one device connected to the bus, the user might configure it only once.

Initialization

To initialize the DSPI master driver, call the dspi_master_init() function and and pass the instance number of the DSPI peripheral, the user config of type dspi_master_user_config_t , a pointer to the variable for retrieving the calculated baud rate (this can be a NULL), and memory allocation for the run-time state structure used by the master driver to keep track of data transfers. For example, to use DSPI1, pass a value of 1 to the init function. This is an example for the other parameters.

Example code to initialize and configure the driver:

```
/* Set up and init the master  */
dspi_master_state_t dspiMasterState;
uint32_t calculatedBaudRate;

/* configure the members of the user config */
dspi_master_user_config_t userConfig;
userConfig.isChipSelectContinuous = false;
userConfig.isSckContinuous = false;
userConfig.pcsPolarity = kDspiPcs_ActiveLow;
userConfig.whichCtar = kDspiCtar0;
userConfig.whichPcs = kDspiPcs1;

/* init the DSPI module */
dspi_master_init(masterInstance, &dspiMasterState, &userConfig, &calculatedBaudRate);

// Define bus configuration.
dspi_device_t spiDevice;
spiDevice.dataBusConfig.bitsPerFrame = 16;
spiDevice.dataBusConfig.clkPhase = kDspiClockPhase_FirstEdge;
spiDevice.dataBusConfig.clkPolarity = kDspiClockPolarity_ActiveHigh
      ;
spiDevice.dataBusConfig.direction = kDspiMsbFirst;
spiDevice.bitsPerSec = 500000;

/* configure the SPI bus */
dspi_master_configure_bus(&dspiMasterState, &spiDevice, &calculatedBaudRate);
```

Transfers

The driver supports two different modes for transferring data: blocking and non-blocking (async). The dspi_master_transfer() is the blocking transfer function. The dspi_master_transfer_async() is a non-blocking function.

Example of a blocking transfer:

```
/* Function prototype */
status_t dspi_master_transfer(dspi_master_state_t * dspiState,
                              const dspi_device_t * restrict device,
                              const uint8_t * sendBuffer,
                              uint8_t * receiveBuffer,
                              uint32_t transferByteCount,
                              uint32_t timeout);

/* Example function call */
dspi_master_transfer(&dspiMasterState, NULL, s_dspiSourceBuffer[masterInstance],
      s_dspiSinkBuffer[masterInstance], 32, 1);
```

Example of a nonb-locking transfer:

```
/* Function prototype */
status_t dspi_master_transfer_async(
      dspi_master_state_t * dspiState,
                                    const dspi_device_t * restrict device,
                                    const uint8_t * sendBuffer,
                                    uint8_t * receiveBuffer,
                                    uint32_t transferByteCount);
/* Example function call */
```

```
dspi_master_transfer_async(&dspiMasterState, NULL, s_dspiSourceBuffer[
      masterInstance], s_dspiSinkBuffer[masterInstance], 32);

/* For non-blocking/a-sync transfers, need to check back to get transfer status, for example */
/* Where framesXfer returns the number of frames transferred */
dspi_master_get_transfer_status(&dspiMasterState, &framesXfer);
```

6.2.1 Data Structure Documentation

6.2.1.1 struct dspi_device_t

The user must fill out these members to set up the DSPI master to properly communicate with the SPI device.

Data Fields

- uint32_t bitsPerSec
  Baud rate in bits per second.

6.2.1.1.0.5 Field Documentation

6.2.1.1.0.5.1 uint32_t dspi_device_t::bitsPerSec

6.2.1.2 struct dspi_master_state_t

This struct holds data that is used by the DSPI master peripheral driver to communicate between the transfer function and the interrupt handler. The interrupt handler also uses this information to keep track of its progress. The user must pass the memory for this run-time state structure and the DSPI master driver will fill out the members.

Data Fields

- uint32_t instance
  DSPI module instance number.
- dspi_ctar_selection_t whichCtar
  Desired Clock and Transfer Attributes Register (CTAR)
- uint32_t bitsPerFrame
  Desired number of bits per frame.
- dspi_which_pcs_config_t whichPcs
  Desired Peripheral Chip Select (pcs)
- bool isChipSelectContinuous
  Option to enable the continuous assertion of chip select between transfers.
- uint32_t dspiSourceClock
  Module source clock.
- volatile bool isTransferInProgress
  True if there is an active transfer.
- bool isTransferAsync
  Whether the transfer is asynchronous (needed in IRQ).

- const uint8_t *restrict sendBuffer
  The buffer from which transmitted bytes are taken.
- uint8_t *restrict receiveBuffer
  The buffer into which received bytes are placed.
- volatile size_t remainingSendByteCount
  Number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount
  Number of bytes remaining to receive.
- sync_object_t irqSync
  Used to wait for ISR to complete its business.

6.2.1.2.0.6 Field Documentation

6.2.1.2.0.6.1 volatile bool dspi_master_state_t::isTransferInProgress

6.2.1.2.0.6.2 bool dspi_master_state_t::isTransferAsync

6.2.1.2.0.6.3 const uint8_t* restrict dspi_master_state_t::sendBuffer

6.2.1.2.0.6.4 uint8_t* restrict dspi_master_state_t::receiveBuffer

6.2.1.2.0.6.5 volatile size_t dspi_master_state_t::remainingSendByteCount

6.2.1.2.0.6.6 volatile size_t dspi_master_state_t::remainingReceiveByteCount

6.2.1.2.0.6.7 sync_object_t dspi_master_state_t::irqSync

6.2.1.3 struct dspi_master_user_config_t

Use an instance of this struct with the dspi_master_init(). This allows the user to configure the most common settings of the DSPI peripheral with a single function call.

Data Fields

- dspi_ctar_selection_t whichCtar
  Desired Clock and Transfer Attributes Register(CTAR)
- bool isSckContinuous
  Disable or Enable continuous SCK operation.
- bool isChipSelectContinuous
  Option to enable the continuous assertion of chip select between transfers.
- dspi_which_pcs_config_t whichPcs
  Desired Peripheral Chip Select (pcs)
- dspi_pcs_polarity_config_t pcsPolarity
  Peripheral Chip Select (pcs) polarity setting.

6.2.1.3.0.7 Field Documentation

6.2.1.3.0.7.1 dspi_pcs_polarity_config_t dspi_master_user_config_t::pcsPolarity

6.2.2 Function Documentation

6.2.2.1 dspi_status_t dspi_master_init (uint32_t *instance*, dspi_master_state_t * *dspiState*, const dspi_master_user_config_t * *userConfig*, uint32_t * *calculatedBaudRate*)

This function initializes the run-time state structure to track the ongoing transfers, ungates the clock to the DSPI module, resets the DSPI module, initializes the module to user defined settings and default settings, configures the IRQ state structure, enables the module-level interrupt to the core, and enables the DSPI module. The CTAR parameter is special in that it allows the user to have different SPI devices connected to the same DSPI module instance in addition to different peripheral chip selects. Each CTAR contains the bus attributes associated with that particular SPI device. For most use cases where only one SPI device is connected per DSPI module instance, use CTAR0. This is an example to set up the dspi_master_state_t and the dspi_master_user_config_t parameters and to call the dspi_master_init function by passing in these parameters:

```
dspi_master_state_t dspiMasterState; <- the user simply allocates memory for this struct
uint32_t calculatedBaudRate;
dspi_master_user_config_t userConfig; <- the user fills out members for this
      struct
userConfig.isChipSelectContinuous = false;
userConfig.isSckContinuous = false;
userConfig.pcsPolarity = kDspiPcs_ActiveLow;
userConfig.whichCtar = kDspiCtar0;
userConfig.whichPcs = kDspiPcs0;
dspi_master_init(masterInstance, &dspiMasterState, &userConfig, &calculatedBaudRate);
```

Parameters

instance	The instance number of the DSPI peripheral.
dspiState	The pointer to the DSPI master driver state structure. The user must pass the memory for this run-time state structure and the DSPI master driver will fill out the members. This run-time state structure keeps track of the transfer in progress.
userConfig	The dspi_master_user_config_t user configuration structure. The user must fill out the members of this structure and pass the pointer of this struct into the function.
calculated-BaudRate	The calculated baud rate passed back to the user to determine if the calculated baud rate is close enough to meet the needs. The value returned may be 0 if the user decides to set the baud rate to 0 in the user configuration struct. The user can later configure the baud rate via the dspi_master_configure_bus function, and hence may not need to configure the baud rate in the dspi_master_init function.

Returns

An error code or kStatus_DSPI_Success.

6.2.2.2 void dspi_master_shutdown (dspi_master_state_t * *dspiState*)

This function resets the DSPI peripheral, gates its clock, and disables the interrupt to the core.

Parameters

dspiState	The pointer to the DSPI master driver state structure.

6.2.2.3 void dspi_master_configure_modified_transfer_format (dspi_master_state_t * *dspiState,* bool *enableOrDisable,* dspi_master_sample_point_t *samplePnt*)

This function allows the user to enable or disable (default setting) the modified transfer format. The modified transfer format is supported to allow high-speed communication with peripherals that require longer setup times. The module can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The data-in sample point can also be configured in this function. Note that the data-in sample point setting is valid only when the CPHA bit in the CTAR is cleared (when the dspi_clock_phase_t is set to kDspiClockPhase_FirstEdge in the dspi_data_format_config_t).

Parameters

dspiState	The pointer to the DSPI master driver state structure.
enableOr-Disable	Enables (true) or disables(false) the modified transfer (timing) format.
samplePnt	Selects when the data in (SIN) is sampled, of type dspi_master_sample_point_t. This value selects either 0, 1, or 2 system clocks between the SCK edge and the SIN (data in) sample.

6.2.2.4 dspi_status_t dspi_master_configure_bus (dspi_master_state_t * *dspiState,* const dspi_device_t * *device,* uint32_t * *calculatedBaudRate*)

The term "device" is used to indicate the SPI device for which the DSPI master is communicating. The user has two options to configure the device parameters: either pass in the pointer to the device configuration structure to the desired transfer function (see dspi_master_transfer or dspi_master_transfer_async) or pass it in to the dspi_master_configure_bus function. The user can pass in a device structure to the transfer function which contains the parameters for the bus (the transfer function will then call this function). However, the user has the option to call this function directly especially to get the calculated baud rate, at which point they may pass in NULL for the device struct in the transfer function (assuming they have

called this configure bus function first). This is an example to set up the dspi_device_t structure to call the dspi_master_configure_bus function by passing in these parameters:

```
dspi_device_t spiDevice;
spiDevice.dataBusConfig.bitsPerFrame = 16;
spiDevice.dataBusConfig.clkPhase = kDspiClockPhase_FirstEdge;
spiDevice.dataBusConfig.clkPolarity = kDspiClockPolarity_ActiveHigh
      ;
spiDevice.dataBusConfig.direction = kDspiMsbFirst;
spiDevice.bitsPerSec = 50000;
dspi_master_configure_bus(&dspiMasterState, &spiDevice, &calculatedBaudRate);
```

Parameters

dspiState	The pointer to the DSPI master driver state structure.
device	Pointer to the device information struct. This struct contains the settings for the SPI bus configuration. The device parameters are the desired baud rate (in bits-per-sec), and the data format field which consists of bits-per-frame, clock polarity and phase, and data shift direction.
calculated-BaudRate	The calculated baud rate passed back to the user to determine if the calculated baud rate is close enough to meet the needs. The baud rate never exceeds the desired baud rate.

Returns

An error code or kStatus_DSPI_Success.

6.2.2.5 dspi_status_t dspi_master_transfer (dspi_master_state_t * *dspiState*, const dspi_device_t *restrict *device*, const uint8_t * *sendBuffer*, uint8_t * *receiveBuffer*, size_t *transferByteCount*, uint32_t *timeout*)

This function simultaneously sends and receives data on the SPI bus, as SPI is naturally a full-duplex bus. The function does not return until the transfer is complete.

Parameters

dspiState	The pointer to the DSPI master driver state structure.
device	Pointer to the device information struct. This struct contains the settings for the SPI bus configuration in this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified. The device can be configured separately by calling the dspi_master_configure_bus function.

sendBuffer	The pointer to the data buffer of the data to send. You may pass NULL for this parameter and bytes with a value of 0 (zero) will be sent.
receiveBuffer	Pointer to the buffer where the received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.
transferByte-Count	The number of bytes to send and receive.
timeout	A timeout for the transfer in microseconds. If the transfer takes longer than this amount of time, the transfer is aborted and a kStatus_SPI_Timeout error returned.

Return values

kStatus_DSPI_Success	The transfer was successful.
kStatus_DSPI_Busy	Cannot perform another transfer because a transfer is already in progress.
kStatus_DSPI_Timeout	The transfer timed out and was aborted.

6.2.2.6 dspi_status_t dspi_master_transfer_async (dspi_master_state_t * *dspiState*, const dspi_device_t *restrict *device*, const uint8_t * *sendBuffer*, uint8_t * *receiveBuffer*, size_t *transferByteCount*)

This function returns immediately. The user must check back to check whether the transfer is complete (using the dspi_master_get_transfer_status function). This function simultaneously sends and receives data on the SPI bus, as SPI is naturally a full-duplex bus.

Parameters

dspiState	The pointer to the DSPI master driver state structure.
device	Pointer to the device information struct. This struct contains the settings for the SPI bus configuration in this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified. The device can be configured separately by calling the dspi_master_configure_bus function.
sendBuffer	The pointer to the data buffer of the data to send. You may pass NULL for this parameter, in which case bytes with a value of 0 (zero) are sent.
receiveBuffer	Pointer to the buffer where the received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.

transferByte-Count	The number of bytes to send and receive.

Return values

kStatus_DSPI_Success	The transfer was successful.
kStatus_DSPI_Busy	Cannot perform another transfer because a transfer is already in progress.

6.2.2.7 dspi_status_t dspi_master_get_transfer_status (dspi_master_state_t * *dspiState*, uint32_t * *framesTransferred*)

When performing an a-sync transfer, the user can call this function to ascertain the state of the current transfer: in progress (or busy) or complete (success). In addition, if the transfer is still in progress, the user can get the number of words that have been transferred up to now.

Parameters

dspiState	The pointer to the DSPI master driver state structure
frames-Transferred	Pointer to value that is filled in with the number of frames that have been sent in the active transfer. A frame is defined as the number of bits per frame.

Return values

kStatus_DSPI_Success	The transfer has completed successfully.
kStatus_DSPI_Busy	The transfer is still in progress. *wordsTransferred* will be filled with the number of words that have been transferred so far.

6.2.2.8 dspi_status_t dspi_master_abort_transfer (dspi_master_state_t * *dspiState*)

During an a-sync transfer, the user has the option to terminate the transfer early if the transfer is still in progress.

Parameters

dspiState	The pointer to the DSPI master driver state structure.

Return values

kStatus_DSPI_Success	The transfer was successful.
kStatus_DSPI_No-TransferInProgress	No transfer is currently in progress.

6.3 DSPI Slave Driver

The chapter describes the programming interface of the DSPI slave mode Peripheral driver.

Data Structures

- struct dspi_slave_callbacks_t
  The set of callbacks for the DSPI slave mode. More...
- struct dspi_slave_state_t
  Runtime state of the DSPI slave driver. More...
- struct dspi_slave_user_config_t
  User configuration structure and callback functions for the DSPI slave driver. More...

Init and shutdown

- dspi_status_t dspi_slave_init (uint32_t instance, dspi_slave_state_t ∗dspiState, const dspi_slave_-user_config_t ∗slaveConfig)
  Initializes a DSPI instance for a slave mode operation.
- void dspi_slave_shutdown (dspi_slave_state_t ∗dspiState)
  Shut down a DSPI instance.

6.3.0.9 DSPI Slave Driver

Overview

The DSPI slave peripheral driver supports using the SPI peripheral in slave mode.

Data transfer is performed entirely through callback functions.

Runtime state of the DSPI slave driver

This struct holds data that the DSPI slave peripheral driver uses to communicate between the transfer function and the interrupt handler. The user needs to pass the memory for this structure and the driver will fill out the members.

Callbacks

To use this driver, first define application callbacks. These are the callback functions that are set in the dspi_slave_callbacks_t struct.

The three callbacks are:

- Data source
- Data sink

- Error notification

The first two callbacks are for sending and receiving data. The third callback is invoked if an error occurs.

The prototypes for the three callbacks are:

```
status_t data_source(uint8_t * sourceWord, uint32_t instance);

status_t data_sink(uint8_t sinkWord, uint32_t instance);

void on_error(status_t error, uint32_t instance);
```

All callbacks are invoked from the IRQ state.

Setup

To initialize the DSPI slave driver, first create and fill in a dspi_slave_user_config_t struct. This struct defines the callbacks and the data format settings for the SPI peripheral. The struct is not required after the driver is initialized and can be allocated on the stack. The user also must pass the memory for the run-time state structure.

Here is an example of a config struct definition:

```
/* Set up and init the slave  */
dspi_slave_state_t dspiSlaveState;

dspi_slave_user_config_t slaveUserConfig;
slaveUserConfig.callbacks.dataSink = data_sink;
slaveUserConfig.callbacks.dataSource = data_source;
slaveUserConfig.callbacks.onError = on_error;
slaveUserConfig.dataConfig.bitsPerFrame = 16;
slaveUserConfig.dataConfig.clkPhase =
      kDspiClockPhase_FirstEdge;
slaveUserConfig.dataConfig.clkPolarity =
      kDspiClockPolarity_ActiveHigh;
slaveUserConfig.isModifiedTimingFormatEnabled = false;

dspi_slave_init(slaveInstance, &slaveUserConfig, &dspiSlaveState);
```

6.3.1 Data Structure Documentation

6.3.1.1 struct dspi_slave_callbacks_t

The user creates the function implementations.

Data Fields

- dspi_status_t(∗ dataSource)(uint8_t ∗sourceWord, uint32_t instance)
  Callback to get word to transmit.
- dspi_status_t(∗ dataSink)(uint8_t sinkWord, uint32_t instance)
  Callback to put received word.

- void(∗ onError)(dspi_status_t error, uint32_t instance)
  Callback to report a DSPI error, such as an under-run or over-run error.

6.3.1.1.0.8 Field Documentation

6.3.1.1.0.8.1 dspi_status_t(∗ dspi_slave_callbacks_t::dataSource)(uint8_t ∗sourceWord, uint32_t instance)

6.3.1.1.0.8.2 dspi_status_t(∗ dspi_slave_callbacks_t::dataSink)(uint8_t sinkWord, uint32_t instance)

6.3.1.1.0.8.3 void(∗ dspi_slave_callbacks_t::onError)(dspi_status_t error, uint32_t instance)

6.3.1.2 struct dspi_slave_state_t

This struct holds data that is used by the DSPI slave peripheral driver to communicate between the transfer function and the interrupt handler. The user needs to pass in the memory for this structure and the driver fills out the members.

Data Fields

- uint32_t instance
  DSPI module instance number.
- dspi_slave_callbacks_t callbacks
  Application/user callbacks.
- uint32_t bitsPerFrame
  Desired number of bits per frame.

6.3.1.3 struct dspi_slave_user_config_t

Data Fields

- dspi_slave_callbacks_t callbacks
  Application/user callbacks.
- dspi_data_format_config_t dataConfig
  Data format configuration structure.

6.3.1.3.0.9 Field Documentation

6.3.1.3.0.9.1 dspi_slave_callbacks_t dspi_slave_user_config_t::callbacks

6.3.2 Function Documentation

6.3.2.1 dspi_status_t dspi_slave_init (uint32_t *instance*, dspi_slave_state_t * *dspiState*, const dspi_slave_user_config_t * *slaveConfig*)

This function saves the callbacks to the run-time state structure for a later use in the interrupt handler. It also ungates the clock to the DSPI module, initializes the DSPI for slave mode, and enables the module and corresponding interrupts. Once initialized, the DSPI module is configured in slave mode and ready to receive data from the SPI master. This is an example to set up the dspi_slave_state_t and the dspi_slave_user_config_t parameters and to call the dspi_slave_init function by passing in these parameters:

```
   dspi_slave_state_t dspiSlaveState; <- the user simply allocates memory for this struct
   dspi_slave_user_config_t slaveUserConfig;
   slaveUserConfig.callbacks.dataSink = data_sink; <- set to user implementation of
      function
   slaveUserConfig.callbacks.dataSource = data_source; <- set to user implementation of function
   slaveUserConfig.callbacks.onError = on_error; <- set to user implementation of function
   slaveUserConfig.dataConfig.bitsPerFrame = 16;
   slaveUserConfig.dataConfig.clkPhase = kDspiClockPhase_FirstEdge;
   slaveUserConfig.dataConfig.clkPolarity = kDspiClockPolarity_ActiveHigh;
   dspi_slave_init(slaveInstance, &slaveUserConfig, &dspiSlaveState);
*
```

Parameters

instance	The instance number of the DSPI peripheral.
dspiState	The pointer to the DSPI slave driver state structure.
user	The configuration structure dspi_slave_user_config_t, including the callbacks.

Returns

An error code or kStatus_DSPI_Success.

6.3.2.2 void dspi_slave_shutdown (dspi_slave_state_t * *dspiState*)

Resets the DSPI peripheral, disables the interrupt to the core, and gates its clock.

Parameters

dspiState	The pointer to the DSPI slave driver state structure.

6.4 DSPI Shared IRQ Driver

The chapter describes the programming interface of the DSPI shared IRQ driver for master and slave Peripheral drivers.

Data Structures

- struct dspi_shared_irq_config_t
  Configuration of the DSPI IRQs shared by master and slave drivers. More...

Functions

- static void dspi_set_shared_irq_state (uint32_t instance, void ∗state, bool isMaster)
  Set the shared IRQ state structure.

Variables

- dspi_shared_irq_config_t g_dspiSharedIrqConfig [HW_SPI_INSTANCE_COUNT]
  Contains global IRQ configuration information for the DSPI drivers.

6.4.1 Data Structure Documentation

6.4.1.1 struct dspi_shared_irq_config_t

Data Fields

- bool isMaster
  Whether the IRQ is used by the master mode driver.
- void ∗ state
  Void pointer to driver state information.

6.4.1.1.0.10 Field Documentation

6.4.1.1.0.10.1 bool dspi_shared_irq_config_t::isMaster

6.4.2 Function Documentation

6.4.2.1 static void dspi_set_shared_irq_state (uint32_t *instance,* void ∗ *state,* bool *isMaster*) [inline], [static]

This function sets whether the master or slave driver IRQ handler will be invoked and to set the pointer to the driver run-time state structure associated with the desired module instance number. This is not a public API.

6.4.3 Variable Documentation

6.4.3.1 dspi_shared_irq_config_t g_dspiSharedIrqConfig[HW_SPI_INSTANCE_COUNT]

Chapter 7
Enhanced Direct Memory Access (eDMA)

The Kinetis SDK provides both HAL and Peripheral drivers for the Direct Memory Access (eDMA) block of Kinetis devices.

Modules

- DMAMUX HAL driver
  The part describes the programming interface of the DMAMUX HAL module.
- eDMA HAL driver
  The part describes the programming interface of the eDMA HAL driver.
- eDMA Peripheral Driver
  The part describes the programming interface of the eDMA Peripheral driver.
- eDMA request
  The part describes the programming interface of the eDMA DMA request resource.

7.1 eDMA HAL driver

The chapter describes the programming interface of the eDMA HAL driver.

Data Structures

- union edma_tcd_control_t
  eDMA TCD control configuration More...
- struct edma_minorloop_offset_config_t
  eDMA TCD Minor loop mapping configuration More...
- union edma_error_status_all_t
  Error status of the eDMA module. More...
- struct edma_software_tcd_t
  eDMA TCD More...
- struct edma_config_t
  DMA configuration structure. More...

Enumerations

- enum edma_status_t { ,
  kStatus_EDMA_InvalidArgument = 1U,
  kStatus_EDMA_Fail = 2U }
  eDMA status
- enum edma_channel_priority_t
  Priority limitation of the eDMA channel.
- enum edma_modulo_t
  eDMA modulo configuration
- enum edma_transfer_size_t
  eDMA transfer size configuration
- enum edma_bandwidth_configuration_t {
  kEdmaBandwidthStallNone = 0,
  kEdmaBandwidthStall4Cycle = 2,
  kEdmaBandwidthStall8Cycle = 3 }
  Bandwidth control configuration.
- enum edma_group_priority_t
  eDMA group priority

EDMA HAL common configuration

- void edma_hal_init (uint32_t instance, const edma_config_t *init)
  Initializes the eDMA module.
- static void edma_hal_cancel_transfer (uint32_t instance)
  Cancels the remaining data transfer.
- static void edma_hal_error_cancel_transfer (uint32_t instance)
  Cancels the remaining data transfer.
- static void edma_hal_set_minor_loop_mapping (uint32_t instance, bool isEnabled)
  Enables/Disables the minor loop mapping.

- static void edma_hal_set_continuous_mode (uint32_t instance, bool isContinuous)
  Configures the continuous mode.
- static void edma_hal_halt (uint32_t instance)
  Halts the DMA Operations.
- static void edma_hal_clear_halt (uint32_t instance)
  Clears the halt bit.
- static void edma_hal_set_halt_on_error (uint32_t instance, bool isHaltOnError)
  Halts the eDMA module when an error occurs.
- static void edma_hal_set_fixed_priority_channel_arbitration (uint32_t instance)
  The fixed priority arbitration is used for the channel selection.
- static void edma_hal_set_roundrobin_channel_arbitration (uint32_t instance)
  The round robin arbitration is used for the channel selection.
- static void edma_hal_set_debug_mode (uint32_t instance, bool isEnabled)
  Enables/Disables the eDMA DEBUG mode.
- static uint32_t edma_hal_get_error_status (uint32_t instance)
  Gets the error status of the eDMA module.
- static void edma_hal_disable_all_enabled_error_interrupt (uint32_t instance)
  Disables the interrupt when an error happens on any of channel in the eDMA module.
- static void edma_hal_enable_all_channel_error_interrupt (uint32_t instance)
  Enables an interrupt when an error happens on any channel in the eDMA module.
- static void edma_hal_disable_all_channel_dma_request (uint32_t instance)
  Disables the DMA request for all eDMA channels.
- static void edma_hal_enable_all_channel_dma_request (uint32_t instance)
  Enables the DMA request for all eDMA channels.
- static void edma_hal_clear_all_channel_done_status (uint32_t instance)
  Clears the done status for all eDMA channels.
- static void edma_hal_trigger_all_channel_start_bit (uint32_t instance)
  Triggers all channel start bits.
- static void edma_hal_clear_all_channel_error_status (uint32_t instance)
  Clears the error status for all eDMA channels.
- static void edma_hal_clear_all_channel_interrupt_request (uint32_t instance)
  Clears an interrupt request for all eDMA channels.
- static uint32_t edma_hal_get_all_channel_interrupt_request_status (uint32_t instance)
  Gets the interrupt status for all eDMA channels.
- static uint32_t edma_hal_get_all_channel_error_status (uint32_t instance)
  Gets the channel error status for all eDMA channels.
- static uint32_t edma_hal_get_all_channel_dma_request_status (uint32_t instance)
  Gets the status of the DMA request for all DMA channels.

EDMA HAL channel configuration.

- static bool edma_hal_check_dma_request_enable_status (uint32_t instance, uint32_t channel)
  Check whether the channel DMA request is enabled.
- static void edma_hal_disable_error_interrupt (uint32_t instance, uint32_t channel)
  Disables an interrupt when an error happens in the eDMA channel.
- static void edma_hal_enable_error_interrupt (uint32_t instance, uint32_t channel)
  Enables an interrupt when an error happens in the eDMA channel.
- static void edma_hal_disable_dma_request (uint32_t instance, uint32_t channel)
  Disables the DMA request for an eDMA channel.
- static void edma_hal_enable_dma_request (uint32_t instance, uint32_t channel)

Enables the DMA request for a specified eDMA channel.

- static void edma_hal_clear_done_status (uint32_t instance, uint32_t channel)
  Clears the done status for an eDMA channel.
- static void edma_hal_trigger_start_bit (uint32_t instance, uint32_t channel)
  Starts an eDMA channel manually.
- static void edma_hal_clear_error_status (uint32_t instance, uint32_t channel)
  Clears an error status for the eDMA channel.
- static void edma_hal_clear_interrupt_request (uint32_t instance, uint32_t channel)
  Clears an interrupt request for the eDMA channel.
- static void edma_hal_set_channel_preemp_ability (uint32_t instance, uint32_t channel, bool isDisabled)
  Configures the preempt feature for an eDMA channel.
- static void edma_hal_set_channel_preemption_ability (uint32_t instance, uint32_t channel, bool isEnabled)
  Configures the preempt feature for the eDMA channel.
- static void edma_hal_set_channel_priority (uint32_t instance, uint32_t channel, uint32_t priority)
  Configures the eDMA channel priority.

eDMA HAL hardware TCD configuration

- static void edma_hal_htcd_configure_source_address (uint32_t instance, uint32_t channel, uint32_t address)
  Configures the source address for the hardware TCD.
- static void edma_hal_htcd_configure_source_offset (uint32_t instance, uint32_t channel, int16_t offset)
  Configures the source address signed offset for the hardware TCD.
- static void edma_hal_htcd_configure_source_modulo (uint32_t instance, uint32_t channel, edma_modulo_t modulo)
  Configures the source modulo for the hardware TCD.
- static void edma_hal_htcd_configure_source_transfersize (uint32_t instance, uint32_t channel, edma_transfer_size_t size)
  Configures the source data transfersize for the hardware TCD.
- static void edma_hal_htcd_configure_dest_modulo (uint32_t instance, uint32_t channel, edma_modulo_t modulo)
  Configures the destination modulo for the hardware TCD.
- static void edma_hal_htcd_configure_dest_transfersize (uint32_t instance, uint32_t channel, edma_transfer_size_t size)
  Configures the destination data transfersize for the hardware TCD.
- static void edma_hal_htcd_configure_nbytes_minorloop_disabled (uint32_t instance, uint32_t channel, uint32_t nbytes)
  Configures the nbytes if minor loop mapping is disabled for the hardware TCD.
- static void edma_hal_htcd_configure_nbytes_minorloop_enabled_offset_disabled (uint32_t instance, uint32_t channel, uint32_t nbytes)
  Configures the nbytes if the minor loop mapping is enabled and offset is disabled for the hardware TCD.
- static void edma_hal_htcd_configure_nbytes_minorloop_enabled_offset_enabled (uint32_t instance, uint32_t channel, uint32_t nbytes)
  Configures the nbytes if the minor loop mapping is enabled and offset is enabled for the hardware TCD.
- uint32_t edma_hal_htcd_get_nbytes_configuration (uint32_t instance, uint32_t channel)

Gets the nbytes configuration data.

- static void edma_hal_htcd_configure_minorloop_offset (uint32_t instance, uint32_t channel, edma_minorloop_offset_config_t config)
  Configures the minorloop offset for the hardware TCD.
- static void edma_hal_htcd_configure_source_last_adjustment (uint32_t instance, uint32_t channel, int32_t size)
  Configures the last source address adjustment for the hardware TCD.
- static void edma_hal_htcd_configure_dest_address (uint32_t instance, uint32_t channel, uint32_t address)
  Configures the destination address for the hardware TCD.
- static void edma_hal_htcd_configure_dest_offset (uint32_t instance, uint32_t channel, int16_t offset)
  Configures the destination address signed offset for the hardware TCD.
- static void edma_hal_htcd_configure_dest_last_adjustment_or_scatter_address (uint32_t instance, uint32_t channel, uint32_t address)
  Configures the last source address adjustment or the memory address for the next transfer control for the hardware TCD.
- static void edma_hal_htcd_configure_bandwidth (uint32_t instance, uint32_t channel, edma_bandwidth_configuration_t bandwidth)
  Configures the bandwidth for the hardware TCD.
- static void edma_hal_htcd_configure_majorlink_channel (uint32_t instance, uint32_t channel, uint32_t majorchannel)
  Configures the major link channel number for the hardware TCD.
- static uint32_t edma_hal_htcd_get_majorlink_channel (uint32_t instance, uint32_t channel)
  Gets the major link channel for the hardware TCD.
- static void edma_hal_htcd_set_majorlink (uint32_t instance, uint32_t channel, bool isEnabled)
  Enables/Disables the major link channel feature for the hardware TCD.
- static void edma_hal_htcd_set_scatter_gather_process (uint32_t instance, uint32_t channel, bool isEnabled)
  Enables/Disables the scatter/gather feature for the hardware TCD.
- static bool edma_hal_htcd_is_gather_scatter_enabled (uint32_t instance, uint32_t channel)
  Checks whether the scatter/gather feature is enabled for the hardware TCD.
- static void edma_hal_htcd_set_disable_dma_request_after_tcd_done (uint32_t instance, uint32_t channel, bool isDisabled)
  Disables/Enables the DMA request after the major loop completes for the hardware TCD.
- static void edma_hal_htcd_set_half_complete_interrupt (uint32_t instance, uint32_t channel, bool isEnabled)
  Enables/Disables the half complete interrupt for the hardware TCD.
- static void edma_hal_htcd_set_complete_interrupt (uint32_t instance, uint32_t channel, bool isEnabled)
  Enables/Disables the interrupt after the major loop completes for the hardware TCD.
- static void edma_hal_htcd_trigger_channel_start (uint32_t instance, uint32_t channel)
  Triggers the start bits for the hardware TCD.
- static bool edma_hal_htcd_is_channel_active (uint32_t instance, uint32_t channel)
  Checks whether the channel is running for the hardware TCD.
- static bool edma_hal_htcd_is_channel_done (uint32_t instance, uint32_t channel)
  Checks whether the major loop is exhausted for the hardware TCD.
- static void edma_hal_htcd_set_minor_link (uint32_t instance, uint32_t channel, bool isEnabled)
  Enables/Disables the channel link after the minor loop for the hardware TCD.
- static void edma_hal_htcd_set_current_minor_link (uint32_t instance, uint32_t channel, bool is-

Enabled)

Enables/Disables the channel link after the minor loop in the current register for the hardware TCD.

- static void edma_hal_htcd_configure_minor_link_channel (uint32_t instance, uint32_t channel, uint32_t minorchannel)

  Configures the minor loop link channel for the hardware TCD.
- static void edma_hal_htcd_configure_current_minor_link_channel (uint32_t instance, uint32_t channel, uint32_t minorchannel)

  Configures the minor loop link channel in the current register for the hardware TCD.
- static void edma_hal_htcd_configure_majorcount_minorlink_disabled (uint32_t instance, uint32_t channel, uint32_t count)

  Configures the major count if the minor loop channel link is disabled for the hardware TCD.
- static void edma_hal_htcd_configure_current_majorcount_minorlink_disabled (uint32_t instance, uint32_t channel, uint32_t count)

  Configures the current major count if the minor loop channel link is disabled for the hardware TCD.
- static void edma_hal_htcd_configure_majorcount_minorlink_enabled (uint32_t instance, uint32_t channel, uint32_t count)

  Configures the major count if the minor loop channel link is enabled for the hardware TCD.
- static void edma_hal_htcd_configure_current_majorcount_minorlink_enabled (uint32_t instance, uint32_t channel, uint32_t count)

  Configures the current major count if the minor loop channel link is enabled for the hardware TCD.
- uint32_t edma_hal_htcd_get_current_major_count (uint32_t instance, uint32_t channel)

  Gets the current major loop count.
- uint32_t edma_hal_htcd_get_begin_major_count (uint32_t instance, uint32_t channel)

  Gets the beginning major loop count.
- uint32_t edma_hal_htcd_get_unfinished_bytes (uint32_t instance, uint32_t channel)

  Gets the bytes number not to be transferred for the hardware TCD.
- uint32_t edma_hal_htcd_get_finished_bytes (uint32_t instance, uint32_t channel)

  Gets the number of already transferred bytes for the hardware TCD.

eDMA HAL software TCD configuration

- static void edma_hal_stcd_configure_source_address (edma_software_tcd_t *stcd, uint32_t address)

  Configures the source address for the software TCD.
- static void edma_hal_stcd_configure_source_offset (edma_software_tcd_t *stcd, uint32_t offset)

  Configures the source address for the software TCD.
- static void edma_hal_stcd_configure_source_modulo (edma_software_tcd_t *stcd, edma_modulo_t modulo)

  Configures the source modulo for the software TCD.
- static void edma_hal_stcd_configure_source_transfersize (edma_software_tcd_t *stcd, edma_transfer_size_t size)

  Configures the source data transfersize for the software TCD.
- static void edma_hal_stcd_configure_dest_modulo (edma_software_tcd_t *stcd, edma_modulo_t modulo)

  Configures the destination modulo for the software TCD.
- static void edma_hal_stcd_configure_dest_transfersize (edma_software_tcd_t *stcd, edma_transfer_size_t size)

  Configures the destination data transfersize for the software TCD.

- static void edma_hal_stcd_configure_nbytes_minorloop_disabled (edma_software_tcd_t *stcd, uint32_t nbytes)
  Configures the nbytes if minor loop mapping is disabled the for software TCD.
- static void edma_hal_stcd_configure_nbytes_minorloop_enabled_offset_disabled (edma_software_tcd_t *stcd, uint32_t nbytes)
  Configures the nbytes if the minor loop mapping is enabled and offset is disabled for the software TCD.
- static void edma_hal_stcd_configure_nbytes_minorloop_enabled_offset_enabled (edma_software_tcd_t *stcd, uint32_t nbytes)
  Configures the nbytes if minor loop mapping is enabled and offset is enabled for the software TCD.
- static void edma_hal_stcd_configure_minorloop_offset (edma_software_tcd_t *stcd, edma_minorloop_offset_config_t *config)
  Configures the minorloop offset for the software TCD.
- static void edma_hal_stcd_configure_source_last_adjustment (edma_software_tcd_t *stcd, int32_t size)
  Configures the last source address adjustment for the software TCD.
- static void edma_hal_stcd_configure_dest_address (edma_software_tcd_t *stcd, uint32_t address)
  Configures the destination address for the software TCD.
- static void edma_hal_stcd_configure_dest_offset (edma_software_tcd_t *stcd, uint32_t offset)
  Configures the destination address signed offset for the software TCD.
- static void edma_hal_stcd_configure_dest_last_adjustment_or_scatter_address (edma_software_tcd_t *stcd, uint32_t address)
  Configures the last source address adjustment or the memory address for the next transfer control for the software TCD.
- static void edma_hal_stcd_configure_bandwidth (edma_software_tcd_t *stcd, edma_bandwidth_configuration_t bandwidth)
  Configures the bandwidth for the software TCD.
- static void edma_hal_stcd_configure_majorlink_channel (edma_software_tcd_t *stcd, uint32_t majorchannel)
  Configures the major link channel number for the software TCD.
- static void edma_hal_stcd_set_majorlink (edma_software_tcd_t *stcd, bool isEnabled)
  Enables/Disables the major link channel feature for the software TCD.
- static void edma_hal_stcd_set_scatter_gather_process (edma_software_tcd_t *stcd, bool isEnabled)
  Enables/Disables the scatter/gather feature for the software TCD.
- static void edma_hal_stcd_set_disable_dma_request_after_tcd_done (edma_software_tcd_t *stcd, bool isDisabled)
  Disables/Enables the DMA request after the major loop completes for the software TCD.
- static void edma_hal_stcd_set_half_complete_interrupt (edma_software_tcd_t *stcd, bool isEnabled)
  Enables/Disables the half complete interrupt for the software TCD.
- static void edma_hal_stcd_set_complete_interrupt (edma_software_tcd_t *stcd, bool isEnabled)
  Enables/Disables the interrupt after major loop complete for the software TCD.
- static void edma_hal_stcd_trigger_channel_start (edma_software_tcd_t *stcd)
  Sets the trigger start bits for the software TCD.
- static void edma_hal_stcd_set_minor_link (edma_software_tcd_t *stcd, bool isEnabled)
  Enables/Disables the channel link after the minor loop for the software TCD.
- static void edma_hal_stcd_set_current_minor_link (edma_software_tcd_t *stcd, bool isEnabled)
  Enables/Disables the current channel link after the minor loop for the software TCD.
- static void edma_hal_stcd_configure_minor_link_channel (edma_software_tcd_t *stcd, uint32_t minorchannel)

Configures the minor loop link channel for the software TCD.

- static void edma_hal_stcd_configure_current_minor_link_channel (edma_software_tcd_t *stcd, uint32_t minorchannel)
  Configures the minor loop link channel for the software TCD.
- static void edma_hal_stcd_configure_majorcount_minorlink_disabled (edma_software_tcd_t *stcd, uint32_t count)
  Configures the major count if the minor loop channel link is disabled for the software TCD.
- static void edma_hal_stcd_configure_current_majorcount_minorlink_disabled (edma_software_tcd_t *stcd, uint32_t count)
  Configure current major count if minor loop channel link is disabled for software TCD.
- static void edma_hal_stcd_configure_majorcount_minorlink_enabled (edma_software_tcd_t *stcd, uint32_t count)
  Configures the major count if the minor loop channel link is enabled for the software TCD.
- static void edma_hal_stcd_configure_current_majorcount_minorlink_enabled (edma_software_tcd_t *stcd, uint32_t count)
  Configures the current major count if the minor loop channel link is enabled for the software TCD.
- void edma_hal_stcd_push_to_htcd (uint32_t instance, uint32_t channel, edma_software_tcd_t *stcd)
  Copy the software TCD configuration to the hardware TCD.

7.1.0.2 EDMA HAL Driver

Overview

The eDMA HAL driver provides a function set to operate the eDMA hardware.

7.1.1 Data Structure Documentation

7.1.1.1 union edma_tcd_control_t

7.1.1.1.0.11 Field Documentation

7.1.1.1.0.11.1 uint16_t edma_tcd_control_t::majorInterrupt

7.1.1.1.0.11.2 uint16_t edma_tcd_control_t::halfInterrupt

7.1.1.1.0.11.3 uint16_t edma_tcd_control_t::disabledDmaRequest

7.1.1.1.0.11.4 uint16_t edma_tcd_control_t::enabledScatterGather

7.1.1.1.0.11.5 uint16_t edma_tcd_control_t::enableMajorLink

7.1.1.2 struct edma_minorloop_offset_config_t

7.1.1.3 union edma_error_status_all_t

7.1.1.4 struct edma_software_tcd_t

7.1.1.5 struct edma_config_t

Data Fields

- bool isEnableMinorLoopping
  Enabled the minor loop mapping.
- bool isEnableContinuousMode
  Enabled the continuous mode.
- bool isHaltOnError
  Halt if error happens.
- bool isEnableRoundrobinArbitration
  Enabled round robin or fixed priority arbitration.
- bool isEnableDebug
  Enabled Debug mode.

7.1.1.5.0.12 Field Documentation

7.1.1.5.0.12.1 bool edma_config_t::isEnableMinorLoopping

7.1.1.5.0.12.2 bool edma_config_t::isEnableContinuousMode

7.1.1.5.0.12.3 bool edma_config_t::isHaltOnError

7.1.1.5.0.12.4 bool edma_config_t::isEnableRoundrobinArbitration

7.1.1.5.0.12.5 bool edma_config_t::isEnableDebug

7.1.2 Enumeration Type Documentation

7.1.2.1 enum edma_status_t

Enumerator

kStatus_EDMA_InvalidArgument Parameter is not available for the current configuration.
kStatus_EDMA_Fail Function operation failed.

7.1.2.2 enum edma_bandwidth_configuration_t

Enumerator

kEdmaBandwidthStallNone No eDMA engine stalls.
kEdmaBandwidthStall4Cycle eDMA engine stalls for 4 cycles after each read/write.
kEdmaBandwidthStall8Cycle eDMA engine stalls for 4 cycles after each read/write.

7.1.3 Function Documentation

7.1.3.1 void edma_hal_init (uint32_t *instance,* const edma_config_t * *init*)

The function configures the eDMA module with the corresponding global configuration. The configuration is for all channels in this module.

Parameters

module	eDMA module
init	Init data structure

7.1.3.2 static void edma_hal_cancel_transfer (uint32_t *instance*) [inline], [static]

Stops the executing channel and forces the minor loop to finish. The cancellation takes effect after the last write of the current read/write sequence. The CX clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop had completed.

Parameters

instance	eDMA module

7.1.3.3 static void edma_hal_error_cancel_transfer (uint32_t *instance*) [inline], [static]

Stops the executing channel and forces the minor loop to finish. The cancellation takes effect after the last write of the current read/write sequence. The ECX bit clears itself after the cancel is honored. In addition to cancelling the transfer, ECX treats the cancel as an error condition.

Parameters

instance	eDMA module

7.1.3.4 static void edma_hal_set_minor_loop_mapping (uint32_t *instance,* bool *isEnabled*) [inline], [static]

If enabled, the NBYTES is redefined to include individual enable fields. And the NBYTES field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The NBYTES field is reduced when either offset is enabled.

Parameters

instance	eDMA module
isEnabled	Enable or disable.

7.1.3.5 static void edma_hal_set_continuous_mode (uint32_t *instance,* bool *isContinuous*) [inline], [static]

If set, a minor loop channel link does not go through the channel arbitration before being activated again. Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

Parameters

module	eDMA module
isContinuous	Whether the minor loop finish triggers itself.

7.1.3.6 static void edma_hal_halt (uint32_t *instance*) [inline], [static]

Stalls the start of any new channels. Executing channels are allowed to complete.

Parameters

instance	eDMA module.

7.1.3.7 static void edma_hal_clear_halt (uint32_t *instance*) [inline], [static]

If a previous eDMA channel is halted, clear operation would resume it back to executing.

Parameters

instance	eDMA module.

7.1.3.8 static void edma_hal_set_halt_on_error (uint32_t *instance,* bool *isHaltOnError*) [inline], [static]

An error causes the HALT bit to be set. Subsequently, all service requests are ignored until the HALT bit is cleared.

Parameters

instance	eDMA module.
isHaltOnError	halts or does not halt when an error occurs.

7.1.3.9 static void edma_hal_set_fixed_priority_channel_arbitration (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module.

7.1.3.10 static void edma_hal_set_roundrobin_channel_arbitration (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module.

7.1.3.11 static void edma_hal_set_debug_mode (uint32_t *instance,* bool *isEnabled*) [inline], [static]

When in debug mode, the DMA stalls the start of a new channel. Executing channels are allowed to complete. Channel execution resumes either when the system exits debug mode or when the EDBG bit is cleared.

Parameters

instance	eDMA module.

7.1.3.12 static uint32_t edma_hal_get_error_status (uint32_t *instance*) [inline], [static]

The detailed reason is listed along with the error channel.

Parameters

instance	eDMA module

Returns

Detailed information of the error type in the eDMA module.

7.1.3.13 static void edma_hal_disable_all_enabled_error_interrupt (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

7.1.3.14 static void edma_hal_enable_all_channel_error_interrupt (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

7.1.3.15 static void edma_hal_disable_all_channel_dma_request (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

7.1.3.16 static void edma_hal_enable_all_channel_dma_request (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

7.1.3.17 static void edma_hal_clear_all_channel_done_status (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

7.1.3.18 static void edma_hal_trigger_all_channel_start_bit (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

7.1.3.19 static void edma_hal_clear_all_channel_error_status (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

7.1.3.20 static void edma_hal_clear_all_channel_interrupt_request (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

7.1.3.21 static uint32_t edma_hal_get_all_channel_interrupt_request_status (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

Returns

32 bit data. Every bit stands for an eDMA channel. For example, bit 0 stands for channel 0 and so on.

7.1.3.22 static uint32_t edma_hal_get_all_channel_error_status (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

Returns

32 bit data. Every bit stands for an eDMA channel. For example, bit 0 stands for channel 0 and so on.

7.1.3.23 static uint32_t edma_hal_get_all_channel_dma_request_status (uint32_t *instance*) [inline], [static]

Parameters

instance	eDMA module

Returns

32 bit data. Every bit stands for an eDMA channel. For example, bit 0 stands for channel 0 and so on.

7.1.3.24 static bool edma_hal_check_dma_request_enable_status (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Check whether the DMA request of a specified channel is enabled.

Parameters

instance	eDMA module
channel	eDMA channel

Returns

True stands for enabled. False stands for disabled.

7.1.3.25 static void edma_hal_disable_error_interrupt (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Disables an error interrupt for the eDMA module.

Parameters

instance	eDMA module
channel	eDMA channel

7.1.3.26 static void edma_hal_enable_error_interrupt (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel

7.1.3.27 static void edma_hal_disable_dma_request (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel

7.1.3.28 static void edma_hal_enable_dma_request (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel

7.1.3.29 static void edma_hal_clear_done_status (uint32_t *instance,* uint32_t *channel*) [inline], [static]

The DONE status of the DMA channel is cleared. If the scatter/gather state is enabled, the DONE status in CSR can be cleared but the global DONE statue is still set. This function is to clear the global done state.

Parameters

instance	eDMA module
channel	eDMA channel

7.1.3.30 static void edma_hal_trigger_start_bit (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel

7.1.3.31 static void edma_hal_clear_error_status (uint32_t *instance,* uint32_t *channel*) [inline], [static]

*

Parameters

instance	eDMA module
channel	eDMA channel

7.1.3.32 static void edma_hal_clear_interrupt_request (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel

7.1.3.33 static void edma_hal_set_channel_preemp_ability (uint32_t *instance,* uint32_t *channel,* bool *isDisabled*) [inline], [static]

If it is disabled, the DMA channel can't suspend a lower priority channel.

Parameters

instance	eDMA module
channel	eDMA channel
preempt	configuration mode for preempt

7.1.3.34 static void edma_hal_set_channel_preemption_ability (uint32_t *instance,* uint32_t *channel,* bool *isEnabled*) [inline], [static]

If enabled, channel can be temporarily suspended by a higher priority channel.

Parameters

instance	eDMA module
channel	eDMA channel
preempt	configuration mode for preempt

7.1.3.35 static void edma_hal_set_channel_priority (uint32_t *instance,* uint32_t *channel,* uint32_t *priority*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
priority	Priority of the DMA channel. Different channels should have different priority inside a group.

7.1.3.36 static void edma_hal_htcd_configure_source_address (uint32_t *instance,* uint32_t *channel,* uint32_t *address*) [inline], [static]

Parameters

instance	eDMA module

channel	eDMA channel
address	memory address pointing to the source data

7.1.3.37 static void edma_hal_htcd_configure_source_offset (uint32_t *instance,* uint32_t *channel,* int16_t *offset*) [inline], [static]

Sign-extended offset applied to the current source address to form the next-state value as each source read is complete.

Parameters

instance	eDMA module
channel	eDMA channel
offset	signed-offset

7.1.3.38 static void edma_hal_htcd_configure_source_modulo (uint32_t *instance,* uint32_t *channel,* edma_modulo_t *modulo*) [inline], [static]

The value defines a specific address range specified as the value after the SADDR + SOFF calculation is performed on the original register value. Setting this field provides the ability to implement a circular data. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of the lower address bits allowed to change. For a circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with SMOD function restricting the addresses to a 0-modulo-size range.

Parameters

instance	eDMA module
channel	eDMA channel
modulo	enum type for an allowed modulo

7.1.3.39 static void edma_hal_htcd_configure_source_transfersize (uint32_t *instance,* uint32_t *channel,* edma_transfer_size_t *size*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
size	enum type for transfer size

7.1.3.40 static void edma_hal_htcd_configure_dest_modulo (uint32_t *instance,* uint32_t *channel,* edma_modulo_t *modulo*) [inline], [static]

The value defines a specific address range as the value after the DADDR + DOFF calculation is performed on the original register value. Setting this field provides the ability to implement a circular data. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits allowed to change. For a circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with DMOD function restricting the addresses to a 0-modulo-size range.

Parameters

instance	eDMA module
channel	eDMA channel
modulo	enum type for an allowed modulo

7.1.3.41 static void edma_hal_htcd_configure_dest_transfersize (uint32_t *instance,* uint32_t *channel,* edma_transfer_size_t *size*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
size	enum type for the transfer size

7.1.3.42 static void edma_hal_htcd_configure_nbytes_minorloop_disabled (uint32_t *instance,* uint32_t *channel,* uint32_t *nbytes*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
nbytes	Number of bytes to be transferred in each service request of the channel

7.1.3.43 static void edma_hal_htcd_configure_nbytes_minorloop_enabled_offset_disabled (uint32_t *instance,* uint32_t *channel,* uint32_t *nbytes*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
nbytes	Number of bytes to be transferred in each service request of the channel

7.1.3.44 static void edma_hal_htcd_configure_nbytes_minorloop_enabled_offset_enabled (uint32_t *instance,* uint32_t *channel,* uint32_t *nbytes*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
nbytes	Number of bytes to be transferred in each service request of the channel

7.1.3.45 uint32_t edma_hal_htcd_get_nbytes_configuration (uint32_t *instance,* uint32_t *channel*)

This function decides whether the minor loop mapping is enabled or whether the source/dest minor loop mapping is enabled. Then, the nbytes are returned accordingly.

Parameters

instance	eDMA module

channel	eDMA channel

Returns

nbytes configuration

7.1.3.46 static void edma_hal_htcd_configure_minorloop_offset (uint32_t *instance,* uint32_t *channel,* edma_minorloop_offset_config_t *config*) [inline], [static]

Configures both the enable bits and the offset value. If neither source nor dest offset is enabled, offset is not configured.

Parameters

instance	eDMA module
channel	eDMA channel
config	Configuration data structure for the minorloop offset

7.1.3.47 static void edma_hal_htcd_configure_source_last_adjustment (uint32_t *instance,* uint32_t *channel,* int32_t *size*) [inline], [static]

Adjustment value added to the source address at the completion of the major iteration count. This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.

Parameters

instance	eDMA module
channel	eDMA channel
size	adjustment value

7.1.3.48 static void edma_hal_htcd_configure_dest_address (uint32_t *instance,* uint32_t *channel,* uint32_t *address*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
address	memory address pointing to destination data

7.1.3.49 static void edma_hal_htcd_configure_dest_offset (uint32_t *instance,* uint32_t *channel,* int16_t *offset*) [inline], [static]

Sign-extended offset applied to the current source address to form the next-state value as each destination write is complete.

Parameters

instance	eDMA module
channel	eDMA channel
offset	signed-offset

7.1.3.50 static void edma_hal_htcd_configure_dest_last_adjustment_or_scatter_address (uint32_t *instance,* uint32_t *channel,* uint32_t *address*) [inline], [static]

If a scatter/gather feature is enabled (edma_hal_htcd_set_scatter_gather_process()):

This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. The channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32-byte. Otherwise, a configuration error is reported.

else:

An adjustment value added to the source address at the completion of the major iteration count. This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.

Parameters

instance	eDMA module

channel	eDMA channel
size	adjustment value

7.1.3.51 static void edma_hal_htcd_configure_bandwidth (uint32_t *instance,* uint32_t *channel,* edma_bandwidth_configuration_t *bandwidth*) [inline], [static]

Throttles the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

instance	eDMA module
channel	eDMA channel
bandwidth	enum type for bandwidth control

7.1.3.52 static void edma_hal_htcd_configure_majorlink_channel (uint32_t *instance,* uint32_t *channel,* uint32_t *majorchannel*) [inline], [static]

If the majorlink is enabled, after the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by these six bits by setting that channel start bits.

Parameters

instance	eDMA module
channel	eDMA channel
majorchannel	channel number for major link

7.1.3.53 static uint32_t edma_hal_htcd_get_majorlink_channel (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel

Returns

major link channel number

7.1.3.54 static void edma_hal_htcd_set_majorlink (uint32_t *instance,* uint32_t *channel,* bool *isEnabled*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
isEnabled	Enable/Disable

7.1.3.55 static void edma_hal_htcd_set_scatter_gather_process (uint32_t *instance,* uint32_t *channel,* bool *isEnabled*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
isEnabled	Enable/Disable

7.1.3.56 static bool edma_hal_htcd_is_gather_scatter_enabled (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel

Returns

True stand for enabled. False stands for disabled.

7.1.3.57 static void edma_hal_htcd_set_disable_dma_request_after_tcd_done (uint32_t *instance,* uint32_t *channel,* bool *isDisabled*) [inline], [static]

If disabled, the eDMA hardware automatically clears the corresponding DMA request when the current major iteration count reaches zero.

Parameters

instance	eDMA module
channel	eDMA channel
isDisabled	Disable/Enable.

7.1.3.58 static void edma_hal_htcd_set_half_complete_interrupt (uint32_t *instance,* uint32_t *channel,* bool *isEnabled*) [inline], [static]

If set, the channel generates an interrupt request by setting the appropriate bit in the interrupt register when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER >> 1)). This half-way point interrupt request is provided to support the double-buffered schemes or other types of data movement where the processor needs an early indication of the transfer's process.

Parameters

instance	eDMA module
channel	eDMA channel
isEnabled	Enable/Disable

7.1.3.59 static void edma_hal_htcd_set_complete_interrupt (uint32_t *instance,* uint32_t *channel,* bool *isEnabled*) [inline], [static]

If enabled, the channel generates an interrupt request by setting the appropriate bit in the interrupt register when the current major iteration count reaches zero.

Parameters

instance	eDMA module
channel	eDMA channel
isEnabled	Enable/Disable

7.1.3.60 static void edma_hal_htcd_trigger_channel_start (uint32_t *instance,* uint32_t *channel*) [inline], [static]

The eDMA hardware automatically clears this flag after the channel begins execution.

Parameters

instance	eDMA module
channel	eDMA channel

7.1.3.61 static bool edma_hal_htcd_is_channel_active (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel

Returns

True stands for running. False stands for not.

7.1.3.62 static bool edma_hal_htcd_is_channel_done (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel

Returns

True stands for running. False stands for not.

7.1.3.63 static void edma_hal_htcd_set_minor_link (uint32_t *instance,* uint32_t *channel,* bool *isEnabled*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
isEnabled	Enable/Disable

7.1.3.64 static void edma_hal_htcd_set_current_minor_link (uint32_t *instance,* uint32_t *channel,* bool *isEnabled*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
isEnabled	Enable/Disable

7.1.3.65 static void edma_hal_htcd_configure_minor_link_channel (uint32_t *instance,* uint32_t *channel,* uint32_t *minorchannel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
minorchannel	minor loop link channel

7.1.3.66 static void edma_hal_htcd_configure_current_minor_link_channel (uint32_t *instance,* uint32_t *channel,* uint32_t *minorchannel*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
minorchannel	minor loop link channel

7.1.3.67 static void edma_hal_htcd_configure_majorcount_minorlink_disabled (uint32_t *instance,* uint32_t *channel,* uint32_t *count*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
count	major loop count

7.1.3.68 static void edma_hal_htcd_configure_current_majorcount_minorlink_disabled (uint32_t *instance,* uint32_t *channel,* uint32_t *count*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
count	major loop count

7.1.3.69 static void edma_hal_htcd_configure_majorcount_minorlink_enabled (uint32_t *instance,* uint32_t *channel,* uint32_t *count*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
count	major loop count

7.1.3.70 static void edma_hal_htcd_configure_current_majorcount_minorlink_enabled (uint32_t *instance,* uint32_t *channel,* uint32_t *count*) [inline], [static]

Parameters

instance	eDMA module
channel	eDMA channel
count	major loop count

7.1.3.71 uint32_t edma_hal_htcd_get_current_major_count (uint32_t *instance,* uint32_t *channel*)

Parameters

instance	eDMA module
channel	eDMA channel

Returns

current major loop count

7.1.3.72 uint32_t edma_hal_htcd_get_begin_major_count (uint32_t *instance,* uint32_t *channel*)

Parameters

instance	eDMA module
channel	eDMA channel

Returns

begin major loop count

7.1.3.73 uint32_t edma_hal_htcd_get_unfinished_bytes (uint32_t *instance,* uint32_t *channel*)

Parameters

instance	eDMA module
channel	eDMA channel

Returns

data bytes to be transferred

7.1.3.74 uint32_t edma_hal_htcd_get_finished_bytes (uint32_t *instance,* uint32_t *channel*)

Parameters

instance	eDMA module
channel	eDMA channel

Returns

data bytes to be transferred

7.1.3.75 static void edma_hal_stcd_configure_source_address (edma_software_tcd_t * *stcd,* uint32_t *address*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
address	memory address pointing to the source data

7.1.3.76 static void edma_hal_stcd_configure_source_offset (edma_software_tcd_t * *stcd,* uint32_t *offset*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
address	memory address pointing to the source data

7.1.3.77 static void edma_hal_stcd_configure_source_modulo (edma_software_tcd_t * *stcd,* edma_modulo_t *modulo*) [inline], [static]

The value defines a specific address range as the value after the SADDR + SOFF calculation is performed on the original register value. Setting this field provides the ability to implement a circular data queue. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits allowed to change. For a circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with SMOD function restricting the addresses to a 0-modulo-size range.

Parameters

STCD	memory pointing to the eDMA software TCD
modulo	enum type for the allowed modulo

7.1.3.78 static void edma_hal_stcd_configure_source_transfersize (edma_software_tcd_t * *stcd,* edma_transfer_size_t *size*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
size	enum type for transfer size

7.1.3.79 static void edma_hal_stcd_configure_dest_modulo (edma_software_tcd_t * *stcd,* edma_modulo_t *modulo*) [inline], [static]

The value defines a specific address range as the value after the DADDR + DOFF calculation is performed on the original register value. Setting this field provides the ability to implement a circular data queue easily. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits allowed to change. For a circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with DMOD function restricting the addresses to a 0-modulo-size range.

Parameters

STCD	memory pointing to the eDMA software TCD
modulo	enum type for allowed modulo

7.1.3.80 static void edma_hal_stcd_configure_dest_transfersize (edma_software_tcd_t * *stcd,* edma_transfer_size_t *size*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
size	enum type for transfer size

7.1.3.81 static void edma_hal_stcd_configure_nbytes_minorloop_disabled (edma_software_tcd_t * *stcd,* uint32_t *nbytes*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
nbytes	Number of bytes to be transferred in each service request of the channel

7.1.3.82 static void edma_hal_stcd_configure_nbytes_minorloop_enabled_offset_disabled (edma_software_tcd_t * *stcd,* uint32_t *nbytes*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
nbytes	Number of bytes to be transferred in each service request of the channel

7.1.3.83 static void edma_hal_stcd_configure_nbytes_minorloop_enabled_offset_enabled (edma_software_tcd_t * *stcd,* uint32_t *nbytes*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD.
nbytes	Number of bytes to be transferred in each service request of the channel.

7.1.3.84 static void edma_hal_stcd_configure_minorloop_offset (edma_software_tcd_t * *stcd,* edma_minorloop_offset_config_t * *config*) [inline], [static]

Configures both the enable bits and the offset value. If neither source nor destination offset is enabled, offset can't be configured.

Parameters

STCD	memory pointing to the eDMA software TCD
config	Configuration data structure for the minorloop offset

7.1.3.85 static void edma_hal_stcd_configure_source_last_adjustment (edma_software_tcd_t * *stcd,* int32_t *size*) [inline], [static]

Adjustment value added to the source address at the completion of the major iteration count. This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.

Parameters

STCD	memory pointing to the eDMA software TCD
size	adjustment value

7.1.3.86 static void edma_hal_stcd_configure_dest_address (edma_software_tcd_t * *stcd,* uint32_t *address*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
address	memory address pointing to destination data

7.1.3.87 static void edma_hal_stcd_configure_dest_offset (edma_software_tcd_t * *stcd,* uint32_t *offset*) [inline], [static]

Sign-extended offset applied to the current source address to form the next-state value as each destination write is complete.

Parameters

STCD	memory pointing to the eDMA software TCD
offset	signed-offset

7.1.3.88 static void edma_hal_stcd_configure_dest_last_adjustment_or_scatter_address (edma_software_tcd_t * *stcd,* uint32_t *address*) [inline], [static]

If the scatter/gather feature is enabled(edma_hal_htcd_set_scatter_gather_process()):

This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. The channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32-byte. Otherwise, a configuration error is reported.

else:

Adjustment value added to the source address at the completion of the major iteration count. This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.

Parameters

STCD	memory pointing to the eDMA software TCD
size	adjustment value

7.1.3.89 static void edma_hal_stcd_configure_bandwidth (edma_software_tcd_t * *stcd,* edma_bandwidth_configuration_t *bandwidth*) [inline], [static]

Throttles the amount of bus bandwidth consumed by the eDMA. As the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

STCD	memory pointing to the eDMA software TCD
bandwidth	enum type for bandwidth control

7.1.3.90 static void edma_hal_stcd_configure_majorlink_channel (edma_software_tcd_t * *stcd,* uint32_t *majorchannel*) [inline], [static]

If majorlink is enabled, after the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by these six bits by setting that channel start bits.

Parameters

STCD	memory pointing to the eDMA software TCD
majorchannel	channel number for major link

7.1.3.91 static void edma_hal_stcd_set_majorlink (edma_software_tcd_t * *stcd,* bool *isEnabled*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
isEnabled	Enable/Disable

7.1.3.92 static void edma_hal_stcd_set_scatter_gather_process (edma_software_tcd_t * *stcd,* bool *isEnabled*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
isEnabled	Enable/Disable

7.1.3.93 static void edma_hal_stcd_set_disable_dma_request_after_tcd_done (edma_software_tcd_t * *stcd,* bool *isDisabled*) [inline], [static]

If disabled, the eDMA hardware automatically clears the corresponding DMA request when the current major iteration count reaches zero.

Parameters

STCD	memory pointing to the eDMA software TCD.
isDisabled	Disable/Enable

7.1.3.94 static void edma_hal_stcd_set_half_complete_interrupt (edma_software_tcd_t * *stcd,* bool *isEnabled*) [inline], [static]

If set, the channel generates an interrupt request by setting the appropriate bit in the interrupt register when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER >> 1)). This half way point interrupt request is provided to

support the double-buffered schemes or other types of data movement where the processor needs an early indication of the transfer process.

Parameters

STCD	memory pointing to the eDMA software TCD
isEnabled	Enable/Disable

7.1.3.95 static void edma_hal_stcd_set_complete_interrupt (edma_software_tcd_t * *stcd,* bool *isEnabled*) [inline], [static]

If enabled, the channel generates an interrupt request by setting the appropriate bit in the interrupt register when the current major iteration count reaches zero.

Parameters

STCD	memory pointing to the eDMA software TCD
isEnabled	Enable/Disable

7.1.3.96 static void edma_hal_stcd_set_minor_link (edma_software_tcd_t * *stcd,* bool *isEnabled*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
isEnabled	Enable/Disable

7.1.3.97 static void edma_hal_stcd_set_current_minor_link (edma_software_tcd_t * *stcd,* bool *isEnabled*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
isEnabled	Enable/Disable

7.1.3.98 static void edma_hal_stcd_configure_minor_link_channel (edma_software_tcd_t * *stcd,* uint32_t *minorchannel*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
minorchannel	minor loop link channel

7.1.3.99 static void edma_hal_stcd_configure_current_minor_link_channel (edma_software_tcd_t * *stcd,* uint32_t *minorchannel*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
minorchannel	minor loop link channel.

7.1.3.100 static void edma_hal_stcd_configure_majorcount_minorlink_disabled (edma_software_tcd_t * *stcd,* uint32_t *count*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
count	major loop count

7.1.3.101 static void edma_hal_stcd_configure_current_majorcount_minorlink_disabled (edma_software_tcd_t * *stcd,* uint32_t *count*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
count	major loop count

7.1.3.102 static void edma_hal_stcd_configure_majorcount_minorlink_enabled (edma_software_tcd_t * *stcd,* uint32_t *count*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
count	major loop count

7.1.3.103 static void edma_hal_stcd_configure_current_majorcount_minorlink_enabled (edma_software_tcd_t * *stcd,* uint32_t *count*) [inline], [static]

Parameters

STCD	memory pointing to the eDMA software TCD
count	major loop count

7.1.3.104 void edma_hal_stcd_push_to_htcd (uint32_t *instance,* uint32_t *channel,* edma_software_tcd_t * *stcd*)

Parameters

STCD	memory pointing to the eDMA software TCD
instance	eDMA module
channel	eDMA channel
STCD	memory pointing to the software TCD

7.2 eDMA Peripheral Driver

The chapter describes the programming interface of the eDMA Peripheral driver.

Data Structures

- struct edma_scatter_list_t
  Data structure for configuring a discrete memory transfer. More...
- struct edma_channel_t
  Data structure for the DMA channel. More...

Typedefs

- typedef void(∗ edma_callback_t)(void ∗parameter, edma_channel_status_t status)
  Definition for the DMA channel callback function.

Enumerations

- enum edma_descriptor_internal_status_t {
  kEdmaDescriptorDone = 0U,
  kEdmaDescriptorPrepared = 0xFFFFFFFFU }
  Status of the DMA hardware descriptor.
- enum edma_tcd_alignment_t {
  kEdmaTcdAlignment = 0x20U,
  kEdmaTcdAlignmentMask = 0x1FU }
  Alignment for eDMA TCD start address.
- enum edma_channel_type_t {
  kEdmaInvalidChannel = 0xFFU,
  kEdmaAnyChannel = 0xFEU }
  A constant for the DMA allocation status.
- enum edma_channel_status_t {
  kEdmaIdle,
  kEdmaNormal,
  kEdmaError }
  Channel status for an eDMA channel.
- enum edma_transfer_type_t {
  kEdmaPeripheralToMemory,
  kEdmaMemoryToPeripheral,
  kEdmaMemoryToMemory,
  kEdmaPeripheralToPeripheral }
  A type for the DMA transfer.

eDMA Peripheral Driver

- edma_status_t edma_init (void)
  Initializes eDMA.
- edma_status_t edma_shutdown (void)
  De-initilizes eDMA.
- void edma_register_callback (edma_channel_t *chn, edma_callback_t callback, void *para)
  Registers the callback function and a parameter.
- edma_status_t edma_get_descriptor_status (edma_channel_t *chn, uint32_t *descriptorStatus)
  Gets the status of the eDMA channel descriptor chain.
- uint32_t edma_request_channel (uint32_t channel, dma_request_source_t source, edma_channel_t *chan)
  Requests an eDMA channel.
- edma_status_t edma_free_channel (edma_channel_t *chn)
  Frees eDMA channel hardware and software resource.
- edma_status_t edma_start_channel (edma_channel_t *chn)
  Starts an eDMA channel.
- edma_status_t edma_stop_channel (edma_channel_t *chn)
  Stops an eDMA channel.
- edma_status_t edma_update_descriptor (edma_channel_t *chn)
  Updates the status of a particular descriptor in the eDMA.
- edma_status_t edma_config_loop (edma_software_tcd_t *stcd, edma_channel_t *chn, edma_transfer_type_t type, uint32_t srcAddr, uint32_t destAddr, uint32_t size, uint32_t watermark, uint32_t length, uint8_t period)
  Configures the DMA transfer in a scatter-gather mode.
- edma_status_t edma_config_scatter_gather (edma_software_tcd_t *stcd, edma_channel_t *chn, edma_transfer_type_t type, uint32_t size, uint32_t watermark, edma_scatter_list_t *srcScatterList, edma_scatter_list_t *destScatterList, uint8_t number)
  Configures the DMA transfer in scatter-gather mode.
- void EDMA_IRQ_HANDLER (uint32_t channel)
  eDMA IRQ Handler
- void DMA_ERR_IRQHandler (uint32_t instance)
  eDMA ERROR IRQ Handler

7.2.0.105 EDMA PERIPHERAL Driver

Overview

The eDMA driver requests, configures, and uses eDMA hardware. It supports module initializations and DMA channel configurations.

Initialization

To initialize the DMA module, call the edma_init() function. The user does not need to pass a configuration data structure. This function enables the eDMA module and clock automatically.

Channel concept

The eDMA module has many channels. Additionally, the EDMA peripheral driver is designed based on a channel concept. All operations should be started by requesting an eDMA channel and ended by freeing an eDMA channel. The user can configure and run operations on the eDMA module by allocating a channel. If a channel is not allocated, a system error may occur.

DMA request concept

A DMA request is used to trigger an eDMA transfer. The DMA request table is available in chip configuration chapters in each chip reference manual.

Memory allocation and alignment

The eDMA peripheral driver does not allocate any memory dynamically. The user needs to provide the allocated memory pointer for the driver and ensure that the memory is valid. If this is not done, a system error may occur. The user needs to prepare three types of memory:

1. The handler memory: [edma_channel_t]. The driver must store the status data for each channel and the edma_channel_t is designed for this purpose.
2. The edma_software_tcd_t. The eDMA supports a TCD chain, which provides either the scatter-gather feature or the loop feature. The eDMA module loads the TCDCs from memory, where TCDs are stored. The user must provide the memory storing software TCDs and the edma_config_scatter_gather() function or the edma_config_loop() function configures the software TCDs.
3. The status memory. If the user wants to know the status of TCD chains, the user needs to provide the status memory and the driver will fill it with the chain status.
   The start address of the software TCDs must be 32 bytes.

Call diagram

To use the DMA driver, follow these steps:

1. Initialize the DMA module: edma_init().
2. Request a DMA channel: edma_request_channel().
3. Configure the TCD:
   - Configure the TCD chain in a scatter-gather list. UART transmit/receive is the common case.
   - Configure the TCD chain in a loop way. Audio playback/Record is the common case.
4. Register callback function: edma_register_callback.
5. Start the DMA channel: edma_start_channel.
6. [OPTION] Stop the DMA channel: edma_stop_channel.
7. Free the DMA channel: edma_free_channel.

This is an example code to initialize and configure the driver by configuring the descriptor:

```
edma_init();

stcd = (edma_software_tcd_t *)(((uint32_t)status + 32) & ~0x1F);

/* Prepare the memory space. */
for ( i = 0; i < kEdmaTestChainLength; i++)
{
    /* Allocate the memory! */
    srcAddr[i] = malloc(kEdmaTestBufferSize);
    destAddr[i] = malloc(kEdmaTestBufferSize);

    /* Check whether the allocation is successfully. */
    if (((uint32_t)srcAddr[i] == 0x0U) & ((uint32_t)destAddr[i] == 0x0U))
    {
        printf("Fali to allocate memory for EDMA test! \r\n");
        goto error;
    }

    /* Init the memory buffer. */
    for (j = 0; j < kEdmaTestBufferSize; j++)
    {
        srcAddr[i][j] = j;
        destAddr[i][j] = 0;
    }

    srcSG[i].address = (uint32_t)srcAddr[i];
    destSG[i].address = (uint32_t)destAddr[i];
    srcSG[i].length = kEdmaTestBufferSize;
    destSG[i].length = kEdmaTestBufferSize;

}

if (edma_request_channel(channel, kDmaRequestMux0AlwaysOn62, &chan_handler) != channel)
{
    printf("Failed to request channel %d !\r\n", channel);
    goto error;
}

edma_config_scatter_gather(
        stcd, &chan_handler, kDmaMemoryToMemory,
        0x1U, kEdmaTestWatermarkLevel,
        srcSG, destSG,
        kEdmaTestChainLength);

edma_register_callback(&chan_handler, test_callback, &chan_handler);

edma_start_channel(&chan_handler);
```

For an example to configure the loop mode, see the SAI module driver.

Extend the driver

The user can call the eDMA HAL driver to extend the application capability.

7.2.1 Data Structure Documentation

7.2.1.1 struct edma_scatter_list_t

Data Fields

- uint32_t address
  Address of buffer.
- uint32_t length
  Length of buffer.

7.2.1.1.0.13 Field Documentation

7.2.1.1.0.13.1 uint32_t edma_scatter_list_t::address

7.2.1.1.0.13.2 uint32_t edma_scatter_list_t::length

7.2.1.2 struct edma_channel_t

Data Fields

- uint8_t module
  eDMA physical module indicator
- uint8_t channel
  eDMA physical channel indicator
- uint8_t dmamuxModule
  DMA Mux module indicator.
- uint8_t dmamuxChannel
  DMA Mux channel indicator.
- edma_callback_t callback
  Callback for the eDMA channel.
- void * parameter
  Parameter for the callback function.
- volatile edma_channel_status_t status
  Channel status.
- uint8_t tcdNumber
  Length of the hardware descriptor chain.
- volatile uint8_t tcdWrite
  Indicator for updates of the user descriptor.
- volatile uint8_t tcdRead
  Indicator for the consuming of the DMA controller.
- uint32_t tcdLeftBytes
  Left bytes to be transferred for a current TCD.
- volatile bool tcdUnderflow
  `Flag indicating whether the user failed to feed a descriptor in time.`

7.2.1.2.0.14 Field Documentation

7.2.1.2.0.14.1 uint8_t edma_channel_t::dmamuxModule

7.2.1.2.0.14.2 uint8_t edma_channel_t::dmamuxChannel

7.2.1.2.0.14.3 edma_callback_t edma_channel_t::callback

7.2.1.2.0.14.4 void* edma_channel_t::parameter

7.2.1.2.0.14.5 volatile edma_channel_status_t edma_channel_t::status

7.2.1.2.0.14.6 uint8_t edma_channel_t::tcdNumber

7.2.1.2.0.14.7 volatile uint8_t edma_channel_t::tcdWrite

7.2.1.2.0.14.8 volatile uint8_t edma_channel_t::tcdRead

7.2.1.2.0.14.9 uint32_t edma_channel_t::tcdLeftBytes

7.2.1.2.0.14.10 volatile bool edma_channel_t::tcdUnderflow

7.2.2 Typedef Documentation

7.2.2.1 typedef void(* edma_callback_t)(void *parameter, edma_channel_status_t status)

Prototype for the callback function registered in the DMA driver.

7.2.3 Enumeration Type Documentation

7.2.3.1 enum edma_descriptor_internal_status_t

Defines the status of the DMA descriptor. The status of the descriptor is not limited to these two states. It can also be any the value between 0~0xFFFFFFFF for left bytes of a specified descriptor.

Enumerator

kEdmaDescriptorDone The descriptor is finished.
kEdmaDescriptorPrepared The descriptor is either consumed or not consumed.

7.2.3.2 enum edma_tcd_alignment_t

Enumerator

kEdmaTcdAlignment Alignment of the eDMA TCD.
kEdmaTcdAlignmentMask Alignment mask of the eDMA TCD.

7.2.3.3 enum edma_channel_type_t

A structure used for the dma_request_channel.

Enumerator

kEdmaInvalidChannel Macros indicate the failure of the channel request.
kEdmaAnyChannel Macros used when requesting channel. kEdmaAnyChannel means a request of a dynamic channel allocation.

7.2.3.4 enum edma_channel_status_t

A structure describing the DMA channel status. The user can get the status from the channel callback function.

Enumerator

kEdmaIdle DMA channel is idle.
kEdmaNormal DMA channel is occupied.
kEdmaError An error occurs in the DMA channel.

7.2.3.5 enum edma_transfer_type_t

Enumerator

kEdmaPeripheralToMemory Transfer from peripheral to memory.
kEdmaMemoryToPeripheral Transfer from memory to peripheral.
kEdmaMemoryToMemory Transfer from memory to memory.
kEdmaPeripheralToPeripheral Transfer from peripheral to peripheral.

7.2.4 Function Documentation

7.2.4.1 void edma_register_callback (edma_channel_t * *chn*, edma_callback_t *callback*, void * *para*)

The user register callback function and parameter for a specified eDMA channel. When channel interrupt or channel error occurs, the callback function is called and a parameter along with user parameter is provided to indicate the channel status.

Parameters

chn	Handler for eDMA channel.
callback	Callback function.
para	A parameter for callback functions.

7.2.4.2 edma_status_t edma_get_descriptor_status (edma_channel_t * *chn*, uint32_t * *descriptorStatus*)

This function indicates the status of descriptor chains. The user needs to provide the memory space to store the descriptor status. A parameter descriptorStatus is the pointer for the descriptorStatus. If the descriptorStaus can't point to a valid memory space and the length of the memory is not enough to store the descriptor status, an error occurs inside the DMA driver. Every descriptor needs a uint32_t to store the status. If the return descriptorStatus[n] is equal to the kEdmaDescriptorPrepared, the descriptor is either consuming or it is already prepared but not consumed. Any other value indicates the left bytes to be transferred for this descriptor. If the value is equal to 0, it means the descriptor is already finished. The user can update the memory for this descriptor safely. To get a precise status of the descriptor, the user can first stop the channel. The ongoing descriptor is updated with a value indicating bytes to be transferred for this descriptor. If not, it indicates that this descriptor is ongoing with a value of kEdmaDescriptorPrepared.

Parameters

chn	Handler for eDMA channel.
descriptor-Status	Status of descriptors.

7.2.4.3 uint32_t edma_request_channel (uint32_t *channel*, dma_request_source_t *source*, edma_channel_t * *chan*)

This function provides two ways to allocate a DMA channel: static allocation and dynamic allocation. To allocate a channel dynamically, the user should set the channel parameter with the value of kDmaAny-Channel. The driver searches all available channels and assigns the first channel to the user. To allocate the channel statically, the user should set the channel parameter with the value of a specified channel. If the channel is available, the driver assigns the channel to the user. Note that the user must provide the handler memory for the DMA channel. The driver initializes the handler and configures the handler memory.

Parameters

channel	eDMA channel number. If a channel is assigned with a valid channel number, the DMA driver tries to assign a specified channel to the user. If a channel is assigned with a kDmaAnyChannel, the DMA driver searches all available channels and assigns the first channel to the user.
source	eDMA hardware request.
chan	Memory pointing to eDMA channel. The user must ensure that the handler memory is valid and that it will not be released or changed by another code before the channel is freed.

Returns

If the channel allocation is successful, the return value indicates the requested channel. If not, the driver returns a kDmaInvalidChannel value to indicate that the requested operation has failed.

7.2.4.4 edma_status_t edma_free_channel (edma_channel_t * *chn*)

This function frees the relevant software and hardware resources. The request and the freeing operation need to be called in a pair.

Parameters

chn	Memory pointing to the eDMA channel

7.2.4.5 edma_status_t edma_start_channel (edma_channel_t * *chn*)

Starts an eDMA channel. The driver starts an eDMA channel by enabling the DMA request. The software start bit is not used in the eDMA peripheral driver.

Parameters

chn	Memory pointing to the eDMA channel.

7.2.4.6 edma_status_t edma_stop_channel (edma_channel_t * *chn*)

This function stops an eDMA channel and updates the descriptor chain status. By calling the edma_get_descriptor_status() function, the ongoing left bytes of the descriptor are updated and the user can simultaneously get the status of all descriptors.

Parameters

chn	Memory pointing to the eDMA channel

7.2.4.7 edma_status_t edma_update_descriptor (edma_channel_t * *chn*)

This function is designed for the loop descriptor chain. When descriptor is chained in loop mode, the DMA constantly consumes descriptors. At the same time, the user may need to update the content belonging to a consumed descriptor. This function is used to update the descriptor state from "CONSUMED" to "TO BE CONSUMED". In this case, the DMA driver can work out whether the underflow happens on the loop descriptor chain. This function can only update descriptors one-by-one in a sequence but not a specified descriptor.

Parameters

chn	Memory pointing to the eDMA channel

7.2.4.8 edma_status_t edma_config_loop (edma_software_tcd_t * *stcd,* edma_channel_t * *chn,* edma_transfer_type_t *type,* uint32_t *srcAddr,* uint32_t *destAddr,* uint32_t *size,* uint32_t *watermark,* uint32_t *length,* uint8_t *period*)

This function configures descriptors in a loop chain. The user passes a block of memory into this function. The memory is divided into "period" sub blocks. The DMA driver configures "period" descriptors. Each descriptor stands for a sub block. The DMA driver transfers data from the 1st descriptor to the last descriptor. Then, the DMA driver wraps to the first descriptor to continue the loop. The interrupt handler is called on every finish of a descriptor. The user can find out whether a descriptor is in the process of being transferred, is already transferred, or to be transferred by calling the edma_get_descriptor_status() function in the interrupt handler or any other task context. At the same time, the user can call the edma_update_descriptor() function to tell the DMA driver that the content belonging to a descriptor is already updated and the DMA needs to count it as and underflow next time it loops to this descriptor.

Parameters

chn	Memory pointing to the eDMA channel
stcd	Memory pointing to software TCDs. The user must prepare this memory block. The required memory size is equal to a "period" * size of(edma_software_tcd_t). At the same time, the "stcd" must align with 32 bytes. If not, an error occurs in the eDMA driver.

srcAddr	A source register address or a start memory address.
destAddr	A destination register address or a start memory address.
size	Size to be transferred on every DMA write/read. Source/Dest share the same write/read size.
watermark	size write/read for every trigger of the DMA request.
length	Total length of Memory. A number of the descriptor that is configured for this transfer configuration.

7.2.4.9 edma_status_t edma_config_scatter_gather (edma_software_tcd_t * *stcd*, edma_channel_t * *chn*, edma_transfer_type_t *type*, uint32_t *size*, uint32_t *watermark*, edma_scatter_list_t * *srcScatterList*, edma_scatter_list_t * *destScatterList*, uint8_t *number*)

This function configure descriptors into sing-end chain. User passed blocks of memory into this function. Interrupt will only be triggered on the last memory block is finished. The information of memory blocks are passed by edma_scatter_list_t data structure which can tell the memory address and length. DMA driver will configure descriptor for every memory block and transfer descriptor from the first one to the last one and then stop.

Parameters

chn	Memory pointing to eDMA channel.
stcd	Memory pointing to software TCDs. The user must prepare this memory block. The required memory size is equal to the "number" * size of(edma_software_tcd_t). At the same time, the "stcd" must align with 32 bytes. If not, an error occurs in the eDMA driver.
type	Transfer type.
srcScatterList	Data structure storing the address and length to be transferred for source memory blocks. If source memory is peripheral, the length is not used.
destScatterList	Data structure storing the address and length to be transferred for dest memory blocks. If in the memory-to-memory transfer mode, the user must ensure that the length of the dest scatter gather list is equal to the source scatter gather list. If the dest memory is a peripheral register, the length is not used.

size	Size to be transferred on each DMA write/read. Source/Dest share the same write/read size.
watermark	Size write/read for each trigger of the DMA request. number A number of memory block contained in the scatter gather list.

7.2.4.10 void EDMA_IRQ_HANDLER (uint32_t *channel*)

7.2.4.11 void DMA_ERR_IRQHandler (uint32_t *instance*)

7.3 eDMA request

The chapter describes the programming interface of the eDMA DMA request resource.

Chapter 8
Ethernet MAC (ENET)

The Kinetis SDK provides both HAL and Peripheral drivers for the Ethernet (ENET) block of Kinetis devices.

Modules

- ENET HAL driver
  The part describes the programming interface of the ENET HAL driver.
- ENET Peripheral Driver
  The part describes the programming interface of the ENET Peripheral Driver.
- ENET Physical Layer Driver
  The part describes the programming interface of the ENET Physical Layer Driver.
- ENET RTCS Adaptor
  The part describes the programming interface of the ENET RTCS Adaptor.

8.1 ENET HAL driver

The chapter describes the programming interface of the ENET HAL driver.

Data Structures

- struct enet_bd_struct_t
  Defines the buffer descriptor structure for the little-Endian system and endianness configurable IP. More...
- struct enet_config_ptp_timer_t
  Defines the configuration structure for the 1588 PTP timer. More...
- struct enet_config_tx_accelerator_t
  Defines the transmit accelerator configuration. More...
- struct enet_config_rx_accelerator_t
  Defines the receive accelerator configuration. More...
- struct enet_config_tx_fifo_t
  Defines the transmit FIFO configuration. More...
- struct enet_config_rx_fifo_t
  Defines the receive FIFO configuration. More...

Macros

- #define SYSTEM_LITTLE_ENDIAN (1)
  Defines the system endian type.
- #define BSWAP_16(x) (uint16_t)((uint16_t)(((uint16_t)(x) & (uint16_t)0xFF00) >> 0x8) | (uint16_t)(((uint16_t)(x) & (uint16_t)0xFF) << 0x8))
  Define macro to do the endianness swap.

Typedefs

- typedef uint8_t enetMacAddr [kEnetMacAddrLen]
  Defines the six-byte Mac address type.

Enumerations

- enum enet_status_t { ,
  kStatus_ENET_InvalidInput,
  kStatus_ENET_InvalidDevice,
  kStatus_ENET_MemoryAllocateFail,
  kStatus_ENET_GetClockFreqFail,
  kStatus_ENET_Initialized,
  kStatus_ENET_Open,
  kStatus_ENET_Close,
  kStatus_ENET_Layer2QueueNull,
  kStatus_ENET_Layer2OverLarge,
  kStatus_ENET_Layer2BufferFull,
  kStatus_ENET_Layer2TypeError,
  kStatus_ENET_PtpringBufferFull,
  kStatus_ENET_PtpringBufferEmpty,
  kStatus_ENET_Miiuninitialized,
  kStatus_ENET_RxbdInvalid,
  kStatus_ENET_RxbdEmpty,
  kStatus_ENET_RxbdTrunc,
  kStatus_ENET_RxbdError,
  kStatus_ENET_RxBdFull,
  kStatus_ENET_SmallBdSize,
  kStatus_ENET_LargeBufferFull,
  kStatus_ENET_TxLarge,
  kStatus_ENET_TxbdFull,
  kStatus_ENET_TxbdNull,
  kStatus_ENET_TxBufferNull,
  kStatus_ENET_NoRxBufferLeft,
  kStatus_ENET_UnknownCommand,
  kStatus_ENET_TimeOut,
  kStatus_ENET_MulticastPointerNull,
  kStatus_ENET_NoMulticastAddr,
  kStatus_ENET_AlreadyAddedMulticast }
  Defines the Status return codes.
- enum enet_rx_bd_control_status_t {

kEnetRxBdEmpty = 0x8000,
kEnetRxBdRxSoftOwner1 = 0x4000,
kEnetRxBdWrap = 0x2000,
kEnetRxBdRxSoftOwner2 = 0x1000,
kEnetRxBdLast = 0x0800,
kEnetRxBdMiss = 0x0100,
kEnetRxBdBroadCast = 0x0080,
kEnetRxBdMultiCast = 0x0040,
kEnetRxBdLengthViolation = 0x0020,
kEnetRxBdNoOctet = 0x0010,
kEnetRxBdCrc = 0x0004,
kEnetRxBdOverRun = 0x0002,
kEnetRxBdTrunc = 0x0001 }

Defines the control and status region of the receive buffer descriptor.

- enum enet_rx_bd_control_extend_t {
kEnetRxBdIpv4 = 0x0001,
kEnetRxBdIpv6 = 0x0002,
kEnetRxBdVlan = 0x0004,
kEnetRxBdProtocolChecksumErr = 0x0010,
kEnetRxBdIpHeaderChecksumErr = 0x0020,
kEnetRxBdIntrrupt = 0x0080,
kEnetRxBdUnicast = 0x0100,
kEnetRxBdCollision = 0x0200,
kEnetRxBdPhyErr = 0x0400,
kEnetRxBdMacErr = 0x8000 }

  Defines the control extended region of the receive buffer descriptor.
- enum enet_tx_bd_control_status_t {
kEnetTxBdReady = 0x8000,
kEnetTxBdTxSoftOwner1 = 0x4000,
kEnetTxBdWrap = 0x2000,
kEnetTxBdTxSoftOwner2 = 0x1000,
kEnetTxBdLast = 0x0800,
kEnetTxBdTransmitCrc = 0x0400 }

  Defines the control status of the transmit buffer descriptor.
- enum enet_tx_bd_control_extend_t {
kEnetTxBdTxErr = 0x8000,
kEnetTxBdTxUnderFlowErr = 0x2000,
kEnetTxBdExcessCollisionErr = 0x1000,
kEnetTxBdTxFrameErr = 0x0800,
kEnetTxBdLatecollisionErr = 0x0400,
kEnetTxBdOverFlowErr = 0x0200,
kEnetTxTimestampErr = 0x0100 }

  Defines the control extended of the transmit buffer descriptor.
- enum enet_tx_bd_control_extend2_t {
kEnetTxBdTxInterrupt = 0x4000,
kEnetTxBdTimeStamp = 0x2000 }

Defines the control extended2 of the transmit buffer descriptor.

- enum enet_constant_parameter_t {
  kEnetMacAddrLen = 6,
  kEnetHashValMask = 0x1f,
  kEnetRxBdCtlJudge1 = 0x0080,
  kEnetRxBdCtlJudge2 = 0x8000 }
  Defines the macro to the different ENET constant value.
- enum enet_config_rmii_t {
  kEnetCfgMii = 0,
  kEnetCfgRmii = 1 }
  Defines the RMII or MII mode for data interface between the MAC and the PHY.
- enum enet_config_speed_t {
  kEnetCfgSpeed100M = 0,
  kEnetCfgSpeed10M = 1 }
  Defines the 10 Mbps or 100 Mbps speed mode for the data transfer.
- enum enet_config_duplex_t {
  kEnetCfgHalfDuplex = 0,
  kEnetCfgFullDuplex = 1 }
  Defines the half or full duplex mode for the data transfer.
- enum enet_mii_operation_t {
  kEnetWriteNoCompliant = 0,
  kEnetWriteValidFrame = 1,
  kEnetReadValidFrame = 2,
  kEnetReadNoCompliant = 3 }
  Defines the write/read operation for the MII.
- enum enet_mdio_holdon_clkcycle_t {
  kEnetMdioHoldOneClkCycle = 0,
  kEnetMdioHoldTwoClkCycle = 1,
  kEnetMdioHoldThreeClkCycle = 2,
  kEnetMdioHoldFourClkCycle = 3,
  kEnetMdioHoldFiveClkCycle = 4,
  kEnetMdioHoldSixClkCycle = 5,
  kEnetMdioHoldSevenClkCycle = 6,
  kEnetMdioHoldEightClkCycle = 7 }
  Define holdon time on MDIO output.
- enum enet_special_address_filter_t {
  kEnetSpecialAddressInit = 0,
  kEnetSpecialAddressEnable = 1,
  kEnetSpecialAddressDisable = 2 }
  Defines the initialization, enables or disables the operation for a special address filter.
- enum enet_timer_channel_mode_t {

kEnetChannelDisable = 0,
kEnetChannelRisingCapture = 1,
kEnetChannelFallingCapture = 2,
kEnetChannelBothCapture = 3,
kEnetChannelSoftCompare = 4,
kEnetChannelToggleCompare = 5,
kEnetChannelClearCompare = 6,
kEnetChannelSetCompare = 7,
kEnetChannelClearCompareSetOverflow = 10,
kEnetChannelSetCompareClearOverflow = 11,
kEnetChannelPulseLowonCompare = 14,
kEnetChannelPulseHighonCompare = 15 }

Defines the capture or compare mode for 1588 timer channels.

- enum enet_interrupt_request_t {
kEnetBabrInterrupt = 0x40000000,
kEnetBabtInterrupt = 0x20000000,
kEnetGraInterrupt = 0x10000000,
kEnetTxFrameInterrupt = 0x8000000,
kEnetTxByteInterrupt = 0x4000000,
kEnetRxFrameInterrupt = 0x2000000,
kEnetRxByteInterrupt = 0x1000000,
kEnetMiiInterrupt = 0x0800000,
kEnetEBERInterrupt = 0x0400000,
kEnetLcInterrupt = 0x0200000,
kEnetRlInterrupt = 0x0100000,
kEnetUnInterrupt = 0x0080000,
kEnetPlrInterrupt = 0x0040000,
kEnetWakeupInterrupt = 0x0020000,
kEnetTsAvailInterrupt = 0x0010000,
kEnetTsTimerInterrupt = 0x0008000,
kEnetAllInterrupt = 0xFFFFFFFF }

  Defines the RXFRAME/RXBYTE/TXFRAME/TXBYTE/MII/TSTIMER/TSAVAIL interrupt source for ENET.

Functions

- static void enet_hal_reset_ethernet (uint32_t instance)

  Resets the ENET module.
- static bool enet_hal_is_reset_completed (uint32_t instance)

  Gets the ENET status to check whether the reset has completed.
- static void enet_hal_enable_stop (uint32_t instance, bool isEnabled)

  Enable or disable stop mode.
- static void enet_hal_enable_sleep (uint32_t instance, bool isEnabled)

  Enable or disable sleep mode.
- void enet_hal_set_mac_address (uint32_t instance, enetMacAddr hwAddr)

Sets the Mac address.

- void enet_hal_set_group_hashtable (uint32_t instance, uint32_t crcValue, enet_special_address_filter_t mode)
  Sets the hardware addressing filtering to a multicast group address.
- void enet_hal_set_individual_hashtable (uint32_t instance, uint32_t crcValue, enet_special_address_filter_t mode)
  Sets the hardware addressing filtering to an individual address.
- static void enet_hal_enable_payloadcheck (uint32_t instance, bool isEnabled)
  Enable/disable payload length check.
- static void enet_hal_enable_txcrcforward (uint32_t instance, bool isEnabled)
  Enable/disable append CRC to transmitted frames.
- static void enet_hal_enable_rxcrcforward (uint32_t instance, bool isEnabled)
  Enable/disable forward the CRC filed of the received frame.
- static void enet_hal_enable_pauseforward (uint32_t instance, bool isEnabled)
  Enable/disable forward PAUSE frames.
- static void enet_hal_enable_padremove (uint32_t instance, bool isEnabled)
  Enable/disable frame padding remove on receive.
- static void enet_hal_enable_flowcontrol (uint32_t instance, bool isEnabled)
  Enable/disable flow control.
- static void enet_hal_enable_broadcastreject (uint32_t instance, bool isEnabled)
  Enable/disable broadcast frame reject.
- static void enet_hal_set_pauseduration (uint32_t instance, uint32_t pauseDuration)
  Sets PAUSE duration for a PAUSE frame.
- static bool enet_hal_get_rxpause_status (uint32_t instance)
  Gets receive PAUSE frame status.
- static void enet_hal_enable_txpause (uint32_t instance, bool isEnabled)
  Enables transmit frame control PAUSE.
- void enet_hal_set_txpause (uint32_t instance, uint32_t pauseDuration)
  Sets transmit PAUSE frame.
- static void enet_hal_set_txipg (uint32_t instance, uint32_t ipgValue)
  Sets the transmit inter-packet gap.
- static void enet_hal_set_truncationlen (uint32_t instance, uint32_t length)
  Sets the receive frame truncation length.
- static void enet_hal_set_rx_max_size (uint32_t instance, uint32_t maxBufferSize, uint32_t maxFrameSize)
  Sets the maximum receive buffer size and the maximum frame size.
- void enet_hal_config_tx_fifo (uint32_t instance, enet_config_tx_fifo_t *thresholdCfg)
  Configures the ENET transmit FIFO.
- void enet_hal_config_rx_fifo (uint32_t instance, enet_config_rx_fifo_t *thresholdCfg)
  Configures the ENET receive FIFO.
- static void enet_hal_set_rxbd_address (uint32_t instance, uint32_t rxBdAddr)
  Sets the start address for ENET receive buffer descriptors.
- static void enet_hal_set_txbd_address (uint32_t instance, uint32_t txBdAddr)
  Sets the start address for ENET transmit buffer descriptors.
- void enet_hal_init_rxbds (void *rxBds, uint8_t *buffer, bool isLastBd)
  Initializes the receive buffer descriptors.
- void enet_hal_init_txbds (void *txBds, bool isLastBd)
  Initializes the transmit buffer descriptors.
- void enet_hal_update_rxbds (void *rxBds, uint8_t *data, bool isbufferUpdate)
  Updates the receive buffer descriptors.
- void enet_hal_update_txbds (void *txBds, uint8_t *buffer, uint16_t length, bool isTxtsCfged)

Updates the transmit buffer descriptors.

- static void enet_hal_clear_txbds (void *curBd)
  Clears the context in the transmit buffer descriptors.
- uint16_t enet_hal_get_rxbd_control (void *curBd)
  Gets the control and the status region of the receive buffer descriptors.
- uint16_t enet_hal_get_txbd_control (void *curBd)
  Gets the control and the status region of the transmit buffer descriptors.
- bool enet_hal_get_rxbd_control_extend (void *curBd, enet_rx_bd_control_extend_t controlRegion)
  Gets the extended control region of the receive buffer descriptors.
- uint16_t enet_hal_get_txbd_control_extend (void *curBd)
  Gets the extended control region of the transmit buffer descriptors.
- uint16_t enet_hal_get_bd_length (void *curBd)
  Gets the data length of the buffer descriptors.
- uint8_t * enet_hal_get_bd_buffer (void *curBd)
  Gets the buffer address of the buffer descriptors.
- uint32_t enet_hal_get_bd_timestamp (void *curBd)
  Gets the timestamp of the buffer descriptors.
- static void enet_hal_active_rxbd (uint32_t instance)
  Activates the receive buffer descriptor.
- static void enet_hal_active_txbd (uint32_t instance)
  Activates the transmit buffer descriptor.
- void enet_hal_config_rmii (uint32_t instance, enet_config_rmii_t mode, enet_config_speed_t speed, enet_config_duplex_t duplex, bool isRxOnTxDisabled, bool isLoopEnabled)
  Configures the (R)MII of ENET.
- static void enet_hal_config_mii (uint32_t instance, uint32_t miiSpeed, enet_mdio_holdon_clkcycle_t clkCycle, bool isPreambleDisabled)
  Configures the MII of ENET.
- static bool enet_hal_is_mii_enabled (uint32_t instance)
  Gets the MII configuration status.
- static uint32_t enet_hal_get_mii_data (uint32_t instance)
  Reads data from PHY.
- void enet_hal_set_mii_command (uint32_t instance, uint32_t phyAddr, uint32_t phyReg, enet_mii_operation_t operation, uint32_t data)
  Sets the MII command.
- void enet_hal_config_ethernet (uint32_t instance, bool isEnhanced, bool isEnabled)
  Enables/Disables the ENET module.
- void enet_hal_config_interrupt (uint32_t instance, uint32_t source, bool isEnabled)
  Enables/Disables the ENET interrupt.
- static void enet_hal_clear_interrupt (uint32_t instance, uint32_t source)
  Clears ENET interrupt events.
- static bool enet_hal_get_interrupt_status (uint32_t instance, uint32_t source)
  Gets the ENET interrupt status.
- static void enet_hal_clear_mib (uint32_t instance, bool isEnabled)
  Enables/disables the clear MIB counter.
- static void enet_hal_enable_mib (uint32_t instance, bool isEnabled)
  Sets the enable/disable of the MIB block.
- static bool enet_hal_get_mib_status (uint32_t instance)
  Gets the MIB idle status.
- void enet_hal_config_tx_accelerator (uint32_t instance, enet_config_tx_accelerator_t *txCfgPtr)
  Sets the transmit accelerator.

- void enet_hal_config_rx_accelerator (uint32_t instance, enet_config_rx_accelerator_t *rxCfgPtr)
  Sets the receive accelerator.
- void enet_hal_init_ptp_timer (uint32_t instance, enet_config_ptp_timer_t *ptpCfgPtr)
  Initializes the 1588 timer.
- static void enet_hal_enable_ptp_timer (uint32_t instance, uint32_t isEnabled)
  Enables or disables the 1588 timer.
- static void enet_hal_restart_ptp_timer (uint32_t instance)
  Restarts the 1588 timer.
- static void enet_hal_adjust_ptp_timer (uint32_t instance, uint32_t increaseCorrection, uint32_t periodCorrection)
  Adjusts the 1588 timer.
- static void enet_hal_init_timer_channel (uint32_t instance, uint32_t channel, enet_timer_channel_mode_t mode)
  Initializes the 1588 timer channel.
- static void enet_hal_set_timer_channel_compare (uint32_t instance, uint32_t channel, uint32_t compareValue)
  Sets the compare value for the 1588 timer channel.
- static bool enet_hal_get_timer_channel_status (uint32_t instance, uint32_t channel)
  Gets the 1588 timer channel status.
- static void enet_hal_clear_timer_channel_flag (uint32_t instance, uint32_t channel)
  Clears the 1588 timer channel flag.
- static void enet_hal_set_timer_capture (uint32_t instance)
  Sets the capture command to the 1588 timer.
- static void enet_hal_set_current_time (uint32_t instance, uint32_t nanSecond)
  Sets the 1588 timer.
- static uint32_t enet_hal_get_current_time (uint32_t instance)
  Gets the time from the 1588 timer.
- static uint32_t enet_hal_get_tx_timestamp (uint32_t instance)
  Gets the transmit timestamp.
- bool enet_hal_get_txbd_timestamp_flag (void *curBd)
  Gets the transmit buffer descriptor timestamp flag.
- static uint32_t enet_hal_get_bd_size (void)
  Gets the buffer descriptor timestamp.

8.1.1 Data Structure Documentation

8.1.1.1 struct enet_bd_struct_t

Data Fields

- uint16_t length
  Buffer descriptor data length.
- uint16_t control
  Buffer descriptor control.
- uint8_t * buffer
  Data buffer pointer.
- uint16_t controlExtend0
  Extend buffer descriptor control0.
- uint16_t controlExtend1

Extend buffer descriptor control1.
- uint16_t payloadCheckSum
  Internal payload checksum.
- uint8_t headerLength
  Header length.
- uint8_t protocalTyte
  Protocol type.
- uint16_t controlExtend2
  Extend buffer descriptor control2.
- uint32_t timestamp
  Timestamp.

8.1.1.2 struct enet_config_ptp_timer_t

Data Fields

- bool isSlaveEnabled
  Master or slave PTP timer.
- uint32_t clockIncease
  Timer increase value each clock period.
- uint32_t period
  Timer period for generate interrupt event.

8.1.1.3 struct enet_config_tx_accelerator_t

Data Fields

- bool isIpCheckEnabled
  Insert IP header checksum.
- bool isProtocolCheckEnabled
  Insert protocol checksum.
- bool isShift16Enabled
  Tx FIFO shift-16.

8.1.1.4 struct enet_config_rx_accelerator_t

Data Fields

- bool isIpcheckEnabled
  Discard with wrong IP header checksum.
- bool isProtocolCheckEnabled
  Discard with wrong protocol checksum.
- bool isMacCheckEnabled
  Discard with Mac layer errors.
- bool isPadRemoveEnabled
  Padding removal for short IP frames.
- bool isShift16Enabled
  Rx FIFO shift-16.

8.1.1.5 struct enet_config_tx_fifo_t

Data Fields

- bool isStoreForwardEnabled
  Transmit FIFO store and forward.
- uint8_t txFifoWrite
  Transmit FIFO write.
- uint8_t txEmpty
  Transmit FIFO chapter empty threshold.
- uint8_t txAlmostEmpty
  Transmit FIFO chapter almost empty threshold.
- uint8_t txAlmostFull
  Transmit FIFO chapter almost full threshold.

8.1.1.6 struct enet_config_rx_fifo_t

Data Fields

- uint8_t rxFull
  Receive FIFO chapter full threshold.
- uint8_t rxAlmostFull
  Receive FIFO chapter almost full threshold.
- uint8_t rxEmpty
  Receive FIFO chapter empty threshold.
- uint8_t rxAlmostEmpty
  Receive FIFO chapter almost empty threshold.

8.1.2 Macro Definition Documentation

8.1.2.1 #define SYSTEM_LITTLE_ENDIAN (1)

8.1.3 Typedef Documentation

8.1.3.1 typedef uint8_t enetMacAddr[kEnetMacAddrLen]

8.1.4 Enumeration Type Documentation

8.1.4.1 enum enet_status_t

Enumerator

kStatus_ENET_InvalidInput Invalid ENET input parameter.
kStatus_ENET_InvalidDevice Invalid ENET device.
kStatus_ENET_MemoryAllocateFail Memory allocate failure.
kStatus_ENET_GetClockFreqFail Get clock frequency failure.

kStatus_ENET_Initialized ENET device already initialized.
kStatus_ENET_Open Open ENET device.
kStatus_ENET_Close Close ENET device.
kStatus_ENET_Layer2QueueNull NULL L2 PTP buffer queue pointer.
kStatus_ENET_Layer2OverLarge Layer2 packet length over large.
kStatus_ENET_Layer2BufferFull Layer2 packet buffer full.
kStatus_ENET_Layer2TypeError Layer2 packet error type.
kStatus_ENET_PtpringBufferFull PTP ring buffer full.
kStatus_ENET_PtpringBufferEmpty PTP ring buffer empty.
kStatus_ENET_Miiuninitialized MII uninitialized.
kStatus_ENET_RxbdInvalid Receive buffer descriptor invalid.
kStatus_ENET_RxbdEmpty Receive buffer descriptor empty.
kStatus_ENET_RxbdTrunc Receive buffer descriptor truncate.
kStatus_ENET_RxbdError Receive buffer descriptor error.
kStatus_ENET_RxBdFull Receive buffer descriptor full.
kStatus_ENET_SmallBdSize Small receive buffer size.
kStatus_ENET_LargeBufferFull Receive large buffer full.
kStatus_ENET_TxLarge Transmit large packet.
kStatus_ENET_TxbdFull Transmit buffer descriptor full.
kStatus_ENET_TxbdNull Transmit buffer descriptor Null.
kStatus_ENET_TxBufferNull Transmit data buffer Null.
kStatus_ENET_NoRxBufferLeft No more receive buffer left.
kStatus_ENET_UnknownCommand Invalid ENET PTP IOCTL command.
kStatus_ENET_TimeOut ENET Timeout.
kStatus_ENET_MulticastPointerNull Null multicast group pointer.
kStatus_ENET_NoMulticastAddr No multicast group address.
kStatus_ENET_AlreadyAddedMulticast Have Already added to multicast group.

8.1.4.2 enum enet_rx_bd_control_status_t

Enumerator

kEnetRxBdEmpty Empty bit.
kEnetRxBdRxSoftOwner1 Receive software owner.
kEnetRxBdWrap Update buffer descriptor.
kEnetRxBdRxSoftOwner2 Receive software owner.
kEnetRxBdLast Last BD in the frame.
kEnetRxBdMiss Receive for promiscuous mode.
kEnetRxBdBroadCast Broadcast.
kEnetRxBdMultiCast Multicast.
kEnetRxBdLengthViolation Receive length violation.
kEnetRxBdNoOctet Receive non-octet aligned frame.
kEnetRxBdCrc Receive CRC error.
kEnetRxBdOverRun Receive FIFO overrun.

kEnetRxBdTrunc Frame is truncated.

8.1.4.3 enum enet_rx_bd_control_extend_t

Enumerator

kEnetRxBdIpv4 Ipv4 frame.
kEnetRxBdIpv6 Ipv6 frame.
kEnetRxBdVlan VLAN.
kEnetRxBdProtocolChecksumErr Protocol checksum error.
kEnetRxBdIpHeaderChecksumErr IP header checksum error.
kEnetRxBdIntrrupt BD interrupt.
kEnetRxBdUnicast Unicast frame.
kEnetRxBdCollision BD collision.
kEnetRxBdPhyErr PHY error.
kEnetRxBdMacErr Mac error.

8.1.4.4 enum enet_tx_bd_control_status_t

Enumerator

kEnetTxBdReady Ready bit.
kEnetTxBdTxSoftOwner1 Transmit software owner.
kEnetTxBdWrap Wrap buffer descriptor.
kEnetTxBdTxSoftOwner2 Transmit software owner.
kEnetTxBdLast Last BD in the frame.
kEnetTxBdTransmitCrc Receive for transmit CRC.

8.1.4.5 enum enet_tx_bd_control_extend_t

Enumerator

kEnetTxBdTxErr Transmit error.
kEnetTxBdTxUnderFlowErr Underflow error.
kEnetTxBdExcessCollisionErr Excess collision error.
kEnetTxBdTxFrameErr Frame error.
kEnetTxBdLatecollisionErr Late collision error.
kEnetTxBdOverFlowErr Overflow error.
kEnetTxTimestampErr Timestamp error.

8.1.4.6 enum enet_tx_bd_control_extend2_t

Enumerator

kEnetTxBdTxInterrupt Transmit interrupt.
kEnetTxBdTimeStamp Transmit timestamp flag.

8.1.4.7 enum enet_constant_parameter_t

Enumerator

kEnetMacAddrLen ENET mac address length.
kEnetHashValMask ENET hash value mask.
kEnetRxBdCtlJudge1 ENET receive buffer descriptor control judge value1.
kEnetRxBdCtlJudge2 ENET receive buffer descriptor control judge value2.

8.1.4.8 enum enet_config_rmii_t

Enumerator

kEnetCfgMii MII mode for data interface.
kEnetCfgRmii RMII mode for data interface.

8.1.4.9 enum enet_config_speed_t

Enumerator

kEnetCfgSpeed100M Speed 100 M mode.
kEnetCfgSpeed10M Speed 10 M mode.

8.1.4.10 enum enet_config_duplex_t

Enumerator

kEnetCfgHalfDuplex Half duplex mode.
kEnetCfgFullDuplex Full duplex mode.

8.1.4.11 enum enet_mii_operation_t

Enumerator

kEnetWriteNoCompliant Write frame operation, but not MII compliant.

kEnetWriteValidFrame Write frame operation for a valid MII management frame.
kEnetReadValidFrame Read frame operation for a valid MII management frame.
kEnetReadNoCompliant Read frame operation, but not MII compliant.

8.1.4.12 enum enet_mdio_holdon_clkcycle_t

Enumerator

kEnetMdioHoldOneClkCycle MDIO output hold on one clock cycle.
kEnetMdioHoldTwoClkCycle MDIO output hold on two clock cycles.
kEnetMdioHoldThreeClkCycle MDIO output hold on three clock cycles.
kEnetMdioHoldFourClkCycle MDIO output hold on four clock cycles.
kEnetMdioHoldFiveClkCycle MDIO output hold on five clock cycles.
kEnetMdioHoldSixClkCycle MDIO output hold on six clock cycles.
kEnetMdioHoldSevenClkCycle MDIO output hold seven two clock cycles.
kEnetMdioHoldEightClkCycle MDIO output hold on eight clock cycles.

8.1.4.13 enum enet_special_address_filter_t

Enumerator

kEnetSpecialAddressInit Initializes the special address filter.
kEnetSpecialAddressEnable Enables the special address filter.
kEnetSpecialAddressDisable Disables the special address filter.

8.1.4.14 enum enet_timer_channel_mode_t

Enumerator

kEnetChannelDisable Disable timer channel.
kEnetChannelRisingCapture Input capture on rising edge.
kEnetChannelFallingCapture Input capture on falling edge.
kEnetChannelBothCapture Input capture on both edges.
kEnetChannelSoftCompare Output compare software only.
kEnetChannelToggleCompare Toggle output on compare.
kEnetChannelClearCompare Clear output on compare.
kEnetChannelSetCompare Set output on compare.
kEnetChannelClearCompareSetOverflow Clear output on compare, set output on overflow.
kEnetChannelSetCompareClearOverflow Set output on compare, clear output on overflow.
kEnetChannelPulseLowonCompare Pulse output low on compare for one 1588 clock cycle.
kEnetChannelPulseHighonCompare Pulse output high on compare for one 1588 clock cycle.

8.1.4.15 enum enet_interrupt_request_t

Enumerator

kEnetBabrInterrupt BABR interrupt source.
kEnetBabtInterrupt BABT interrupt source.
kEnetGraInterrupt GRA interrupt source.
kEnetTxFrameInterrupt TXFRAME interrupt source.
kEnetTxByteInterrupt TXBYTE interrupt source.
kEnetRxFrameInterrupt RXFRAME interrupt source.
kEnetRxByteInterrupt RXBYTE interrupt source.
kEnetMiiInterrupt MII interrupt source.
kEnetEBERInterrupt EBERR interrupt source.
kEnetLcInterrupt LC interrupt source.
kEnetRlInterrupt RL interrupt source.
kEnetUnInterrupt UN interrupt source.
kEnetPlrInterrupt PLR interrupt source.
kEnetWakeupInterrupt WAKEUP interrupt source.
kEnetTsAvailInterrupt TS AVAIL interrupt source.
kEnetTsTimerInterrupt TS WRAP interrupt source.
kEnetAllInterrupt All interrupt.

8.1.5 Function Documentation

8.1.5.1 static void enet_hal_reset_ethernet (uint32_t *instance*) [inline], [static]

Parameters

instance	The ENET instance number

8.1.5.2 static bool enet_hal_is_reset_completed (uint32_t *instance*) [inline], [static]

Parameters

instance	The ENET instance number

Returns

Current status of the reset operation
- true if ENET reset completed.
- false if ENET reset has not completed.

8.1.5.3 static void enet_hal_enable_stop (uint32_t *instance,* bool *isEnabled*) [inline], [static]

Enable stop mode will control device behavior in doze mode. In doze mode, if this filed is set then all clock of the enet assemably are disabled, except the RMII/MII clock.

Parameters

instance	The ENET instance number.
isEnabled	The switch to enable/disable stop mode. • true to enabale the stop mode. • false to disable the stop mode.

8.1.5.4 static void enet_hal_enable_sleep (uint32_t *instance,* bool *isEnabled*) [inline], [static]

Enable sleep mode will disable normal operating mode. When enable the sleep mode, the magic packet detection is also enabled so that a remote agent can wakeup the node.

Parameters

instance	The ENET instance number.
isEnabled	The switch to enable/disable the sleep mode. • true to enabale the sleep mode. • false to disable the sleep mode.

8.1.5.5 void enet_hal_set_mac_address (uint32_t *instance,* enetMacAddr *hwAddr*)

This interface sets the six-byte Mac address of the ENET interface.

Parameters

instance	The ENET instance number
hwAddr	The mac address pointer store for six bytes Mac address

8.1.5.6 void enet_hal_set_group_hashtable (uint32_t *instance,* uint32_t *crcValue,* enet_special_address_filter_t *mode*)

This interface is used to add the ENET device to a multicast group address. After joining the group, Mac receives all frames with the group Mac address.

Parameters

instance	The ENET instance number
crcValue	The CRC value of the special address
mode	The operation for init/enable/disable the specified hardware address

8.1.5.7 void enet_hal_set_individual_hashtable (uint32_t *instance,* uint32_t *crcValue,* enet_special_address_filter_t *mode*)

This interface is used to add an individual address to the hardware address filter. Mac receives all frames with the individual address as a destination address.

Parameters

instance	The ENET instance number
crcValue	The CRC value of the special address
mode	The operation for init/enable/disable the specified hardware address

8.1.5.8 static void enet_hal_enable_payloadcheck (uint32_t *instance,* bool *isEnabled*) [inline], [static]

If the length/type is less than 0x600,When enable payload length check the core checks the fame's payload length. If the length/type is greater than or equal to 0x600. The MAC interprets the field as a type and no payload length check is performanced.

Parameters

instance	The ENET instance number
isEnabled	The switch to enable/disable payload length check • True to enabale payload length check. • False to disable payload legnth check.

8.1.5.9 static void enet_hal_enable_txcrcforward (uint32_t *instance,* bool *isEnabled*) [inline], [static]

If transmit CRC forward is enabled, the transmit buffer descriptor controls whether the frame has a C-RC from the application. If transmit CRC forward is disabled, transmitter does not append any CRC to transmitted frames.

Parameters

instance	The ENET instance number
isEnabled	The switch to enable/disable transmit the receive CRC • True the transmitter control CRC through transmit buffer descriptor. • False the transmitter does not append any CRC to transmitted frames.

8.1.5.10 static void enet_hal_enable_rxcrcforward (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This is used to deceide whether the CRC field of received frame is transmitted or stripped. Enable this feature to strip CRC field from the frame. If padding remove is enabled, this feature will be ignored and the CRC field is checked and always terminated and removed.

Parameters

instance	The ENET instance number
isEnabled	The switch to enable/disable transmit the receive CRC • True to transmit the received CRC. • False to strip the received CRC.

8.1.5.11 static void enet_hal_enable_pauseforward (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This is used to deceide whether PAUSE frames is forwarded or discarded.

Parameters

instance	The ENET instance number
isEnabled	The switch to enable/disable forward PAUSE frames • True to forward PAUSE frames. • False to terminate and discard PAUSE frames.

8.1.5.12 static void enet_hal_enable_padremove (uint32_t *instance,* bool *isEnabled*) [inline], [static]

Enable frame padding remove will remove the padding from the received frames.

Parameters

instance	The ENET instance number
isEnabled	The switch to enable/disable remove padding • True to remove padding from frames. • False to disable padding remove.

8.1.5.13 static void enet_hal_enable_flowcontrol (uint32_t *instance,* bool *isEnabled*) [inline], [static]

If flow control is enabled, the receive detects PAUSE frames. Upon PAUSE frame detection, the transmitter stops transmitting data frames for a given duration.

Parameters

instance	The ENET instance number
isEnabled	The switch to enable/disable flow control • True to enable the flow control. • False to disable the flow control.

8.1.5.14 static void enet_hal_enable_broadcastreject (uint32_t *instance,* bool *isEnabled*) [inline], [static]

If broadcast frame reject is enabled, frames with destination address equal to 0xffff_ffff_ffff are rejected unless the promiscuous mode is open.

Parameters

instance	The ENET instance number
isEnabled	The switch to enable/disable reject broadcast frames • True to reject broadcast frames. • False to accept broadcast frames.

8.1.5.15 static void enet_hal_set_pauseduration (uint32_t *instance,* uint32_t *pauseDuration*) [inline], [static]

This function is used to set the pause duraion used in transmission of a PAUSE frame. When another node detects a PAUSE frame, that node pauses transmission for the pause duration.

Parameters

instance	The ENET instance number
pauseDuration	The PAUSE duration for the transmitted PAUSE frame the maximum pause duration is 0xFFFF.

8.1.5.16 static bool enet_hal_get_rxpause_status (uint32_t *instance*) [inline], [static]

This function is used to get the received PAUSE frame status.

Parameters

instance	The ENET instance number

Returns

The status of the received flow control frames true if the flow control pause frame is received. false if there is no flow control frame received or the pause duration is complete.

8.1.5.17 static void enet_hal_enable_txpause (uint32_t *instance,* bool *isEnabled*) [inline], [static]

This function enables pauses frame transmission. When this is set, with transmission of data frames stopped, the MAC transmits a MAC control PAUSE frame. NEXT, the MAC clear the and resumes transmitting data frames.

Parameters

instance	The ENET instance number
isEnabled	The switch to enable/disable PAUSE control frame transmission • True enable PAUSE control frame transmission. • Flase disable PAUSE control frame transmission.

8.1.5.18 void enet_hal_set_txpause (uint32_t *instance,* uint32_t *pauseDuration*)

This function Sets ENET transmit controller with pause duration. And set the transmit control to do PAUSE frame transmission This should be called when a PAUSE frame is dynamically wanted.

Parameters

instance	The ENET instance number

8.1.5.19 static void enet_hal_set_txipg (uint32_t *instance,* uint32_t *ipgValue*) [inline], [static]

This function indicates the IPG, in bytes, between transmitted frames. Valid values range from 8 to 27. If value is less than 8, the IPG is 8. If value is greater than 27, the IPG is 27.

Parameters

instance	The ENET instance number
ipgValue	The IPG for transmitted frames The default value is 12, the maximum value set to ipg is 0x1F.

8.1.5.20 static void enet_hal_set_truncationlen (uint32_t *instance,* uint32_t *length*) [inline], [static]

This function indicates the value a receive frame is truncated, if it is greater than this value. The frame truncation length must be greater than or equal to the receive maximum frame length.

Parameters

instance	The ENET instance number
length	The truncation length. The maximum value is 0x3FFF The default truncation length is 2047(0x7FF).

8.1.5.21 static void enet_hal_set_rx_max_size (uint32_t *instance,* uint32_t *maxBufferSize,* uint32_t *maxFrameSize*) [inline], [static]

Parameters

instance	The ENET instance number
maxBufferSize	The maximum receive buffer size, which should not be smaller than 256 It should be evenly divisible by 16 and the maximum receive size should not be larger than 0x3ff0.

maxFrameSize	The maximum receive frame size, the reset value is 1518 or 1522 if the VLAN tags are supported. The length is measured starting at DA and including the CRC.

8.1.5.22 void enet_hal_config_tx_fifo (uint32_t *instance,* enet_config_tx_fifo_t * *thresholdCfg*)

Parameters

instance	The ENET instance number
thresholdCfg	The FIFO threshold configuration

8.1.5.23 void enet_hal_config_rx_fifo (uint32_t *instance,* enet_config_rx_fifo_t * *thresholdCfg*)

Parameters

instance	The ENET instance number
thresholdCfg	The FIFO threshold configuration

8.1.5.24 static void enet_hal_set_rxbd_address (uint32_t *instance,* uint32_t *rxBdAddr*) [inline], [static]

This interface provides the beginning of the receive and receive buffer descriptor queue in the external memory. The txbdAddr is recommended to be 128-bit aligned, must be evenly divisible by 16.

Parameters

instance	The ENET instance number
rxBdAddr	The start address of receive buffer descriptors

8.1.5.25 static void enet_hal_set_txbd_address (uint32_t *instance,* uint32_t *txBdAddr*) [inline], [static]

This interface provides the beginning of the receive and transmit buffer descriptor queue in the external memory. The txbdAddr is recommended to be 128-bit aligned, must be evenly divisible by 16.

Parameters

instance	The ENET instance number
txBdAddr	The start address of transmit buffer descriptors

8.1.5.26 void enet_hal_init_rxbds (void * *rxBds*, uint8_t * *buffer*, bool *isLastBd*)

To make sure the uDMA will do the right data transfer after you activate with wrap flag and all the buffer descriptors should be initialized with an empty bit.

Parameters

rxBds	The current receive buffer descriptor
buffer	The data buffer on buffer descriptor
isLastBd	The flag to indicate the last receive buffer descriptor

8.1.5.27 void enet_hal_init_txbds (void * *txBds*, bool *isLastBd*)

To make sure the uDMA will do the right data transfer after you active with wrap flag.

Parameters

txBds	The current transmit buffer descriptor.
isLastBd	The last transmit buffer descriptor flag.

Ensures that the uDMA transfer data correctly after the user activates with the wrap flag.

Parameters

txBds	The current transmit buffer descriptor
isLastBd	The last transmit buffer descriptor flag

8.1.5.28 void enet_hal_update_rxbds (void * *rxBds*, uint8_t * *data*, bool *isbufferUpdate*)

This interface mainly clears the status region and updates the received buffer descriptor to ensure that the BD is correctly used.

Parameters

rxBds	The current receive buffer descriptor
data	The data buffer address
isbufferUpdate	The data buffer update flag. When you want to update the data buffer of the buffer descriptor ensure that this flag is set.

8.1.5.29 void enet_hal_update_txbds (void * *txBds*, uint8_t * *buffer*, uint16_t *length*, bool *isTxtsCfged*)

This interface mainly clears the status region and updates the transmit buffer descriptor to ensure tat this B-D is correctly used again. You should set the isTxtsCfged when the transmit timestamp feature is required.

Parameters

txBds	The current transmit buffer descriptor
buffer	The data buffer on buffer descriptor
length	The data length on buffer descriptor
isTxtsCfged	The timestamp configure flag. The timestamp is added to the transmit buffer descriptor when this flag is set.

8.1.5.30 static void enet_hal_clear_txbds (void * *curBd*) [inline], [static]

Clears the data, length, control, and status region of the transmit buffer descriptor.

Parameters

curBd	The current buffer descriptor

8.1.5.31 uint16_t enet_hal_get_rxbd_control (void * *curBd*)

This interface can get the whole control and status region of the receive buffer descriptor. The enet_rx_-bd_control_status_t enum type definition should be used if you want to get each status bit of the control and status region.

Parameters

curBd	The current receive buffer descriptor

Returns

The control and status data on buffer descriptors

8.1.5.32 uint16_t enet_hal_get_txbd_control (void * *curBd*)

This interface can get the whole control and status region of the transmit buffer descriptor. The enet_tx_bd_control_status_t enum type definition should be used if you want to get each status bit of the control and status region.

Parameters

curBd	The current transmit buffer descriptor

Returns

The extended control region of transmit buffer descriptor

8.1.5.33 bool enet_hal_get_rxbd_control_extend (void * *curBd,* enet_rx_bd_control_extend_t *controlRegion*)

This interface can get the whole control and status region of the receive buffer descriptor. The enet_rx_bd_control_extend_t enum type definition should be used if you want to get each status bit of the control and status region.

Parameters

curBd	The current receive buffer descriptor
controlRegion	The different control region

Returns

The extended control region data of receive buffer descriptor
- true when the control region is set
- false when the control region is not set

8.1.5.34 uint16_t enet_hal_get_txbd_control_extend (void * *curBd*)

This interface can get the whole control and status region of the transmit buffer descriptor. The enet_tx_bd_control_extend_t enum type definition should be used if you want to get each status bit of the control and status region.

Parameters

curBd	The current transmit buffer descriptor

Returns

The extended control data

8.1.5.35 uint16_t enet_hal_get_bd_length (void * *curBd*)

Parameters

curBd	The current buffer descriptor

Returns

The data length of the buffer descriptor

8.1.5.36 uint8_t* enet_hal_get_bd_buffer (void * *curBd*)

Parameters

curBd	The current buffer descriptor

Returns

The buffer address of the buffer descriptor

8.1.5.37 uint32_t enet_hal_get_bd_timestamp (void * *curBd*)

Parameters

curBd	The current buffer descriptor

Returns

The time stamp of the frame in the buffer descriptor. Notice that the frame timestamp is only set in the last buffer descriptor of the frame.

8.1.5.38 static void enet_hal_active_rxbd (uint32_t *instance*) [inline], [static]

The buffer descriptor activation should be done after the ENET module is enabled. Otherwise, the activation fails.

Parameters

instance	The ENET instance number

8.1.5.39 static void enet_hal_active_txbd (uint32_t *instance*) [inline], [static]

The buffer descriptor activation should be done after the ENET module is enabled. Otherwise, the activation fails.

Parameters

instance	The ENET instance number

8.1.5.40 void enet_hal_config_rmii (uint32_t *instance,* enet_config_rmii_t *mode,* enet_config_speed_t *speed,* enet_config_duplex_t *duplex,* bool *isRxOnTxDisabled,* bool *isLoopEnabled*)

Parameters

instance	The ENET instance number
mode	The RMII or MII mode
speed	The speed of RMII
duplex	The full or half duplex mode
isRxOnTx-Disabled	The Receive on transmit disable flag
isLoopEnabled	The loop enable flag

8.1.5.41 static void enet_hal_config_mii (uint32_t *instance,* uint32_t *miiSpeed,* enet_mdio_holdon_clkcycle_t *clkCycle,* bool *isPreambleDisabled*) [inline], [static]

Sets the MII interface between Mac and PHY. The miiSpeed is a value that controls the frequency of the MDC, relative to the internal module clock(InterClockSrc). A value of zero in this parameter turns the MDC off and leaves it in the low voltage state. Any non-zero value results in the MDC frequency MDC = InterClockSrc/((miiSpeed + 1)∗2). So miiSpeed = InterClockSrc/(2∗MDC) - 1. The Maximum MDC clock is 2.5MHZ(maximum). We should round up and plus one to simlplify: miiSpeed = InterClock-Src/(2∗2.5MHZ).

Parameters

instance	The ENET instance number
miiSpeed	The MII speed and it is ranged from 0∼0x3F
time	The holdon clock cycles for MDIO output
isPreamble-Disabled	The preamble disabled flag

8.1.5.42 static bool enet_hal_is_mii_enabled (uint32_t *instance*) [inline], [static]

This interface is usually called to check the MII interface before the Mac writes or reads the PHY registers.

Parameters

instance	The ENET instance number

Returns

The MII configuration status
- true if the MII has been configured.
- false if the MII has not been configured.

8.1.5.43 static uint32_t enet_hal_get_mii_data (uint32_t *instance*) [inline], [static]

Parameters

instance	The ENET instance number

Returns

The data read from PHY

8.1.5.44 void enet_hal_set_mii_command (uint32_t *instance,* uint32_t *phyAddr,* uint32_t *phyReg,* enet_mii_operation_t *operation,* uint32_t *data*)

Parameters

instance	The ENET instance number
phyAddr	The PHY address
phyReg	The PHY register
operation	The read or write operation
data	The data written to PHY

8.1.5.45 void enet_hal_config_ethernet (uint32_t *instance,* bool *isEnhanced,* bool *isEnabled*)

Parameters

instance	The ENET instance number
isEnhanced	The enhanced 1588 feature switch
isEnabled	The ENET module enable switch

8.1.5.46 void enet_hal_config_interrupt (uint32_t *instance,* uint32_t *source,* bool *isEnabled*)

Parameters

instance	The ENET instance number
source	The interrupt sources. enet_interrupt_request_t enum types is recommended as the interrupt source.
isEnabled	The interrupt enable switch

8.1.5.47 static void enet_hal_clear_interrupt (uint32_t *instance,* uint32_t *source*) [inline], [static]

Parameters

instance	The ENET instance number

source	The interrupt source to be cleared. enet_interrupt_request_t enum types is recommended as the interrupt source.

8.1.5.48 static bool enet_hal_get_interrupt_status (uint32_t *instance,* uint32_t *source*) [inline], [static]

Parameters

instance	The ENET instance number
source	The interrupt sources. enet_interrupt_request_t enum types is recommended as the interrupt source.

Returns

The event status of the interrupt source
- true if the interrupt event happened.
- false if the interrupt event has not happened.

8.1.5.49 static void enet_hal_clear_mib (uint32_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	The ENET instance number
isEnabled	The enable switch

8.1.5.50 static void enet_hal_enable_mib (uint32_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	The ENET instance number
isEnabled	The enable flag • True to enabale MIB block. • False to disable MIB block.

8.1.5.51 static bool enet_hal_get_mib_status (uint32_t *instance*) [inline], [static]

Parameters

instance	The ENET instance number

Returns

true if in MIB idle and MIB is not updating else false.

8.1.5.52 void enet_hal_config_tx_accelerator (uint32_t *instance,* enet_config_tx_accelerator_t * *txCfgPtr*)

Parameters

instance	The ENET instance number
txCfgPtr	The transmit accelerator configuration

8.1.5.53 void enet_hal_config_rx_accelerator (uint32_t *instance,* enet_config_rx_accelerator_t * *rxCfgPtr*)

Parameters

instance	The ENET instance number
rxCfgPtr	The receive accelerator configuration

8.1.5.54 void enet_hal_init_ptp_timer (uint32_t *instance,* enet_config_ptp_timer_t * *ptpCfgPtr*)

This interface initializes the 1588 context structure. Initialize 1588 parameters according to the user configuration structure.

Parameters

instance	The ENET instance number
ptpCfg	The 1588 timer configuration

8.1.5.55 static void enet_hal_enable_ptp_timer (uint32_t *instance,* uint32_t *isEnabled*) [inline], [static]

Enable the PTP timer will starts the timer. Disable the timer will stop timer at the current value.

Parameters

instance	The ENET instance number.
isEnabled	The 1588 timer Enable switch • True enbaled the 1588 PTP timer. • False disable or stop the 1588 PTP timer.

8.1.5.56 static void enet_hal_restart_ptp_timer (uint32_t *instance*) [inline], [static]

Restarting the PTP timer clears all PTP-timer counters to zero.

Parameters

instance	The ENET instance number

8.1.5.57 static void enet_hal_adjust_ptp_timer (uint32_t *instance,* uint32_t *increaseCorrection,* uint32_t *periodCorrection*) [inline], [static]

Adjust the 1588 timer according to the increase and correction period of the configured correction.

Parameters

instance	The ENET instance number
incease-Correction	The increase correction for 1588 timer
period-Correction	The period correction for 1588 timer

8.1.5.58 static void enet_hal_init_timer_channel (uint32_t *instance,* uint32_t *channel,* enet_timer_channel_mode_t *mode*) [inline], [static]

Parameters

instance	The ENET instance number channel The 1588 timer channel number
mode	Compare or capture mode for the 1588 timer channel

8.1.5.59 static void enet_hal_set_timer_channel_compare (uint32_t *instance,* uint32_t *channel,* uint32_t *compareValue*) [inline], [static]

Parameters

instance	The ENET instance number channel The 1588 timer channel number
compareValue	Compare value for 1588 timer channel

8.1.5.60 static bool enet_hal_get_timer_channel_status (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	The ENET instance number
channel	The 1588 timer channel number

Returns

Compare or capture operation status
- True if the compare or capture has occurred.
- False if the compare or capture has not occurred.

8.1.5.61 static void enet_hal_clear_timer_channel_flag (uint32_t *instance,* uint32_t *channel*) [inline], [static]

Parameters

instance	The ENET instance number
channel	The 1588 timer channel number

8.1.5.62 static void enet_hal_set_timer_capture (uint32_t *instance*) [inline], [static]

This is used before reading the current time register. After set timer capture, please wait for about 1us before read the captured timer.

Parameters

instance	The ENET instance number

8.1.5.63 static void enet_hal_set_current_time (uint32_t *instance,* uint32_t *nanSecond*) [inline], [static]

Parameters

instance	The ENET instance number
nanSecond	The nanosecond set to 1588 timer

8.1.5.64 static uint32_t enet_hal_get_current_time (uint32_t *instance*) [inline], [static]

Parameters

instance	The ENET instance number

Returns

the current time from 1588 timer

8.1.5.65 static uint32_t enet_hal_get_tx_timestamp (uint32_t *instance*) [inline], [static]

Parameters

instance	The ENET instance number

Returns

The timestamp of the last transmitted frame

8.1.5.66 bool enet_hal_get_txbd_timestamp_flag (void * *curBd*)

Parameters

curBd	The ENET transmit buffer descriptor

Returns

true if timestamp region is set else false.

8.1.5.67 static uint32_t enet_hal_get_bd_size (void) [inline], [static]

Parameters

null	

Returns

The the size of the buffer descriptor

8.2 ENET Peripheral Driver

The chapter describes the programming interface of the ENET Peripheral Driver.

Data Structures

- struct enet_multicast_group_t
  Defines the multicast group structure for the ENET device. More...
- struct enet_rxbd_config_t
  Defines the receive buffer descriptor configure structure. More...
- struct enet_txbd_config_t
  Defines the transmit buffer descriptor configure structure. More...
- struct enet_mac_config_t
  Defines the basic configuration structure for the ENET device. More...
- struct enet_phy_config_t
  Defines the basic configuration for PHY. More...
- struct enet_ethernet_header_t
  Defines the ENET header structure. More...
- struct enet_8021vlan_header_t
  Defines the ENET VLAN frame header structure. More...
- struct enet_mac_context_t
  Defines the ENET MAC context structure for the buffer address, buffer descriptor address, etc. More...
- struct enet_stats_t
  Defines the ENET packets statistic structure. More...
- struct enet_mac_packet_buffer_t
  Defines the ENET MAC packet buffer structure. More...
- struct enet_dev_if_t
  Defines the ENET device data structure for the ENET. More...
- struct enet_mac_api_t
  Defines the basic application for the ENET device. More...

Macros

- #define ENET_RECEIVE_ALL_INTERRUPT 0
  Defines the approach: ENET interrupt handler do receive.
- #define ENET_ENABLE_DETAIL_STATS 0
  Defines the statistic enable macro.
- #define ENET_ALIGN(x, align) ((unsigned int)((x) + ((align)-1)) & (unsigned int)(~(unsigned int)((align)- 1)))
  Defines the alignment operation.

Enumerations

- enum enet_interrupt_number_t {
  kEnetTstimerInt = 0,
  kEnetTsAvailInt,
  kEnetWakeUpInt,
  kEnetPlrInt,
  kEnetUnInt,
  kEnetRlInt,
  kEnetLcInt,
  kEnetEberrInt,
  kEnetMiiInt,
  kEnetRxbInt,
  kEnetRxfInt,
  kEnetTxbInt,
  kEnetTxfInt,
  kEnetGraInt,
  kEnetBabtInt,
  kEnetBabrInt,
  kEnetIntNum }
  Defines the interrupt source index for the interrupt vector change table.
- enum enet_frame_max_t {
  kEnetMaxTimeout = 0x10000,
  kEnetMaxFrameSize = 1518,
  kEnetMaxFrameVlanSize = 1522,
  kEnetMaxFrameDateSize = 1500,
  kEnetMaxFrameBdNumbers = 7,
  kEnetFrameFcsLen = 4,
  kEnetEthernetHeadLen = 14,
  kEnetEthernetVlanHeadLen = 18 }
  Defines the ENET main constant.
- enum enet_crc_parameter_t {
  kEnetCrcData = 0xFFFFFFFFU,
  kEnetCrcOffset = 8,
  kEnetCrcMask1 = 0x3F }
  Defines the CRC data for a hash value calculation.
- enum enet_protocol_type_t {
  kEnetProtocolIeee8023 = 0x88F7,
  kEnetProtocolIpv4 = 0x0800,
  kEnetProtocolIpv6 = 0x86dd,
  kEnetProtocol8021QVlan = 0x8100,
  kEnetPacketUdpVersion = 0x11,
  kEnetPacketIpv4Version = 0x4,
  kEnetPacketIpv6Version = 0x6 }
  Defines the ENET protocol type and main parameters.
- enum enet_mac_control_flag_t {

kEnetSleepModeEnable = 0x1,
kEnetPayloadlenCheckEnable = 0x2,
kEnetRxFlowControlEnable = 0x4,
kEnetRxCrcFwdEnable = 0x8,
kEnetRxPauseFwdEnable = 0x10,
kEnetRxPadRemoveEnable = 0x20,
kEnetRxBcRejectEnable = 0x40,
kEnetRxPromiscuousEnable = 0x80,
kEnetRxMiiLoopback = 0x100,
kEnetRxDisRxOnTx = 0x200 }
Defines the ENET MAC control Configure.

ENET Driver

- uint32_t enet_mii_read (uint32_t instance, uint32_t phyAddr, uint32_t phyReg, uint32_t *dataPtr)
  (R)MII Read function.
- uint32_t enet_mii_write (uint32_t instance, uint32_t phyAddr, uint32_t phyReg, uint32_t data)
  (R)MII Read function.
- uint32_t enet_mac_bd_init (enet_dev_if_t *enetIfPtr)
  Initializes ENET buffer descriptors.
- uint32_t enet_mac_mii_init (enet_dev_if_t *enetIfPtr)
  Initializes the ENET MAC MII(MDC/MDIO) interface.
- uint32_t enet_mac_rxbd_init (enet_dev_if_t *enetIfPtr, enet_rxbd_config_t *rxbdCfg)
  Initialize the ENET receive buffer descriptors.
- uint32_t enet_mac_rxbd_deinit (enet_dev_if_t *enetIfPtr)
  Deinitialize the ENET receive buffer descriptors.
- uint32_t enet_mac_txbd_init (enet_dev_if_t *enetIfPtr, enet_txbd_config_t *txbdCfg)
  Initialize the ENET transmit buffer descriptors.
- uint32_t enet_mac_txbd_deinit (enet_dev_if_t *enetIfPtr)
  Deinitialize the ENET transmit buffer descriptors.
- uint32_t enet_mac_configure_fifo_accel (enet_dev_if_t *enetIfPtr)
  Initializes ENET MAC FIFO and accelerator with the basic configuration.
- uint32_t enet_mac_configure_controller (enet_dev_if_t *enetIfPtr)
  the ENET controller with the basic configuration.
- uint32_t enet_mac_deinit (enet_dev_if_t *enetIfPtr)
  Deinit the ENET device.
- uint32_t enet_mac_update_rxbd (enet_dev_if_t *enetIfPtr, bool isBufferUpdate)
  Updates the receive buffer descriptor.
- bool enet_mac_rx_error_stats (enet_dev_if_t *enetIfPtr, uint32_t data)
  Processes the ENET receive frame error statistics.
- void enet_mac_tx_error_stats (enet_dev_if_t *enetIfPtr, void *curBd)
  Processes the ENET transmit frame statistics.
- uint32_t enet_mac_tx_cleanup (enet_dev_if_t *enetIfPtr)
  ENET transmit buffer descriptor cleanup.
- uint32_t enet_mac_receive (enet_dev_if_t *enetIfPtr, enet_mac_packet_buffer_t *packBuffer)
  Receives ENET packets.
- uint32_t enet_mac_send (enet_dev_if_t *enetIfPtr, uint8_t *packet, uint32_t size)
  Transmits ENET packets.
- void enet_mac_rx_isr (void *enetIfPtr)

The ENET receive interrupt handler.

- void enet_mac_tx_isr (void *enetIfPtr)
  The ENET transmit interrupt handler.
- void enet_mac_calculate_crc32 (enetMacAddr address, uint32_t *crcValue)
  Calculates the CRC hash value.
- uint32_t enet_mac_add_multicast_group (uint32_t instance, enet_multicast_group_t *multiGroupPtr, enetMacAddr address)
  Adds the ENET device to a multicast group.
- uint32_t enet_mac_leave_multicast_group (uint32_t instance, enet_multicast_group_t *multiGroupPtr, enetMacAddr address)
  Moves the ENET device from a multicast group.
- uint32_t enet_mac_init (enet_dev_if_t *enetIfPtr, enet_rxbd_config_t *rxbdCfg, enet_txbd_config_t *txbdCfg)
  Initializes the ENET with the basic configuration.
- void enet_mac_enqueue_buffer (void **queue, void *buffer)
  Enqueues a data buffer to the buffer queue.
- void * enet_mac_dequeue_buffer (void **queue)
  Dequeues a buffer from the buffer queue.

8.2.0.68 ENET Driver

Overview

The ENET driver receives data from and transmits data to the wired network. The enhanced 1588 feature supports clock synchronization.

ENET device data structure

The ENET device data structure, enet_dev_if_t, includes configuration structure, application structure, and data context structure. This structure should be initialized before the ENET device initialization function. Then, the data structure is passed from the API layer to the device.

Configuration structure

The ENET device data structure uses the enet_mac_config_t and the enet_phy_config_t configuration structure for MAC and PHY configurations. This allows users to configure the most common settings of the ENET peripheral. The enet_mac_config_t mac configuration structure includes the receive and transmit buffer descriptor numbers, data buffer number, buffer size, MII/RMII mode, MAC loop mode. The enet_phy_config_t PHY configuration structure includes the PHY address and PHY loop mode, which need to be configured. If the PHY address is unknown, use the isPhyAutoDiscover flag in MAC enet_mac_config_t configuration to find the PHY address. Note:

1. The recommended maximum frame length is 1518 or 1522(VLAN).

2. The receive buffer size should use the maximum frame size 1518 or 1522(VLAN). In this case, the rxLargeBuffer is not needed and the rxLargeBufferNumber should be set to 0.
3. If a small buffer size is used and the stack you choose does not support a frame with fragments, the drivers will allocate a large external buffer and do memcopy for collecting frames. In this case, the rxLargeBufferNumber value needs to be at least 4.
4. In current driver, transmit frames are with crc.

Application structure

The ENET device data structure uses the enet_mac_api_t and the enet_phy_api_t as the driver API structure. This API structure allows higher layers to call specific device functions. This is still flexible for different stack porting. The MAC and PHY application structure are defined like this:

```
    const enet_mac_api_t g_enetMacApi =
    {
        enet_mac_init,
        enet_mac_close,
        enet_mac_send,
#if !ENET_RECEIVE_ALL_INTERRUPT
        enet_mac_receive,
#endif
        enet_mii_read,
        enet_mii_write,
        enet_mac_add_multicast_group,
        enet_mac_leave_multicast_group,
    };

    const enet_phy_api_t g_enetPhyApi =
    {
        phy_auto_discover,
        phy_init,
        phy_get_link_speed,
        phy_get_link_status,
        phy_get_link_duplex,
    };
```

Context data structure

The ENET device data structure uses the enet_mac_context_t to keep the user configuration and keep track of data transfer. Data receive and transmit is easily managed with this context structure.

Initialization

To initialize the ENET module, do this:

1. First, initialize the ENET MAC and PHY configuration structures, and initialize the upper layer callback function.
2. Call the enet_mac_init() function through the enet_mac_api_t API structure in the enet_dev_if_t and pass in the enet_dev_if_t device data structure. This function enables the ENET module.
   This is an example code to initialize the ENET device data structure:

```
    enet_dev_if_t  enetIfPtr;
    enet_rxbd_config_t rxbdCfg;
    enet_txbd_config_t txbdCfg;

    /* Set up the MAC configuration structure*/
    enet_mac_config_t g_enetMacCfg =
    {
        kEnetMaxFrameSize,
        ENET_RX_LARGE_BUFFER_NUM,
        ENET_RX_RING_LEN,
        ENET_TX_RING_LEN,
        {0},
        kEnetCfgRmii,
        kEnetCfgSpeed100M,
        kEnetCfgFullDuplex,
        kEnetRxCrcFwdEnable, /* No loop, receive Broadcast receive crc forward*/
        false,
        false,
        false,
        {true, true, false, false, false},
        {true, true, false},
        false,
        true,
        ENET_MII_CLOCK,
#if FSL_FEATURE_ENET_SUPPORT_PTP
        ENET_PTP_RING_BUFFER_NUM,
        false,
#endif
    };

    /* Set up the PHY configuration structure*/
    enet_phy_config_t g_enetPhyCfg =
    {{0, false }};

    /* Initialize ENET device data structure*/
    enetIfPtr->macCfgPtr = &g_enetMacCfg[device];
    enetIfPtr->phyCfgPtr = &g_enetPhyCfg[device];
    enetIfPtr->macApiPtr = &g_enetMacApi;
    enetIfPtr->phyApiPtr = (void *)&g_enetPhyApi;
#if ENET_RECEIVE_ALL_INTERRUPT
    /* Initialize ENET callback functions if choose interrupt only approach to do mac receive*/
    enetIfPtr->enetNetifcall = enet_Callback;
#endif
    /* Initialize ENET buffer and prepare configuration before enet mac init*/
    enet_mac_buffer_init(enetIfPtr, &rxbdCfg, &txbdCfg);
    /* Initialize ENET device*/
    ((enet_mac_api_t *)(enetIfPtr->macApiPtr))->
      enet_mac_init(enetIfPtr, rxbdCfg, txbdCfg);
    ((enet_phy_api_t *)(enetIfPtr->phyApiPtr))->phy_init(enetIfPtr);
```

Data Receive

There are two approaches on ENET receive and the MACRO ENET_RECEIVE_ALL_INTERRUPT is used to choose the approach.

- interrupt add poll(define ENET_RECEIVE_ALL_INTERRUPT with 0)
- interrupt only (define ENET_RECEIVE_ALL_INTERRUPT with 1)

Interrupt add task poll approach: The ENET driver receives data directly from the buffer descriptor to the upper layer. To shorten the receive process time on the receive interrupt, receive data uses the interrupt and poll combination:

1. Receive interrupt: releases the receive synchronize signal(enetReceiveSync).
2. Poll: receives task and waits for the receive synchronize signal when no data is received. The receive data returns the address and data length of the received data. To receive data from the ENET device, create a receive task as follows:

```
    /* Create the task when do net device initialize*/
    task_create(ENET_receive, ENET_TEST_TASK_PRIO, enetIfPtr, &revHandle);

      ....................................


    ENET_receive(void *param)
    {
        uint8_t *packet;
        uint16_t length;
        enet_mac_packet_buffer_t packetBuffer[
      kEnetMaxFrameBdNumbers];

        enet_dev_if_t * enetIfPtr = (enet_dev_if_t *)param;
        while(1)
       {
            /* Receive frame*/
           result = enetIfPtr->macApiPtr->enet_mac_receive(enetIfPtr, &packetBuffer[0]);
           if((result == kStatus_ENET_RxbdEmpty) || (result ==
      kStatus_ENET_InvalidInput))
        {
            /* Block when there is no data received*/
            event_wait(&enetIfPtr->enetReceiveSync,
      kSyncWaitForever, &flag);
        }

          .................

        /* The packets delivery to upper layer*/
          .................
      }
  }



/* Receive interrupt handler to wake up blocked receive task*/
void enet_mac_rx_isr(void *enetIfPtr)
{
    event_group_t flag = 0x1;

    /*Check input parameter*/
    if (!enetIfPtr)
    {
        return;
    }
    /* Get interrupt status.*/
    while (enet_hal_get_interrupt_status(((
      enet_dev_if_t *)enetIfPtr)->deviceNumber, (kEnetRxFrameInterrupt |
      kEnetRxByteInterrupt)))
    {
        /* Clear interrupt*/
        enet_hal_clear_interrupt(((enet_dev_if_t *)enetIfPtr)->
      deviceNumber,
            (kEnetRxFrameInterrupt | kEnetRxByteInterrupt));
        /* Release sync signal and wake up ENET receive task to process received data*/
        event_set(&((enet_dev_if_t *)enetIfPtr)->enetReceiveSync, flag);
    }
}
Interrupt only approach for MAC receive:
The ENET driver receives data directly from the buffer descriptor to the upper layer.
In this case, data is received on receive interrupt handler:
  1. Receive interrupt handler calls the receive peripheral driver.
```

2. Receive peripheral driver calls the initialized callback function.
3. The callback function checks the protocol and delivers the received data to the upper layer of the TCP/IP stack.

These are the details:
~~~~~{.c}

```
void enet_mac_rx_isr(void *enetIfPtr)
{
    /*Check input parameter*/
    if(!enetIfPtr)
    {
        return;
    }
    /* Get interrupt status.*/
    while(enet_hal_get_interrupt_status(((
      enet_dev_if_t *)enetIfPtr)->deviceNumber, (kEnetRxFrameInterrupt |
      kEnetRxByteInterrupt)))
    {
        /*Clear interrupt*/
        enet_hal_clear_interrupt(((enet_dev_if_t *)enetIfPtr)->
      deviceNumber,
            (kEnetRxFrameInterrupt | kEnetRxByteInterrupt));
        /* Receive peripheral driver*/
        enet_mac_receive((enet_dev_if_t *)enetIfPtr);
    }

}

  ....................................


uint32_t enet_mac_receive(enet_dev_if_t * enetIfPtr)
{
    void *curBd;
    uint32_t length;
    uint8_t *packet;
    uint32_t controlStatus;

    /* Check input parameters*/
    if(!enetIfPtr->macContextPtr)
    {
        return kStatus_ENET_InvalidInput;
    }

    /* Check the current buffer descriptor address*/
    curBd = enetIfPtr->macContextPtr->rxBdCurPtr;
    if(!curBd)
    {
        return kStatus_ENET_RxbdInvalid;
    }


      ...........
      /* callback function to delivery to stack upper layer */
       enetIfPtr->enetNetifcall(enetIfPtr, packet, length);

      ..........


    return kStatus_ENET_Success;
}


uint32_t void enet_callback(void *param, uint8_t *packet, uint32_t length)
{
    uint16_t type;
```
~~~~~

```
        /* Process the received frame*/
        type = NTOHS(*(uint16_t *)&((enet_etherent_header_t *)packet)->type);
        /* Collect frame to PCB structure for upper layer process*/
        QUEUEGET(packbuffer[enetIfPtr->deviceNumber].pcbHead,packbuffer[enetIfPtr->
          deviceNumber].pcbTail, pcbPtr);
        if(pcbPtr)
        {
            pcbPtr->FRAG[0].LENGTH = length;
            pcbPtr->FRAG[0].FRAGMENT = packet;
            pcbPtr->PRIVATE = (void *)enetIfPtr;

            switch(type)
            {
                case ENETPROT_IP:
                IPE_recv_IP((PCB *)pcbPtr,enetIfPtr->netIfPtr);
                    break;
                case ENETPROT_ARP:
                IPE_recv_ARP((PCB *)pcbPtr,enetIfPtr->netIfPtr);
                    break;
                case ENETPROT_IP6:
                IP6E_recv_IP((PCB *)pcbPtr,enetIfPtr->netIfPtr);
                    break;
                case ENETPROT_ETHERNET:
                enet_ptp_service_l2packet(enetIfPtr, packet, length);
                    break;
                default:
                PCB_free((PCB *)pcbPtr);
                    break;
            }
        }
        else
        {

            enetIfPtr->stats.statsRxMissed++;
        }
    }
```

Data Transmit

Data Transmit transmits data from the TCP/IP layer to the device and, then, to the network. The data transmit function should be used like this: PCB_PTR is the data buffer structure of the frame which contains many PCB_FRAGMENT(including the data buffer and length).

```
void ENET_send(PCB_PTR packet, void *handle)
{
    uint16_t length = 0;
    PCB_FRAGMENT *fragPtr;
    enet_dev_if_t *enetIfPtr = (enet_dev_if_t *)handle;

    /* dequeue an available transmit data buffer from txbuffer queue */
    frame = enet_mac_dequeue_buffer((void * *)&enetIfPtr->
      macContextPtr->txBufferPtr);
    if( frame == NULL)
    {
        return ENETERR_ALLOC;
    }

    /* Fill the buffer with a whole frame you want to transmit*/
    for(fragPtr = packet->FRAG; fragPtr->LENGTH; fragPtr++)
    {
        memcpy(frame + length, fragPtr->FRAGMENT, fragPtr->LENGTH);
        length += fragPtr->LENGTH;
```

```
    }

    /*Send the data out*/
    ((enet_mac_api_t *)(enetIfPtr->macApiPtr))->
      enet_mac_send(enetIfPtr, frame, length);
}
```

#Notification:

- TWR-MK70F120M board only routes the RMII interface signals from the CPU board to the primary connectors. Hence, the rmiiCfgMode in the enet_mac_config_t should be set to the kEnetCfgRmii. TWR-MK64F120M board can configure both RMII and MII modes.
- Jumper setting When ENET is uses the RMII interface signals by default, the RMII input clock must be kept in phase with the clock supplied to the external PHY. Jumpers should be set like this:
  - TWR-SER J2 shunt across 3-4 , J3 shunt across 2-3, J12 shunt across 9-10
  - TWR-MK64f120M board make sure J32 jumper is on to disable the OSC on the CPU board.
  - TWR-MK70f120M board make sure J18 jumper is on to disable the OSC on the CPU board. When ENET chooses the MII interface signals, the jumper should be set as: TWR-SER J2 shunt across 1-2, J12 no shunt across 9-10.

8.2.1 Data Structure Documentation

8.2.1.1 struct enet_multicast_group_t

Data Fields

- enetMacAddr groupAdddr
  Multicast group address.
- uint32_t hash
  Hash value of the multicast group address.
- struct ENETMulticastGroup * next
  Pointer of the next group structure.
- struct ENETMulticastGroup * prv
  Pointer of the previous structure.

8.2.1.2 struct enet_rxbd_config_t

Data Fields

- uint8_t * rxBdPtrAlign
  Aligned receive buffer descriptor pointer.
- uint8_t * rxBufferAlign
  Aligned receive data buffer pointer.
- uint8_t * rxLargeBufferAlign
  Aligned receive large data buffer pointer.
- uint8_t rxBdNum
  Aligned receive buffer descriptor pointer.
- uint8_t rxBufferNum

Receive buffer number.
- uint8_t rxLargeBufferNum
  Large receive buffer number.
- uint32_t rxLargeBufferSizeAlign
  Aligned large receive buffer size.

8.2.1.3 struct enet_txbd_config_t

Data Fields

- uint8_t * txBdPtrAlign
  Aligned transmit buffer descriptor pointer.
- uint8_t * txBufferAlign
  Aligned transmit buffer descriptor pointer.
- uint8_t txBufferNum
  Transmit buffer number.
- uint32_t txBufferSizeAlign
  Aligned transmit buffer size.

8.2.1.4 struct enet_mac_config_t

Data Fields

- uint16_t rxBufferSize
  Receive buffer size.
- uint16_t rxLargeBufferNumber
  Receive large buffer number; Needed only when the BD size is smaller than the maximum frame length.
- uint16_t rxBdNumber
  Receive buffer descriptor number.
- uint16_t txBdNumber
  Transmit buffer descriptor number.
- enetMacAddr macAddr
  MAC hardware address.
- enet_config_rmii_t rmiiCfgMode
  RMII configure mode.
- enet_config_speed_t speed
  Speed configuration.
- enet_config_duplex_t duplex
  Duplex configuration.
- bool isTxAccelEnabled
  Switcher to enable transmit accelerator.
- bool isRxAccelEnabled
  Switcher to enable receive accelerator.
- bool isStoreAndFwEnabled
  Switcher to enable store and forward.
- enet_config_rx_accelerator_t rxAcceler
  Receive accelerator configure.
- enet_config_tx_accelerator_t txAcceler
  Transmit accelerator configure.

- bool isVlanEnabled
  Switcher to enable VLAN frame.
- bool isPhyAutoDiscover
  Switcher to use PHY auto discover.
- uint32_t miiClock
  MII speed.

8.2.1.4.0.15 Field Documentation

8.2.1.4.0.15.1 uint16_t enet_mac_config_t::rxLargeBufferNumber

8.2.1.4.0.15.2 enet_config_duplex_t enet_mac_config_t::duplex

Mac control configure, it is recommended to use enet_mac_control_flag_t it is special control set for loop mode, sleep mode, crc forward/terminate etc

8.2.1.5 struct enet_phy_config_t

Data Fields

- uint8_t phyAddr
  PHY address.
- bool isLoopEnabled
  Switcher to enable the HY loop mode.

8.2.1.6 struct enet_ethernet_header_t

Data Fields

- enetMacAddr destAddr
  Destination address.
- enetMacAddr sourceAddr
  Source address.
- uint16_t type
  Protocol type.

8.2.1.7 struct enet_8021vlan_header_t

Data Fields

- enetMacAddr destAddr
  Destination address.
- enetMacAddr sourceAddr
  Source address.
- uint16_t tpidtag
  ENET 8021tag header tag region.
- uint16_t othertag
  ENET 8021tag header type region.

- uint16_t type
  Protocol type.

8.2.1.8 struct enet_mac_context_t

Data Fields

- uint8_t * rxBufferPtr
  Receive buffer pointer.
- uint8_t * rxLargeBufferPtr
  Receive large buffer descriptor.
- uint8_t * txBufferPtr
  Transmit buffer pointer.
- uint8_t * rxBdBasePtr
  Receive buffer descriptor base address pointer.
- uint8_t * rxBdCurPtr
  Current receive buffer descriptor pointer.
- uint8_t * rxBdDirtyPtr
  Receive dirty buffer descriptor.
- uint8_t * txBdBasePtr
  Transmit buffer descriptor base address pointer.
- uint8_t * txBdCurPtr
  Current transmit buffer descriptor pointer.
- uint8_t * txBdDirtyPtr
  Last cleaned transmit buffer descriptor pointer.
- bool isTxFull
  Transmit buffer descriptor full.
- bool isRxFull
  Receive buffer descriptor full.
- uint32_t bufferdescSize
  ENET buffer descriptor size.
- uint16_t rxBufferSizeAligned
  Receive buffer alignment size.
- bool isVlanTagEnabled
  ENET VLAN-TAG frames enabled.

8.2.1.9 struct enet_stats_t

Data Fields

- uint32_t statsRxTotal
  Total number of receive packets.
- uint32_t statsRxMissed
  Total number of receive packets.
- uint32_t statsRxDiscard
  Receive discarded with error.
- uint32_t statsRxError
  Receive discarded with error packets.
- uint32_t statsTxTotal

Total number of transmit packets.
- uint32_t statsTxMissed
  Transmit missed.
- uint32_t statsTxDiscard
  Transmit discarded with error.
- uint32_t statsTxError
  Transmit error.
- uint32_t statsRxAlign
  Receive non-octet alignment.
- uint32_t statsRxFcs
  Receive CRC error.
- uint32_t statsRxTruncate
  Receive truncate.
- uint32_t statsRxLengthGreater
  Receive length greater than RCR[MAX_FL].
- uint32_t statsRxCollision
  Receive collision.
- uint32_t statsRxOverRun
  Receive over run.
- uint32_t statsTxOverFlow
  Transmit overflow.
- uint32_t statsTxLateCollision
  Transmit late collision.
- uint32_t statsTxExcessCollision
  Transmit excess collision.
- uint32_t statsTxUnderFlow
  Transmit under flow.
- uint32_t statsTxLarge
  Transmit large packet.
- uint32_t statsTxSmall
  Transmit small packet.

8.2.1.10 struct enet_mac_packet_buffer_t

8.2.1.11 struct enet_dev_if_t

Data Fields

- struct ENETDevIf * next
  Next device structure address.
- void * netIfPtr
  Store the connected upper layer in the structure.
- enet_multicast_group_t * multiGroupPtr
  Multicast group chain.
- uint32_t deviceNumber
  Device number.
- bool isInitialized
  Device initialized.
- uint16_t maxFrameSize
  MAC maximum frame size.

- enet_mac_config_t * macCfgPtr
  MAC configuration structure.
- enet_phy_config_t * phyCfgPtr
  PHY configuration structure.
- struct ENETMacApi * macApiPtr
  MAC application interface structure.
- void * phyApiPtr
  PHY application interface structure.
- enet_mac_context_t * macContextPtr
  MAC context pointer.
- event_object_t enetReceiveSync
  Receive sync signal.
- lock_object_t enetContextSync
  Sync signal.

8.2.1.12 struct enet_mac_api_t

Data Fields

- uint32_t(* enet_mac_init)(enet_dev_if_t *enetIfPtr, enet_rxbd_config_t *rxbdCfg, enet_txbd_config_t *txbdCfg)
  MAC initialize interface.
- uint32_t(* enet_mac_deinit)(enet_dev_if_t *enetIfPtr)
  MAC close interface.
- uint32_t(* enet_mac_send)(enet_dev_if_t *enetIfPtr, uint8_t *packet, uint32_t size)
  MAC send packets.
- uint32_t(* enet_mac_receive)(enet_dev_if_t *enetIfPtr, enet_mac_packet_buffer_t *packBuffer)
  MAC receive interface.
- uint32_t(* enet_mii_read)(uint32_t instance, uint32_t phyAddr, uint32_t phyReg, uint32_t *dataPtr)
  MII reads PHY.
- uint32_t(* enet_mii_write)(uint32_t instance, uint32_t phyAddr, uint32_t phyReg, uint32_t data)
  MII writes PHY.
- uint32_t(* enet_add_multicast_group)(uint32_t instance, enet_multicast_group_t *multiGroupPtr, uint8_t *groupAddr)
  Add multicast group.
- uint32_t(* enet_leave_multicast_group)(uint32_t instance, enet_multicast_group_t *multiGroupPtr, uint8_t *groupAddr)
  Leave multicast group.

8.2.2 Macro Definition Documentation

8.2.2.1 #define ENET_ENABLE_DETAIL_STATS 0

8.2.2.2 #define ENET_ALIGN(*x, align*) ((unsigned int)((x) + ((align)-1)) & (unsigned int)(∼(unsigned int)((align)- 1)))

8.2.3 Enumeration Type Documentation

8.2.3.1 enum enet_interrupt_number_t

Enumerator

kEnetTstimerInt Timestamp interrupt.
kEnetTsAvailInt TS-avail interrupt.
kEnetWakeUpInt Wakeup interrupt.
kEnetPlrInt Plr interrupt.
kEnetUnInt Un interrupt.
kEnetRlInt RL interrupt.
kEnetLcInt LC interrupt.
kEnetEberrInt Eberr interrupt.
kEnetMiiInt MII interrupt.
kEnetRxbInt Receive byte interrupt.
kEnetRxfInt Receive frame interrupt.
kEnetTxbInt Transmit byte interrupt.
kEnetTxfInt Transmit frame interrupt.
kEnetGraInt Gra interrupt.
kEnetBabtInt Babt interrupt.
kEnetBabrInt Babr interrupt.
kEnetIntNum Interrupt number.

8.2.3.2 enum enet_frame_max_t

Enumerator

kEnetMaxTimeout Maximum timeout.
kEnetMaxFrameSize Maximum frame size.
kEnetMaxFrameVlanSize Maximum VLAN frame size.
kEnetMaxFrameDateSize Maximum frame data size.
kEnetMaxFrameBdNumbers Maximum buffer descriptor numbers of a frame.
kEnetFrameFcsLen FCS length.
kEnetEthernetHeadLen Ethernet Frame header length.
kEnetEthernetVlanHeadLen Ethernet Vlan frame header length.

8.2.3.3 enum enet_crc_parameter_t

Enumerator

kEnetCrcData CRC-32 maximum data.
kEnetCrcOffset CRC-32 offset2.
kEnetCrcMask1 CRC-32 mask.

8.2.3.4 enum enet_protocol_type_t

Enumerator

kEnetProtocolIeee8023 Packet type Ethernet ieee802.3.
kEnetProtocolIpv4 Packet type IPv4.
kEnetProtocolIpv6 Packet type IPv6.
kEnetProtocol8021QVlan Packet type VLAN.
kEnetPacketUdpVersion UDP protocol type.
kEnetPacketIpv4Version Packet IP version IPv4.
kEnetPacketIpv6Version Packet IP version IPv6.

8.2.3.5 enum enet_mac_control_flag_t

Enumerator

kEnetSleepModeEnable ENET control sleep mode Enable.
kEnetPayloadlenCheckEnable ENET receive payload length check Enable.
kEnetRxFlowControlEnable ENET flow control, receiver detects PAUSE frames and stops transmitting data when a PAUSE frame is detected.
kEnetRxCrcFwdEnable Received frame crc is stripped from the frame.
kEnetRxPauseFwdEnable Pause frames are forwarded to the user application.
kEnetRxPadRemoveEnable Padding is removed from received frames.
kEnetRxBcRejectEnable Broadcast frame reject.
kEnetRxPromiscuousEnable Promiscuous mode enabled.
kEnetRxMiiLoopback MAC MII loopback mode.
kEnetRxDisRxOnTx MAC disable receive on transmit for half-duplex.

8.2.4 Function Documentation

8.2.4.1 uint32_t enet_mii_read (uint32_t *instance,* uint32_t *phyAddr,* uint32_t *phyReg,* uint32_t * *dataPtr*)

Parameters

instance	The ENET instance number.
phyAddr	The PHY address.
phyReg	The PHY register.
dataPtr	The data read from MII.

Returns

The execution status.

8.2.4.2 uint32_t enet_mii_write (uint32_t *instance,* uint32_t *phyAddr,* uint32_t *phyReg,* uint32_t *data*)

Parameters

instance	The ENET instance number.
phyAddr	The PHY address.
phyReg	The PHY register.
data	The data write to MII.

Returns

The execution status.

8.2.4.3 uint32_t enet_mac_bd_init (enet_dev_if_t * *enetIfPtr*)

Parameters

enetIfPtr	The ENET context structure.

Returns

The execution status.

8.2.4.4 uint32_t enet_mac_mii_init (enet_dev_if_t * *enetIfPtr*)

Parameters

enetIfPtr	The ENET context structure.

Returns

The execution status.

8.2.4.5 uint32_t enet_mac_rxbd_init (enet_dev_if_t * *enetIfPtr,* enet_rxbd_config_t * *rxbdCfg*)

If you open ENET_RECEIVE_ALL_INTERRUPT to do receive data buffer numbers can be the same as the receive descriptor numbers. But if you close ENET_RECEIVE_ALL_INTERRUPT and choose polling receive frames please make sure the receive data buffers are more than buffer descriptor numbers to guarantee a good performance.

Parameters

enetIfPtr	The ENET context structure.
rxbdCfg	The receive buffer descriptor configuration.

Returns

The execution status.

8.2.4.6 uint32_t enet_mac_rxbd_deinit (enet_dev_if_t * *enetIfPtr*)

Deinitialize the ENET receive buffer descriptors.

Parameters

enetIfPtr	The ENET context structure.

Returns

The execution status.

8.2.4.7 uint32_t enet_mac_txbd_init (enet_dev_if_t * *enetIfPtr,* enet_txbd_config_t * *txbdCfg*)

Parameters

enetIfPtr	The ENET context structure.
txbdCfg	The transmit buffer descriptor configuration.

Returns

The execution status.

8.2.4.8 uint32_t enet_mac_txbd_deinit (enet_dev_if_t * *enetIfPtr*)

Deinitialize the ENET transmit buffer descriptors.

Parameters

enetIfPtr	The ENET context structure.

Returns

The execution status.

8.2.4.9 uint32_t enet_mac_configure_fifo_accel (enet_dev_if_t * *enetIfPtr*)

Parameters

enetIfPtr	The ENET context structure.

Returns

The execution status.

8.2.4.10 uint32_t enet_mac_configure_controller (enet_dev_if_t * *enetIfPtr*)

Parameters

enetIfPtr	The ENET context structure.

Returns

The execution status.

8.2.4.11 uint32_t enet_mac_deinit (enet_dev_if_t * *enetIfPtr*)

Parameters

enetIfPtr	The ENET context structure.

Returns

The execution status.

8.2.4.12 uint32_t enet_mac_update_rxbd (enet_dev_if_t * *enetIfPtr,* bool *isBufferUpdate*)

This updates the used receive buffer descriptor ring to ensure that the used BDS is correctly used again. It cleans the status region and sets the control region of the used receive buffer descriptor. If the isBuffer-Update flag is set, the data buffer in the buffer descriptor is updated.

Parameters

enetIfPtr	The ENET context structure.
isBufferUpdate	The data buffer update flag.

Returns

The execution status.

8.2.4.13 bool enet_mac_rx_error_stats (enet_dev_if_t * *enetIfPtr,* uint32_t *data*)

This interface gets the error statistics of the received frame. Because the error information is in the last BD of a frame, this interface should be called when processing the last BD of a frame.

Parameters

enetIfPtr	The ENET context structure.
data	The current control and status data of the buffer descriptor.

Returns

The frame error status.
- True if the frame has an error.
- False if the frame does not have an error.

8.2.4.14 void enet_mac_tx_error_stats (enet_dev_if_t * *enetIfPtr,* void * *curBd*)

This interface gets the error statistics of the transmit frame. Because the error information is in the last BD of a frame, this interface should be called when processing the last BD of a frame.

Parameters

enetIfPtr	The ENET context structure.
curBd	The current buffer descriptor.

8.2.4.15 uint32_t enet_mac_tx_cleanup (enet_dev_if_t * *enetIfPtr*)

First, store the transmit frame error statistic and PTP timestamp of the transmitted packets. Second, clean up the used transmit buffer descriptors. If the PTP 1588 feature is open, this interface captures the 1588 timestamp. It is called by the transmit interrupt handler.

Parameters

enetIfPtr	The ENET context structure.

Returns

The execution status.

8.2.4.16 uint32_t enet_mac_receive (enet_dev_if_t * *enetIfPtr,* enet_mac_packet_buffer_t * *packBuffer*)

Parameters

enetIfPtr	The ENET context structure.
packBuffer	The received data buffer.

Returns

The execution status.

8.2.4.17 uint32_t enet_mac_send (enet_dev_if_t * *enetIfPtr,* uint8_t * *packet,* uint32_t *size*)

Parameters

enetIfPtr	The ENET context structure.
packet	The frame to be transmitted.
size	The frame size.

Returns

The execution status.

8.2.4.18 void enet_mac_rx_isr (void * *enetIfPtr*)

Parameters

enetIfPtr	The ENET context structure pointer.

8.2.4.19 void enet_mac_tx_isr (void * *enetIfPtr*)

Parameters

enetIfPtr	The ENET context structure pointer.

8.2.4.20 void enet_mac_calculate_crc32 (enetMacAddr *address,* uint32_t * *crcValue*)

Parameters

address	The ENET MAC hardware address.
crcVlaue	The calculated CRC value of the Mac address.

8.2.4.21 uint32_t enet_mac_add_multicast_group (uint32_t *instance,* enet_multicast_group_t * *multiGroupPtr,* enetMacAddr *address*)

Parameters

instance	The ENET instance number.
multiGroupPtr	The ENET multicast group structure.
address	The ENET MAC hardware address.

Returns

The execution status.

8.2.4.22 uint32_t enet_mac_leave_multicast_group (uint32_t *instance,* enet_multicast_group_t * *multiGroupPtr,* enetMacAddr *address*)

Parameters

instance	The ENET instance number.
multiGroupPtr	The ENET multicast group structure.
address	The ENET MAC hardware address.

Returns

The execution status.

8.2.4.23 uint32_t enet_mac_init (enet_dev_if_t * *enetIfPtr,* enet_rxbd_config_t * *rxbdCfg,* enet_txbd_config_t * *txbdCfg*)

Parameters

enetIfPtr	The pointer to the basic configuration structure.

Returns

The execution status.

8.2.4.24 void enet_mac_enqueue_buffer (void ** *queue*, void * *buffer*)

Parameters

queue	The buffer queue.
buffer	The buffer to add to the buffer queue.

8.2.4.25 void* enet_mac_dequeue_buffer (void ** *queue*)

Parameters

queue	The buffer queue.

Returns

The dequeued data buffer.

8.3 ENET RTCS Adaptor

The chapter describes the programming interface of the ENET RTCS Adaptor.

Data Structures

- struct ENET_HEADER_PTR
  Define Structure for Ethernet packet header. More...
- struct PCB_FRAGMENT_PTR
  Define Structure that contains fragment of PCB. More...
- struct PCB_PTR
  Define PCB structure for RTCS adaptor. More...
- struct PCB2_PTR
  Define PCB structure contains two fragments. More...
- struct pcb_queue
  Define PCB structure contains two fragments. More...
- struct ENET_ECB_STRUCT_PTR
  Define ECB structure contains protocol type and it's related service function. More...
- struct enet_8022_header_ptr
  Define 8022 header. More...
- struct ENET_COMMON_STATS_STRUCT_PTR
  Define common status structure. More...

Macros

- #define ENET_RX_RING_LEN (8)
  Define parameter for configuration.
- #define ENETPROT_IP 0x0800
  Define ENET protocol parameter.
- #define ENET_OPTION_HW_TX_IP_CHECKSUM 0x00001000
  Define ENET option macro.
- #define ENET_DEFAULT_MAC_ADD { 0x00, 0x00, 0x5E, 0, 0, 0 }
  Define for ENET default MAC.
- #define htonl(p, x)
  Define macro for byte-swap.
- #define QUEUEADD(head, tail, pcb)
  Define add to queue.
- #define QUEUEGET(head, tail, pcb)
  Define get from queue.

Typedefs

- typedef void * task_param_t
  Define Error codes.
- typedef unsigned char _enet_address [6]
  Define for ENET six-byte MAC type.
- typedef void * _enet_handle
  Define the structure for ipcfg.

Variables

- unsigned long _RTCSTASK_priority
  Define Task parameter.

ENET RTCS ADAPTOR

- uint32_t ENET_initialize (uint32_t device, _enet_address address, uint32_t flag, _enet_handle *handle)
  Initialize the ENET device.
- uint32_t ENET_open (_enet_handle handle, uint16_t type, void(*service)(PCB_PTR, void *), void *private)
  Open the ENET device.
- uint32_t ENET_shutdown (_enet_handle handle)
  Shutdown the ENET device.
- static void ENET_receive (task_param_t param)
  ENET frame receive.
- uint32_t ENET_send (_enet_handle handle, PCB_PTR packet, uint32_t type, _enet_address dest, uint32_t flags)
  ENET frame transmit.
- uint32_t ENET_get_address (_enet_handle handle, _enet_address address)
  ENET get address with initialized device.
- uint32_t ENET_get_mac_address (uint32_t device, uint32_t value, _enet_address address)
  ENET get address with uninitialized device.
- uint32_t ENET_join (_enet_handle handle, uint16_t type, _enet_address address)
  ENET join a multicast group address.
- uint32_t ENET_leave (_enet_handle handle, uint16_t type, _enet_address address)
  ENET leave a multicast group address.
- bool ENET_link_status (_enet_handle handle)
  ENET get link status.
- uint32_t ENET_get_speed (_enet_handle handle)
  ENET get link speed.
- uint32_t ENET_get_MTU (_enet_handle handle)
  ENET get MTU.
- bool ENET_phy_registers (_enet_handle handle, uint32_t numRegs, uint32_t *regPtr)
  Get ENET PHY registers.
- uint32_t ENET_get_options (_enet_handle handle)
  Get ENET options.
- uint32_t ENET_close (_enet_handle handle, uint16_t type)
  Unregisters a protocol type on an Ethernet channel.
- uint32_t ENET_mediactl (_enet_handle handle, uint32_t commandId, void *inOutParam)
  ENET mediactl .
- _enet_handle ENET_get_next_device_handle (_enet_handle handle)
  Get the next ENET device handle address.
- void ENET_free (PCB_PTR packet)
  ENET free .
- const char * ENET_strerror (uint32_t error)
  ENET error description.

8.3.1 Data Structure Documentation

8.3.1.1 struct ENET_HEADER

Data Fields

- _enet_address DEST
  destination MAC address
- _enet_address SOURCE
  source MAC address
- unsigned char TYPE [2]
  protocol type

8.3.1.2 struct PCB_FRAGMENT

Data Fields

- uint32_t LENGTH
  Packet fragment length.
- unsigned char * FRAGMENT
  brief Pointer to fragment

8.3.1.3 struct PCB

Data Fields

- PCB_FREE_FPTR FREE
  Function that frees PCB.
- void * PRIVATE
  Private PCB information.
- PCB_FRAGMENT FRAG [1]
  Pointer to PCB fragment.

8.3.1.4 struct PCB2

Data Fields

- PCB_FREE_FPTR FREE
  Function that frees PCB.
- void * PRIVATE
  Private PCB information.
- PCB_FRAGMENT FRAG [2]
  Pointers to two PCB fragments.

8.3.1.5 struct pcb_queue

Data Fields

- PCB * pcbHead
  PCB buffer head.
- PCB * pcbTail
  PCB buffer tail.

8.3.1.6 struct ENET_ECB_STRUCT

8.3.1.7 struct enet_8022_header_t

Data Fields

- uint8_t dsap [1]
  DSAP region.
- uint8_t ssap [1]
  SSAP region.
- uint8_t command [1]
  Command region.
- uint8_t oui [3]
  OUI region.
- uint16_t type
  type region

8.3.1.8 struct ENET_COMMON_STATS_STRUCT

Data Fields

- uint32_t ST_RX_TOTAL
  Total number of received packets.
- uint32_t ST_RX_MISSED
  Number of missed packets.
- uint32_t ST_RX_DISCARDED
  Discarded unrecognized protocol.
- uint32_t ST_RX_ERRORS
  Discarded error during reception.
- uint32_t ST_TX_TOTAL
  Total number of transmitted packets.
- uint32_t ST_TX_MISSED
  Discarded transmit ring full.
- uint32_t ST_TX_DISCARDED
  Discarded bad packet.
- uint32_t ST_TX_ERRORS
  Error during transmission.

8.3.2 Function Documentation

8.3.2.1 uint32_t ENET_initialize (uint32_t *device*, _enet_address *address*, uint32_t *flag*, _enet_handle * *handle*)

Parameters

device	The ENET device number.
address	The hardware address.
flag	The flag for upper layer.
handle	The address pointer for ENET device structure.

Returns

The execution status.

8.3.2.2 uint32_t ENET_open (_enet_handle *handle,* uint16_t *type,* void(∗)(PCB_PTR, void ∗) *service,* void ∗ *private*)

Parameters

handle	The address pointer for ENET device structure.
type	The ENET protocol type.
service	The service function for type.
private	The private data for ENET device.

Returns

The execution status.

8.3.2.3 uint32_t ENET_shutdown (_enet_handle *handle*)

Parameters

handle	The address pointer for ENET device structure.

Returns

The execution status.

8.3.2.4 static void ENET_receive (task_param_t *param*) [static]

Parameters

enetIfPtr	The address pointer for ENET device structure.

8.3.2.5 uint32_t ENET_send (_enet_handle *handle*, PCB_PTR *packet*, uint32_t *type*, _enet_address *dest*, uint32_t *flags*)

Parameters

handle	The address pointer for ENET device structure.
packet	The ENET packet buffer.
type	The ENET protocol type.
dest	The destination hardware address.
flag	The flag for upper layer.

Returns

The execution status.

8.3.2.6 uint32_t ENET_get_address (_enet_handle *handle*, _enet_address *address*)

Parameters

handle	The address pointer for ENET device structure.
address	The destination hardware address.

Returns

The execution status.

8.3.2.7 uint32_t ENET_get_mac_address (uint32_t *device*, uint32_t *value*, _enet_address *address*)

Parameters

handle	The address pointer for ENET device structure.
value	The value to change the last three bytes of hardware.
address	The destination hardware address.

Returns

True if the execution status is success else false.

8.3.2.8 uint32_t ENET_join (_enet_handle *handle*, uint16_t *type*, _enet_address *address*)

Parameters

handle	The address pointer for ENET device structure.
type	The ENET protocol type.
address	The destination hardware address.

Returns

The execution status.

8.3.2.9 uint32_t ENET_leave (_enet_handle *handle*, uint16_t *type*, _enet_address *address*)

Parameters

handle	The address pointer for ENET device structure.
type	The ENET protocol type.
address	The destination hardware address.

Returns

The execution status.

8.3.2.10 bool ENET_link_status (_enet_handle *handle*)

Parameters

handle	The address pointer for ENET device structure.

Returns

The link status.

8.3.2.11 uint32_t ENET_get_speed (_enet_handle *handle*)

Parameters

handle	The address pointer for ENET device structure.

Returns

The link speed.

8.3.2.12 uint32_t ENET_get_MTU (_enet_handle *handle*)

Parameters

handle	The address pointer for ENET device structure.

Returns

The link MTU

8.3.2.13 bool ENET_phy_registers (_enet_handle *handle,* uint32_t *numRegs,* uint32_t * *regPtr*)

Parameters

handle	The address pointer for ENET device structure.

numRegs	The number of registers.
regPtr	The buffer for data read from PHY registers.

Returns

True if all numRegs registers are read succeed else false.

8.3.2.14 uint32_t ENET_get_options (_enet_handle *handle*)

Parameters

handle	The address pointer for ENET device structure.

Returns

ENET options.

8.3.2.15 uint32_t ENET_close (_enet_handle *handle,* uint16_t *type*)

Parameters

handle	The address pointer for ENET device structure.

Returns

ENET options.

8.3.2.16 uint32_t ENET_mediactl (_enet_handle *handle,* uint32_t *commandId,* void * *inOutParam*)

Parameters

handle	The address pointer for ENET device structure.

The	command Id.
The	buffer for input or output parameters.

Returns

ENET options.

8.3.2.17 _enet_handle ENET_get_next_device_handle (_enet_handle *handle*)

Parameters

handle	The address pointer for ENET device structure.

Returns

The address of next ENET device handle.

8.3.2.18 void ENET_free (PCB_PTR *packet*)

Parameters

packet	The buffer address.

8.3.2.19 const char* ENET_strerror (uint32_t *error*)

Parameters

error	The ENET error code.

Returns

The error string.

8.4 ENET Physical Layer Driver

The chapter describes the programming interface of the ENET Physical Layer Driver.

Chapter 9
FlexTimer (FTM)

The Kinetis SDK provides both HAL and Peripheral drivers for the FlexTimer (FTM) block of Kinetis devices.

Modules

- FlexTimer HAL driver
  The part describes the programming interface of the FlexTimer HAL driver.
- FlexTimer Peripheral Driver
  The part describes the programming interface of the FlexTimer Peripheral driver.

9.1 FlexTimer HAL driver

The chapter describes the programming interface of the FlexTimer HAL driver.

Data Structures

- union ftm_edge_mode_t
  FlexTimer edge mode. More...
- struct ftm_config_t
  FlexTimer module configuration. More...

Macros

- #define HW_FTM_CHANNEL_COUNT (8U)
  Number of channels for one FTM instance.
- #define HW_FTM_CHANNEL_PAIR_COUNT (4U)
  Number of combined channel of one FTM instance.
- #define HW_CHAN0 (0U)
  Channel number for CHAN0.
- #define HW_CHAN1 (1U)
  Channel number for CHAN1.
- #define HW_CHAN2 (2U)
  Channel number for CHAN2.
- #define HW_CHAN3 (3U)
  Channel number for CHAN3.
- #define HW_CHAN4 (4U)
  Channel number for CHAN4.
- #define HW_CHAN5 (5U)
  Channel number for CHAN5.
- #define HW_CHAN6 (6U)
  Channel number for CHAN6.
- #define HW_CHAN7 (7U)
  Channel number for CHAN7.

Enumerations

- enum ftm_clock_source_t
  FlexTimer clock source selection.
- enum ftm_counting_mode_t
  FlexTimer counting mode, up-down.
- enum ftm_clock_ps_t
  FlexTimer pre-scaler factor selection for the clock source.
- enum ftm_phase_t
  FlexTimer phase for the quadrature.
- enum ftm_deadtime_ps_t
  FlexTimer pre-scaler factor for the deadtime insertion.
- enum ftm_config_mode_t
  FlexTimer operation mode, capture, output, dual, or quad.

- enum ftm_input_capture_edge_mode_t
  FlexTimer input capture edge mode, rising edge, or falling edge.
- enum ftm_output_compare_edge_mode_t
  FlexTimer output compare edge mode.
- enum ftm_pwm_edge_mode_t
  FlexTimer PWM output pulse mode, high-true or low-true on match up.
- enum ftm_dual_capture_edge_mode_t
  FlexTimer dual capture edge mode, one shot or continuous.

Functions

- static void ftm_hal_set_clock_source (uint8_t instance, ftm_clock_source_t clock)
  Sets the FTM clock source.
- static void ftm_hal_set_clock_ps (uint8_t instance, ftm_clock_ps_t ps)
  Sets the FTM clock divider.
- static void ftm_hal_enable_timer_overflow_interrupt (uint8_t instance)
  Enables the FTM peripheral timer overflow interrupt.
- static void ftm_hal_disable_timer_overflow_interrupt (uint8_t instance)
  Disables the FTM peripheral timer overflow interrupt.
- static bool ftm_is_timer_overflow (uint8_t instance)
  Returns the FTM peripheral timer overflow interrupt flag.
- static void ftm_hal_set_cpwms (uint8_t instance, uint8_t mode)
  Sets the FTM center-aligned PWM select.
- static void ftm_hal_set_counter (uint8_t instance, uint16_t val)
  Sets the FTM peripheral current counter value.
- static uint16_t ftm_hal_get_counter (uint8_t instance)
  Returns the FTM peripheral current counter value.
- static void ftm_hal_set_mod (uint8_t instance, uint16_t val)
  Sets the FTM peripheral timer modulo value.
- static uint16_t ftm_hal_get_mod (uint8_t instance)
  Returns the FTM peripheral counter modulo value.
- static void ftm_hal_set_counter_init_val (uint8_t instance, uint16_t val)
  Sets the FTM peripheral timer counter initial value.
- static uint16_t ftm_hal_get_counter_init_val (uint8_t instance)
  Returns the FTM peripheral counter initial value.
- static void ftm_hal_set_channel_MSnBA_mode (uint8_t instance, uint8_t channel, uint8_t selection)
  Sets the FTM peripheral timer channel mode.
- static void ftm_hal_set_channel_edge_level (uint8_t instance, uint8_t channel, uint8_t level)
  Sets the FTM peripheral timer channel edge level.
- static uint8_t ftm_hal_get_channel_mode (uint8_t instance, uint8_t channel)
  Gets the FTM peripheral timer channel mode.
- static uint8_t ftm_hal_get_channel_edge_level (uint8_t instance, uint8_t channel)
  Gets the FTM peripheral timer channel edge level.
- static void ftm_hal_enable_channle_dma (uint8_t instance, uint8_t channel, bool val)
  Enables or disables the FTM peripheral timer channel DMA.
- static bool ftm_hal_is_channel_dma (uint8_t instance, uint8_t channel, bool val)
  Returns whether the FTM peripheral timer channel DMA is enabled.
- static void ftm_hal_enable_channel_interrupt (uint8_t instance, uint8_t channel)
  Enables the FTM peripheral timer channel(n) interrupt.

- static void ftm_hal_disable_channel_interrupt (uint8_t instance, uint8_t channel)
  Disables the FTM peripheral timer channel(n) interrupt.
- static bool ftm_is_channel_event_occurred (uint8_t instance, uint8_t channel)
  Returns whether any event for the FTM peripheral timer channel has occurred.
- static void ftm_hal_set_channel_count_value (uint8_t instance, uint8_t channel, uint16_t val)
  Sets the FTM peripheral timer channel counter value.
- static uint16_t ftm_hal_get_channel_count_value (uint8_t instance, uint8_t channel, uint16_t val)
  Gets the FTM peripheral timer channel counter value.
- static uint32_t ftm_hal_get_channel_event_status (uint8_t instance, uint8_t channel)
  Gets the FTM peripheral timer channel event status.
- static void ftm_hal_clear_channel_event_status (uint8_t instance, uint8_t channel)
  Clears the FTM peripheral timer all channel event status.
- static void ftm_hal_set_channel_output_mask (uint8_t instance, uint8_t channel, bool mask)
  Sets the FTM peripheral timer channel output mask.
- static void ftm_hal_set_channel_output_init_state (uint8_t instance, uint8_t channel, uint8_t state)
  Sets the FTM peripheral timer channel output initial state 0 or 1.
- static void ftm_hal_set_channel_output_polarity (uint8_t instance, uint8_t channel, uint8_t pol)
  Sets the FTM peripheral timer channel output polarity.
- static void ftm_hal_set_channel_fault_input_polarity (uint8_t instance, uint8_t channel, uint8_t pol)
  Sets the FTM peripheral timer channel input polarity.
- static void ftm_hal_enable_fault_interrupt (uint8_t instance)
  Enables the FTM peripheral timer fault interrupt.
- static void ftm_hal_disable_fault_interrupt (uint8_t instance)
  Disables the FTM peripheral timer fault interrupt.
- static void ftm_hal_set_fault_control_mode (uint8_t instance, uint8_t mode)
  Sets the FTM peripheral timer fault control mode.
- static void ftm_hal_enable_capture_test (uint8_t instance, bool enable)
  Enables the FTM peripheral timer capture test.
- static void ftm_hal_enable_write_protection (uint8_t instance, bool enable)
  Enables the FTM peripheral timer write protection.
- static void ftm_hal_ftm_enable (uint8_t instance, bool enable)
  Enables the FTM peripheral timer group.
- static void ftm_hal_enable_channel_init_output (uint8_t instance, bool enable)
  Enables the FTM peripheral timer channel output initialization.
- static void ftm_hal_set_pwm_sync_mdoe (uint8_t instance, bool enable)
  Sets the FTM peripheral timer sync mode.
- static void ftm_hal_enable_software_trigger (uint8_t instance, bool enable)
  Enables the FTM peripheral timer software trigger.
- void ftm_hal_set_hardware_trigger (uint8_t instance, uint8_t trigger_num, bool enable)
  Sets the FTM peripheral timer hardware trigger.
- static void ftm_hal_enable_output_mask_sync_by_pwm (uint8_t instance, bool enable)
  Enables the FTM peripheral timer output mask update by PWM sync.
- static void ftm_hal_enable_count_reinit_sync (uint8_t instance, bool enable)
  Enables the FTM peripheral timer counter re-initialized by sync.
- static void ftm_hal_enable_max_loading (uint8_t instance, bool enable)
  Enables the FTM peripheral timer maximum loading points.
- static void ftm_hal_enable_min_loading (uint8_t instance, bool enable)
  Enables the FTM peripheral timer minimum loading points.
- static uint32_t get_channel_pair_index (uint8_t channel)
  Combines the channel control.
- static void ftm_hal_enable_dual_capture (uint8_t instance, uint8_t channel, bool enable)

Enables the FTM peripheral timer dual edge capture mode.
- static void ftm_hal_enable_dual_channel_fault (uint8_t instance, uint8_t channel, bool enable)
  Enables the FTM peripheral timer channel pair fault control.
- static void ftm_hal_enable_dual_channel_pwm_sync (uint8_t instance, uint8_t channel, bool enable)
  Enables the FTM peripheral timer channel pair counter PWM sync.
- static void ftm_hal_enable_dual_channel_deadtime (uint8_t instance, uint8_t channel, bool enable)
  Enables the FTM peripheral timer channel pair deadtime.
- static void ftm_hal_enable_dual_channel_decap (uint8_t instance, uint8_t channel, bool enable)
  Enables the FTM peripheral timer channel dual edge capture decap, not decapen.
- static void ftm_hal_enable_dual_channel_comp (uint8_t instance, uint8_t channel, bool enable)
  Enables the FTM peripheral timer channel pair output complement mode.
- static void ftm_hal_enable_dual_channel_combine (uint8_t instance, uint8_t channel, bool enable)
  Enables the FTM peripheral timer channel pair output combine mode.
- static void ftm_hal_set_deadtime_prescale (uint8_t instance, ftm_deadtime_ps_t divider)
  Set the FTM deadtime divider.
- static void ftm_hal_set_deadtime_count (uint8_t instance, uint8_t count)
  Sets the FTM deadtime value.
- void ftm_hal_enable_channel_trigger (uint8_t instance, uint8_t channel, bool val)
  Enables the generation of the FTM peripheral timer channel trigger when the FTM counter is equal to its initial value.
- static bool ftm_hal_is_channel_trigger_generated (uint8_t instance, uint8_t channel)
  Checks whether any channel trigger event has occurred.
- static uint8_t ftm_hal_get_detected_fault_input (uint8_t instance)
  Gets the FTM detected fault input.
- static bool ftm_hal_is_write_protection_enable (uint8_t instance)
  Checks whether the write protection is enabled.
- static void ftm_hal_enable_quad_capture (uint8_t instance, bool enable)
  Enables the channel quadrature decoder.
- void ftm_hal_set_channel_input_capture_filter (uint8_t instance, uint8_t channel, uint8_t val)
  Sets the FTM peripheral timer channel input capture filter value.
- static void ftm_hal_enable_channel_fault_input_filter (uint8_t instance, uint8_t channel, bool val)
  Enables the channel input filter.
- static void ftm_hal_enable_channel_fault_input (uint8_t instance, uint8_t channel, bool val)
  Enables the channel fault input.
- static void ftm_hal_enable_dual_channel_invert (uint8_t instance, uint8_t channel, bool val)
  Enables the channel invert.
- static void ftm_hal_enable_channel_software_ctrl (uint8_t instance, uint8_t channel, bool val)
  Enables the channel software control.
- static void ftm_hal_set_channel_software_ctrl_val (uint8_t instance, uint8_t channel, bool val)
  Sets the channel software control value.
- static void ftm_hal_enable_pwm_load (uint8_t instance, bool enable)
  Enables the FTM timer PWM loading of MOD, CNTIN and CV.
- static void ftm_hal_enable_pwm_load_matching_channel (uint8_t instance, uint8_t channel, bool val)
  Enables the channel matching process.
- static void ftm_hal_enable_global_time_base_output (uint8_t instance, bool enable)
  Enables the FTM timer global time base output.
- static void ftm_hal_enable_global_time_base (uint8_t instance, bool enable)
  Enables the FTM timer global time base.
- static void ftm_hal_set_bdm_mode (uint8_t instance, uint8_t val)

Sets the FTM timer TOF Frequency.
- static void ftm_hal_set_tof_frequency (uint8_t instance, uint8_t val)
  Sets the BDM mode.
- static void ftm_hal_enable_hardware_sync_software_output_ctrl (uint8_t instance, bool enable)
  Enables the FTM timer hardware sync activation.
- static void ftm_hal_enable_hardware_sync_invert_ctrl (uint8_t instance, bool enable)
  Enables the FTM timer hardware inverting control sync.
- static void ftm_hal_enable_hardware_sync_output_mask (uint8_t instance, bool enable)
  Enables the FTM timer hardware outmask sync.
- static void ftm_hal_enable_hardware_sycn_mod_cntin_cv (uint8_t instance, bool enable)
  MOD, CNTIN, and CV registers synchronization is activated.
- static void ftm_hal_enable_hardware_sync_counter (uint8_t instance, bool enable)
  The FTM counter synchronization is activated by a hardware trigger.
- static void ftm_hal_enable_pwm_sync_swoctrl (uint8_t instance, bool enable)
  Enables the FTM timer software sync activation.
- static void ftm_hal_enable_enhanced_pwm_sync_mdoe (uint8_t instance, bool enable)
  Enables the FTM timer enhanced PWM sync mode.
- static void ftm_hal_enable_software_sync_swoctrl (uint8_t instance, bool enable)
  Enables the FTM timer software output control sync.
- static void ftm_hal_enable_software_sync_invert_ctrl (uint8_t instance, bool enable)
  Enables the FTM timer software inverting control sync.
- static void ftm_hal_enable_software_sync_output_mask (uint8_t instance, bool enable)
  Enables the FTM timer software outmask sync.
- static void ftm_hal_enable_software_sycn_mod_cntin_cv (uint8_t instance, bool enable)
  Enables the FTM timer software outmask sync.
- static void ftm_hal_enable_software_sync_counter (uint8_t instance, bool enable)
  Enables the FTM timer counter software sync.
- static void ftm_hal_enable_invert_sync_with_rising_edge (uint8_t instance, bool enable)
  Enables the FTM timer INVCTRL update by PWM.
- static void ftm_hal_enable_cntin_sync_with_rising_edge (uint8_t instance, bool enable)
  Enables the FTM timer cntin update by PWM.
- void ftm_hal_reset (uint8_t instance)
  Resets the FTM registers.
- void ftm_hal_init (uint8_t instance, ftm_config_t *config)
  Initializes the FTM.
- void ftm_hal_enable_pwm_mode (uint8_t instance, ftm_config_t *config)
  Enables the FTM timer when it is PWM output mode.
- void ftm_hal_disable_pwm_mode (uint8_t instance, ftm_config_t *config)
  Initializes the FTM timer when it is PWM output mode.

9.1.1 Data Structure Documentation

9.1.1.1 union ftm_edge_mode_t

9.1.1.2 struct ftm_config_t

9.1.2 Macro Definition Documentation

9.1.2.1 #define HW_FTM_CHANNEL_COUNT (8U)

9.1.2.2 #define HW_FTM_CHANNEL_PAIR_COUNT (4U)

9.1.2.3 #define HW_CHAN0 (0U)

9.1.2.4 #define HW_CHAN1 (1U)

9.1.2.5 #define HW_CHAN2 (2U)

9.1.2.6 #define HW_CHAN3 (3U)

9.1.2.7 #define HW_CHAN4 (4U)

9.1.2.8 #define HW_CHAN5 (5U)

9.1.2.9 #define HW_CHAN6 (6U)

9.1.2.10 #define HW_CHAN7 (7U)

9.1.3 Enumeration Type Documentation

9.1.3.1 enum ftm_output_compare_edge_mode_t

Toggle, clear or set.

9.1.4 Function Documentation

9.1.4.1 static void ftm_hal_set_clock_source (uint8_t *instance,* ftm_clock_source_t *clock*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
clock	The FTM peripheral clock selection bits:00: No clock 01: system clock 10 :fixed clock 11:External clock

9.1.4.2 static void ftm_hal_set_clock_ps (uint8_t *instance,* ftm_clock_ps_t *ps*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
ps	The FTM peripheral clock pre-scale divider

9.1.4.3 static void ftm_hal_enable_timer_overflow_interrupt (uint8_t *instance*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

9.1.4.4 static void ftm_hal_disable_timer_overflow_interrupt (uint8_t *instance*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

9.1.4.5 static bool ftm_is_timer_overflow (uint8_t *instance*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

Return values

true	if overflow, false if not

9.1.4.6 static void ftm_hal_set_cpwms (uint8_t *instance,* uint8_t *mode*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
mode	1:upcounting mode 0:up_down counting mode

9.1.4.7 static void ftm_hal_set_counter (uint8_t *instance,* uint16_t *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
val	FTM timer counter value to be set

9.1.4.8 static uint16_t ftm_hal_get_counter (uint8_t *instance*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

Return values

current	FTM timer counter value

9.1.4.9 static void ftm_hal_set_mod (uint8_t *instance,* uint16_t *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
val	The value to be set to the timer modulo

9.1.4.10 static uint16_t ftm_hal_get_mod (uint8_t *instance*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

Return values

FTM	timer modulo value

9.1.4.11 static void ftm_hal_set_counter_init_val (uint8_t *instance,* uint16_t *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
val	initial value to be set

9.1.4.12 static uint16_t ftm_hal_get_counter_init_val (uint8_t *instance*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

Return values

FTM	timer counter initial value

9.1.4.13 static void ftm_hal_set_channel_MSnBA_mode (uint8_t *instance,* uint8_t *channel,* uint8_t *selection*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
selection	The mode to be set valid value MSnB:MSnA :00,01, 10, 11

9.1.4.14 static void ftm_hal_set_channel_edge_level (uint8_t *instance,* uint8_t *channel,* uint8_t *level*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
level	The rising or falling edge to be set, valid value ELSnB:ELSnA :00,01, 10, 11

9.1.4.15 static uint8_t ftm_hal_get_channel_mode (uint8_t *instance,* uint8_t *channel*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number

Return values

The	MSnB:MSnA mode value, will be 00,01, 10, 11

9.1.4.16 static uint8_t ftm_hal_get_channel_edge_level (uint8_t *instance,* uint8_t *channel*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

channel	The FTM peripheral channel number

Return values

The	ELSnB:ELSnA mode value, will be 00,01, 10, 11

9.1.4.17 static void ftm_hal_enable_channle_dma (uint8_t *instance,* uint8_t *channel,* bool *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
val	enable or disable

9.1.4.18 static bool ftm_hal_is_channel_dma (uint8_t *instance,* uint8_t *channel,* bool *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number

Return values

true	if enabled, false if disabled

9.1.4.19 static void ftm_hal_enable_channel_interrupt (uint8_t *instance,* uint8_t *channel*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

channel	The FTM peripheral channel number

9.1.4.20 static void ftm_hal_disable_channel_interrupt (uint8_t *instance,* uint8_t *channel*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number

9.1.4.21 static bool ftm_is_channel_event_occurred (uint8_t *instance,* uint8_t *channel*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number

Return values

true	if event occurred, false otherwise.

9.1.4.22 static void ftm_hal_set_channel_count_value (uint8_t *instance,* uint8_t *channel,* uint16_t *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
val	counter value to be set

9.1.4.23 static uint16_t ftm_hal_get_channel_count_value (uint8_t *instance,* uint8_t *channel,* uint16_t *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number

Return values

val	return current channel counter value

9.1.4.24 static uint32_t ftm_hal_get_channel_event_status (uint8_t *instance,* uint8_t *channel*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number

Return values

val	return current channel event status value

9.1.4.25 static void ftm_hal_clear_channel_event_status (uint8_t *instance,* uint8_t *channel*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number

Return values

val	return current channel counter value

9.1.4.26 static void ftm_hal_set_channel_output_mask (uint8_t *instance,* uint8_t *channel,* bool *mask*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
mask	mask to be set 0 or 1, unmasked or masked

9.1.4.27 static void ftm_hal_set_channel_output_init_state (uint8_t *instance,* uint8_t *channel,* uint8_t *state*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
state	counter value to be set 0 or 1

9.1.4.28 static void ftm_hal_set_channel_output_polarity (uint8_t *instance,* uint8_t *channel,* uint8_t *pol*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
pol	polarity to be set 0 or 1

9.1.4.29 static void ftm_hal_set_channel_fault_input_polarity (uint8_t *instance,* uint8_t *channel,* uint8_t *pol*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
pol	polarity to be set, 0: active high, 1:active low

9.1.4.30 static void ftm_hal_enable_fault_interrupt (uint8_t *instance*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

9.1.4.31 static void ftm_hal_disable_fault_interrupt (uint8_t *instance*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

9.1.4.32 static void ftm_hal_set_fault_control_mode (uint8_t *instance,* uint8_t *mode*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
mode,valid	number bits:00, 01, 10,11 (1, 2, 3, 4)

9.1.4.33 static void ftm_hal_enable_capture_test (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	true to enable, false to disable

9.1.4.34 static void ftm_hal_enable_write_protection (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

enable	true to enable, false to disable

9.1.4.35 static void ftm_hal_ftm_enable (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.36 static void ftm_hal_enable_channel_init_output (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.37 static void ftm_hal_set_pwm_sync_mdoe (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True no restriction both software and hardware sync, false only software sync.

9.1.4.38 static void ftm_hal_enable_software_trigger (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number.

enable	True to enable, false to disable

9.1.4.39 void ftm_hal_set_hardware_trigger (uint8_t *instance,* uint8_t *trigger_num,* bool *enable*)

Parameters

instance	The FTM peripheral instance number
trigger_num	0, 1,2 for trigger0, trigger1 and trigger3
enable	True to enable, 1 to enable

9.1.4.40 static void ftm_hal_enable_output_mask_sync_by_pwm (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable PWM sync, false to enable outmask in the rising edges of the system clock

9.1.4.41 static void ftm_hal_enable_count_reinit_sync (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to update FTM counter when triggered , false to count normally

9.1.4.42 static void ftm_hal_enable_max_loading (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.43 static void ftm_hal_enable_min_loading (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.44 static uint32_t get_channel_pair_index (uint8_t *channel*) [static]

Returns an index for each channel pair.

Parameters

channel	The FTM peripheral channel number.

Returns

0 for channel pair 0 & 1
1 for channel pair 2 & 3
2 for channel pair 4 & 5
3 for channel pair 6 & 7

9.1.4.45 static void ftm_hal_enable_dual_capture (uint8_t *instance,* uint8_t *channel,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
enable	True to enable, false to disable

9.1.4.46 static void ftm_hal_enable_dual_channel_fault (uint8_t *instance,* uint8_t *channel,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
enable	True to enable, false to disable

9.1.4.47 static void ftm_hal_enable_dual_channel_pwm_sync (uint8_t *instance,* uint8_t *channel,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
enable	True to enable, false to disable

9.1.4.48 static void ftm_hal_enable_dual_channel_deadtime (uint8_t *instance,* uint8_t *channel,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
enable	True to enable, false to disable

9.1.4.49 static void ftm_hal_enable_dual_channel_decap (uint8_t *instance,* uint8_t *channel,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
enable	True to enable, false to disable

9.1.4.50 static void ftm_hal_enable_dual_channel_comp (uint8_t *instance,* uint8_t *channel,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
enable	True to enable, false to disable

9.1.4.51 static void ftm_hal_enable_dual_channel_combine (uint8_t *instance,* uint8_t *channel,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
enable	True to enable channel pair to combine, false to disable

9.1.4.52 static void ftm_hal_set_deadtime_prescale (uint8_t *instance,* ftm_deadtime_ps_t *divider*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
divider	The FTM peripheral prescale divider 0x :divided by 1, 10: divided by 4 11:divided by 16

9.1.4.53 static void ftm_hal_set_deadtime_count (uint8_t *instance,* uint8_t *count*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
divider	The FTM peripheral prescale divider count: 0, no counts inserted 1: 1 count is inserted 2: 2 count is inserted....

9.1.4.54 void ftm_hal_enable_channel_trigger (uint8_t *instance,* uint8_t *channel,* bool *val*)

Channels 6 and 7 cannot be used as triggers.

Parameters

instance	The FTM peripheral instance number
channel	Channel to be enabled, valid value 0, 1, 2, 3, 4, 5
enable	True to enable, false to disable

9.1.4.55 static bool ftm_hal_is_channel_trigger_generated (uint8_t *instance,* uint8_t *channel*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

Return values

True	if there is a trigger event, false if not.

9.1.4.56 static uint8_t ftm_hal_get_detected_fault_input (uint8_t *instance*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

Return values

Return	faulty byte

9.1.4.57 static bool ftm_hal_is_write_protection_enable (uint8_t *instance*) [inline], [static]

Parameters

instance	The FTM peripheral instance number

Return values

True	if enabled, false if not

9.1.4.58 static void ftm_hal_enable_quad_capture (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.59 void ftm_hal_set_channel_input_capture_filter (uint8_t *instance,* uint8_t *channel,* uint8_t *val*)

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number, only 0,1,2,3, channel 4, 5,6, 7 don't have.
val	Filter value to be set

9.1.4.60 static void ftm_hal_enable_channel_fault_input_filter (uint8_t *instance,* uint8_t *channel,* bool *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	Channel to be enabled, valid value 0, 1, 2, 3
enable	True to enable, false to disable

9.1.4.61 static void ftm_hal_enable_channel_fault_input (uint8_t *instance,* uint8_t *channel,* bool *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	Channel to be enabled, valid value 0, 1, 2, 3
enable	True to enable, false to disable

9.1.4.62 static void ftm_hal_enable_dual_channel_invert (uint8_t *instance,* uint8_t *channel,* bool *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	The FTM peripheral channel number
enable	True to enable, false to disable

9.1.4.63 static void ftm_hal_enable_channel_software_ctrl (uint8_t *instance,* uint8_t *channel,* bool *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	Channel to be enabled, valid value 0, 1, 2, 3, 4,5,6,7
enable	True to enable, false to disable

9.1.4.64 static void ftm_hal_set_channel_software_ctrl_val (uint8_t *instance,* uint8_t *channel,* bool *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number.
channel	Channel to be enabled, valid value 0, 1, 2, 3,5,6,7,
bool	True to set 1, false to set 0

9.1.4.65 static void ftm_hal_enable_pwm_load (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.66 static void ftm_hal_enable_pwm_load_matching_channel (uint8_t *instance,* uint8_t *channel,* bool *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
channel	Channel to be enabled, valid value 0, 1, 2, 3, 4,5,6,7
enable	True to enable, false to disable

9.1.4.67 static void ftm_hal_enable_global_time_base_output (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.68 static void ftm_hal_enable_global_time_base (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.69 static void ftm_hal_set_bdm_mode (uint8_t *instance,* uint8_t *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
val	Value of the TOF bit set frequency

9.1.4.70 static void ftm_hal_set_tof_frequency (uint8_t *instance,* uint8_t *val*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
val	Value of the TOF bit set frequency

9.1.4.71 static void ftm_hal_enable_hardware_sync_software_output_ctrl (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.72 static void ftm_hal_enable_hardware_sync_invert_ctrl (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.73 static void ftm_hal_enable_hardware_sync_output_mask (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.74 static void ftm_hal_enable_hardware_sycn_mod_cntin_cv (uint8_t *instance,* bool *enable*) [inline], [static]

A hardware trigger activates the synchronization.

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.75 static void ftm_hal_enable_hardware_sync_counter (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.76 static void ftm_hal_enable_pwm_sync_swoctrl (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.77 static void ftm_hal_enable_enhanced_pwm_sync_mdoe (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.78 static void ftm_hal_enable_software_sync_swoctrl (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.79 static void ftm_hal_enable_software_sync_invert_ctrl (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.80 static void ftm_hal_enable_software_sync_output_mask (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.81 static void ftm_hal_enable_software_sycn_mod_cntin_cv (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable.

9.1.4.82 static void ftm_hal_enable_software_sync_counter (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to enable, false to disable

9.1.4.83 static void ftm_hal_enable_invert_sync_with_rising_edge (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to update with PWM, false to update with rising edge of system clock.

9.1.4.84 static void ftm_hal_enable_cntin_sync_with_rising_edge (uint8_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The FTM peripheral instance number
enable	True to update with PWM, false to update with rising edge of system clock.

9.1.4.85 void ftm_hal_reset (uint8_t *instance*)

Parameters

instance	The FTM peripheral instance number

9.1.4.86 void ftm_hal_init (uint8_t *instance,* ftm_config_t * *config*)

Parameters

instance	The FTM peripheral instance number.

9.1.4.87 void ftm_hal_enable_pwm_mode (uint8_t *instance,* ftm_config_t * *config*)

Parameters

instance	The FTM peripheral instance number
config	pwm config parameter

9.1.4.88 void ftm_hal_disable_pwm_mode (uint8_t *instance,* ftm_config_t * *config*)

Parameters

instance	The FTM peripheral instance number

9.2 FlexTimer Peripheral Driver

The chapter describes the programming interface of the FlexTimer Peripheral driver.

Data Structures

- struct ftm_channel_info_t
  Channel information. More...
- struct ftm_combined_channel_info_t
  Combines channel information. More...
- struct ftm_user_config_t
  Internal driver state information grouped by naming. More...
- struct ftm_pwm_param_t
  FlexTimer driver PWM parameter. More...

Functions

- void ftm_init (uint8_t instance, ftm_user_config_t *info)
  Initializes the FTM driver.
- void ftm_shutdown (ftm_user_config_t *state)
  Shuts down the FTM driver.
- void ftm_pwm_start (ftm_user_config_t *info, uint8_t channel)
  Starts channel PWM.
- void ftm_pwm_stop (ftm_user_config_t *info, uint8_t channel)
  Stops channel PWM.
- void ftm_pwm_configure (ftm_user_config_t *info, uint8_t channel, ftm_pwm_param_t *param)
  Configures duty cycle and frequency of the channel PWM.

9.2.1 Data Structure Documentation

9.2.1.1 struct ftm_channel_info_t

Data Fields

- ftm_config_mode_t mode
  FlexTimer operation mode.
- ftm_edge_mode_t edge_mode
  capture mode
- bool isSoftwareOutput
  1:output high 0:output low

9.2.1.2 struct ftm_combined_channel_info_t

9.2.1.3 struct ftm_user_config_t

User needs to set the relevant ones.

Data Fields

- uint8_t instance
  name FTM instance FTM0, FTM1, FTM2, FTM3

9.2.1.4 struct ftm_pwm_param_t

Setting uPulseHigPercentage = 1, uPulseLowPulsePercentage = 1 means sin wave equal to low and high width.

Data Fields

- uint32_t uFrequencyHZ
  PWM period.
- uint32_t uPulseHighPercentage
  High pulse width period.
- uint32_t uPulseLowPercentage
  Low pulse width period.
- uint16_t uCnV
  In combined mode, the n channel count value, the duty cycle=|Cn+1V -CnV|.

9.2.2 Function Documentation

9.2.2.1 void ftm_init (uint8_t *instance,* ftm_user_config_t * *info*)

Parameters

instance	The FTM peripheral instance number.
info	The FTM user config structure.

9.2.2.2 void ftm_shutdown (ftm_user_config_t * *state*)

Parameters

state	The FTM user config structure.

9.2.2.3 void ftm_pwm_start (ftm_user_config_t * *info,* uint8_t *channel*)

Parameters

info	The FTM user config structure.
channel	The channel or channel pair number(combined mode).

9.2.2.4 void ftm_pwm_stop (ftm_user_config_t * *info,* uint8_t *channel*)

Parameters

info	The FTM user config structure.
channel	The channel or channel pair number(combined mode).

9.2.2.5 void ftm_pwm_configure (ftm_user_config_t * *info,* uint8_t *channel,* ftm_pwm_param_t * *param*)

Parameters

info	The FTM user config structure.
channel	The channel or channel pair number(combined mode). For channel pair, channel= 0(0,1),1(2,3),2(4,5),3(6,7)
param	FTM driver PWM parameter to configure PWM options

Chapter 10
General Purpose Input/Output (GPIO)

The Kinetis SDK provides both HAL and Peripheral drivers for the General Purpose Input/Output (GPIO) block of Kinetis devices.

Modules

- GPIO HAL driver
  The part describes the programming interface of the GPIO HAL driver.
- GPIO ISR Definitions
  The part describes the programming interface of the GPIO interrupt definitions.
- GPIO Peripheral Driver
  The part describes the programming interface of the GPIO Peripheral driver.

10.1 GPIO HAL driver

The chapter describes the programming interface of the GPIO HAL driver.

Enumerations

- enum gpio_pin_direction_t {
  kGpioDigitalInput = 0,
  kGpioDigitalOutput = 1 }
  GPIO direction definition.

Configuration

- void gpio_hal_set_pin_direction (uint32_t instance, uint32_t pin, gpio_pin_direction_t direction)
  Sets the individual GPIO pin to general input or output.
- static void gpio_hal_set_port_direction (uint32_t instance, uint32_t direction)
  Sets the GPIO port pins to general input or output.

Status

- static uint32_t gpio_hal_get_pin_direction (uint32_t instance, uint32_t pin)
  Gets the current direction of the individual GPIO pin.
- static uint32_t gpio_hal_get_port_direction (uint32_t instance)
  Gets the GPIO port pins direction.

Output Operation

- void gpio_hal_write_pin_output (uint32_t instance, uint32_t pin, uint32_t output)
  Sets the output level of the individual GPIO pin to logic 1 or 0.
- static uint32_t gpio_hal_read_pin_output (uint32_t instance, uint32_t pin)
  Reads the current pin output.
- static void gpio_hal_set_pin_output (uint32_t instance, uint32_t pin)
  Sets the output level of the individual GPIO pin to logic 1.
- static void gpio_hal_clear_pin_output (uint32_t instance, uint32_t pin)
  Clears the output level of the individual GPIO pin to logic 0.
- static void gpio_hal_toggle_pin_output (uint32_t instance, uint32_t pin)
  Reverses the current output logic of the individual GPIO pin.
- static void gpio_hal_write_port_output (uint32_t instance, uint32_t portOutput)
  Sets the output of the GPIO port to a specific logic value.
- static uint32_t gpio_hal_read_port_output (uint32_t instance)
  Reads out all pin output status of the current port.

Input Operation

- static uint32_t gpio_hal_read_pin_input (uint32_t instance, uint32_t pin)

Reads the current input value of the individual GPIO pin.

- static uint32_t gpio_hal_read_port_input (uint32_t instance)
  Reads the current input value of a specific GPIO port.

10.1.0.6 GPIO HAL Driver

Overview

The GPIO HAL driver is designed to access GPIO hardware registers. It provides sets of APIs for users to configure and control GPIO ports/pins.

10.1.1 Enumeration Type Documentation

10.1.1.1 enum gpio_pin_direction_t

Enumerator

kGpioDigitalInput Set current pin as digital input.
kGpioDigitalOutput Set current pin as digital output.

10.1.2 Function Documentation

10.1.2.1 void gpio_hal_set_pin_direction (uint32_t *instance,* uint32_t *pin,* gpio_pin_direction_t *direction*)

Parameters

instance	GPIO instance number(HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)
pin	GPIO port pin number
direction	GPIO directions • kGpioDigitalInput: set to input • kGpioDigitalOutput: set to output

10.1.2.2 static void gpio_hal_set_port_direction (uint32_t *instance,* uint32_t *direction*) [inline], [static]

This function operates all 32 port pins.

Parameters

instance	GPIO instance number (HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)
direction	GPIO directions • 0: set to input • 1: set to output • LSB: pin 0 • MSB: pin 31

10.1.2.3 static uint32_t gpio_hal_get_pin_direction (uint32_t *instance,* uint32_t *pin*) [inline], [static]

Parameters

instance	GPIO instance number(HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)
pin	GPIO port pin number

Returns

GPIO directions
- 0: corresponding pin is set to input.
- 1: corresponding pin is set to output.

10.1.2.4 static uint32_t gpio_hal_get_port_direction (uint32_t *instance*) [inline], [static]

This function gets all 32-pin directions as a 32-bit integer.

Parameters

instance	GPIO instance number (HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)

Returns

GPIO directions. Each bit represents one pin. For each bit:
- 0: corresponding pin is set to input
- 1: corresponding pin is set to output
- LSB: pin 0
- MSB: pin 31

10.1.2.5 void gpio_hal_write_pin_output (uint32_t *instance,* uint32_t *pin,* uint32_t *output*)

Parameters

instance	GPIO instance number(HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)
pin	GPIO port pin number
output	pin output logic level

10.1.2.6 static uint32_t gpio_hal_read_pin_output (uint32_t *instance,* uint32_t *pin*) [inline], [static]

Parameters

instance	GPIO instance number (HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)
pin	GPIO port pin number

Returns

current pin output status. 0 - Low logic, 1 - High logic

10.1.2.7 static void gpio_hal_set_pin_output (uint32_t *instance,* uint32_t *pin*) [inline], [static]

Parameters

instance	GPIO instance number(HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)
pin	GPIO port pin number

10.1.2.8 static void gpio_hal_clear_pin_output (uint32_t *instance,* uint32_t *pin*) [inline], [static]

Parameters

instance	GPIO instance number(HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)
pin	GPIO port pin number

10.1.2.9 static void gpio_hal_toggle_pin_output (uint32_t *instance,* uint32_t *pin*) [inline], [static]

Parameters

instance	GPIO instance number(HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)
pin	GPIO port pin number

10.1.2.10 static void gpio_hal_write_port_output (uint32_t *instance,* uint32_t *portOutput*) [inline], [static]

This function operates all 32 port pins.

Parameters

instance	GPIO instance number (HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)
portOutput	data to configure the GPIO output. Each bit represents one pin. For each bit: • 0: set logic level 0 to pin • 1: set logic level 1 to pin • LSB: pin 0 • MSB: pin 31

10.1.2.11 static uint32_t gpio_hal_read_port_output (uint32_t *instance*) [inline], [static]

This function operates all 32 port pins.

Parameters

instance	GPIO instance number (HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)

Returns

current port output status. Each bit represents one pin. For each bit:
- 0: corresponding pin is outputting logic level 0
- 1: corresponding pin is outputting logic level 1
- LSB: pin 0
- MSB: pin 31

10.1.2.12 static uint32_t gpio_hal_read_pin_input (uint32_t *instance,* uint32_t *pin*) [inline], [static]

Parameters

instance	GPIO instance number(HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)
pin	GPIO port pin number

Returns

GPIO port input value
- 0: Pin logic level is 0, or is not configured for use by digital function.
- 1: Pin logic level is 1

10.1.2.13 static uint32_t gpio_hal_read_port_input (uint32_t *instance*) [inline], [static]

This function gets all 32-pin input as a 32-bit integer.

Parameters

instance	GPIO instance number(HW_GPIOA, HW_GPIOB, HW_GPIOC, etc.)

Returns

GPIO port input data. Each bit represents one pin. For each bit:
- 0: Pin logic level is 0, or is not configured for use by digital function.
- 1: Pin logic level is 1.
- LSB: pin 0
- MSB: pin 31

10.2 GPIO Peripheral Driver

The chapter describes the programming interface of the GPIO Peripheral driver.

Data Structures

- struct gpio_input_pin_t
  The GPIO input pin configuration structure. More...
- struct gpio_output_pin_t
  The GPIO output pin configuration structure. More...
- struct gpio_input_pin_user_config_t
  The GPIO input pin structure. More...
- struct gpio_output_pin_user_config_t
  The GPIO output pin structure. More...

Typedefs

- typedef void(∗ gpio_isr_callback_t)(void)
  The GPIO ISR callback function.

GPIO Pin Macros

- #define GPIO_PINS_OUT_OF_RANGE (0xFFFFFFFFU)
  Indicates the end of a pin configuration structure.
- #define GPIO_PORT_SHIFT (0x8U)
  Bits shifted for the GPIO port number.
- #define GPIO_MAKE_PIN(r, p) (((r)<< GPIO_PORT_SHIFT) | (p))
  Combines the port number and the pin number into a single scalar value.
- #define GPIO_EXTRACT_PORT(v) (((v) >> GPIO_PORT_SHIFT) & 0xFFU)
  Extracts the port number from a combined port and pin value.
- #define GPIO_EXTRACT_PIN(v) ((v) & 0xFFU)
  Extracts the pin number from a combined port and pin value.

Initialization

- void gpio_init (const gpio_input_pin_user_config_t ∗inputPins, const gpio_output_pin_user_-config_t ∗outputPins)
  Initialize all GPIO pins used by board.
- void gpio_input_pin_init (const gpio_input_pin_user_config_t ∗inputPin)
  Initializes one GPIO input pin used by board.
- void gpio_output_pin_init (const gpio_output_pin_user_config_t ∗outputPin)
  Initializes one GPIO output pin used by board.

Pin Direction

- uint32_t gpio_get_pin_direction (uint32_t pinName)
  Gets the current direction of the individual GPIO pin.
- void gpio_set_pin_direction (uint32_t pinName, gpio_pin_direction_t direction)
  Sets the current direction of the individual GPIO pin.

Output Operations

- void gpio_write_pin_output (uint32_t pinName, uint32_t output)
  Sets the output level of the individual GPIO pin to the logic 1 or 0.
- void gpio_set_pin_output (uint32_t pinName)
  Sets the output level of the individual GPIO pin to the logic 1.
- void gpio_clear_pin_output (uint32_t pinName)
  Sets the output level of the individual GPIO pin to the logic 0.
- void gpio_toggle_pin_output (uint32_t pinName)
  Reverses current output logic of the individual GPIO pin.

Input Operations

- uint32_t gpio_read_pin_input (uint32_t pinName)
  Reads the current input value of the individual GPIO pin.

Interrupt

- void gpio_clear_pin_interrupt_flag (uint32_t pinName)
  Clears the individual GPIO pin interrupt status flag.
- void gpio_register_isr_callback_function (uint32_t pinName, gpio_isr_callback_t function)
  Registers the GPIO ISR callback function.

10.2.0.14 GPIO Peripheral Driver

Overview

The GPIO peripheral driver configures pins to digital input/output and provides APIs for input/output operations.

GPIO Pin Definitions

The user should define GPIO pins according to the target board configuration and ensure that the definitions are correct. One header file should be defined to save GPIO pin names and the input/output configuration arrays defined in source files.

Include this header file in source files where you want to use GPIO driver to operate GPIO pins.

GPIO pin header file example:

```
// Feel free to change the pin names as what you want
enum _gpio_pins
{
    kGpioLED1  = GPIO_MAKE_PIN(HW_PORTC, 0x0),
    kGpioLED2  = GPIO_MAKE_PIN(HW_PORTC, 0x1),
    kGpioLED3  = GPIO_MAKE_PIN(HW_PORTC, 0x2),
    kGpioLED4  = GPIO_MAKE_PIN(HW_PORTC, 0x3),
};

// Extern input/output arrays defined in source files.
extern gpio_input_pin_t inputPin[];
extern gpio_output_pin_t outputPin[];
```

Initialization

To initialize the GPIO driver, two arrays, the gpio_input_pin_t inputPin[] and the gpio_output_pin_t outputPin[], should be defined first. Then , call the gpio_init() function and pass these two arrays.

Example of the inputPin and outputPin array definition of:

```
#include "gpio/fsl_gpio_driver.h"
#include "gpio_pin_header_file.h"

    // Configure kGpioPTA2 as a digital input and enable the interrupt on the rising edge.
    gpio_input_pin_t inputPin[] = {
        {
            .pinName = kGpioPTA2,
            .config.isPullEnable = false,
            .config.pullSelect = kPortPullDown,
            .config.isPassiveFilterEnabled = false,
            .config.interrupt = kPortIntRisingEdge,
        },
        {
            //Note: This pinName must be defined here to indicate end of array.
            .pinName = GPIO_PINS_OUT_OF_RANGE,
        }
    };

    // Configure the kGpioLED4 and kGpioPTB9 as a digital output and enable the high drive strength.
    gpio_output_pin_t outputPin[] = {
        {
            .pinName = kGpioLED4,
            .config.outputLogic = 0,
            .config.slewRate = kPortFastSlewRate,
            .config.driveStrength = kPortHighDriveStrength,
            .config.interrupt = kPortIntDisabled,
        },
        {
            .pinName = kGpioPTB9,
            .config.outputLogic = 0,
            .config.slewRate = kPortFastSlewRate,
            .config.driveStrength = kPortHighDriveStrength,
            .config.interrupt = kPortIntDisabled,
        },
        {
            //Note: This pinName must be defined here to indicate the end of array.
            .pinName = GPIO_PINS_OUT_OF_RANGE,
        }
```

```
};

// Initializes GPIO pins.
gpio_init(inputPin, outputPin);
```

Note

If the digital filter is enabled, the digital filter clock source and width must be configured before calling the gpio_init function. To configure the digital filter, use this API from porting the HAL driver. Each pin in the same port shares the same digital filter settings.

```
void port_hal_configure_digital_filter_clock(uint32_t instance,
                        port_digital_filter_clock_source_t clockSource);
void port_hal_configure_digital_filter_width(uint32_t instance, uint8_t width);
```

Output Operations

To use the output operation, configure the target GPIO pin as a digital output in the gpio_init function. The output operations include set, clear, and toggle of the output logic level. For these operations, three APIs are provided:

```
void gpio_set_pin_output(uint32_t pinName);
void gpio_clear_pin_output(uint32_t pinName);
void gpio_toggle_pin_output(uint32_t pinName);
```

These functions are used when the logic level of the GPIO output is known.

Otherwise, a different function is provided to configure the output logic level according to a passed parameter:

```
void gpio_write_pin_output(uint32_t pinName, uint32_t output);
```

Use this function to change the GPIO output according to the results of other code. Pass an integer as the "uint32_t output" parameter. If that integer is not 0, it generates the high logic. If the integer is 0, it generates the low logic.

Input Operations

To use the input operation, configure the target GPIO pin as a digital input in the gpio_init function. For the input operation, the most commonly used API is:

```
uint32_t gpio_read_pin_input(uint32_t pinName);
```

This function returns the logic level read from a specific GPIO pin.

If the digital filter is enabled, use this function to disable it:

```
void gpio_configure_digital_filter(uint32_t pinName, bool isDigitalFilterEnabled);
```

Interrupt

Enable a specific pin interrupt in GPIO initialization structures. Both output and input can trigger an interrupt. This API clears the interrupt flag inside the interrupt handler:

```
void gpio_clear_pin_interrupt_flag(uint32_t pinName);
```

10.2.1 Data Structure Documentation

10.2.1.1 struct gpio_input_pin_t

Although every pin is configurable, valid configurations depend on a specific SoC. Users should check the related reference manual to ensure that the specific feature is valid in an individual pin. A configuration of unavailable features is harmless, but takes no effect.

Data Fields

- bool isPullEnable
  Enable or disable pull.
- port_pull_t pullSelect
  Select internal pull(up/down) resistor.
- bool isPassiveFilterEnabled
  Enable or disable passive filter.
- port_interrupt_config_t interrupt
  Select interrupt/DMA request.

10.2.1.1.0.16 Field Documentation

10.2.1.1.0.16.1 bool gpio_input_pin_t::isPullEnable

10.2.1.1.0.16.2 port_pull_t gpio_input_pin_t::pullSelect

10.2.1.1.0.16.3 bool gpio_input_pin_t::isPassiveFilterEnabled

10.2.1.1.0.16.4 port_interrupt_config_t gpio_input_pin_t::interrupt

10.2.1.2 struct gpio_output_pin_t

Although every pin is configurable, valid configurations depend on a specific SoC. Users should check the related reference manual to ensure that the specific feature is valid in an individual pin. The configuration of unavailable features is harmless, but takes no effect.

Data Fields

- uint32_t outputLogic
  Set default output logic.

- port_drive_strength_t driveStrength
  Select fast/slow slew rate.

10.2.1.2.0.17 Field Documentation

10.2.1.2.0.17.1 uint32_t gpio_output_pin_t::outputLogic

10.2.1.2.0.17.2 port_drive_strength_t gpio_output_pin_t::driveStrength

Select low/high drive strength.

10.2.1.3 struct gpio_input_pin_user_config_t

Although the pinName is defined as a uint32_t type, values assigned to the pinName should be the enumeration names defined in the enum _gpio_pins.

Data Fields

- uint32_t pinName
  Virtual pin name from enum defined by the user.
- gpio_input_pin_t config
  Input pin configuration structure.

10.2.1.3.0.18 Field Documentation

10.2.1.3.0.18.1 uint32_t gpio_input_pin_user_config_t::pinName

10.2.1.3.0.18.2 gpio_input_pin_t gpio_input_pin_user_config_t::config

10.2.1.4 struct gpio_output_pin_user_config_t

Although the pinName is defined as a uint32_t type, values assigned to the pinName should be the enumeration names defined in the enum _gpio_pins.

Data Fields

- uint32_t pinName
  Virtual pin name from enum defined by the user.
- gpio_output_pin_t config
  Input pin configuration structure.

10.2.1.4.0.19 Field Documentation

10.2.1.4.0.19.1 uint32_t gpio_output_pin_user_config_t::pinName

10.2.1.4.0.19.2 gpio_output_pin_t gpio_output_pin_user_config_t::config

10.2.2 Macro Definition Documentation

10.2.2.1 #define GPIO_PINS_OUT_OF_RANGE (0xFFFFFFFFU)

10.2.2.2 #define GPIO_PORT_SHIFT (0x8U)

10.2.2.3 #define GPIO_MAKE_PIN(*r, p*) (((r)<< GPIO_PORT_SHIFT) | (p))

10.2.2.4 #define GPIO_EXTRACT_PORT(*v*) (((v) >> GPIO_PORT_SHIFT) & 0xFFU)

10.2.2.5 #define GPIO_EXTRACT_PIN(*v*) ((v) & 0xFFU)

10.2.3 Function Documentation

10.2.3.1 void gpio_init (const gpio_input_pin_user_config_t * *inputPins*, const gpio_output_pin_user_config_t * *outputPins*)

To initialize the GPIO driver, define two arrays similar to the gpio_input_pin_user_config_t inputPin[] and the gpio_output_pin_user_config_t outputPin[] in the user file. Then, call the gpio_init() function and pass in the two arrays. If the input or output pins are not needed, pass in a NULL.

This is an example to define an input pin array:

```
// Configure the kGpioPTA2 as digital input.
gpio_input_pin_user_config_t inputPin[] = {
  {
      .pinName = kGpioPTA2,
      .config.isPullEnable = false,
      .config.pullSelect = kPortPullDown,
      .config.isPassiveFilterEnabled = false,
      .config.interrupt = kPortIntDisabled,
  },
  {
     // Note: This pinName must be defined here to indicate the end of the array.
     .pinName = GPIO_PINS_OUT_OF_RANGE,
  }
};
```

Parameters

inputPins	input GPIO pins pointer.
outputPins	output GPIO pins pointer.

10.2.3.2 void gpio_input_pin_init (const gpio_input_pin_user_config_t * *inputPin*)

Parameters

inputPins	input GPIO pins pointer.

10.2.3.3 void gpio_output_pin_init (const gpio_output_pin_user_config_t * *outputPin*)

Parameters

outputPins	output GPIO pins pointer.

10.2.3.4 uint32_t gpio_get_pin_direction (uint32_t *pinName*)

Parameters

pinName	GPIO pin name defined by the user in the GPIO pin enumeration list.

Returns

GPIO directions.
- 0: corresponding pin is set as digital input.
- 1: corresponding pin is set as digital output.

10.2.3.5 void gpio_set_pin_direction (uint32_t *pinName,* gpio_pin_direction_t *direction*)

Parameters

pinName	GPIO pin name defined by the user in the GPIO pin enumeration list.

direction	GPIO directions. • kGpioDigitalInput: corresponding pin is set as digital input. • kGpioDigitalOutput: corresponding pin is set as digital output.

10.2.3.6 void gpio_write_pin_output (uint32_t *pinName,* uint32_t *output*)

Parameters

pinName	GPIO pin name defined by the user in the GPIO pin enumeration list.
output	pin output logic level. • 0: corresponding pin output low logic level. • Non-0: corresponding pin output high logic level.

10.2.3.7 void gpio_set_pin_output (uint32_t *pinName*)

Parameters

pinName	GPIO pin name defined by the user in the GPIO pin enumeration list.

10.2.3.8 void gpio_clear_pin_output (uint32_t *pinName*)

Parameters

pinName	GPIO pin name defined by the user in the GPIO pin enumeration list.

10.2.3.9 void gpio_toggle_pin_output (uint32_t *pinName*)

Parameters

pinName	GPIO pin name defined by the user in the GPIO pin enumeration list.

10.2.3.10 uint32_t gpio_read_pin_input (uint32_t *pinName*)

Parameters

pinName	GPIO pin name defined by the user in the GPIO pin enumeration list.

Returns

GPIO port input value.
- 0: Pin logic level is 0, or is not configured for use by digital function.
- 1: Pin logic level is 1.

10.2.3.11 void gpio_clear_pin_interrupt_flag (uint32_t *pinName*)

Parameters

pinName	GPIO pin name defined by the user in the GPIO pin enumeration list.

10.2.3.12 void gpio_register_isr_callback_function (uint32_t *pinName,* gpio_isr_callback_t *function*)

Parameters

pinName	GPIO pin name defined by the user in the GPIO pin enumeration list.
function	Pointer to the GPIO ISR callback function.

10.3 GPIO ISR Definitions

The chapter describes the programming interface of the GPIO interrupt definitions.

Chapter 11
Inter-Integrated Circuit (I2C)

The Kinetis SDK provides both HAL and Peripheral drivers for the Inter-Integrated Circuit (I2C) block of Kinetis devices.

Modules

- I2C Classes
  The part describes the C++ classes of the I2C Peripheral driver.
- I2C HAL driver
  The part describes the programming interface of the I2C HAL driver.
- I2C Master peripheral
  The part describes the programming interface of the I2C master mode Peripheral driver.
- I2C Slave peripheral driver
  The part describes the programming interface of the I2C slave mode Peripheral driver.

11.1 I2C HAL driver

The chapter describes the programming interface of the I2C HAL driver.

Data Structures

- struct i2c_config_t
  I2C module configuration. More...

Enumerations

- enum i2c_status_t { ,
  kStatus_I2C_Busy,
  kStatus_I2C_Timeout,
  kStatus_I2C_ReceivedNak,
  kStatus_I2C_SlaveTxUnderrun,
  kStatus_I2C_SlaveRxOverrun,
  kStatus_I2C_AribtrationLost }
  I2C status return codes.
- enum i2c_transmit_receive_mode_t {
  kI2CReceive = 0,
  kI2CTransmit = 1 }
  Direction of master and slave transfers.

Module controls

- void i2c_hal_init (uint32_t instance, const i2c_config_t *config, uint32_t sourceClockInHz)
  Initializes and configures the I2C peripheral.
- void i2c_hal_reset (uint32_t instance)
  Restores the I2C peripheral to reset state.
- static void i2c_hal_enable (uint32_t instance)
  Enables the I2C module operation.
- static void i2c_hal_disable (uint32_t instance)
  Disables the I2C module operation.

DMA

- static void i2c_hal_set_dma_enable (uint32_t instance, bool enable)
  Enables or disables the DMA support.

Pin functions

- static void i2c_hal_set_high_drive (uint32_t instance, bool enable)

Controls the drive capability of the I2C pads.
- static void i2c_hal_set_glitch_filter (uint32_t instance, uint8_t glitchWidth)
  Controls the width of the programmable glitch filter.

Low power

- static void i2c_hal_set_wakeup_enable (uint32_t instance, bool enable)
  Controls the I2C wakeup enable.

Baud rate

- uint32_t i2c_hal_get_max_baud (uint32_t instance, uint32_t sourceClockInHz)
  brief Returns the maximum supported baud rate in kilohertz.
- i2c_status_t i2c_hal_set_baud (uint32_t instance, uint32_t sourceClockInHz, uint32_t kbps, uint32_t *absoluteError_Hz)
  Sets the I2C bus frequency for master transactions.
- static void i2c_hal_set_baud_icr (uint32_t instance, uint8_t mult, uint8_t icr)
  Sets the I2C baud rate multiplier and table entry.
- static void i2c_hal_set_independent_slave_baud (uint32_t instance, bool enable)
  Slave baud rate control.

Bus operations

- void i2c_hal_send_start (uint32_t instance)
  Sends a START or a Repeated START signal on the I2C bus.
- static void i2c_hal_send_stop (uint32_t instance)
  Sends a STOP signal on the I2C bus.
- static void i2c_hal_set_direction (uint32_t instance, i2c_transmit_receive_mode_t mode)
  Selects either transmit or receive modes.
- static i2c_transmit_receive_mode_t i2c_hal_get_direction (uint32_t instance)
  Returns the currently selected transmit or receive mode.
- static void i2c_hal_send_ack (uint32_t instance)
  Causes an ACK to be sent on the bus.
- static void i2c_hal_send_nak (uint32_t instance)
  Causes a NAK to be sent on the bus.

Data transfer

- static uint8_t i2c_hal_read (uint32_t instance)
  Returns the last byte of data read from the bus and initiate another read.
- static void i2c_hal_write (uint32_t instance, uint8_t data)
  Writes one byte of data to the I2C bus.

Slave address

- void i2c_hal_set_slave_address_7bit (uint32_t instance, uint8_t address)
  Sets the primary 7-bit slave address.
- void i2c_hal_set_slave_address_10bit (uint32_t instance, uint16_t address)
  Sets the primary slave address and enables 10-bit address mode.
- static void i2c_hal_set_general_call_enable (uint32_t instance, bool enable)
  Controls whether the general call address is recognized.
- static void i2c_hal_set_slave_range_address_enable (uint32_t instance, bool enable)
  Enables or disables the slave address range matching.
- static void i2c_hal_set_upper_slave_address_7bit (uint32_t instance, uint8_t address)
  Sets the upper slave address.

Status

- static bool i2c_hal_is_master (uint32_t instance)
  Returns whether the I2C module is in master mode.
- static bool i2c_hal_is_transfer_complete (uint32_t instance)
  Gets the transfer complete flag.
- static bool i2c_hal_is_addressed_as_slave (uint32_t instance)
  Returns whether the I2C slave was addressed.
- static bool i2c_hal_is_bus_busy (uint32_t instance)
  Determines whether the I2C bus is busy.
- static bool i2c_hal_was_arbitration_lost (uint32_t instance)
  Returns whether the arbitration procedure was lost.
- static void i2c_hal_clear_arbitration_lost (uint32_t instance)
  Clears the arbitration lost flag.
- static bool i2c_hal_is_range_address_match (uint32_t instance)
  Get the range address match flag.
- static i2c_transmit_receive_mode_t i2c_hal_get_slave_direction (uint32_t instance)
  Returns whether the I2C slave was addressed in read or write mode.
- static bool i2c_hal_get_receive_ack (uint32_t instance)
  Returns whether an ACK was received after the last byte was transmitted.

Interrupt

- static void i2c_hal_enable_interrupt (uint32_t instance)
  Enables the I2C interrupt requests.
- static void i2c_hal_disable_interrupt (uint32_t instance)
  Disables the I2C interrupt requests.
- static bool i2c_hal_is_interrupt_enabled (uint32_t instance)
  Returns whether the I2C interrupts are enabled.
- static bool i2c_hal_get_interrupt_status (uint32_t instance)
  Returns the current I2C interrupt flag.
- static void i2c_hal_clear_interrupt (uint32_t instance)
  Clears the I2C interrupt if set.

11.1.0.13 I2C HAL Driver

ICR Table

ICR (hex)	SCL divider	SDA hold value	SCL start hold value	SCL stop hold value
00	20	7	6	11
01	22	7	7	12
02	24	8	8	13
03	26	8	9	14
04	28	9	10	15
05	30	9	11	16
06	34	10	13	18
07	40	10	16	21

Clock rate formulas

I2C baud rate = bus_clock_Hz / (mult ∗ SCL_divider)

SDA hold time = bus_clock_period_s ∗ mult ∗ SDA_hold_value

SCL start hold time = bus_clock_period_s ∗ mult ∗ SCL_start_hold_value

SCL stop hold time = bus_clock_period_s ∗ mult ∗ SCL_stop_hold_value

11.1.1 Data Structure Documentation

11.1.1.1 struct i2c_config_t

Pass an instance of this structure to the i2c_hal_init() to configure the entire I2C peripheral in a single function call.

Data Fields

- bool enableModule
  Whether the I2C peripheral operation is enabled.
- uint32_t baudRate_kbps
  Requested baud rate in kilobits per second, for example, 100 or 400.
- bool useIndependentSlaveBaud
  Enables independent slave mode baud rate at maximum frequency.
- bool enableInterrupt
  Enable for the I2C interrupt.
- bool enableDma

Enable DMA transfer signalling.

- bool enableHighDrive
  Enable high drive pin mode.
- bool enableWakeup
  Enable low power wakeup.
- uint8_t glitchFilterWidth
  Specify the glitch filter width in terms of bus clock cycles.
- uint16_t slaveAddress
  7-bit or 10-bit slave address.
- uint8_t upperSlaveAddress
  7-bit upper slave address, or zero to disable.
- bool use10bitSlaveAddress
  Controls whether 10-bit slave addresses are enabled.
- bool enableGeneralCallAddress
  Enable general call address matching.
- bool enableRangeAddressMatch
  Determines if addresses between slaveAddress and upperSlaveAddress are matched.

11.1.1.1.0.20 Field Documentation

11.1.1.1.0.20.1 bool i2c_config_t::enableModule

11.1.1.1.0.20.2 uint32_t i2c_config_t::baudRate_kbps

Pass zero to not set the baud rate.

11.1.1.1.0.20.3 bool i2c_config_t::useIndependentSlaveBaud

11.1.1.1.0.20.4 bool i2c_config_t::enableInterrupt

11.1.1.1.0.20.5 bool i2c_config_t::enableDma

11.1.1.1.0.20.6 bool i2c_config_t::enableHighDrive

11.1.1.1.0.20.7 bool i2c_config_t::enableWakeup

11.1.1.1.0.20.8 uint8_t i2c_config_t::glitchFilterWidth

Set this value to zero to disable the glitch filter.

11.1.1.1.0.20.9 uint16_t i2c_config_t::slaveAddress

11.1.1.1.0.20.10 uint8_t i2c_config_t::upperSlaveAddress

If 10-bit addresses are enabled, the top 3 bits are provided by the *slaveAddress* field.

11.1.1.1.0.20.11 bool i2c_config_t::use10bitSlaveAddress

11.1.1.1.0.20.12 bool i2c_config_t::enableGeneralCallAddress

11.1.1.1.0.20.13 bool i2c_config_t::enableRangeAddressMatch

Both of those fields must be non-zero.

11.1.2 Enumeration Type Documentation

11.1.2.1 enum i2c_status_t

Enumerator

kStatus_I2C_Busy The master is already performing a transfer.
kStatus_I2C_Timeout The transfer timed out.
kStatus_I2C_ReceivedNak The slave device sent a NAK in response to a byte.
kStatus_I2C_SlaveTxUnderrun I2C Slave TX Underrun error.
kStatus_I2C_SlaveRxOverrun I2C Slave RX Overrun error.
kStatus_I2C_AribtrationLost I2C Arbitration Lost error.

11.1.2.2 enum i2c_transmit_receive_mode_t

Enumerator

kI2CReceive Master and slave receive.
kI2CTransmit Master and slave transmit.

11.1.3 Function Documentation

11.1.3.1 void i2c_hal_init (uint32_t *instance*, const i2c_config_t * *config*, uint32_t *sourceClockInHz*)

Parameters

instance	The I2C peripheral instance number
config	Pointer to the configuration settings

sourceClockInHz	I2C source input clock in Hertz.

11.1.3.2 void i2c_hal_reset (uint32_t *instance*)

Parameters

instance	The I2C peripheral instance number

11.1.3.3 static void i2c_hal_enable (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

11.1.3.4 static void i2c_hal_disable (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

11.1.3.5 static void i2c_hal_set_dma_enable (uint32_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The I2C peripheral instance number
enable	Pass true to enable DMA transfer signalling

11.1.3.6 static void i2c_hal_set_high_drive (uint32_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The I2C peripheral instance number
enable	Passing true will enable high drive mode of the I2C pads. False sets normal drive mode.

11.1.3.7 static void i2c_hal_set_glitch_filter (uint32_t *instance,* uint8_t *glitchWidth*) [inline], [static]

Controls the width of the glitch, in terms of bus clock cycles, that the filter must absorb. The filter does not allow any glitch whose size is less than or equal to this width setting, to pass.

Parameters

instance	The I2C peripheral instance number
glitchWidth	Maximum width in bus clock cycles of the glitches that is filtered. Pass zero to disable the glitch filter.

11.1.3.8 static void i2c_hal_set_wakeup_enable (uint32_t *instance,* bool *enable*) [inline], [static]

The I2C module can wake the MCU from low power mode with no peripheral bus running when slave address matching occurs.

Parameters

instance	The I2C peripheral instance number.
enable	true - Enables the wakeup function in low power mode. false - Normal operation. No interrupt is generated when address matching in low power mode.

11.1.3.9 uint32_t i2c_hal_get_max_baud (uint32_t *instance,* uint32_t *sourceClockInHz*)

Parameters

instance	The I2C peripheral instance number
sourceClockInHz	I2C source input clock in Hertz

Returns

The maximum baud rate in kilohertz

11.1.3.10 i2c_status_t i2c_hal_set_baud (uint32_t *instance,* uint32_t *sourceClockInHz,* uint32_t *kbps,* uint32_t * *absoluteError_Hz*)

Parameters

instance	The I2C peripheral instance number
sourceClockInHz	I2C source input clock in Hertz
kbps	Requested bus frequency in kilohertz. Common values are either 100 or 400.
absoluteError_Hz	If this parameter is not NULL, it is filled in with the difference in Hertz between the requested bus frequency and the closest frequency possible given available divider values.

Return values

kStatus_Success	The baud rate was changed successfully. However, there is no guarantee on the minimum error. If you want to ensure that the baud was set to within a certain error, then use the *absoluteError_Hz* parameter.
kStatus_OutOfRange	The requested baud rate was not within the range of rates supported by the peripheral.

11.1.3.11 static void i2c_hal_set_baud_icr (uint32_t *instance,* uint8_t *mult,* uint8_t *icr*) [inline], [static]

Use this function to set the I2C bus frequency register values directly, if they are known in advance.

Parameters

instance	The I2C peripheral instance number
mult	Value of the MULT bitfield, ranging from 0-2.
icr	The ICR bitfield value, which is the index into an internal table in the I2C hardware that selects the baud rate divisor and SCL hold time.

11.1.3.12 static void i2c_hal_set_independent_slave_baud (uint32_t *instance,* bool *enable*) [inline], [static]

Enables an independent slave mode baud rate at the maximum frequency. This forces clock stretching on the SCL in very fast I2C modes.

Parameters

instance	The I2C peripheral instance number
enable	true - Slave baud rate is independent of the master baud rate; false - The slave baud rate follows the master baud rate and clock stretching may occur.

11.1.3.13 void i2c_hal_send_start (uint32_t *instance*)

This function is used to initiate a new master mode transfer by sending the START signal. It is also used to send a Repeated START signal when a transfer is already in progress.

Parameters

instance	The I2C peripheral instance number

11.1.3.14 static void i2c_hal_send_stop (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

11.1.3.15 static void i2c_hal_set_direction (uint32_t *instance,* i2c_transmit_receive_-mode_t *mode*) [inline], [static]

Parameters

instance	The I2C peripheral instance number
mode	Specifies either transmit mode or receive mode. The valid values are: • kI2CTransmit • kI2CReceive

11.1.3.16 static i2c_transmit_receive_mode_t i2c_hal_get_direction (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

kI2CTransmit	I2C is configured for master or slave transmit mode.
kI2CReceive	I2C is configured for master or slave receive mode.

11.1.3.17 static void i2c_hal_send_ack (uint32_t *instance*) [inline], [static]

This function specifies that an ACK signal is sent in response to the next received byte.

Note that the behavior of this function is changed when the I2C peripheral is placed in Fast ACK mode. In this case, this function causes an ACK signal to be sent in response to the current byte, rather than the next received byte.

Parameters

instance	The I2C peripheral instance number

11.1.3.18 static void i2c_hal_send_nak (uint32_t *instance*) [inline], [static]

This function specifies that a NAK signal is sent in response to the next received byte.

Note that the behavior of this function is changed when the I2C peripheral is placed in the Fast ACK mode. In this case, this function causes an NAK signal to be sent in response to the current byte, rather than the next received byte.

Parameters

instance	The I2C peripheral instance number

11.1.3.19 static uint8_t i2c_hal_read (uint32_t *instance*) [inline], [static]

In a master receive mode, calling this function initiates receiving the next byte of data.

Parameters

instance	The I2C peripheral instance number

Returns

This function returns the last byte received while the I2C module is configured in master receive or slave receive mode.

11.1.3.20 static void i2c_hal_write (uint32_t *instance,* uint8_t *data*) [inline], [static]

When this function is called in the master transmit mode, a data transfer is initiated. In slave mode, the same function is available after an address match occurs.

In a master transmit mode, the first byte of data written following the start bit or repeated start bit is used for the address transfer and must consist of the slave address (in bits 7-1) concatenated with the required R/#W bit (in position bit 0).

Parameters

instance	The I2C peripheral instance number
data	The byte of data to transmit

11.1.3.21 void i2c_hal_set_slave_address_7bit (uint32_t *instance,* uint8_t *address*)

Parameters

instance	The I2C peripheral instance number
address	The slave address in the upper 7 bits. Bit 0 of this value must be 0.

11.1.3.22 void i2c_hal_set_slave_address_10bit (uint32_t *instance,* uint16_t *address*)

Parameters

instance	The I2C peripheral instance number
address	The 10-bit slave address, in bits [10:1] of the value. Bit 0 must be 0.

11.1.3.23 static void i2c_hal_set_general_call_enable (uint32_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The I2C peripheral instance number
enable	Whether to enable the general call address.

11.1.3.24 static void i2c_hal_set_slave_range_address_enable (uint32_t *instance,* bool *enable*) [inline], [static]

Parameters

instance	The I2C peripheral instance number
enable	Pass true to enable the range address matching. You must also call the i2c_hal_set_upper_slave_address_7bit() to set the upper address.

11.1.3.25 static void i2c_hal_set_upper_slave_address_7bit (uint32_t *instance,* uint8_t *address*) [inline], [static]

This slave address is used as a secondary slave address. If range address matching is enabled, this slave address acts as the upper bound on the slave address range.

This function sets only a 7-bit slave address. If 10-bit addressing was enabled by calling the i2c_hal_set_slave_address_10bit(), then the top 3 bits set with that function are also used with the address set with this function to form a 10-bit address.

Passing 0 for the *address* parameter disables matching the upper slave address.

Parameters

instance	The I2C peripheral instance number
address	The upper slave address in the upper 7 bits. Bit 0 of this value must be 0. This address must be greater than the primary slave address that is set by calling the i2c_hal_set_slave_address_7bit().

11.1.3.26 static bool i2c_hal_is_master (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

true	The module is in master mode, which implies it is also performing a transfer.
false	The module is in slave mode.

11.1.3.27 static bool i2c_hal_is_transfer_complete (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

true	Transfer is complete.
false	Transfer is in progress.

11.1.3.28 static bool i2c_hal_is_addressed_as_slave (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

true	Addressed as slave.
false	Not addressed.

11.1.3.29 static bool i2c_hal_is_bus_busy (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

true	Bus is busy.
false	Bus is idle.

11.1.3.30 static bool i2c_hal_was_arbitration_lost (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

true	Loss of arbitration
false	Standard bus operation

11.1.3.31 static void i2c_hal_clear_arbitration_lost (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

11.1.3.32 static bool i2c_hal_is_range_address_match (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

true	Addressed as slave.
false	Not addressed.

11.1.3.33 static i2c_transmit_receive_mode_t i2c_hal_get_slave_direction (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

kI2CReceive	Slave receive, master writing to slave
kI2CTransmit	Slave transmit, master reading from slave

11.1.3.34 static bool i2c_hal_get_receive_ack (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

true	Acknowledges that the signal was received after the completion of one byte of data transmission on the bus.
false	No acknowledgement of the signal is detected.

11.1.3.35 static void i2c_hal_enable_interrupt (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

11.1.3.36 static void i2c_hal_disable_interrupt (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

11.1.3.37 static bool i2c_hal_is_interrupt_enabled (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

true	I2C interrupts are enabled.
false	I2C interrupts are disabled.

11.1.3.38 static bool i2c_hal_get_interrupt_status (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

Return values

true	An interrupt is pending.
false	No interrupt is pending.

11.1.3.39 static void i2c_hal_clear_interrupt (uint32_t *instance*) [inline], [static]

Parameters

instance	The I2C peripheral instance number

11.2 I2C Slave peripheral driver

The chapter describes the programming interface of the I2C slave mode Peripheral driver.

Data Structures

- struct i2c_slave_user_config_t
  brief Definition of application-implemented callback functions used by the I2C slave driver. More...

I2C Slave

- void i2c_slave_init (uint32_t instance, i2c_slave_user_config_t ∗appInfo)
  Initializes the I2C module.
- void i2c_slave_shutdown (uint32_t instance)
  Shuts down the I2C slave driver.

11.2.1 Data Structure Documentation

11.2.1.1 struct i2c_slave_user_config_t

Data Fields

- i2c_status_t(∗ data_source)(uint8_t ∗source_byte)
  Callback to get byte to transmit.
- i2c_status_t(∗ data_sink)(uint8_t sink_byte)
  Callback to put received byte.
- void(∗ on_error)(i2c_status_t error)
  Callback to report an I2C error.
- uint8_t slaveAddress
  7-bit slave address.

11.2.1.1.0.21 Field Documentation

11.2.1.1.0.21.1 i2c_status_t(∗ i2c_slave_user_config_t::data_source)(uint8_t ∗source_byte)

11.2.1.1.0.21.2 i2c_status_t(∗ i2c_slave_user_config_t::data_sink)(uint8_t sink_byte)

11.2.1.1.0.21.3 void(∗ i2c_slave_user_config_t::on_error)(i2c_status_t error)

11.2.1.1.0.21.4 uint8_t i2c_slave_user_config_t::slaveAddress

11.2.2 Function Documentation

11.2.2.1 void i2c_slave_init (uint32_t *instance,* i2c_slave_user_config_t ∗ *appInfo*)

Saves the application callback info, turns on the clock to the module, enables the device, and enables interrupts. Sets the I2C to slave mode. IOMux is expected to be already handled in the init_hardware().

Parameters

instance	Instance number of the I2C module.
appInfo	Pointer of the application information.

11.2.2.2 void i2c_slave_shutdown (uint32_t *instance*)

Clears the control register and turns off the clock to the module.

Parameters

instance	Instance number of the I2C module.

11.3 I2C Master peripheral

The chapter describes the programming interface of the I2C master mode Peripheral driver.

Data Structures

- struct i2c_device_t
  Information necessary to communicate with an I2C slave device. More...
- struct i2c_master_t
  Internal driver state information. More...

Enumerations

- enum i2c_direction_t {
  kI2CRead = 1,
  kI2CWrite = 0 }
  Constants for the direction of an I2C transfer.
- enum _i2c_transfer_flags {
  kI2CNoStart = 1 << 1,
  kI2CNoStop = 1 << 2 }
  Optional flags to control a transfer.

I2C Master

- void i2c_master_init (uint32_t instance, i2c_master_t *master)
  Initialize the I2C master mode driver.
- void i2c_master_shutdown (i2c_master_t *master)
  Shut down the driver.
- i2c_status_t i2c_master_configure_bus (i2c_master_t *master, const i2c_device_t *device)
  Configure the I2C bus to access a device.
- i2c_status_t i2c_master_transfer_basic (i2c_master_t *master, uint32_t flags, i2c_direction_t direction, uint8_t *data, size_t dataLength, size_t *actualLengthTransferred, uint32_t timeout_ms)
  Low-level I2C transfer function.
- i2c_status_t i2c_master_transfer (i2c_master_t *master, const i2c_device_t *device, i2c_direction_t direction, bool stopAfterTransfer, uint32_t subaddress, size_t subaddressLength, uint8_t *data, size_t dataLength, size_t *actualLengthTransferred, uint32_t timeout_ms)
  Perform a blocking read or write transaction on the I2C bus.

11.3.1 Data Structure Documentation

11.3.1.1 struct i2c_device_t

Data Fields

- uint8_t address

The slave's 7-bit address.

- uint32_t baudRate_kbps

  The baud rate in kbps to use with this slave device.

11.3.1.1.0.22 Field Documentation

11.3.1.1.0.22.1 uint8_t i2c_device_t::address

11.3.1.1.0.22.2 uint32_t i2c_device_t::baudRate_kbps

11.3.1.2 struct i2c_master_t

Note

The contents of this structure are internal to the driver and should not be modified by users. Also, contents of the structure are subject to change in future releases.

11.3.2 Enumeration Type Documentation

11.3.2.1 enum i2c_direction_t

Enumerator

kI2CRead Read from slave device.
kI2CWrite Write to slave device.

11.3.2.2 enum _i2c_transfer_flags

Enumerator

kI2CNoStart Set this flag to prevent sending a START signal.
kI2CNoStop Set this flag to prevent sending a STOP signal.

11.3.3 Function Documentation

11.3.3.1 void i2c_master_init (uint32_t *instance,* i2c_master_t * *master*)

Parameters

instance	The I2C peripheral instance number.
master	The pointer to the I2C master driver state structure.

11.3.3.2 void i2c_master_shutdown (i2c_master_t * *master*)

Parameters

master	The pointer to the I2C master driver state structure.

11.3.3.3 i2c_status_t i2c_master_configure_bus (i2c_master_t * *master,* const i2c_device_t * *device*)

Parameters

master	The pointer to the I2C master driver state structure.
device	The pointer to the I2C device information struct.

11.3.3.4 i2c_status_t i2c_master_transfer_basic (i2c_master_t * *master,* uint32_t *flags,* i2c_direction_t *direction,* uint8_t * *data,* size_t *dataLength,* size_t * *actualLengthTransferred,* uint32_t *timeout_ms*)

Parameters

master	The pointer to the I2C master driver state structure.
flags	The flags to control a transfer.
direction	The direction of an I2C transfer.
data	The pointer to the data to be transfered.
dataLength	The length in bytes of the data to be transfered.
actualLength-Transferred	The length in bytes of the data thansfered.

timeous_ms	A timeout for the transfer in microseconds.

11.3.3.5 i2c_status_t i2c_master_transfer (i2c_master_t * *master*, const i2c_device_t * *device*, i2c_direction_t *direction*, bool *stopAfterTransfer*, uint32_t *subaddress*, size_t *subaddressLength*, uint8_t * *data*, size_t *dataLength*, size_t * *actualLengthTransferred*, uint32_t *timeout_ms*)

Parameters

master	The pointer to the I2C master driver state structure.
device	The pointer to the I2C device information struct.
direction	The direction of an I2C transfer.
stopAfter-Transfer	Send STOP signal after this transfer or not.
subaddress	The subaddress for a device if it has.
subaddress-Length	The length of the subaddress.
data	The pointer to the data to be transfered.
dataLength	The length in bytes of the data to be transfered.
actualLength-Transferred	The length in bytes of the data thansfered.
timeous_ms	A timeout for the transfer in microseconds.

11.4 I2C Classes

The chapter describes the C++ classes of the I2C Peripheral driver.

Chapter 12
Interrupt Manager (Interrupt)

The Kinetis SDK Interrupt Manager provides a set of API/services to configure the Interrupt Controller (NVIC).

interrupt_manager APIs

- void interrupt_register_handler (IRQn_Type irqNumber, void(∗handler)(void))
  Installs an interrupt handler routine for a given IRQ number.
- static void interrupt_enable (IRQn_Type irqNumber)
  Enables an interrupt for a given IRQ number.
- static void interrupt_disable (IRQn_Type irqNumber)
  Disables an interrupt for a given IRQ number.
- void interrupt_enable_global (void)
  Enables system interrupt.
- void interrupt_disable_global (void)
  Disable system interrupt.

12.0.1 Interrupt Manager

Overview

The Interrupt Manager provides a set of APIs so that the application can enable or disable an interrupt for a specific device and also set/get interrupt status, priority and other features. Also it provides a way to update the vector table for a specific device interrupt handler.

Interrupt Names

Each chip has its own set of supported interrupt names defined in the chip-specific header file. For example, for K70, the header file is MK70F12.h or MK70F15.h as an IRQn_Type.

Example to set/update the vector table for the I2C0_IRQn interrupt handler:

```
#include "interrupt/fsl_interrupt_manager.h"

interrupt_register_handler(I2C0_IRQn, irq_handler_I2C0_IRQn);
```

Example to enable/disable an interrupt for the I2C0_IRQn

```
#include "interrupt/fsl_interrupt_manager.h"

interrupt_enable(I2C0_IRQn);

interrupt_disable(I2C0_IRQn);
```

12.1 Function Documentation

12.1.1 void interrupt_register_handler (IRQn_Type *irqNumber,* void(∗)(void) *handler*)

This function lets the application register/replace the interrupt handler for a specified IRQ number. The IRQ number is different than the vector number. IRQ 0 starts from the vector 16 address. See a chip-specific reference manual for details and the startup_MKxxxx.s file for each chip family to find out the default interrupt handler for each device. This function converts the IRQ number to the vector number by adding 16 to it.

Parameters

irqNumber	IRQ number
handler	Interrupt handler routine address pointer

12.1.2 static void interrupt_enable (IRQn_Type *irqNumber*) [inline], [static]

This function enables the individual interrupt for a specified IRQ number. It calls the system NVIC API to access the interrupt control register. The input IRQ number does not include the core interrupt, only the peripheral interrupt, from 0 to a maximum supported IRQ.

Parameters

irqNumber	IRQ number

12.1.3 static void interrupt_disable (IRQn_Type *irqNumber*) [inline], [static]

This function enables the individual interrupt for a specified IRQ number. It calls the system NVIC API to access the interrupt control register.

Parameters

irqNumber	IRQ number

12.1.4 void interrupt_enable_global (void)

This function enables the global interrupt by calling the core API.

12.1.5 void interrupt_disable_global (void)

This function disables the global interrupt by calling the core API.

Chapter 13
Low Power Universal Asynchronous Receiver/Transmitter (LPUART)

The Kinetis SDK provides both HAL and Peripheral drivers for the Low Power Universal Asynchronous Receiver/Transmitter (LPUART) block of Kinetis devices.

Modules

- LPUART HAL driver
  The part describes the programming interface of the LPUART HAL driver.
- LPUART Types Definitions
  The part describes the LPUART type definitions.
- LPUART peripheral Driver
  The part describes the programming interface of the LPUART Peripheral driver.

13.1 LPUART HAL driver

The chapter describes the programming interface of the LPUART HAL driver.

Data Structures

- struct lpuart_idle_line_config_t
  Structure for idle line configuration settings. *More...*
- struct lpuart_status_t
  Structure for all LPUART status flags. *More...*
- struct lpuart_interrupt_config_t
  LPUART interrupt configuration structure, default settings are 0 (disabled) *More...*
- struct lpuart_config_t
  LPUART configuration structure. *More...*

Enumerations

- enum lpuart_operation_config_t {
  kLpuartOperates = 0,
  kLpuartStops = 1 }
  LPUART operation configuration constants.
- enum lpuart_wakeup_method_t {
  kLpuartIdleLineWake = 0,
  kLpuartAddrMarkWake = 1 }
  LPUART wakeup from standby method constants.
- enum lpuart_idle_line_select_t {
  kLpuartIdleLineAfterStartBit = 0,
  kLpuartIdleLineAfterStopBit = 1 }
  LPUART idle line detect selection types.
- enum lpuart_break_char_length_t {
  kLpuartBreakChar10BitMinimum = 0,
  kLpuartBreakChar13BitMinimum = 1 }
  LPUART break character length settings for transmit/detect.
- enum lpuart_singlewire_txdir_t {
  kLpuartSinglewireTxdirIn = 0,
  kLpuartSinglewireTxdirOut = 1 }
  LPUART single-wire mode TX direction.
- enum lpuart_match_config_t {
  kLpuartAddressMatchWakeup = 0,
  kLpuartIdleMatchWakeup = 1,
  kLpuartMatchOnAndMatchOff = 2,
  kLpuartEnablesRwuOnDataMatch = 3 }
  LPUART Configures the match addressing mode used.
- enum lpuart_status_flag_t {

kLpuartTransmitDataRegisterEmpty,
kLpuartTransmissionComplete,
kLpuartReceiveDataRegisterFull,
kLpuartIdleLineDetect,
kLpuartReceiveOverrun,
kLpuartNoiseDetect,
kLpuartFrameError,
kLpuartParityError,
kLpuartLineBreakDetect,
kLpuartReceiveActiveEdgeDetect,
kLpuartReceiverActive }
LPUART status flags.

- enum lpuart_ir_tx_pulsewidth_t {
  kLpuartIrThreeSixteenthsWidth = 0,
  kLpuartIrOneSixteenthWidth = 1,
  kLpuartIrOneThirtysecondsWidth = 2,
  kLpuartIrOneFourthWidth = 3 }
  LPUART infrared transmitter pulse width options.
- enum lpuart_idle_config_t {
  kLpuart_1_IdleChar = 0,
  kLpuart_2_IdleChar = 1,
  kLpuart_4_IdleChar = 2,
  kLpuart_8_IdleChar = 3,
  kLpuart_16_IdleChar = 4,
  kLpuart_32_IdleChar = 5,
  kLpuart_64_IdleChar = 6,
  kLpuart_128_IdleChar = 7 }
  LPUART Configures the number of idle characters that must be received before the IDLE flag is set.
- enum lpuart_cts_source_t {
  kLpuartCtsSourcePin = 0,
  kLpuartCtsSourceInvertedReceiverMatch = 1 }
  LPUART Transmits the CTS Configuration.
- enum lpuart_cts_config_t {
  kLpuartCtsSampledOnEachCharacter = 0,
  kLpuartCtsSampledOnIdle = 1 }
  LPUART Transmits CTS Source.Configures if the CTS state is checked at the start of each character or only when the transmitter is idle.

LPUART Common Configurations

- status_t lpuart_hal_init (uint32_t lpuartInstance, const lpuart_config_t *config)
  Initializes the LPUART controller (see lpuart_config_t struct for initialization details).
- status_t lpuart_hal_set_baud_rate (uint32_t lpuartInstance, uint32_t sourceClockInHz, uint32_t desiredBaudRate)
  Configures the LPUART baud rate.
- status_t lpuart_hal_set_baud_rate_divisor (uint32_t lpuartInstance, uint32_t baudRateDivisor)

Sets the LPUART baud rate modulo divisor.

- status_t lpuart_hal_configure_bit_count_per_char (uint32_t lpuartInstance, lpuart_bit_count_per_-char_t bitCountPerChar)
  Configures the number of bits per character in the LPUART controller In some LPUART instances, the user should disable the transmitter/receiver before calling this function.
- void lpuart_hal_configure_parity_mode (uint32_t lpuartInstance, lpuart_parity_mode_t parityModeType)
  Configures parity mode in the LPUART controller.
- status_t lpuart_hal_configure_stop_bit_count (uint32_t lpuartInstance, lpuart_stop_bit_count_t stopBitCount)
  Configures the number of stop bits in the LPUART controller.
- void lpuart_hal_configure_tx_rx_inversion (uint32_t lpuartInstance, uint32_t rxInvert, uint32_t txInvert)
  Configures the transmit and receive inversion control in the LPUART controller.
- void lpuart_hal_enable_transmitter (uint32_t lpuartInstance)
  Enables the LPUART transmitter.
- void lpuart_hal_disable_transmitter (uint32_t lpuartInstance)
  Disables the LPUART transmitter.
- bool lpuart_hal_is_transmitter_enabled (uint32_t lpuartInstance)
  Gets the LPUART transmitter enabled/disabled configuration.
- void lpuart_hal_enable_receiver (uint32_t lpuartInstance)
  Enables the LPUART receiver.
- void lpuart_hal_disable_receiver (uint32_t lpuartInstance)
  Disables the LPUART receiver.
- bool lpuart_hal_is_receiver_enabled (uint32_t lpuartInstance)
  Gets the LPUART receiver enabled/disabled configuration.

LPUART Interrupts and DMA

- void lpuart_hal_configure_interrupts (uint32_t lpuartInstance, const lpuart_interrupt_config_t *interruptConfig)
  Configures the LPUART module interrupts to enable/disable various interrupt sources.
- void lpuart_hal_enable_break_detect_interrupt (uint32_t lpuartInstance)
  Enables the break_detect_interrupt.
- void lpuart_hal_disable_break_detect_interrupt (uint32_t lpuartInstance)
  Disables the break_detect_interrupt.
- bool lpuart_hal_is_break_detect_interrupt_enabled (uint32_t lpuartInstance)
  Gets the configuration of the break_detect_interrupt enable.
- void lpuart_hal_enable_rx_active_edge_interrupt (uint32_t lpuartInstance)
  Enables the rx_active_edge_interrupt.
- void lpuart_hal_disable_rx_active_edge_interrupt (uint32_t lpuartInstance)
  Disables the rx_active_edge_interrupt.
- bool lpuart_hal_is_rx_active_edge_interrupt_enabled (uint32_t lpuartInstance)
  Enables the configuration of the rx_active_edge_interrupt.
- void lpuart_hal_enable_tx_data_register_empty_interrupt (uint32_t lpuartInstance)
  Enables the tx_data_register_empty_interrupt.
- void lpuart_hal_disable_tx_data_register_empty_interrupt (uint32_t lpuartInstance)
  Disables the tx_data_register_empty_interrupt.
- bool lpuart_hal_is_tx_data_register_empty_interrupt_enabled (uint32_t lpuartInstance)

Gets the configuration of the tx_data_register_empty_interrupt enable.
- void lpuart_hal_enable_transmission_complete_interrupt (uint32_t lpuartInstance)
  Enables the transmission_complete_interrupt.
- void lpuart_hal_disable_transmission_complete_interrupt (uint32_t lpuartInstance)
  Disables the transmission_complete_interrupt.
- bool lpuart_hal_is_transmission_complete_interrupt_enabled (uint32_t lpuartInstance)
  Gets the configuration of the transmission_complete_interrupt enable.
- void lpuart_hal_enable_rx_data_register_full_interrupt (uint32_t lpuartInstance)
  Enables the rx_data_register_full_interrupt.
- void lpuart_hal_disable_rx_data_register_full_interrupt (uint32_t lpuartInstance)
  Disables the rx_data_register_full_interrupt.
- bool lpuart_hal_is_receive_data_full_interrupt_enabled (uint32_t lpuartInstance)
  Gets the configuration of the rx_data_register_full_interrupt enable.
- void lpuart_hal_enable_idle_line_interrupt (uint32_t lpuartInstance)
  Enables the idle_line_interrupt.
- void lpuart_hal_disable_idle_line_interrupt (uint32_t lpuartInstance)
  Disables the idle_line_interrupt.
- bool lpuart_hal_is_idle_line_interrupt_enabled (uint32_t lpuartInstance)
  Gets the configuration of the idle_line_interrupt enable.
- void lpuart_hal_enable_rx_overrun_interrupt (uint32_t lpuartInstance)
  Enables the rx_overrun_interrupt.
- void lpuart_hal_disable_rx_overrun_interrupt (uint32_t lpuartInstance)
  Disables the rx_overrun_interrupt.
- bool lpuart_hal_is_rx_overrun_interrupt_enabled (uint32_t lpuartInstance)
  Gets the configuration of the rx_overrun_interrupt enable.
- void lpuart_hal_enable_noise_error_interrupt (uint32_t lpuartInstance)
  Enables the noise_error_interrupt.
- void lpuart_hal_disable_noise_error_interrupt (uint32_t lpuartInstance)
  Disables the noise_error_interrupt.
- bool lpuart_hal_is_noise_error_interrupt_enabled (uint32_t lpuartInstance)
  Gets the configuration of the noise_error_interrupt enable.
- void lpuart_hal_enable_framing_error_interrupt (uint32_t lpuartInstance)
  Enables the framing_error_interrupt.
- void lpuart_hal_disable_framing_error_interrupt (uint32_t lpuartInstance)
  Disables the framing_error_interrupt.
- bool lpuart_hal_is_framing_error_interrupt_enabled (uint32_t lpuartInstance)
  Gets the configuration of the framing_error_interrupt enable.
- void lpuart_hal_enable_parity_error_interrupt (uint32_t lpuartInstance)
  Enables the parity_error_interrupt.
- void lpuart_hal_disable_parity_error_interrupt (uint32_t lpuartInstance)
  Disables the parity_error_interrupt.
- bool lpuart_hal_is_parity_error_interrupt_enabled (uint32_t lpuartInstance)
  Gets the configuration of the parity_error_interrupt enable.
- void lpuart_hal_configure_dma (uint32_t lpuartInstance, bool txDmaConfig, bool rxDmaConfig)
  LPUART configures DMA requests for Transmitter and Receiver.
- bool lpuart_hal_is_txdma_enabled (uint32_t lpuartInstance)
  Gets the LPUART Transmit DMA request configuration.
- bool lpuart_hal_is_rxdma_enabled (uint32_t lpuartInstance)
  Gets the LPUART receive DMA request configuration.

LPUART Transfer Functions

- void lpuart_hal_putchar (uint32_t lpuartInstance, uint8_t data)
  Sends the LPUART 8-bit character.
- void lpuart_hal_putchar9 (uint32_t lpuartInstance, uint16_t data)
  Sends the LPUART 9-bit character.
- status_t lpuart_hal_putchar10 (uint32_t lpuartInstance, uint16_t data)
  Sends the LPUART 10-bit character (Note: Feature available on select LPUART instances).
- void lpuart_hal_getchar (uint32_t lpuartInstance, uint8_t *readData)
  Gets the LPUART 8-bit character.
- void lpuart_hal_getchar9 (uint32_t lpuartInstance, uint16_t *readData)
  Gets the LPUART 9-bit character.
- status_t lpuart_hal_getchar10 (uint32_t lpuartInstance, uint16_t *readData)
  Gets the LPUART 10-bit character.
- void lpuart_hal_idleconfig (uint32_t lpuartInstance, lpuart_idle_config_t idle_config)
  Configures the number of idle characters that must be received before the IDLE flag is set.
- lpuart_idle_config_t lpuart_hal_get_idleconfig (uint32_t lpuartInstance)
  Gets the configuration of the number of idle characters that must be received before the IDLE flag is set.

LPUART Special Feature Configurations

- void lpuart_hal_configure_wait_mode_operation (uint32_t lpuartInstance, lpuart_operation_config_t mode)
  Configures the LPUART operation in wait mode (operates or stops operations in wait mode).
- lpuart_operation_config_t lpuart_hal_get_wait_mode_operation_config (uint32_t lpuartInstance)
  Gets the LPUART operation in wait mode (operates or stops operations in wait mode).
- void lpuart_hal_configure_loopback_mode (uint32_t lpuartInstance, bool enable)
  Configures the LPUART loopback operation (enable/disable loopback operation) In some LPUART instances, the user should disable the transmitter/receiver before calling this function.
- void lpuart_hal_configure_singlewire_mode (uint32_t lpuartInstance, bool enable)
  Configures the LPUART single-wire operation (enable/disable single-wire mode) In some LPUART instances, the user should disable the transmitter/receiver before calling this function.
- void lpuart_hal_configure_txdir_in_singlewire_mode (uint32_t lpuartInstance, lpuart_singlewire_txdir_t direction)
  Configures the LPUART transmit direction while in single-wire mode.
- status_t lpuart_hal_put_receiver_in_standby_mode (uint32_t lpuartInstance)
  Places the LPUART receiver in standby mode.
- void lpuart_hal_put_receiver_in_normal_mode (uint32_t lpuartInstance)
  Places the LPUART receiver in a normal mode (disable standby mode operation).
- bool lpuart_hal_is_receiver_in_standby (uint32_t lpuartInstance)
  Checks whether the LPUART receiver is in a standby mode.
- void lpuart_hal_select_receiver_wakeup_method (uint32_t lpuartInstance, lpuart_wakeup_method_t method)
  LPUART receiver wakeup method (idle line or addr-mark) from standby mode.
- lpuart_wakeup_method_t lpuart_hal_get_receiver_wakeup_method (uint32_t lpuartInstance)
  Gets the LPUART receiver wakeup method (idle line or addr-mark) from standby mode.
- void lpuart_hal_configure_idle_line_detect (uint32_t lpuartInstance, const lpuart_idle_line_config_t *config)
  LPUART idle-line detect operation configuration (idle line bit-count start and wake up affect on IDLE

status bit).

- void lpuart_hal_set_break_char_transmit_length (uint32_t lpuartInstance, lpuart_break_char_length_t length)
  LPUART break character transmit length configuration In some LPUART instances, the user should disable the transmitter before calling this function.
- void lpuart_hal_set_break_char_detect_length (uint32_t lpuartInstance, lpuart_break_char_length_t length)
  LPUART break character detect length configuration.
- void lpuart_hal_queue_break_char_to_send (uint32_t lpuartInstance, bool enable)
  LPUART transmit send break character configuration.
- status_t lpuart_hal_configure_match_address_operation (uint32_t lpuartInstance, bool matchAddrMode1, bool matchAddrMode2, uint8_t matchAddrValue1, uint8_t matchAddrValue2, lpuart_match_config_t config)
  LPUART configure match address mode control (Note: Feature available on select LPUART instances)
- status_t lpuart_hal_configure_send_msb_first_operation (uint32_t lpuartInstance, bool enable)
  LPUART sends the MSB first configuration (Note: Feature available on select LPUART instances) In some LPUART instances, the user should disable the transmitter/receiver before calling this function.
- status_t lpuart_hal_configure_receive_resync_disable_operation (uint32_t lpuartInstance, bool enable)
  LPUART disables re-sync of received data configuration (Note: Feature available on select LPUART instances).

LPUART Status Flags

- void lpuart_hal_get_all_status_flag (uint32_t lpuartInstance, lpuart_status_t *allStatusFlag)
  LPUART get all status flag.
- bool lpuart_hal_is_transmit_data_register_empty (uint32_t lpuartInstance)
  Gets the LPUART transmit data register empty flag.
- bool lpuart_hal_is_transmission_complete (uint32_t lpuartInstance)
  Gets the LPUART transmission complete flag.
- bool lpuart_hal_is_receive_data_register_full (uint32_t lpuartInstance)
  Gets the LPUART receive data register full flag.
- bool lpuart_hal_is_idle_line_detected (uint32_t lpuartInstance)
  Gets the LPUART idle line detect flag.
- bool lpuart_hal_is_receive_overrun_detected (uint32_t lpuartInstance)
  Gets the LPUART receiver overrun status.
- bool lpuart_hal_is_noise_detected (uint32_t lpuartInstance)
  Gets the LPUART noise flag.
- bool lpuart_hal_is_frame_error_detected (uint32_t lpuartInstance)
  Gets the LPUART frame error flag.
- bool lpuart_hal_is_parity_error_detected (uint32_t lpuartInstance)
  Gets the LPUART parity error flag.
- bool lpuart_hal_is_line_break_detected (uint32_t lpuartInstance)
  Gets the LPUART LIN break detect interrupt flag.
- bool lpuart_hal_is_receive_active_edge_detected (uint32_t lpuartInstance)
  Gets the LPUART receive pin active edge interrupt flag.
- bool lpuart_hal_is_receiver_active (uint32_t lpuartInstance)
  Gets LPUART Receiver Active Flag (RAF).
- status_t lpuart_hal_clear_status_flag (uint32_t lpuartInstance, lpuart_status_flag_t statusFlag)

LPUART clears an individual status flag (see lpuart_status_flag_t for list of status bits).

- void lpuart_hal_clear_all_non_autoclear_status_flags (uint32_t lpuartInstance)
  LPUART clears ALL status flags.

13.1.1 Data Structure Documentation

13.1.1.1 struct lpuart_idle_line_config_t

Data Fields

- unsigned idleLineType: 1
  ILT, Idle bit count start: 0 - after start bit (default),.
- unsigned rxWakeIdleDetect: 1
  1 - after stop bit

13.1.1.1.0.23 Field Documentation

13.1.1.1.0.23.1 unsigned lpuart_idle_line_config_t::rxWakeIdleDetect

RWUID, Receiver Wake Up Idle Detect. IDLE status bit

13.1.1.2 struct lpuart_status_t

Data Fields

- unsigned transmitDataRegisterEmpty: 1
  Transmit data register empty flag, sets when.
- unsigned transmissionComplete: 1
  transmit buffer is empty
- unsigned receiveDataRegisterFull: 1
  is idle (transmission activity complete)
- unsigned idleLineDetect: 1
  receive data buffer is full
- unsigned receiveOverrun: 1
  Receiver Overrun sets when new data is received.
- unsigned noiseDetect: 1
  before data is read from receive register
- unsigned frameError: 1
  If any of these samples differ, noise flag sets.
- unsigned parityError: 1
  where stop bit expected
- unsigned lineBreakDetect: 1
  error detection
- unsigned receiveActiveEdgeDetect: 1
  LIN break char detected and LIN circuit enabled.
- unsigned receiverActive: 1
  when active edge detected

13.1.1.2.0.24 Field Documentation

13.1.1.2.0.24.1 unsigned lpuart_status_t::transmissionComplete

Transmission complete flag, sets when transmitter

13.1.1.2.0.24.2 unsigned lpuart_status_t::receiveDataRegisterFull

Receive data register full flag, sets when the

13.1.1.2.0.24.3 unsigned lpuart_status_t::idleLineDetect

Idle line detect flag, sets when idle line detected

13.1.1.2.0.24.4 unsigned lpuart_status_t::noiseDetect

Receiver takes 3 samples of each received bit.

13.1.1.2.0.24.5 unsigned lpuart_status_t::frameError

Frame error flag, sets if logic 0 was detected

13.1.1.2.0.24.6 unsigned lpuart_status_t::parityError

If parity enabled, will set upon parity

13.1.1.2.0.24.7 unsigned lpuart_status_t::lineBreakDetect

LIN break detect interrupt flag, sets when

13.1.1.2.0.24.8 unsigned lpuart_status_t::receiveActiveEdgeDetect

Receive pin active edge interrupt flag, sets

13.1.1.2.0.24.9 unsigned lpuart_status_t::receiverActive

Receiver Active Flag (RAF), sets a beginning of

13.1.1.3 struct lpuart_interrupt_config_t

Data Fields

- unsigned linBreakDetect: 1
  LIN break detect: 0 - disable interrupt,.
- unsigned rxActiveEdge: 1
  1 - enable interrupt
- unsigned transmitDataRegisterEmpty: 1
  1 - enable interrupt
- unsigned transmitComplete: 1
  0 - disable interrupt, 1 - enable interrupt

- unsigned receiverDataRegisterFull: 1
  1 - enable interrupt
- unsigned idleLine: 1
  0 - disable interrupt, 1 - enable interrupt
- unsigned receiverOverrun: 1
  Receiver Overrun: 0 - disable interrupt,.
- unsigned noiseErrorFlag: 1
  1 - enable interrupt
- unsigned frameErrorFlag: 1
  1 - enable interrupt
- unsigned parityErrorFlag: 1
  1 - enable interrupt

13.1.1.3.0.25 Field Documentation

13.1.1.3.0.25.1 unsigned lpuart_interrupt_config_t::rxActiveEdge

RX Active Edge: 0 - disable interrupt,

13.1.1.3.0.25.2 unsigned lpuart_interrupt_config_t::transmitDataRegisterEmpty

Transmit data register empty:

13.1.1.3.0.25.3 unsigned lpuart_interrupt_config_t::transmitComplete

Transmission complete: 0 - disable interrupt,

13.1.1.3.0.25.4 unsigned lpuart_interrupt_config_t::receiverDataRegisterFull

Receiver data register full:

13.1.1.3.0.25.5 unsigned lpuart_interrupt_config_t::idleLine

Idle line: 0 - disable interrupt, 1 - enable interrupt

13.1.1.3.0.25.6 unsigned lpuart_interrupt_config_t::noiseErrorFlag

Noise error flag: 0 - disable interrupt,

13.1.1.3.0.25.7 unsigned lpuart_interrupt_config_t::frameErrorFlag

Framing error flag: 0 - disable interrupt,

13.1.1.3.0.25.8 unsigned lpuart_interrupt_config_t::parityErrorFlag

Parity error flag: 0 - disable interrupt,

13.1.1.4 struct lpuart_config_t

Data Fields

- uint32_t lpuartSourceClockInHz
  LPUART module source clock in Hertz.
- uint32_t baudRate
  LPUART baud rate.
- lpuart_parity_mode_t parityMode
  parity mode, disabled (default), even, odd
- lpuart_stop_bit_count_t stopBitCount
  number of stop bits, 1 stop bit (default)
- lpuart_bit_count_per_char_t bitCountPerChar
  or 2 stop bits
- unsigned rxDataInvert: 1
  in a word (up to 10-bits in
- unsigned txDataInvert: 1
  1 - inverted

13.1.1.4.0.26 Field Documentation

13.1.1.4.0.26.1 lpuart_bit_count_per_char_t lpuart_config_t::bitCountPerChar

number of bits, 8-bit (default) or 9-bit

13.1.1.4.0.26.2 unsigned lpuart_config_t::rxDataInvert

some LPUART instances) Receive Data Inversion: 0 - not inverted (default),

13.1.1.4.0.26.3 unsigned lpuart_config_t::txDataInvert

Transmit Data Inversion: 0 - not inverted (default),

13.1.2 Enumeration Type Documentation

13.1.2.1 enum lpuart_operation_config_t

Enumerator

kLpuartOperates LPUART continues to operate normally.
kLpuartStops LPUART stops operation.

13.1.2.2 enum lpuart_wakeup_method_t

Enumerator

kLpuartIdleLineWake Idle-line wakes the LPUART receiver from standby.
kLpuartAddrMarkWake Addr-mark wakes LPUART receiver from standby.

13.1.2.3 enum lpuart_idle_line_select_t

Enumerator

kLpuartIdleLineAfterStartBit LPUART idle character bit count start after start bit.
kLpuartIdleLineAfterStopBit LPUART idle character bit count start after stop bit.

13.1.2.4 enum lpuart_break_char_length_t

The actual maximum bit times may vary depending on the LPUART instance.

Enumerator

kLpuartBreakChar10BitMinimum LPUART break char length 10 bit times (if M = 0, SBNS = 0) or.
kLpuartBreakChar13BitMinimum 11 (if M = 1, SBNS = 0 or M = 0, SBNS = 1) or 12 (if M = 1, SBNS = 1 or M10 = 1, SNBS = 0) or 13 (if M10 = 1, SNBS = 1 LPUART break char length 13 bit times (if M = 0, SBNS = 0) or

13.1.2.5 enum lpuart_singlewire_txdir_t

Enumerator

kLpuartSinglewireTxdirIn LPUART Single Wire mode TXDIR input.
kLpuartSinglewireTxdirOut LPUART Single Wire mode TXDIR output.

13.1.2.6 enum lpuart_match_config_t

Enumerator

kLpuartAddressMatchWakeup LPUART Address Match Wakeup.
kLpuartIdleMatchWakeup LPUART Idle Match Wakeup.
kLpuartMatchOnAndMatchOff LPUART Match On and Match Off.
kLpuartEnablesRwuOnDataMatch LPUART Enables RWU on Data Match and Match On/Off for transmitter CTS input.

13.1.2.7 enum lpuart_status_flag_t

Enumerator

kLpuartTransmitDataRegisterEmpty Transmit data register empty flag, sets when transmit.
kLpuartTransmissionComplete buffer is empty Transmission complete flag, sets when transmitter is idle

kLpuartReceiveDataRegisterFull (transmission activity complete) Receive data register full flag, sets when the receive data
kLpuartIdleLineDetect buffer is full Idle line detect flag, sets when idle line detected
kLpuartReceiveOverrun Receiver Overrun sets when new data is received before data.
kLpuartNoiseDetect is read from the receive register Receiver takes 3 samples of each received bit. If any of
kLpuartFrameError these samples differs, noise flag sets Frame error flag, sets if logic 0 was detected where stop
kLpuartParityError bit expected If parity enabled, will set upon parity error detection
kLpuartLineBreakDetect LIN break detect interrupt flag, sets when.
kLpuartReceiveActiveEdgeDetect LIN break char detected and LIN circuit enabled. Receive pin active edge interrupt flag, sets when active
kLpuartReceiverActive edge detected Receiver Active Flag (RAF), sets a beginning of valid start bit

13.1.2.8 enum lpuart_ir_tx_pulsewidth_t

Enumerator

kLpuartIrThreeSixteenthsWidth 3/16 pulse
kLpuartIrOneSixteenthWidth 1/16 pulse
kLpuartIrOneThirtysecondsWidth 1/32 pulse
kLpuartIrOneFourthWidth 1/4 pulse

13.1.2.9 enum lpuart_idle_config_t

Enumerator

kLpuart_1_IdleChar 1 idle character
kLpuart_2_IdleChar 2 idle character
kLpuart_4_IdleChar 4 idle character
kLpuart_8_IdleChar 8 idle character
kLpuart_16_IdleChar 16 idle character
kLpuart_32_IdleChar 32 idle character
kLpuart_64_IdleChar 64 idle character
kLpuart_128_IdleChar 128 idle character

13.1.2.10 enum lpuart_cts_source_t

Configures the source of the CTS input.

Enumerator

kLpuartCtsSourcePin LPUART CTS input is the LPUART_CTS pin.

kLpuartCtsSourceInvertedReceiverMatch LPUART CTS input is the inverted Receiver Match result.

13.1.2.11 enum lpuart_cts_config_t

Enumerator

kLpuartCtsSampledOnEachCharacter LPUART CTS input is sampled at the start of each character.

kLpuartCtsSampledOnIdle LPUART CTS input is sampled when the transmitter is idle.

13.1.3 Function Documentation

13.1.3.1 status_t lpuart_hal_init (uint32_t *lpuartInstance,* const lpuart_config_t * *config*)

Parameters

lpuartInstance	LPUART instance number.
config	LPUART configuration data.

Returns

An error code or kStatus_Success

13.1.3.2 status_t lpuart_hal_set_baud_rate (uint32_t *lpuartInstance,* uint32_t *sourceClockInHz,* uint32_t *desiredBaudRate*)

In some LPUART instances the user must disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

lpuartInstance	LPUART instance number.
sourceClockInHz	LPUART source input clock in Hz.

desiredBaud-Rate	LPUART desired baud rate.

Returns

An error code or kStatus_Success

13.1.3.3 status_t lpuart_hal_set_baud_rate_divisor (uint32_t *lpuartInstance,* uint32_t *baudRateDivisor*)

Parameters

lpuartInstance	LPUART instance number.
baudRate-Divisor	The baud rate modulo division "SBR"

Returns

An error code or kStatus_Success

13.1.3.4 status_t lpuart_hal_configure_bit_count_per_char (uint32_t *lpuartInstance,* lpuart_bit_count_per_char_t *bitCountPerChar*)

Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

lpuartInstance	LPUART instance number.
bitCountPer-Char	Number of bits per char (8, 9, or 10, depending on the LPUART instance)

Returns

An error code or kStatus_Success

13.1.3.5 void lpuart_hal_configure_parity_mode (uint32_t *lpuartInstance,* lpuart_parity_mode_t *parityModeType*)

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

lpuartInstance	LPUART instance number.
parityMode-Type	Parity mode (enabled, disable, odd, even - see parity_mode_t struct)

13.1.3.6 status_t lpuart_hal_configure_stop_bit_count (uint32_t *lpuartInstance,* lpuart_stop_bit_count_t *stopBitCount*)

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

lpuartInstance	LPUART instance number.
stopBitCount	Number of stop bits (1 or 2 - see lpuart_stop_bit_count_t struct)

Returns

An error code (an unsupported setting in some LPUARTs) or kStatus_Success

13.1.3.7 void lpuart_hal_configure_tx_rx_inversion (uint32_t *lpuartInstance,* uint32_t *rxInvert,* uint32_t *txInvert*)

This function should only be called when the LPUART is between transmit and receive packets.

Parameters

lpuartInstance	LPUART instance number.
rxInvert	Enable (1) or disable (0) receive inversion
txInvert	Enable (1) or disable (0) transmit inversion

13.1.3.8 void lpuart_hal_enable_transmitter (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number.

13.1.3.9 void lpuart_hal_disable_transmitter (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.10 bool lpuart_hal_is_transmitter_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

State of LPUART transmitter enable(1)/disable(0)

13.1.3.11 void lpuart_hal_enable_receiver (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.12 void lpuart_hal_disable_receiver (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.13 bool lpuart_hal_is_receiver_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

State of LPUART receiver enable(1)/disable(0)

13.1.3.14 void lpuart_hal_configure_interrupts (uint32_t *lpuartInstance,* const lpuart_interrupt_config_t * *interruptConfig*)

Parameters

lpuartInstance	LPUART instance number
interruptConfig	LPUART interrupt configuration data

13.1.3.15 void lpuart_hal_enable_break_detect_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.16 void lpuart_hal_disable_break_detect_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.17 bool lpuart_hal_is_break_detect_interrupt_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Bit setting of the interrupt enable bit

13.1.3.18 void lpuart_hal_enable_rx_active_edge_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.19 void lpuart_hal_disable_rx_active_edge_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number.

13.1.3.20 bool lpuart_hal_is_rx_active_edge_interrupt_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Bit setting of the interrupt enable bit

13.1.3.21 void lpuart_hal_enable_tx_data_register_empty_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.22 void lpuart_hal_disable_tx_data_register_empty_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.23 bool lpuart_hal_is_tx_data_register_empty_interrupt_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Bit setting of the interrupt enable bit

13.1.3.24 void lpuart_hal_enable_transmission_complete_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.25 void lpuart_hal_disable_transmission_complete_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.26 bool lpuart_hal_is_transmission_complete_interrupt_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Bit setting of the interrupt enable bit

13.1.3.27 void lpuart_hal_enable_rx_data_register_full_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.28 void lpuart_hal_disable_rx_data_register_full_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.29 bool lpuart_hal_is_receive_data_full_interrupt_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Bit setting of the interrupt enable bit

13.1.3.30 void lpuart_hal_enable_idle_line_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.31 void lpuart_hal_disable_idle_line_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.32 bool lpuart_hal_is_idle_line_interrupt_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Bit setting of the interrupt enable bit

13.1.3.33 void lpuart_hal_enable_rx_overrun_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.34 void lpuart_hal_disable_rx_overrun_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.35 bool lpuart_hal_is_rx_overrun_interrupt_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Bit setting of the interrupt enable bit

13.1.3.36 void lpuart_hal_enable_noise_error_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.37 void lpuart_hal_disable_noise_error_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.38 bool lpuart_hal_is_noise_error_interrupt_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Bit setting of the interrupt enable bit

13.1.3.39 void lpuart_hal_enable_framing_error_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.40 void lpuart_hal_disable_framing_error_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.41 bool lpuart_hal_is_framing_error_interrupt_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Bit setting of the interrupt enable bit

13.1.3.42 void lpuart_hal_enable_parity_error_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.43 void lpuart_hal_disable_parity_error_interrupt (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.44 bool lpuart_hal_is_parity_error_interrupt_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Bit setting of the interrupt enable bit

13.1.3.45 void lpuart_hal_configure_dma (uint32_t *lpuartInstance,* bool *txDmaConfig,* bool *rxDmaConfig*)

Parameters

lpuartInstance	LPUART instance number
txDmaConfig	Transmit DMA request configuration (enable:1 /disable: 0)
rxDmaConfig	Receive DMA request configuration (enable: 1/disable: 0)

13.1.3.46 bool lpuart_hal_is_txdma_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Transmit DMA request configuration (enable: 1/disable: 0)

13.1.3.47 bool lpuart_hal_is_rxdma_enabled (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Receives the DMA request configuration (enable: 1/disable: 0).

13.1.3.48 void lpuart_hal_putchar (uint32_t *lpuartInstance,* uint8_t *data*)

Parameters

lpuartInstance	LPUART Instance
data	data to send (8-bit)

13.1.3.49 void lpuart_hal_putchar9 (uint32_t *lpuartInstance,* uint16_t *data*)

Parameters

lpuartInstance	LPUART Instance
data	data to send (9-bit)

13.1.3.50 status_t lpuart_hal_putchar10 (uint32_t *lpuartInstance,* uint16_t *data*)

Parameters

lpuartInstance	LPUART Instance
data	data to send (10-bit)

Returns

An error code or kStatus_Success

13.1.3.51 void lpuart_hal_getchar (uint32_t *lpuartInstance,* uint8_t * *readData*)

Parameters

lpuartInstance	LPUART instance number
readData	data read from receive (8-bit)

13.1.3.52 void lpuart_hal_getchar9 (uint32_t *lpuartInstance,* uint16_t * *readData*)

Parameters

lpuartInstance	LPUART instance number
readData	data read from receive (9-bit)

13.1.3.53 status_t lpuart_hal_getchar10 (uint32_t *lpuartInstance,* uint16_t * *readData*)

Parameters

lpuartInstance	LPUART instance number
readData	data read from receive (10-bit)

Returns

An error code or kStatus_Success

13.1.3.54 void lpuart_hal_idleconfig (uint32_t *lpuartInstance,* lpuart_idle_config_t *idle_config*)

Parameters

lpuartInstance	LPUART instance number
idle_config	idle characters configuration

13.1.3.55 lpuart_idle_config_t lpuart_hal_get_idleconfig (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

idle characters configuration

13.1.3.56 void lpuart_hal_configure_wait_mode_operation (uint32_t *lpuartInstance,* lpuart_operation_config_t *mode*)

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

lpuartInstance	LPUART instance number

mode	LPUART wait mode operation - operates or stops to operate in wait mode.

13.1.3.57 lpuart_operation_config_t lpuart_hal_get_wait_mode_operation_config (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

LPUART wait mode operation configuration - kLpuartOperates or KLpuartStops in wait mode

13.1.3.58 void lpuart_hal_configure_loopback_mode (uint32_t *lpuartInstance,* bool *enable*)

Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

lpuartInstance	LPUART instance number
enable	LPUART loopback mode - disabled (0) or enabled (1)

13.1.3.59 void lpuart_hal_configure_singlewire_mode (uint32_t *lpuartInstance,* bool *enable*)

Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

lpuartInstance	LPUART instance number
enable	LPUART loopback mode - disabled (0) or enabled (1)

13.1.3.60 void lpuart_hal_configure_txdir_in_singlewire_mode (uint32_t *lpuartInstance,* lpuart_singlewire_txdir_t *direction*)

Parameters

lpuartInstance	LPUART instance number
direction	LPUART single-wire transmit direction - input or output

13.1.3.61 status_t lpuart_hal_put_receiver_in_standby_mode (uint32_t *lpuartInstance*)

In some LPUART instances, before placing LPUART in standby mode, first determine whether the receiver is set to wake on idle or whether it is already in idle state. NOTE that the RWU should only be set with C1[WAKE] = 0 (wakeup on idle) if the channel is currently not idle. This can be determined by the S2[RAF] flag. If it is set to wake up an IDLE event and the channel is already idle, it is possible that the LPUART will discard data since data must be received (or a LIN break detect) after an IDLE is detected and before IDLE is allowed to reasserted.

Parameters

lpuartInstance	LPUART instance number

Returns

Error code or kStatus_Success

13.1.3.62 void lpuart_hal_put_receiver_in_normal_mode (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.1.3.63 bool lpuart_hal_is_receiver_in_standby (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

LPUART in normal more (0) or standby (1)

13.1.3.64 void lpuart_hal_select_receiver_wakeup_method (uint32_t *lpuartInstance,* lpuart_wakeup_method_t *method*)

Parameters

lpuartInstance	LPUART instance number
method	LPUART wakeup method: 0 - Idle-line wake (default), 1 - addr-mark wake

13.1.3.65 lpuart_wakeup_method_t lpuart_hal_get_receiver_wakeup_method (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

LPUART wakeup method: kLpuartIdleLineWake: 0 - Idle-line wake (default), kLpuartAddrMark-Wake: 1 - addr-mark wake

13.1.3.66 void lpuart_hal_configure_idle_line_detect (uint32_t *lpuartInstance,* const lpuart_idle_line_config_t * *config*)

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

lpuartInstance	LPUART instance number
config	LPUART configuration data for idle line detect operation

13.1.3.67 void lpuart_hal_set_break_char_transmit_length (uint32_t *lpuartInstance,* lpuart_break_char_length_t *length*)

Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

lpuartInstance	LPUART instance number

length	LPUART break character length setting: 0 - minimum 10-bit times (default), 1 - minimum 13-bit times

13.1.3.68 void lpuart_hal_set_break_char_detect_length (uint32_t *lpuartInstance,* lpuart_break_char_length_t *length*)

Parameters

lpuartInstance	LPUART instance number
length	LPUART break character length setting: 0 - minimum 10-bit times (default), 1 - minimum 13-bit times

13.1.3.69 void lpuart_hal_queue_break_char_to_send (uint32_t *lpuartInstance,* bool *enable*)

Parameters

lpuartInstance	LPUART instance number
enable	LPUART normal/queue break char - disabled (normal mode, default: 0) or enabled (queue break char: 1)

13.1.3.70 status_t lpuart_hal_configure_match_address_operation (uint32_t *lpuartInstance,* bool *matchAddrMode1,* bool *matchAddrMode2,* uint8_t *matchAddrValue1,* uint8_t *matchAddrValue2,* lpuart_match_config_t *config*)

Parameters

lpuartInstance	LPUART instance number
matchAddr-Mode1	MAEN1: match address mode1 enable (1)/disable (0)
matchAddr-Mode2	MAEN2: match address mode2 enable (1)/disable (0)

matchAddr-Value1	MA: match address value to program into match address register 1
matchAddr-Value2	MA: match address value to program into match address register 2
config	MATCFG: Configures the match addressing mode used.

Returns

An error code or kStatus_Success

13.1.3.71 status_t lpuart_hal_configure_send_msb_first_operation (uint32_t *lpuartInstance,* bool *enable*)

Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

lpuartInstance	LPUART instance number
enable	MSB first mode configuration, MSBF: 0 - LSB (default, feature disabled), 1 - MSB (feature enabled)

Returns

An error code or kStatus_Success

13.1.3.72 status_t lpuart_hal_configure_receive_resync_disable_operation (uint32_t *lpuartInstance,* bool *enable*)

Parameters

lpuartInstance	LPUART instance number
enable	disable re-sync of received data word configuration, RESYNCDIS: 0 - re-sync of received data word (default, feature disabled), 1 - disable the re-sync (feature enabled)

Returns

An error code or kStatus_Success

13.1.3.73 void lpuart_hal_get_all_status_flag (uint32_t *lpuartInstance,* lpuart_status_t * *allStatusFlag*)

Parameters

lpuartInstance	LPUART instance number
allStatusFlag	Structure of status of all LPUART status flags

13.1.3.74 bool lpuart_hal_is_transmit_data_register_empty (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of Transmit data register empty flag, sets when transmit buffer is empty

13.1.3.75 bool lpuart_hal_is_transmission_complete (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of Transmission complete flag, sets when transmitter is idle (transmission activity complete)

13.1.3.76 bool lpuart_hal_is_receive_data_register_full (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of the receive data register full flag, sets when the receive data buffer is full.

13.1.3.77 bool lpuart_hal_is_idle_line_detected (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of Idle line detect flag, sets when idle line detected

13.1.3.78 bool lpuart_hal_is_receive_overrun_detected (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of receiver overrun, sets when new data is received before data is read from receive register

13.1.3.79 bool lpuart_hal_is_noise_detected (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of noise flag, sets if any of the 3 samples taken on receive differ

13.1.3.80 bool lpuart_hal_is_frame_error_detected (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of the frame error flag, sets if logic 0 was detected where stop bit expected

13.1.3.81 bool lpuart_hal_is_parity_error_detected (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of the parity error detection flag, if parity enabled

13.1.3.82 bool lpuart_hal_is_line_break_detected (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of LIN break detect interrupt flag, sets when LIN break char detected if LIN circuit enabled

13.1.3.83 bool lpuart_hal_is_receive_active_edge_detected (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of Receive pin active edge interrupt flag, sets when active edge detected

13.1.3.84 bool lpuart_hal_is_receiver_active (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

Returns

Status of Receiver Active Flag (RAF), sets a beginning of valid start bit

13.1.3.85 status_t lpuart_hal_clear_status_flag (uint32_t *lpuartInstance,* lpuart_status_flag_t *statusFlag*)

Parameters

lpuartInstance	LPUART instance number
statusFlag	Desired LPUART status flag to clear

Returns

An error code or kStatus_Success

13.1.3.86 void lpuart_hal_clear_all_non_autoclear_status_flags (uint32_t *lpuartInstance*)

Parameters

lpuartInstance	LPUART instance number

13.2 LPUART peripheral Driver

The chapter describes the programming interface of the LPUART Peripheral driver.

Data Structures

- struct lpuart_state_t
  Runtime state of the LPUART driver. More...

LPUART Driver

- status_t lpuart_init (uint32_t lpuartInstance, const lpuart_user_config_t *lpuartUserConfig, lpuart_state_t *lpuartState)
  Initialize a LPUART instance for operation.
- status_t lpuart_send_data (lpuart_state_t *lpuartState, uint8_t *sendBuffer, uint32_t txByteCount, uint32_t timeout)
  Send data out through the LPUART module using a blocking method.
- status_t lpuart_send_data_async (lpuart_state_t *lpuartState, uint8_t *sendBuffer, uint32_t txByteCount)
  Send data out through the LPUART module using a non-blocking method.
- status_t lpuart_get_transmit_status (lpuart_state_t *lpuartState, uint32_t *bytesTransmitted)
  Returns whether the previous transmit has finished yet.
- status_t lpuart_get_receive_status (lpuart_state_t *lpuartState, uint32_t *bytesReceived)
  Returns whether the previous receive has finished yet.
- void lpuart_shutdown (lpuart_state_t *lpuartState)
  Shutdown the lpuart by disabling interrupts and transmitter/receiver.
- status_t lpuart_receive_data (lpuart_state_t *lpuartState, uint8_t *rxBuffer, uint32_t requestedByteCount, uint32_t timeout)
  Get data from the LPUART module using a blocking method.
- status_t lpuart_receive_data_async (lpuart_state_t *lpuartState, uint8_t *rxBuffer, uint32_t requestedByteCount)
  Get data from the LPUART module using a non-blocking method.
- status_t lpuart_abort_sending_data (lpuart_state_t *lpuartState)
  Terminates an asynchronous transmission early.
- status_t lpuart_abort_receiving_data (lpuart_state_t *lpuartState)
  Terminates an asynchronous receive early.

13.2.0.87 LPUART Peripheral Driver

Overview

The LPUART peripheral driver transfers data to and from external devices on the Low Power Universal Asynchronous Receiver/Transmitter (LPUART) serial bus. It provides a way to transmit or receive buffers of data with calls to a single function.

Device structures

The driver uses instantiations of the lpuart_tx_state_t and the lpuart_rx_state_t structure to maintain the current state of a particular LPUART instance module driver. The caller provides memory for the driver state structures during the initialization as the driver itself does not statically allocate memory. The structures are provided below:

```
// Runtime transmit state of the LPUART driver.
typedef struct LpuartTxState {
    uint32_t instance;
    bool isTransmitInProgress;
    bool isTransmitAsync;
    const uint8_t * sendBuffer;
    size_t remainingSendByteCount;
    size_t transmittedByteCount;
    sync_object_t irqSync;
    uint8_t fifoEntryCount;
} lpuart_tx_state_t;

// Runtime receive state of the LPUART driver.
typedef struct LpuartRxState {
    uint32_t instance;
    bool isReceiveInProgress;
    bool isReceiveAsync;
    uint8_t * receiveBuffer;
    size_t remainingReceiveByteCount;
    size_t receivedByteCount;
    sync_object_t irqSync;
    uint8_t fifoEntryCount;
} lpuart_rx_state_t;
```

Initialization

To initialize the LPUART driver, call the lpuart_init() function and pass the instance number of the LPUART peripheral you want to use. For instance, to use LPUART0 pass a value of 0 to the initialization function. In addition, you also have to pass a user configuration structure lpuart_user_config_t shown here:

```
// LPUART configuration structure for user
typedef struct LpuartUserConfig {
    uint32_t baudRate;
    lpuart_parity_mode_t parityMode;
    lpuart_stop_bit_count_t stopBitCount;
    lpuart_bit_count_per_char_t bitCountPerChar;
} lpuart_user_config_t;
```

Typically the user configures the lpuart_user_config_t instantiation as an 8-bit-char, no-parity, 1-stop-bit (8-n-1) with a baud rate of 9600 bps. The user can easily modify the lpuart_user_config_t instantiation to configure the LPUART peripheral to a different baud rate or character transfer features. This is a code example to set up a user LPUART configuration instantiation:

```
lpuart_user_config_t lpuartConfig;
lpuartConfig.baudRate = 9600;
lpuartConfig.bitCountPerChar = kLpuart8BitsPerChar;
lpuartConfig.parityMode = kLpuartParityDisabled;
lpuartConfig.stopBitCount = kLpuartOneStopBit;
```

Transfers

The driver implements transmit and receive functions to transfer buffers of data. The driver supports two different modes for transferring data: blocking and non-blocking.

The blocking transmit and receive functions are the lpuart_send_data() and the lpuart_receive_data().

The non-blocking (async) transmit and receive functions are the lpuart_send_data_async() and the lpuart_receive_data_async().

In all of these cases, the functions are interrupt driven.

13.2.1 Data Structure Documentation

13.2.1.1 struct lpuart_state_t

Note, the caller provides memory for the driver state structures during init as the

driver does not statically allocate memory.

Data Fields

- uint32_t instance
  LPUART module instance number.
- bool isTransmitInProgress
  True if there is an active transmit.
- bool isTransmitAsync
  Whether the transmit is asynchronous.
- const uint8_t * sendBuffer
  The buffer of data being sent.
- size_t remainingSendByteCount
  The remaining number of bytes to be transmitted.
- size_t transmittedByteCount
  Number of bytes transmitted so far.
- bool isReceiveInProgress
  True if there is an active receive.
- bool isReceiveAsync
  Whether the receive is asynchronous.
- uint8_t * receiveBuffer
  The buffer of received data.
- size_t remainingReceiveByteCount
  The remaining number of bytes to be received.
- size_t receivedByteCount
  Number of bytes received so far.
- sync_object_t txIrqSync
  Used to wait for ISR to complete its tx business.
- sync_object_t rxIrqSync
  Used to wait for ISR to complete its rx business.
- uint8_t txFifoEntryCount

Number of data word entries in tx FIFO.

- uint8_t rxFifoEntryCount

  Number of data word entries in rx FIFO.

13.2.1.1.0.27 Field Documentation

13.2.1.1.0.27.1 bool lpuart_state_t::isTransmitInProgress

13.2.1.1.0.27.2 bool lpuart_state_t::isTransmitAsync

13.2.1.1.0.27.3 const uint8_t* lpuart_state_t::sendBuffer

13.2.1.1.0.27.4 size_t lpuart_state_t::remainingSendByteCount

13.2.1.1.0.27.5 size_t lpuart_state_t::transmittedByteCount

13.2.1.1.0.27.6 bool lpuart_state_t::isReceiveInProgress

13.2.1.1.0.27.7 bool lpuart_state_t::isReceiveAsync

13.2.1.1.0.27.8 uint8_t* lpuart_state_t::receiveBuffer

13.2.1.1.0.27.9 size_t lpuart_state_t::remainingReceiveByteCount

13.2.1.1.0.27.10 size_t lpuart_state_t::receivedByteCount

13.2.1.1.0.27.11 sync_object_t lpuart_state_t::txIrqSync

13.2.1.1.0.27.12 sync_object_t lpuart_state_t::rxIrqSync

13.2.2 Function Documentation

13.2.2.1 status_t lpuart_init (uint32_t *lpuartInstance*, const lpuart_user_config_t * *lpuartUserConfig*, lpuart_state_t * *lpuartState*)

```
The caller provides memory for the driver state structures during init.
```

Parameters

lpuartInstance	LPUART instance number
lpuartUser-Config	user configuration structure of type lpuart_user_config_t

lpuartState	Pointer to the LPUART driver state structure

Returns

An error code or kStatus_Success

13.2.2.2 status_t lpuart_send_data (lpuart_state_t * *lpuartState*, uint8_t * *sendBuffer*, uint32_t *txByteCount*, uint32_t *timeout*)

By blocking, this means that the function will not return until the transmit is complete.

Parameters

lpuartState	Pointer to the LPUART driver state structure
sendBuffer	source buffer containing 8-bit data chars to send
txByteCount	the number of bytes to send
timeout	timeout value for RTOS abstraction sync control

Returns

An error code or kStatus_Success

13.2.2.3 status_t lpuart_send_data_async (lpuart_state_t * *lpuartState*, uint8_t * *sendBuffer*, uint32_t *txByteCount*)

This allows an async method for transmitting data and when coupled with a non-blocking receive, the LPUART can perform a full duplex operation. By non-blocking, this means that the function will return immediately. The application will have to get the transmit status to see when the transmit is complete.

Parameters

lpuartState	Pointer to the LPUART driver state structure
sendBuffer	source buffer containing 8-bit data chars to send
txByteCount	the number of bytes to send

Returns

An error code or kStatus_Success

13.2.2.4 status_t lpuart_get_transmit_status (lpuart_state_t * *lpuartState*, uint32_t * *bytesTransmitted*)

Parameters

lpuartState	Pointer to the LPUART driver state structure
bytes-Transmitted	Pointer to value that will be filled in with the number of bytes that have been sent in the active transfer

Return values

kStatus_Success	The transmit has completed successfully.
kStatus_LPUART_TxBusy	The transmit is still in progress. *bytesTransmitted* will be filled with the number of bytes that have been transmitted so far.

13.2.2.5 status_t lpuart_get_receive_status (lpuart_state_t * *lpuartState,* uint32_t * *bytesReceived*)

Parameters

lpuartState	Pointer to the LPUART driver state structure
bytesReceived	Pointer to value that will be filled in with the number of bytes that have been received in the active transfer

Return values

kStatus_Success	The receive has completed successfully.
kStatus_LPUART_Rx-Busy	The receive is still in progress. *bytesReceived* will be filled with the number of bytes that have been received so far.

13.2.2.6 void lpuart_shutdown (lpuart_state_t * *lpuartState*)

Parameters

lpuartState	Pointer to the LPUART driver state structure

13.2.2.7 status_t lpuart_receive_data (lpuart_state_t * *lpuartState,* uint8_t * *rxBuffer,* uint32_t *requestedByteCount,* uint32_t *timeout*)

By blocking, this means that the function will not return until the receive is complete.

Parameters

lpuartState	Pointer to the LPUART driver state structure
rxBuffer	buffer containing 8-bit read data chars received
requestedByte-Count	the number of bytes to receive
timeout	timeout value for RTOS abstraction sync control

Returns

An error code or kStatus_Success

13.2.2.8 status_t lpuart_receive_data_async (lpuart_state_t * *lpuartState,* uint8_t * *rxBuffer,* uint32_t *requestedByteCount*)

This allows an async method for receiving data and when coupled with a non-blocking transmit, the L-PUART can perform a full duplex operation. By non-blocking, this means that the function will return immediately. The application will have to get the receive status to see when the receive is complete.

Parameters

lpuartState	Pointer to the LPUART driver state structure
rxBuffer	buffer containing 8-bit read data chars received
requestedByte-Count	the number of bytes to receive

Returns

An error code or kStatus_Success

13.2.2.9 status_t lpuart_abort_sending_data (lpuart_state_t * *lpuartState*)

Parameters

lpuartState	Pointer to the LPUART driver state structure

Return values

#kStatus_Success	The transmit was successful.

13.2.2.10 status_t lpuart_abort_receiving_data (lpuart_state_t * *lpuartState*)

Parameters

lpuartState	Pointer to the LPUART driver state structure

Return values

#kStatus_Success	The receive was successful.

13.3 LPUART Types Definitions

The chapter describes the LPUART type definitions.

Data Structures

- struct lpuart_user_config_t
  LPUART configuration structure for user. More...

Enumerations

- enum _lpuart_errors {
  kStatus_LPUART_BaudRateCalculationError = 1,
  kStatus_LPUART_BaudRatePercentDiffExceeded,
  kStatus_LPUART_BitCountNotSupported,
  kStatus_LPUART_StopBitCountNotSupported,
  kStatus_LPUART_RxStandbyModeError,
  kStatus_LPUART_ClearStatusFlagError,
  kStatus_LPUART_MSBFirstNotSupported,
  kStatus_LPUART_Resync_NotSupported,
  kStatus_LPUART_TxNotDisabled,
  kStatus_LPUART_RxNotDisabled,
  kStatus_LPUART_TxOrRxNotDisabled,
  kStatus_LPUART_TxBusy,
  kStatus_LPUART_RxBusy,
  kStatus_LPUART_NoTransmitInProgress,
  kStatus_LPUART_NoReceiveInProgress,
  kStatus_LPUART_InvalidInstanceNumber,
  kStatus_LPUART_InvalidBitSetting,
  kStatus_LPUART_OverSamplingNotSupported,
  kStatus_LPUART_BothEdgeNotSupported,
  kStatus_LPUART_Timeout }
  Error codes for the LPUART driver.
- enum lpuart_stop_bit_count_t {
  kLpuartOneStopBit = 0,
  kLpuartTwoStopBit = 1 }
  LPUART number of stop bits.
- enum lpuart_parity_mode_t {
  kLpuartParityDisabled = 0x0,
  kLpuartParityEven = 0x2,
  kLpuartParityOdd = 0x3 }
  LPUART parity mode.
- enum lpuart_bit_count_per_char_t {
  kLpuart8BitsPerChar = 0,
  kLpuart9BitsPerChar = 1,

kLpuart10BitsPerChar = 2 }
LPUART number of bits in a character.

13.3.1 Data Structure Documentation

13.3.1.1 struct lpuart_user_config_t

Data Fields

- uint32_t baudRate
  LPUART baud rate.
- lpuart_parity_mode_t parityMode
  parity mode, disabled (default), even, odd
- lpuart_stop_bit_count_t stopBitCount
  number of stop bits, 1 stop bit (default)
- lpuart_bit_count_per_char_t bitCountPerChar
  or 2 stop bits

13.3.1.1.0.28 Field Documentation

13.3.1.1.0.28.1 lpuart_bit_count_per_char_t lpuart_user_config_t::bitCountPerChar

number of bits, 8-bit (default) or 9-bit in

13.3.2 Enumeration Type Documentation

13.3.2.1 enum _lpuart_errors

Enumerator

kStatus_LPUART_BaudRateCalculationError LPUART Baud Rate calculation error out of range.

kStatus_LPUART_BaudRatePercentDiffExceeded LPUART Baud Rate exceeds percentage difference.

kStatus_LPUART_BitCountNotSupported LPUART bit count config not supported.
kStatus_LPUART_StopBitCountNotSupported LPUART stop bit count config not supported.
kStatus_LPUART_RxStandbyModeError LPUART unable to place receiver in standby mode.
kStatus_LPUART_ClearStatusFlagError LPUART clear status flag error.
kStatus_LPUART_MSBFirstNotSupported LPUART MSB first feature not supported.
kStatus_LPUART_Resync_NotSupported LPUART resync disable operation not supported.
kStatus_LPUART_TxNotDisabled LPUART Transmitter not disabled before enabling feature.
kStatus_LPUART_RxNotDisabled LPUART Receiver not disabled before enabling feature.
kStatus_LPUART_TxOrRxNotDisabled LPUART Transmitter or Receiver not disabled.
kStatus_LPUART_TxBusy LPUART transmit still in progress.
kStatus_LPUART_RxBusy LPUART receive still in progress.

kStatus_LPUART_NoTransmitInProgress LPUART no transmit in progress.
kStatus_LPUART_NoReceiveInProgress LPUART no receive in progress.
kStatus_LPUART_InvalidInstanceNumber Invalid LPUART instance number.
kStatus_LPUART_InvalidBitSetting Invalid setting for desired LPUART register bit field.
kStatus_LPUART_OverSamplingNotSupported LPUART oversampling not supported.
kStatus_LPUART_BothEdgeNotSupported LPUART both edge sampling not supported.
kStatus_LPUART_Timeout LPUART transfer timed out.

13.3.2.2 enum lpuart_stop_bit_count_t

Enumerator

kLpuartOneStopBit one stop bit
kLpuartTwoStopBit two stop bits

13.3.2.3 enum lpuart_parity_mode_t

Enumerator

kLpuartParityDisabled parity disabled
kLpuartParityEven parity enabled, type even, bit setting: PE|PT = 10
kLpuartParityOdd parity enabled, type odd, bit setting: PE|PT = 11

13.3.2.4 enum lpuart_bit_count_per_char_t

Enumerator

kLpuart8BitsPerChar 8-bit data characters
kLpuart9BitsPerChar 9-bit data characters
kLpuart10BitsPerChar 10-bit data characters

Chapter 14
Microseconds Timer (MSTIMER)

The Kinetis SDK provides both HAL and Peripheral drivers for the Microseconds Timer (MSTIMER) block of Kinetis devices, which is based on the PIT driver.

Functions

- void microseconds_init (void)
  Initialize timer facilities.
- void microseconds_shutdown (void)
  Shutdown the microsecond timer.
- uint64_t microseconds_get (void)
  Read back current absolute time in microseconds.
- void microseconds_delay (uint32_t us)
  Delay a specified time.

14.0.3 Microseconds Driver

Overview

Microseconds driver provides an easy way to delay specific time or to measure running time of one segment code. The timer will run endlessly before microseconds_shutdown being called. Since the timer is 64 bits, the counter will never reach to 0 in centuries.

NOTE: using microseconds driver will change pit settings, so avoid using this if pit is used for other purpose.

Initialization

To initialize the pit module, simply call microseconds_init(). This will chain pit timer 0 and timer 1 together as a 64 bit lifetime timer, and set period of both timer 0 and timer 1 to maximum value.

Microseconds Delay

Simply call microseconds_delay with passing in the delay time in unit of microseconds. It performs as busy waiting and could be affected by interrupts.

Example code to use microseconds delay:

```
// Initialize microseconds driver.
microseconds_init();

// Do something.

// Delay for 100 us
```

```
microseconds_delay(100);

// Do some other thing.
```

Microseconds Get

Microseconds_get function can be used to measure running time of one segment of code. Call microseconds_get at the beginning of target code segment, call microseconds_get at the end of segment. The running time in microseconds unit could be get by subtracting end time from start time.

Example code to use microseconds get:

```
// Get start time
startTime = microseconds_get();

// Code needed to measure.

// Get end time
endTime = microseconds_get();

// Calculate running time.
runningTime = startTime - endTime;
```

14.1 Function Documentation

14.1.1 void microseconds_init (void)

This initializes pit to lifetime timer by chained channel 0 and channel 1 together, and set both channels to maximum counting period. This function should be called before using microseconds driver.

14.1.2 void microseconds_shutdown (void)

Disable PIT module and gate control.

14.1.3 uint64_t microseconds_get (void)

Returns

absolute time in unit of microseconds from lifetime timer.

14.1.4 void microseconds_delay (uint32_t *us*)

This function does not return until the specified number of microseconds has elapsed.

Parameters

us	The requested length of time to delay in microseconds.

Chapter 15
Periodic Interrup Timer (PIT)

The Kinetis SDK provides both HAL and Peripheral drivers for the Periodic Interrupt Timer (PIT) block of Kinetis devices.

Modules

- PIT HAL driver
  The part describes the programming interface of the PIT HAL driver.
- PIT ISR Definitions
  The part describes the PIT interrupt definitions.
- PIT Peripheral Driver
  The part describes the programming interface of the PIT Peripheral driver.

15.1 PIT HAL driver

The chapter describes the programming interface of the PIT HAL driver.

Initialization

- static void pit_hal_enable (void)
  Enables the PIT module.
- static void pit_hal_disable (void)
  Disables the PIT module.
- static void pit_hal_configure_timer_run_in_debug (bool timerRun)
  Configures the timers to continue running or stop in debug mode.

Timer Start and Stop

- static void pit_hal_timer_start (uint32_t timer)
  Starts the timer counting.
- static void pit_hal_timer_stop (uint32_t timer)
  Stops the timer from counting.
- static bool pit_hal_is_timer_started (uint32_t timer)
  Checks to see whether the current timer is started or not.

Timer Period

- static void pit_hal_set_timer_period_count (uint32_t timer, uint32_t count)
  Sets the timer period in units of count.
- static uint32_t pit_hal_read_timer_period_count (uint32_t timer)
  Returns the current timer period in units of count.
- static uint32_t pit_hal_read_timer_count (uint32_t timer)
  Reads the current timer counting value.

Interrupt

- static void pit_hal_configure_interrupt (uint32_t timer, bool enable)
  Enables or disables the timer interrupt.
- static bool pit_hal_is_interrupt_enabled (uint32_t timer)
  Checks whether the timer interrupt is enabled or not.
- static void pit_hal_clear_interrupt_flag (uint32_t timer)
  Clears the timer interrupt flag.
- static bool pit_hal_is_timeout_occurred (uint32_t timer)
  Reads the current timer timeout flag.

15.1.0.1 PIT HAL Driver

Overview

PIT hal driver is a set of APIs to access and configure PIT hardware registers.

15.1.1 Function Documentation

15.1.1.1 static void pit_hal_enable (void) [inline], [static]

This function enables the PIT timer clock (Note: this function does not un-gate the system clock gating control). It should be called before any other timer related setup.

15.1.1.2 static void pit_hal_disable (void) [inline], [static]

This function disables all PIT timer clocks(Note: it does not affect the SIM clock gating control).

15.1.1.3 static void pit_hal_configure_timer_run_in_debug (bool *timerRun*) [inline], [static]

In debug mode, the timers may or may not be frozen, based on the configuration of this function. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (for example, the timer values), and continue the operation.

Parameters

timerRun	Timers run or stop in debug mode. • true: Timers continue to run in debug mode. • false: Timers stop in debug mode.

15.1.1.4 static void pit_hal_timer_start (uint32_t *timer*) [inline], [static]

After calling this function, timers load the start value as specified by the function pit_hal_set_timer_-period_count(uint32_t timer, uint32_t count), count down to 0, and load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the time-out interrupt flag.

Parameters

timer	Timer channel number

15.1.1.5 static void pit_hal_timer_stop (uint32_t *timer*) [inline], [static]

This function stops every timer from counting. Timers reload their periods respectively after they call the pit_hal_timer_start the next time.

Parameters

timer	Timer channel number

15.1.1.6 static bool pit_hal_is_timer_started (uint32_t *timer*) [inline], [static]

Parameters

timer	Timer channel number

Returns

Current timer running status -true: Current timer is running. -false: Current timer has stopped.

15.1.1.7 static void pit_hal_set_timer_period_count (uint32_t *timer,* uint32_t *count*) [inline], [static]

Timers begin counting from the value set by this function. The counter period of a running timer can be modified by first stopping the timer, setting a new load value, and starting the timer again. If timers are not restarted, the new value is loaded after the next trigger event.

Parameters

timer	Timer channel number
count	Timer period in units of count

15.1.1.8 static uint32_t pit_hal_read_timer_period_count (uint32_t *timer*) [inline], [static]

Parameters

timer	Timer channel number

Returns

Timer period in units of count

15.1.1.9 static uint32_t pit_hal_read_timer_count (uint32_t *timer*) [inline], [static]

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

Parameters

timer	Timer channel number

Returns

Current timer counting value

15.1.1.10 static void pit_hal_configure_interrupt (uint32_t *timer,* bool *enable*) [inline], [static]

If enabled, an interrupt happens when a timeout event occurs (Note: NVIC should be called to enable pit interrupt in system level).

Parameters

timer	Timer channel number
enable	Enable or disable interrupt. • true: Generate interrupt when timer counts to 0. • false: No interrupt is generated.

15.1.1.11 static bool pit_hal_is_interrupt_enabled (uint32_t *timer*) [inline], [static]

Parameters

timer	Timer channel number

Returns

Status of enabled or disabled interrupt
- true: Interrupt is enabled.
- false: Interrupt is disabled.

15.1.1.12 static void pit_hal_clear_interrupt_flag (uint32_t *timer*) [inline], [static]

This function clears the timer interrupt flag after a timeout event occurs.

Parameters

timer	Timer channel number

15.1.1.13 static bool pit_hal_is_timeout_occurred (uint32_t *timer*) [inline], [static]

Every time the timer counts to 0, this flag is set.

Parameters

timer	Timer channel number

Returns

Current status of the timeout flag
- true: Timeout has occurred.
- false: Timeout has not yet occurred.

15.2 PIT Peripheral Driver

The chapter describes the programming interface of the PIT Peripheral driver.

Data Structures

- struct pit_user_config_t
  PIT timer configuration structure. More...

Typedefs

- typedef void(* pit_isr_callback_t)(void)
  PIT ISR callback function typedef.

Initialize and Shutdown

- void pit_init_module (bool isRunInDebug)
  Initialize PIT module.
- void pit_init_channel (uint32_t timer, const pit_user_config_t *config)
  Initialize PIT channel.
- void pit_shutdown (void)
  Disable PIT module and gate control.

Timer Start and Stop

- void pit_timer_start (uint32_t timer)
  Start timer counting.
- void pit_timer_stop (uint32_t timer)
  Stop timer counting.

Timer Period

- void pit_set_timer_period_us (uint32_t timer, uint32_t us)
  Set timer period in microsecond units.
- uint32_t pit_read_timer_us (uint32_t timer)
  Read current timer value in microsecond units.

ISR Callback Function

- void pit_register_isr_callback_function (uint32_t timer, pit_isr_callback_t function)
  Register pit isr callback function.

15.2.0.14 PIT Peripheral Driver

Overview

The pit driver is used to configure pit timers. It provides an easy way to make necessary module initializations and configure timer periods.

Initialization

To initialize the pit module, call pit_init_module first. This function will enable the PIT module and clock automatically. The parameter passed in pit_init_module will configure timers run or stop in debug mode. To use one timer channel, pit_init_channel should be called to init that channel.

This is example code for initializing and configuring the driver:

```
// Define device configuration.
const pit_config_t pitInit = {
    isInterruptEnabled = false,// Disable timer interrupt.
    isTimerChained = false,    // Meaningless for timer 0.
    periodUs = 20U             // Set timer period to 20 us.
};

// Initialize PIT module. Timers will stop running in debug mode.
pit_init_module(stop);

// Initialize PIT timer 0.
pit_init_channel(0, &pitInit);
```

Timer Period

The pit driver provides four ways to set the timer period.

1. The pit_init_channel function sets the timer period in units of microseconds. It is only applicable when initializing the channel.
2. The void pit_set_timer_period_us(uint32_t timer, uint32_t us) function sets the timer period in microsecond units. It is applicable at any time.
3. The void pit_set_lifetime_timer_period_us(uint64_t us) function sets the lifetime timer period in microsecond units. It only supports specific chips. Check the reference manual before using this function.
4. The void pit_hal_set_timer_period_count(uint32_t timer, uint32_t count) function sets the timer period in units of count. To use this function, the fsl_pit_hal.h needs to be included.

To read the current timer counting value in microsecond units, call uint32_t pit_read_timer_us(uint32_t timer).

Timer Operation

After timer setting is finished, call void pit_timer_start(uint32_t timer) to start timer counting. Call void pit_timer_stop(uint32_t timer) to stop it at any time.

If you want to close the PIT module entirely, call void pit_shutdown(void). This will disable the PIT module and the clock gate.

15.2.1 Data Structure Documentation

15.2.1.1 struct pit_user_config_t

Define structure PitConfig and use pit_init_channel() to make necessary initializations. You may also use remaining functions for PIT configuration.

Note

the timer chain feature is not valid in all devices, please check fsl_pit_features.h for accurate setting. If it's not valid, the value set here will be bypassed inside function pit_init_channel().

Data Fields

- bool isInterruptEnabled
  Timer interrupt 0-disable/1-enable.
- bool isTimerChained
  Chained with previous timer, 0-not/1-chained.
- uint32_t periodUs
  Timer period in unit of microseconds.

15.2.2 Function Documentation

15.2.2.1 void pit_init_module (bool *isRunInDebug*)

This function must be called before calling all the other PIT driver functions. This function un-gates the PIT clock and enables the PIT module. The isRunInDebug passed into function will affect all timer channels.

Parameters

isRunInDebug	Timers run or stop in debug mode. • true: Timers continue to run in debug mode. • false: Timers stop in debug mode.

15.2.2.2 void pit_init_channel (uint32_t *timer*, const pit_user_config_t * *config*)

This function initialize PIT timers by channel. Pass in timer number and its config structure. Timers do not start counting by default after calling this function. Function pit_timer_start must be called to start timer counting. Call pit_set_timer_period_us to re-set the period.

Here is an example demonstrating how to define a PIT channel config structure:

```
pit_user_config_t pitTestInit = {
     .isInterruptEnabled = true,
     // Only takes effect when chain feature is available.
     // Otherwise, pass in arbitrary value(true/false).
     .isTimerChained = false,
     // In unit of microseconds.
     .periodUs = 1000,
};
```

Parameters

timer	Timer channel number.
config	PIT channel configuration structure.

15.2.2.3 void pit_shutdown (void)

This function disables all PIT interrupts and PIT clock. It then gates the PIT clock control. pit_init_module must be called if you want to use PIT again.

15.2.2.4 void pit_timer_start (uint32_t *timer*)

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

timer	Timer channel number.

15.2.2.5 void pit_timer_stop (uint32_t *timer*)

This function stops every timer counting. Timers reload their periods respectively after the next time they call pit_timer_start.

Parameters

timer	Timer channel number.

15.2.2.6 void pit_set_timer_period_us (uint32_t *timer,* uint32_t *us*)

The period range depends on the frequency of PIT source clock. If the required period is out of range, use the lifetime timer, if applicable.

Parameters

timer	Timer channel number.
us	Timer period in microseconds.

15.2.2.7 uint32_t pit_read_timer_us (uint32_t *timer*)

This function returns an absolute time stamp in microsecond units. One common use of this function is to measure the running time of a part of code. Call this function at both the beginning and end of code; the time difference between these two time stamps is the running time (Make sure the running time will not exceed the timer period). The time stamp returned is up-counting.

Parameters

timer	Timer channel number.

Returns

Current timer value in microseconds.

15.2.2.8 void pit_register_isr_callback_function (uint32_t *timer,* pit_isr_callback_t *function*)

System default ISR interfaces are already defined in fsl_pit_irq.c. Users can either edit these ISRs or use this function to register a callback function. The default ISR runs the callback function if there is one

installed.

Parameters

timer	Timer channel number.
function	Pointer to pit isr callback function.

15.3 PIT ISR Definitions

The chapter describes the PIT interrupt definitions.

Chapter 16
Real Time Clock (RTC)

The Kinetis SDK provides both HAL and Peripheral drivers for the Real Time Clock (RTC) block of Kinetis devices.

Modules

- RTC HAL driver
  The part describes the programming interface of the RTC HAL driver.
- RTC Peripheral Driver
  The part describes the programming interface of the RTC Peripheral Driver.

16.1 RTC HAL driver

The chapter describes the programming interface of the RTC HAL driver.

Functions

- void rtc_hal_init (rtc_hal_init_config_t *configs)
  Initializes the RTC module.
- static void rtc_hal_reset_reg_TSR (void)
  Resets the RTC Time Seconds Register (RTC_TSR).
- static void rtc_hal_reset_reg_TPR (void)
  Resets the RTC Time Prescaler Register (RTC_TPR).
- static void rtc_hal_reset_reg_TAR (void)
  Resets the RTC Time Alarm Register (RTC_TAR).
- static void rtc_hal_reset_reg_TCR (void)
  Resets the RTC Time Compensation Register (RTC_TCR).
- static void rtc_hal_reset_reg_CR (void)
  Resets the RTC Control Register (RTC_CR).
- static void rtc_hal_reset_reg_SR (void)
  Resets the RTC Status Register (RTC_SR).
- static void rtc_hal_reset_reg_LR (void)
  Resets the RTC Lock Register (RTC_LR).
- static void rtc_hal_reset_reg_IER (void)
  Resets the RTC Interrupt Enable Register (RTC_IER).
- static void rtc_hal_get_seconds (uint32_t *seconds)
  Reads the value of the time seconds counter.
- static bool rtc_hal_set_seconds (const uint32_t *seconds)
  Writes to the time seconds counter.
- static void rtc_hal_get_prescaler (uint16_t *prescale)
  Reads the value of the time prescaler.
- static bool rtc_hal_set_prescaler (const uint16_t *prescale)
  Sets the time prescaler.
- static void rtc_hal_get_alarm (uint32_t *seconds)
  Reads the value of the time alarm.
- static void rtc_hal_set_alarm (const uint32_t *seconds)
  Sets the time alarm, this clears the time alarm flag.
- static void rtc_hal_get_comp_intrvl_counter (uint8_t *counter)
  Reads the compensation interval counter value.
- static void rtc_hal_get_current_time_compensation (uint8_t *tcValue)
  Reads the current time compensation interval counter value.
- static void rtc_hal_get_compensation_interval (uint8_t *value)
  Reads the compensation interval.
- static void rtc_hal_set_compensation_interval (const uint8_t *value)
  Writes the compensation interval.
- static void rtc_hal_get_time_compensation (uint8_t *value)
  Reads the time compensation value which is the configured number of the 32.768 kHz clock cycles in each second.
- static void rtc_hal_set_time_compensation (const uint8_t *enable)
  Writes to the RTC Time Compensation Register (RTC_TCR), field Time Compensation Register (TCR).
- static void rtc_hal_config_osc_2pf_load (bool enable)

Enables/disables oscillator configuration for 2pF load.
- static void rtc_hal_config_osc_4pf_load (bool enable)
  Enables/disables oscillator configuration for 4pF load.
- static void rtc_hal_config_osc_8pf_load (bool enable)
  Enables/disables oscillator configuration for 8pF load.
- static void rtc_hal_config_osc_16pf_load (bool enable)
  Enables/disables oscillator configuration for 16pF load.
- static void rtc_hal_config_clock_out (bool enable)
  Enables/disables the 32kHz clock output to other peripherals.
- static void rtc_hal_config_oscillator (bool enable)
  Enables/disables the oscillator.
- static void rtc_hal_configure_update_mode (bool lock)
  Enables/disables the update mode.
- static void rtc_hal_configure_supervisor_access (bool enable_reg_write)
  Enables/disables the supervisor access, which configures non-supervisor mode write access to all RTC registers and non-supervisor mode read access to RTC tamper/monotonic registers.
- static void rtc_hal_software_reset (void)
  Performs a software reset on the RTC module.
- static void rtc_hal_software_reset_flag_clear (void)
  Clears the software reset flag.
- static bool rtc_hal_is_counter_enabled (void)
  Reads the time counter enabled/disabled status.
- static void rtc_hal_counter_enable (bool enable)
  Changes the time counter enabled/disabled status.
- static bool rtc_hal_is_alarm_occured (void)
  Checks if the configured time alarm occurred.
- static bool rtc_hal_is_counter_overflow (void)
  Checks whether a counter overflow happened.
- static bool rtc_hal_is_time_invalid (void)
  Checks whether the time is marked as invalid.
- static void rtc_hal_config_lock_registers (hw_rtc_lr_t bitfields)
  Configures the register lock to other module fields.
- static bool rtc_hal_get_lock_reg_lock (void)
  Obtains the lock status of the lock register.
- static void rtc_hal_set_lock_reg_lock (bool set_to)
  Changes the lock status of the lock register.
- static bool rtc_hal_get_status_reg_lock (void)
  Obtains the state of the status register lock.
- static void rtc_hal_set_status_reg_lock (bool set_to)
  Changes the state of the status register lock.
- static bool rtc_hal_get_control_reg_lock (void)
  Obtains the state of the control register lock.
- static void rtc_hal_set_control_reg_lock (bool set_to)
  Changes the state of the control register lock.
- static bool rtc_hal_get_time_comp_lock (void)
  Obtains the state of the time compensation lock.
- static void rtc_hal_set_time_comp_lock (bool set_to)
  Changes the state of the time compensation lock.
- static void rtc_hal_config_interrupts (hw_rtc_ier_t *bitfields)
  Enables/disables RTC interrupts.
- static bool rtc_hal_read_seconds_int_enable (void)

Checks whether the Time Seconds Interrupt is enabled/disabled.
- static void rtc_hal_config_seconds_int (bool enable)
  Enables/disables the Time Seconds Interrupt.
- static bool rtc_hal_read_alarm_int_enable (void)
  Checks whether the Time Alarm Interrupt is enabled/disabled.
- static void rtc_hal_config_alarm_int_enable (bool enable)
  Enables/disables the Time Alarm Interrupt.
- static bool rtc_hal_read_time_overflow_int_enable (void)
  Checks whether the Time Overflow Interrupt is enabled/disabled .
- static void rtc_hal_config_time_overflow_int_enable (bool enable)
  Enables/disables the Time Overflow Interrupt.
- static bool rtc_hal_read_time_interval_int_enable (void)
  Checks whether the Time Invalid Interrupt is enabled/disabled.
- static void rtc_hal_config_time_interval_int (bool enable)
  Enables/disables the Time Invalid Interrupt.

16.1.0.9 RTC HAL Driver

Overview

The RTC HAL driver initializes the RTC registers and also provides functions to read or modify the RTC registers. These are mostly invoked by the RTC peripheral driver.

User configuration structures

The RTC HAL driver uses an instantiation of the rtc_hal_init_config_t structure, which is passed in by the user, to initialize the RTC registers.

This is an example initialization of this structure:

```
/* rtc hal init */
rtc_hal_init_config_t rtc_init_configs = {
    .enableOscillatorLoadConfg = 2,
    .disableClockOutToPeripheral = false,
    .enable32kOscillator = true,
    .enableWakeupPin = false,
    .startSecondsCounterAt = 0,
    .prescalerAt = 1,
    .alarmCounterAt = 0,
    .compensationInterval = 0,
    .timeCompensation = 0,
    .enableInterrupts = 0,
};
```

Enable or Disable Alarm Interrupts

The user calls the rtc_hal_config_alarm_int_enable() function to enable or disable the RTC alarm interrupt. The user calls the rtc_hal_read_alarm_int_enable() function to get the state of the RTC alarm interrupt bit.

This is an example:

```
rtc_datetime_t date;

/* alarm with no arguments */
if (rtc_hal_read_alarm_int_enable())
{
    rtc_get_alarm(&date);
    printf("Current alarm: %02d:%02d:%02d\r\n",
                   date.hour, date.minute, date.second);
    return;
}
else
{
    printf("No alarm armed\r\n");
}

rtc_hal_config_alarm_int_enable(true);
```

16.1.1 Function Documentation

16.1.1.1 void rtc_hal_init (rtc_hal_init_config_t * *configs*)

Parameters

configs	Pointer to a structure where the configuration details are stored at. The structure values that do NOT apply to the MCU in use are ignored.

16.1.1.2 static void rtc_hal_reset_reg_TSR (void) [inline], [static]

16.1.1.3 static void rtc_hal_reset_reg_TPR (void) [inline], [static]

16.1.1.4 static void rtc_hal_reset_reg_TAR (void) [inline], [static]

16.1.1.5 static void rtc_hal_reset_reg_TCR (void) [inline], [static]

16.1.1.6 static void rtc_hal_reset_reg_CR (void) [inline], [static]

16.1.1.7 static void rtc_hal_reset_reg_SR (void) [inline], [static]

16.1.1.8 static void rtc_hal_reset_reg_LR (void) [inline], [static]

16.1.1.9 static void rtc_hal_reset_reg_IER (void) [inline], [static]

16.1.1.10 static void rtc_hal_get_seconds (uint32_t * *seconds*) [inline], [static]

Parameters

Seconds	[out] pointer to variable where the seconds are stored.

16.1.1.11 static bool rtc_hal_set_seconds (const uint32_t * *seconds*) [inline], [static]

Parameters

seconds	[in] pointer to a variable from where to write the seconds.

Returns

true: write success since time counter is disabled. false: write error since time counter is enabled.

16.1.1.12 static void rtc_hal_get_prescaler (uint16_t * *prescale*) [inline], [static]

Parameters

prescale	[out] pointer to variable where the prescaler's value is stored.

16.1.1.13 static bool rtc_hal_set_prescaler (const uint16_t * *prescale*) [inline], [static]

Parameters

prescale	[in] pointer to variable from where to write the seconds.

Returns

true: set successfull; false: error, unable to set prescaler since the the time counter is enabled.

16.1.1.14 static void rtc_hal_get_alarm (uint32_t * *seconds*) [inline], [static]

Parameters

seconds	[out] pointer to a variable where the alarm value in seconds will be stored.

16.1.1.15 static void rtc_hal_set_alarm (const uint32_t * *seconds*) [inline], [static]

Parameters

seconds	[in] pointer to variable from where to write alarm value in seconds.

16.1.1.16 static void rtc_hal_get_comp_intrvl_counter (uint8_t * *counter*) [inline], [static]

Parameters

counter	[out] pointer to variable where the value is stored.

16.1.1.17 static void rtc_hal_get_current_time_compensation (uint8_t * *tcValue*) [inline], [static]

Parameters

tcValue	[out] pointer to variable where the value is stored.

16.1.1.18 static void rtc_hal_get_compensation_interval (uint8_t * *value*) [inline], [static]

The value is the configured compensation interval in seconds from 1 to 256 to control how frequently the time compensation register should adjust the number of 32.768 kHz cycles in each second. The value is one less than the number of seconds (for example. Zero means a configuration for a compensation interval of one second).

Parameters

value	[out] pointer to variable where the value is stored.

16.1.1.19 static void rtc_hal_set_compensation_interval (const uint8_t * *value*) [inline], [static]

This configures the compensation interval in seconds from 1 to 256 to control how frequently the TCR should adjust the number of 32.768 kHz cycles in each second. The value written should be one less than the number of seconds (for example, write zero to configure for a compensation interval of one second). This register is double buffered and writes do not take affect until the end of the current compensation interval.

Parameters

value	[in] pointer to a variable from where to write the value.

16.1.1.20 static void rtc_hal_get_time_compensation (uint8_t * *value*) [inline], [static]

Parameters

value	[out] pointer to variable where the value is stored.

16.1.1.21 static void rtc_hal_set_time_compensation (const uint8_t * *enable*) [inline], [static]

Configuring the number of 32.768 kHz clock cycles in each second. This register is double buffered and writes do not take affect until the end of the current compensation interval.

80h Time prescaler register overflows every 32896 clock cycles.

... ...

FFh Time prescaler register overflows every 32769 clock cycles.

00h Time prescaler register overflows every 32768 clock cycles.

01h Time prescaler register overflows every 32767 clock cycles.

... ...

7Fh Time prescaler register overflows every 32641 clock cycles.

Parameters

enable	[in] pointer to variable from where to write the value.

16.1.1.22 static void rtc_hal_config_osc_2pf_load (bool *enable*) [inline], [static]

Parameters

enable	true: enables load; false: disables load.

16.1.1.23 static void rtc_hal_config_osc_4pf_load (bool *enable*) [inline], [static]

Parameters

enable	true: enables load; false: disables load.

16.1.1.24 static void rtc_hal_config_osc_8pf_load (bool *enable*) [inline], [static]

Parameters

enable	true: enables load; false: disables load.

16.1.1.25 static void rtc_hal_config_osc_16pf_load (bool *enable*) [inline], [static]

Parameters

enable	true: enables load; false: disables load.

16.1.1.26 static void rtc_hal_config_clock_out (bool *enable*) [inline], [static]

Parameters

enable	true: enables clock out; false: disables clock out.

16.1.1.27 static void rtc_hal_config_oscillator (bool *enable*) [inline], [static]

After enablement, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

Parameters

enable	true: enables oscillator; false: disables oscillator.

16.1.1.28 static void rtc_hal_configure_update_mode (bool *lock*) [inline], [static]

This mode allows the time counter enabled to be written even when the status register is locked. When set, the time counter enable, can always be written if the TIF (Time Invalid Flag) or TOF (Time Overflow Flag) are set or if the time counter enable is clear. For devices with the monotonic counter, it allows the monotonic enable to be written when it is locked. When set, the monotonic enable can always be written if the TIF (Time Invalid Flag) or TOF (Time Overflow Flag) are set or if the monotonic counter enable is clear. For devices with tamper detect, it allows the it to be written when it is locked. When set, the tamper detect can always be written if the TIF (Time Invalid Flag) is clear. Note: Tamper and Monotonic features are not available in all MCUs.

Parameters

lock	true: enables register lock, registers cannot be written when locked; False: disables register lock, registers can be written when locked under limited conditions.

16.1.1.29 static void rtc_hal_configure_supervisor_access (bool *enable_reg_write*) [inline], [static]

```
Note: Tamper and Monotonic features are NOT available in all MCUs.
```

Parameters

enable_reg_write	true: enables register lock, Non-supervisor mode write accesses are supported; false: disables register lock, non-supervisor mode write accesses are not supported and generate a bus error.

16.1.1.30 static void rtc_hal_software_reset (void) [inline], [static]

This resets all RTC registers except for the SWR bit and the RTC_WAR and RTC_RAR registers. The SWR bit is cleared after VBAT POR and by software explicitly clearing it. Note: access control features (RTC_WAR and RTC_RAR registers) are not available in all MCUs.

16.1.1.31 static void rtc_hal_software_reset_flag_clear (void) [inline], [static]

16.1.1.32 static bool rtc_hal_is_counter_enabled (void) [inline], [static]

Returns

true: time counter is enabled, time seconds register and time prescaler register are not writeable, but increment; false: time counter is disabled, time seconds register and time prescaler register are writeable, but do not increment.

16.1.1.33 static void rtc_hal_counter_enable (bool *enable*) [inline], [static]

Parameters

enable	true: enables the time counter; false: disables the time counter.

16.1.1.34 static bool rtc_hal_is_alarm_occured (void) [inline], [static]

Returns

true: time alarm has occurred. false: NO time alarm occurred.

16.1.1.35 static bool rtc_hal_is_counter_overflow (void) [inline], [static]

Returns

true: time overflow occurred and time counter is zero. false: NO time overflow occurred.

16.1.1.36 static bool rtc_hal_is_time_invalid (void) [inline], [static]

Returns

true: time is INVALID and time counter is zero. false: time is valid.

Reads the value of the RTC Status Register (RTC_SR), field Time Invalid Flag (TIF). This flag is set on the VBAT POR or the software reset. The TSR and TPR do not increment and read as zero when this bit is set. This flag is cleared by writing the TSR register when the time counter is disabled.

16.1.1.37 static void rtc_hal_config_lock_registers (hw_rtc_lr_t *bitfields*) [inline], [static]

Parameters

bitfields	[in] configuration flags: Valid bitfields: LRL: Lock Register Lock SRL: Status Register Lock CRL: Control Register Lock TCL: Time Compensation Lock For MCUs that have the Tamper Detect only: TIL: Tamper Interrupt Lock TTL: Tamper Trim Lock TDL: Tamper Detect Lock TEL: Tamper Enable Lock TTSL: Tamper Time Seconds Lock For MCUs that have the Monotonic Counter only: MCHL: Monotonic Counter High Lock MCLL: Monotonic Counter Low Lock MEL: Monotonic Enable Lock

16.1.1.38 static bool rtc_hal_get_lock_reg_lock (void) [inline], [static]

Returns

true: lock register is not locked and writes complete normally. false: lock register is locked and writes are ignored.

16.1.1.39 static void rtc_hal_set_lock_reg_lock (bool *set_to*) [inline], [static]

Once cleared, this can only be set by the VBAT POR or the software reset.

Parameters

set_to	true: Lock register is not locked and writes complete normally. false: Lock register is locked and writes are ignored.

16.1.1.40 static bool rtc_hal_get_status_reg_lock (void) [inline], [static]

Returns

true: Status register is not locked and writes complete normally. false: Status register is locked and writes are ignored.

16.1.1.41 static void rtc_hal_set_status_reg_lock (bool *set_to*) [inline], [static]

Once cleared, this can only be set by the VBAT POR or the software reset.

Parameters

set_to	true: Status register is not locked and writes complete normally. false: Status register is locked and writes are ignored.

16.1.1.42 static bool rtc_hal_get_control_reg_lock (void) [inline], [static]

Returns

true: Control register is not locked and writes complete normally. false: Control register is locked and writes are ignored.

16.1.1.43 static void rtc_hal_set_control_reg_lock (bool *set_to*) [inline], [static]

Once cleared, this can only be set by the VBAT POR or the software reset.

Parameters

set_to	true: Control register is not locked and writes complete normally. false: Control register is locked and writes are ignored.

16.1.1.44 static bool rtc_hal_get_time_comp_lock (void) [inline], [static]

Returns

true: Time compensation register is not locked and writes complete normally. false: Time compensation register is locked and writes are ignored.

16.1.1.45 static void rtc_hal_set_time_comp_lock (bool *set_to*) [inline], [static]

Once cleared, this can only be set by the VBAT POR or the software reset.

Parameters

set_to	true: Time compensation register is not locked and writes complete normally. false: Time compensation register is locked and writes are ignored.

16.1.1.46 static void rtc_hal_config_interrupts (hw_rtc_ier_t * *bitfields*) [inline], [static]

Parameters

bitfields	[in] set/clear respective bitfields to enabled/disabled interrupts. [out] resulting interrupt enable state. Valid bitfields: TSIE: Time Seconds Interrupt Enable TAIE: Time Alarm Interrupt Enable TOIE: Time Overflow Interrupt Enable TIIE: Time Invalid Interrupt Enable For MCUs that have the Wakeup Pin only: WPON: Wakeup Pin On (see the corresponding MCU's reference manual) For MCUs that have the Monotonic Counter only: MOIE: Monotonic Overflow Interrupt Enable

16.1.1.47 static bool rtc_hal_read_seconds_int_enable (void) [inline], [static]

Returns

true: Seconds interrupt is enabled. false: Seconds interrupt is disabled.

16.1.1.48 static void rtc_hal_config_seconds_int (bool *enable*) [inline], [static]

```
Note: The seconds interrupt is an edge-sensitive interrupt with a
dedicated interrupt vector. It is generated once a second and
```

```
requires no software overhead (there is no corresponding status
flag to clear).
```

Parameters

enable	true: Seconds interrupt is enabled. false: Seconds interrupt is disabled.

16.1.1.49 static bool rtc_hal_read_alarm_int_enable (void) [inline], [static]

Returns

true: Time alarm flag does generate an interrupt. false: Time alarm flag does not generate an interrupt.

16.1.1.50 static void rtc_hal_config_alarm_int_enable (bool *enable*) [inline], [static]

Parameters

enable	true: Time alarm flag does generate an interrupt. false: Time alarm flag does not generate an interrupt.

16.1.1.51 static bool rtc_hal_read_time_overflow_int_enable (void) [inline], [static]

Returns

true: Time overflow flag does generate an interrupt. false: Time overflow flag does not generate an interrupt.

16.1.1.52 static void rtc_hal_config_time_overflow_int_enable (bool *enable*) [inline], [static]

Parameters

enable	true: Time overflow flag does generate an interrupt. false: Time overflow flag does not generate an interrupt.

16.1.1.53 static bool rtc_hal_read_time_interval_int_enable (void) [inline], [static]

Returns

true: Time invalid flag does generate an interrupt. false: Time invalid flag does not generate an interrupt.

16.1.1.54 static void rtc_hal_config_time_interval_int (bool *enable*) [inline], [static]

Parameters

enable	true: Time invalid flag does generate an interrupt. false: Time invalid flag does not generate an interrupt.

16.2 RTC Peripheral Driver

The chapter describes the programming interface of the RTC Peripheral Driver.

Data Structures

- struct rtc_datetime_t
  Structure is used to hold the time in a simple "date" format. More...
- struct rtc_time_t
  Time representation: hours, minutes, second and total seconds. More...
- struct rtc_user_config_t
  RTC timer configuration structure. More...

Typedefs

- typedef void(∗ rtc_isr_callback_t)(void)
  RTC ISR callback function typedef.

Initialize and Shutdown

- bool rtc_init (const rtc_user_config_t ∗config)
  Initializes the Real Time Clock module.
- void rtc_shutdown (void)
  disable RTC module clock gate control.

RTC interrupt configuration and status

- void rtc_configure_int (hw_rtc_ier_t ∗bitfields)
  Enables/disables RTC interrupts.
- void rtc_get_int_status (hw_rtc_sr_t ∗int_status_flags)
  Returns the RTC interrupt status flags.

RTC datetime set and get

- bool rtc_set_datetime (const rtc_datetime_t ∗datetime, bool start_after_set)
  Sets the RTC date and time according to the given time structure, if indicated in the corresponding parameter.
- void rtc_get_datetime (rtc_datetime_t ∗datetime)
  Gets the actual RTC time and stores it in the given time structure.

RTC alarm set and get

- bool rtc_set_alarm (const rtc_datetime_t ∗date)

Sets the RTC alarm.

- bool rtc_get_alarm (rtc_datetime_t *date)
  Returns the RTC alarm time.

RTC time counter start and stop

- void rtc_start_time_counter (void)
  Enables the RTC oscillator and starts time counter.
- void rtc_stop_time_counter (void)
  Halts running time counter.

Copy time data

- void rtc_cp_datetime_time (const rtc_datetime_t *datetime, rtc_time_t *time)
  Utility to copy time data from datetime structure to a time structure.
- void rtc_cp_time_datetime (const rtc_time_t *time, rtc_datetime_t *datetime)
  Utility to copy time data from time structure to a datetime.

ISR Callback Function

- void rtc_register_isr_callback_function (uint8_t rtc_irq_number, rtc_isr_callback_t function)
  Register RTC ISR callback function.

16.2.0.55 RTC Peripheral Driver

Overview

The RTC peripheral driver is used to set and get the RTC clock, set and read the RTC alarm time and receive notifications when an alarm is triggered.

User configuration structures

The RTC driver uses an instantiation of the rtc_user_config_t structure to configure the RTC driver. This allows the user to configure the most common settings of the RTC driver. The rtc_user_config_t structure holds an instance of the rtc_hal_init_config_t (see RTC HAL Driver for details) and rtc_datetime_t data-types. The rtc_hal_init_config_t holds information used by the RTC HAL driver & rtc_datetime_t, which allows the user to pass in the date and time information stored in the RTC seconds register.

Initialization

To initialize, the user calls rtc_init() with a pointer to the rtc_user_config_t structure. The driver initialization function ungates the RTC module clock. It then uses the information passed in to initialize the RTC HAL layer driver (this part is skipped if the "general_config" element is set to NULL) and the RTC time (this part is skipped if the "start_at_datetime" element is set to NULL).

This is an example code to set up a user RTC configuration instantiation:

```
rtc_user_config_t rtc_drv_configs = {
    .general_config = &rtc_init_configs,
    .start_at_datetime = NULL,
};
```

This is an example to make the rtc_init() function.

```
/* init the rtc module */
rtc_init(&rtc_drv_configs);
```

Setting and reading the RTC time

The RTC driver uses an instantiation of the rtc_datetime_t structure to configure or read the date & time.

This is an example code to set up a user RTC configuration instantiation:

```
rtc_datetime_t datetimeToSet;

datetimeToSet.year = 2013;
datetimeToSet.month = 10;
datetimeToSet.day = 13;
datetimeToSet.hour = 18;
datetimeToSet.minute = 55;
datetimeToSet.second = 30;
```

The user can pass the date and time as part of the rtc_user_config_t structure during driver initialization or can call the rtc_set_datetime() or rtc_get_datetime() functions to configure or read the date & time at a later time. This is example to use these functions.

```
/* get datetime  */
rtc_get_datetime(&datetime);
printf("Current datetime: %04hd-%02hd-%02hd %02hd:%02hd:%02hd\r\n",
          datetime.year, datetime.month, datetime.day,
          datetime.hour, datetime.minute, datetime.second);

/* set the datetime */
result = rtc_set_datetime(&datetime, true);
```

Triggering an Alarm

The RTC driver uses interrupts when an alarm is triggered at a user-specified time. The user has to register a callback function that is invoked when the alarm interrupt is triggered.

The user can call the rtc_set_alarm() to set the alarm time or the rtc_get_alarm() to read the configured alarm time. The user has to set the current time using the steps mentioned earlier prior to using the call to set the alarm time. This is an example of how to use these functions.

```
rtc_datetime_t datetimeToSet;

rtc_set_datetime(&datetimeToSet, false);
datetimeToSet.second += 5;
rtc_set_alarm(&datetimeToSet);

/*Register a callback function.*/
rtc_register_isr_callback_function(0, alarmISR);

/* Enable the Alarm interrupt */
rtc_hal_config_alarm_int_enable(true);
```

16.2.1 Data Structure Documentation

16.2.1.1 struct rtc_datetime_t

Warning

If you violate the ranges, undefined behavior results.

Data Fields

- uint16_t year
  Range from 200 to 2099.
- uint16_t month
  Range from 1 to 12.
- uint16_t day
  Range from 1 to 31 (depending on month).
- uint16_t hour
  Range from 0 to 23.
- uint16_t minute
  Range from 0 to 59.
- uint8_t second
  Range from 0 to 59.

16.2.1.1.0.29 Field Documentation

16.2.1.1.0.29.1 uint16_t rtc_datetime_t::year

16.2.1.1.0.29.2 uint16_t rtc_datetime_t::month

16.2.1.1.0.29.3 uint16_t rtc_datetime_t::day

16.2.1.1.0.29.4 uint16_t rtc_datetime_t::hour

16.2.1.1.0.29.5 uint16_t rtc_datetime_t::minute

16.2.1.1.0.29.6 uint8_t rtc_datetime_t::second

16.2.1.2 struct rtc_time_t

Data Fields

- uint16_t hour
  Range from 0 to 23.
- uint16_t minute
  Range from 0 to 59.
- uint8_t second
  Range from 0 to 59.

16.2.1.2.0.30 Field Documentation

16.2.1.2.0.30.1 uint16_t rtc_time_t::hour

16.2.1.2.0.30.2 uint16_t rtc_time_t::minute

16.2.1.2.0.30.3 uint8_t rtc_time_t::second

16.2.1.3 struct rtc_user_config_t

Data Fields

- rtc_hal_init_config_t * general_config
  Pointer to a structure most of the configurations.
- rtc_datetime_t * start_at_datetime
  are found.
- bool start_counter
  Set to true to start the real time counter.

16.2.1.3.0.31 Field Documentation

16.2.1.3.0.31.1 rtc_datetime_t* rtc_user_config_t::start_at_datetime

Set to NULL to skip related configuration.

See the 'rtc_hal_init_config' definition for details. Initial datetime. Set to NULL to skip.

16.2.1.3.0.31.2 bool rtc_user_config_t::start_counter

Will

16.2.2 Function Documentation

16.2.2.1 bool rtc_init (const rtc_user_config_t * *config*)

Parameters

Config	[in] initialization configuration details.

Returns

True: success; false: datetime format is invalid or the counter indicated to start but it was already started.

16.2.2.2 void rtc_shutdown (void)

16.2.2.3 void rtc_configure_int (hw_rtc_ier_t * *bitfields*)

Parameters

bitfields	[in] set/clear respective bitfields to enabled/disabled interrupts. [out] resulting interrupt enable state. Valid bitfields: TSIE: Time Seconds Interrupt Enable TAIE: Time Alarm Interrupt Enable TOIE: Time Overflow Interrupt Enable TIIE: Time Invalid Interrupt Enable For MCUs that have the Wakeup Pin only: WPON: Wakeup Pin On (see the corresponding MCU's reference manual) For MCUs that have the Monotonic Counter only: MOIE: Monotonic Overflow Interrupt Enable

16.2.2.4 void rtc_get_int_status (hw_rtc_sr_t * *int_status_flags*)

Parameters

int_status_flags	[out] pointer to where to store bitfield with an actual RTC interrupt flags. ONLY valid bitfields: TIF: Time Invalid Flag TOF: Time Overflow Flag TAF: Time Alarm Flag MOF: Monotonic Overflow Flag (Not in all devices) Note: other bitfields are not to be taken in account.

16.2.2.5 bool rtc_set_datetime (const rtc_datetime_t * *datetime*, bool *start_after_set*)

After, the time counter is started.

Parameters

datetime	[in] pointer to a structure where the date and time details are stored.
start_after_set	true: the RTC oscillator will be enabled and the counter will start. False: otherwise.

Returns

True: success; false: error, datetime format incorrect, datetime format is invalid or unable to set the counter value because it is already enabled.

16.2.2.6 void rtc_get_datetime (rtc_datetime_t * *datetime*)

Parameters

datetime	[out] pointer to structure where the date and time details will be stored.

16.2.2.7 bool rtc_set_alarm (const rtc_datetime_t * *date*)

Parameters

date	[in] pointer to structure where the alarm date and time details will be stored.

Returns

true: success; false: error.

16.2.2.8 bool rtc_get_alarm (rtc_datetime_t * *date*)

Parameters

date	[out] pointer to structure where the alarm date and time details will be stored.

Returns

True: success; false: error.

16.2.2.9 void rtc_start_time_counter (void)

16.2.2.10 void rtc_stop_time_counter (void)

16.2.2.11 void rtc_cp_datetime_time (const rtc_datetime_t * *datetime*, rtc_time_t * *time*)

Parameters

datetime	[in] data origin
time	[out] data destination

16.2.2.12 void rtc_cp_time_datetime (const rtc_time_t * *time*, rtc_datetime_t * *datetime*)

```
structure.
```

Parameters

time	[in] data origin
datetime	[out] data destination

16.2.2.13 void rtc_register_isr_callback_function (uint8_t *rtc_irq_number*, rtc_isr_callback_t *function*)

System default ISR interfaces are already defined in the fsl_rtc_irq.c. Users can either edit these ISRs or use this function to register a callback function. The default ISR runs the callback function if one is installed.

Parameters

rtc_irq_number	RTC interrupt number.
function	Pointer to the RTC ISR callback function.

Chapter 17
Synchronous Audio Interface (SAI)

The Kinetis SDK provides both HAL and Peripheral drivers for the Synchronous Audio Interface (SAI) block of Kinetis devices.

Modules

- SAI HAL driver
  The part describes the programming interface of the SAI HAL driver.
- SAI peripheral driver
  The part describes the programming interface of the SAI Peripheral driver.

17.1 SAI HAL driver

The chapter describes the programming interface of the SAI HAL driver.

Enumerations

- enum sai_bus_t {
  kSaiBusI2SLeft = 0x0,
  kSaiBusI2SRight = 0x1,
  kSaiBusI2SType = 0x2 }
  Defines the SAI bus type.
- enum sai_io_mode_t {
  kSaiIOModeTransmit = 0x0,
  kSaiIOModeReceive = 0x1,
  kSaiIOModeDuplex = 0x2 }
  Transmits or receives data; Reads and writes at the same time.
- enum sai_master_slave_t {
  kSaiMaster = 0x0,
  kSaiSlave = 0x1 }
  Master or slave mode.
- enum sai_sync_mode_t {
  kSaiModeAsync = 0x0,
  kSaiModeSync = 0x1,
  kSaiModeSyncWithOtherTx = 0x2,
  kSaiModeSyncWithOtherRx = 0x3 }
  Synchronous or asynchronous mode.
- enum sai_mclk_source_t {
  kSaiMclkSourceSysclk = 0x0,
  kSaiMclkSourceExtal = 0x1,
  kSaiMclkSourceAltclk = 0x2,
  kSaiMclkSourcePllout = 0x3 }
  Mater clock source.
- enum sai_bclk_source_t {
  kSaiBclkSourceBusclk = 0x0,
  kSaiBclkSourceMclkDiv = 0x1,
  kSaiBclkSourceOtherSai0 = 0x2,
  kSaiBclkSourceOtherSai1 = 0x3 }
  Bit clock source.
- enum sai_interrupt_request_t {
  kSaiIntrequestWordStart = 0x0,
  kSaiIntrequestSyncError = 0x1,
  kSaiIntrequestFIFOWarning = 0x2,
  kSaiIntrequestFIFOError = 0x3,
  kSaiIntrequestFIFORequest = 0x4 }
  The SAI state flag.
- enum sai_dma_request_t {
  kSaiDmaReqFIFOWarning = 0x0,

kSaiDmaReqFIFORequest = 0x1 }
The DMA request sources.
- enum sai_state_flag_t {
kSaiStateFlagWordStart = 0x0,
kSaiStateFlagSyncError = 0x1,
kSaiStateFlagFIFOError = 0x2,
kSaiStateFlagSoftReset = 0x3 }
The SAI state flag.
- enum sai_reset_type_t {
kSaiResetTypeSoftware = 0x0,
kSaiResetTypeFIFO = 0x1 }
The reset type.
- enum sai_mode_t {
kSaiRunModeDebug = 0x0,
kSaiRunModeStop = 0x1 }

Functions

- void sai_hal_init (uint8_t instance)
Initializes the SAI device.
- void sai_hal_set_tx_bus (uint8_t instance, sai_bus_t bus_mode)
Sets the bus protocol relevant settings for Tx.
- void sai_hal_set_rx_bus (uint8_t instance, sai_bus_t bus_mode)
Sets the bus protocol relevant settings for Rx.
- static void sai_hal_set_mclk_source (uint8_t instance, sai_mclk_source_t source)
Sets the master clock source.
- void sai_hal_set_mclk_divider (uint8_t instance, uint32_t mclk, uint32_t src_clk)
Sets the divider of the master clock.
- static void sai_hal_set_tx_bclk_source (uint8_t instance, sai_bclk_source_t source)
Sets the bit clock source of Tx.
- static void sai_hal_set_rx_bclk_source (uint8_t instance, sai_bclk_source_t source)
Sets the bit clock source of Rx.
- static void sai_hal_set_tx_blck_divider (uint8_t instance, uint32_t divider)
Sets the bit clock divider value of Tx.
- static void sai_hal_set_rx_blck_divider (uint8_t instance, uint32_t divider)
Sets the bit clock divider value of Tx.
- static void sai_hal_set_tx_frame_size (uint8_t instance, uint8_t size)
Sets the frame size for Tx.
- static void sai_hal_set_rx_frame_size (uint8_t instance, uint8_t size)
Set the frame size for rx.
- static void sai_hal_set_tx_word_size (uint8_t instance, uint8_t bits)
Set the word size for tx.
- static void sai_hal_set_rx_word_size (uint8_t instance, uint8_t bits)
Sets the word size for Rx.
- static void sai_hal_set_tx_word_zero_size (uint8_t instance, uint8_t size)
Sets the size of the first word of the frame for Tx.
- static void sai_hal_set_rx_word_zero_size (uint8_t instance, uint8_t size)
Sets the size of the first word of the frame for Rx.
- static void sai_hal_set_tx_sync_width (uint8_t instance, uint8_t width)

Sets the sync width for Tx.
- static void sai_hal_set_rx_sync_width (uint8_t instance, uint8_t width)
  Sets the sync width for Rx.
- static void sai_hal_set_tx_watermark (uint8_t instance, uint8_t watermark)
  Sets the watermark value for Tx FIFO.
- static void sai_hal_set_rx_watermark (uint8_t instance, uint8_t watermark)
  Sets the watermark value for Rx FIFO.
- void sai_hal_set_tx_master_slave (uint8_t instance, sai_master_slave_t master_slave_mode)
  Sets the master or slave mode of Tx.
- void sai_hal_set_rx_master_slave (uint8_t instance, sai_master_slave_t master_slave_mode)
  Sets the Rx master or slave mode.
- void sai_hal_set_tx_sync_mode (uint8_t instance, sai_sync_mode_t sync_mode)
  Transmits the mode setting.
- void sai_hal_set_rx_sync_mode (uint8_t instance, sai_sync_mode_t sync_mode)
  Receives the mode setting.
- uint8_t sai_hal_get_fifo_read_pointer (uint8_t instance, sai_io_mode_t io_mode, uint8_t fifo_-channel)
  Gets the FIFO read pointer.
- uint8_t sai_hal_get_fifo_write_pointer (uint8_t instance, sai_io_mode_t io_mode, uint8_t fifo_-channel)
  Gets the FIFO read pointer.
- uint32_t * sai_hal_get_fifo_address (uint8_t instance, sai_io_mode_t io_mode, uint8_t fifo_-channel)
  Gets the TDR/RDR register address.
- static void sai_hal_enable_tx (uint8_t instance)
  Enables the Tx transmit.
- static void sai_hal_enable_rx (uint8_t instance)
  Enables the Rx receive.
- static void sai_hal_disable_tx (uint8_t instance)
  Disables the Tx transmit.
- static void sai_hal_disable_rx (uint8_t instance)
  Disables the Rx receive.
- void sai_hal_enable_tx_interrupt (uint8_t instance, sai_interrupt_request_t source)
  Enables the Tx interrupt from different interrupt sources.
- void sai_hal_enable_rx_interrupt (uint8_t instance, sai_interrupt_request_t source)
  Enables the Rx interrupt from different sources.
- void sai_hal_disable_tx_interrupt (uint8_t instance, sai_interrupt_request_t source)
  Disables the Tx interrupts from different interrupt sources.
- void sai_hal_disable_rx_interrupt (uint8_t instance, sai_interrupt_request_t source)
  Disables Rx interrupts from different interrupt sources.
- void sai_hal_enable_tx_dma (uint8_t instance, sai_dma_request_t request)
  Enables the Tx DMA request from different sources.
- void sai_hal_enable_rx_dma (uint8_t instance, sai_dma_request_t request)
  Enables the Rx DMA request from different sources.
- void sai_hal_disable_tx_dma (uint8_t instance, sai_dma_request_t request)
  Disables the Tx DMA request from different sources.
- void sai_hal_disable_rx_dma (uint8_t instance, sai_dma_request_t request)
  Disables the Rx DMA request from different sources.
- void sai_hal_clear_tx_state_flag (uint8_t instance, sai_state_flag_t flag)
  Clears the Tx state flags.
- void sai_hal_clear_rx_state_flag (uint8_t instance, sai_state_flag_t flag)

Clears the state flags for Rx.
- void sai_hal_reset_tx (uint8_t instance, sai_reset_type_t mode)
  Resets the Tx.
- void sai_hal_reset_rx (uint8_t instance, sai_reset_type_t mode)
  Resets the Rx.
- static void sai_hal_set_tx_word_mask (uint8_t instance, uint32_t mask)
  Sets the mask word of the frame in Tx.
- static void sai_hal_set_rx_word_mask (uint8_t instance, uint32_t mask)
  Sets the mask word of the frame in Rx.
- static void sai_hal_set_tx_fifo_channel (uint8_t instance, uint8_t fifo_channel)
  Sets the FIFO Tx channel.
- static void sai_hal_set_rx_fifo_channel (uint8_t instance, uint8_t fifo_channel)
  Sets the Rx FIFO channel.
- void sai_hal_set_tx_mode (uint8_t instance, sai_mode_t mode)
  Sets the running mode.
- void sai_hal_set_rx_mode (uint8_t instance, sai_mode_t mode)
  Sets the Rx running mode.
- void sai_hal_set_tx_bclk_swap (uint8_t instance, bool ifswap)
  Set Tx bit clock swap.
- void sai_hal_set_rx_bclk_swap (uint8_t instance, bool ifswap)
  Sets the Rx bit clock swap.
- static void sai_hal_set_tx_word_start_index (uint8_t instance, uint8_t index)
  Configures on which word the start of the word flag is set.
- static void sai_hal_set_rx_word_start_index (uint8_t instance, uint8_t index)
  Configures on which word the start of the word flag is set.
- static void sai_hal_set_tx_fbt (uint8_t instance, uint8_t index)
  Sets the index in FIFO for the first bit data .
- static void sai_hal_set_rx_fbt (uint8_t instance, uint8_t index)
  Sets the index in FIFO for the first bit data.
- bool sai_hal_mclk_divider_is_update (uint8_t instance)
  Flags whether the master clock divider is re-divided.
- bool sai_hal_word_start_is_detected (uint8_t instance, sai_io_mode_t io_mode)
  Word start is detected.
- bool sai_hal_sync_error_is_detected (uint8_t instance, sai_io_mode_t io_mode)
  Sync error is detected.
- bool sai_hal_fifo_warning_is_detected (uint8_t instance, sai_io_mode_t io_mode)
  FIFO warning is detected.
- bool sai_hal_fifo_error_is_detected (uint8_t instance, sai_io_mode_t io_mode)
  FIFO error is detected.
- bool sai_hal_fifo_request_is_detected (uint8_t instance, sai_io_mode_t io_mode)
  FIFO request is detected.
- static void sai_hal_receive_data (uint8_t instance, uint8_t rx_channel, uint32_t *data)
  Receives the data from FIFO.
- static void sai_hal_transmit_data (uint8_t instance, uint8_t tx_channel, uint32_t data)
  Transmits data to the FIFO.

17.1.0.14 SAI HAL Driver

Overview

The SAI HAL driver is used to mask the hardware and provide user a comprehensible way to use sai.

17.1.1 Enumeration Type Documentation

17.1.1.1 enum sai_bus_t

Enumerator

kSaiBusI2SLeft Use I2S left aligned format.
kSaiBusI2SRight Use I2S right aligned format.
kSaiBusI2SType Use I2S format.

17.1.1.2 enum sai_io_mode_t

Enumerator

kSaiIOModeTransmit Write data to FIFO.
kSaiIOModeReceive Read data from FIFO.
kSaiIOModeDuplex Read data and write data at the same time.

17.1.1.3 enum sai_master_slave_t

Enumerator

kSaiMaster Master mode.
kSaiSlave Slave mode.

17.1.1.4 enum sai_sync_mode_t

Enumerator

kSaiModeAsync Asynchronous mode.
kSaiModeSync Synchronous mode (with receiver or transmit)
kSaiModeSyncWithOtherTx Synchronous with another SAI transmit.
kSaiModeSyncWithOtherRx Synchronous with another SAI receiver.

17.1.1.5 enum sai_mclk_source_t

Enumerator

kSaiMclkSourceSysclk Master clock from the system clock.
kSaiMclkSourceExtal Master clock from the extal.
kSaiMclkSourceAltclk Master clock from the ALT.
kSaiMclkSourcePllout Master clock from the PLL.

17.1.1.6 enum sai_bclk_source_t

Enumerator

kSaiBclkSourceBusclk Bit clock using bus clock.
kSaiBclkSourceMclkDiv Bit clock using master clock divider.
kSaiBclkSourceOtherSai0 Bit clock from other SAI device.
kSaiBclkSourceOtherSai1 Bit clock from other SAI device.

17.1.1.7 enum sai_interrupt_request_t

Enumerator

kSaiIntrequestWordStart Word start flag, means the first word in a frame detected.
kSaiIntrequestSyncError Sync error flag, means the sync error is detected.
kSaiIntrequestFIFOWarning FIFO warning flag, means the FIFO is empty.
kSaiIntrequestFIFOError FIFO error flag.
kSaiIntrequestFIFORequest FIFO request, means reached watermark.

17.1.1.8 enum sai_dma_request_t

Enumerator

kSaiDmaReqFIFOWarning FIFO warning caused by the DMA request.
kSaiDmaReqFIFORequest FIFO request caused by the DMA request.

17.1.1.9 enum sai_state_flag_t

Enumerator

kSaiStateFlagWordStart Word start flag, means the first word in a frame detected.
kSaiStateFlagSyncError Sync error flag, means the sync error is detected.
kSaiStateFlagFIFOError FIFO error flag.
kSaiStateFlagSoftReset Software reset flag.

17.1.1.10 enum sai_reset_type_t

Enumerator

kSaiResetTypeSoftware Software reset, reset the logic state.
kSaiResetTypeFIFO FIFO reset, reset the FIFO read and write pointer.

17.1.1.11 enum sai_mode_t

Enumerator

kSaiRunModeDebug In debug mode.
kSaiRunModeStop In stop mode.

17.1.2 Function Documentation

17.1.2.1 void sai_hal_init (uint8_t *instance*)

The initialization resets the SAI module by setting the SR bit of TCSR and the RCSR register. Note that the function would write 0 to every control registers.

Parameters

instance	The SAI peripheral instance number.

17.1.2.2 void sai_hal_set_tx_bus (uint8_t *instance,* sai_bus_t *bus_mode*)

The bus mode means which protocol SAI uses. It can be I2S left, right, and so on. Each protocol would have different configuration on bit clock and frame sync.

Parameters

instance	The SAI peripheral instance number.
bus_mode	The protocol selection, it can be I2S left aligned, I2S right aligned, etc.

17.1.2.3 void sai_hal_set_rx_bus (uint8_t *instance,* sai_bus_t *bus_mode*)

The bus mode means which protocol SAI uses. It can be I2S left, right and so on. Each protocol has a different configuration on bit clock and frame sync.

Parameters

instance	The SAI peripheral instance number.
bus_mode	The protocol selection, it can be I2S left aligned, I2S right aligned, etc.

17.1.2.4 static void sai_hal_set_mclk_source (uint8_t *instance,* sai_mclk_source_t *source*) [inline], [static]

The source of the clock can be: PLL_OUT, ALT_CLK, EXTAL, SYS_CLK. This function sets the clock source for SAI master clock source. Master clock is used to produce the bit clock for the data transfer.

Parameters

instance	The SAI peripheral instance number.
source	Mater clock source

17.1.2.5 void sai_hal_set_mclk_divider (uint8_t *instance,* uint32_t *mclk,* uint32_t *src_clk*)

Using the divider to get the master clock frequency wanted from the source. mclk = clk_source $*$ fract/divide. The input is the master clock frequency needed and the source clock frequency. The master clock is decided by the sample rate and the multi-clock number.

Parameters

instance	The SAI peripheral instance number.
mclk	Master clock frequency needed.
src_clk	The source clock frequency.

17.1.2.6 static void sai_hal_set_tx_bclk_source (uint8_t *instance,* sai_bclk_source_t *source*) [inline], [static]

It is generated by the master clock, bus clock, and other devices.

The function sets the source of the bit clock. The bit clock can be produced by the master clock, and from the bus clock or other SAI Tx/Rx. Tx and Rx in the SAI module can use the same bit clock either from Tx or Rx.

Parameters

instance	The SAI peripheral instance number.
source	Bit clock source.

17.1.2.7 static void sai_hal_set_rx_bclk_source (uint8_t *instance,* sai_bclk_source_t *source*) [inline], [static]

It is generated by the master clock, bus clock, and other devices.

The function sets the source of the Rx bit clock. The bit clock can be produced by the master clock and from the bus clock or other SAI Tx/Rx. Tx and Rx in the SAI module use the same bit clock either from Tx or Rx.

Parameters

instance	The SAI peripheral instance number.
source	Bit clock source.

17.1.2.8 static void sai_hal_set_tx_blck_divider (uint8_t *instance,* uint32_t *divider*) [inline], [static]

bclk = mclk / divider. At the same time, bclk = sample_rate ∗ channel ∗ bits. This means how much time is needed to transfer one bit. Notice: The function is called while the bit clock source is the master clock.

Parameters

instance	The SAI peripheral instance number.
div	The divide number of bit clock.

17.1.2.9 static void sai_hal_set_rx_blck_divider (uint8_t *instance,* uint32_t *divider*) [inline], [static]

bclk = mclk / divider. At the same time, bclk = sample_rate ∗ channel ∗ bits. This means how much time is needed to transfer one bit. Notice: The function is called while the bit clock source is the master clock.

Parameters

instance	The SAI peripheral instance number.
div	The divide number of bit clock.

17.1.2.10 static void sai_hal_set_tx_frame_size (uint8_t *instance,* uint8_t *size*) [inline], [static]

The frame size means how many words are in a frame. For example 2-channel audio data, the frame size is 2. This means there are 2 words in a frame.

Parameters

instance	The SAI peripheral instance number.
size	Words number in a frame.

17.1.2.11 static void sai_hal_set_rx_frame_size (uint8_t *instance,* uint8_t *size*) [inline], [static]

The frame size means how many words in a frame. In the usual case, for example 2-channel audio data, the frame size is 2, means 2 words in a frame.

Parameters

instance	The sai peripheral instance number.
size	Words number in a frame.

17.1.2.12 static void sai_hal_set_tx_word_size (uint8_t *instance,* uint8_t *bits*) [inline], [static]

The word size means the quantization level of audio file. Generally, there are 8bit, 16bit, 24bit, 32bit format which sai would all support.

Parameters

instance	The sai peripheral instance number.
bits	How many bits in a word.

17.1.2.13 static void sai_hal_set_rx_word_size (uint8_t *instance,* uint8_t *bits*) [inline], [static]

The word size means the quantization level of the audio file. Generally, SAI supports 8 bit, 16 bit, 24 bit, and 32 bit formats.

Parameters

instance	The SAI peripheral instance number.
bits	How many bits in a word.

17.1.2.14 static void sai_hal_set_tx_word_zero_size (uint8_t *instance,* uint8_t *size*) [inline], [static]

In I2S protocol, the size of the first word is the same as the size of other words. In some protocols, for example, AC'97, the size of the first word is not the same as other sizes. This function sets the length of the first word which is, in most situations, the same as others.

Parameters

instance	The SAI peripheral instance number.
size	The length of frame head word.

17.1.2.15 static void sai_hal_set_rx_word_zero_size (uint8_t *instance,* uint8_t *size*) [inline], [static]

In I2S protocol, the size of the first word is the same as the size of other words. In some protocols, for example, AC'97, the first word is not the same size as others. This function sets the length of the first word, which is, in most situations, the same as others.

Parameters

instance	The SAI peripheral instance number.
size	The length of frame head word.

17.1.2.16 static void sai_hal_set_tx_sync_width (uint8_t *instance,* uint8_t *width*) [inline], [static]

A sync is the number of bit clocks of a frame. The sync width cannot be longer than the length of the first word of the frame.

Parameters

instance	The SAI peripheral instance number.
width	How many bit clock in a sync.

17.1.2.17 static void sai_hal_set_rx_sync_width (uint8_t *instance,* uint8_t *width*) [inline], [static]

A sync is the number of bit clocks of a frame. The sync width cannot be longer than the length of the first word of the frame.

Parameters

instance	The SAI peripheral instance number.
width	How many bit clock in a sync.

17.1.2.18 static void sai_hal_set_tx_watermark (uint8_t *instance,* uint8_t *watermark*) [inline], [static]

While the value in the Tx FIFO is less or equal to the watermark , it generates an interrupt request or a DMA request. The watermark value cannot be greater than the depth of FIFO.

Parameters

instance	The SAI peripheral instance number.
watermark	Watermark value of a FIFO.

17.1.2.19 static void sai_hal_set_rx_watermark (uint8_t *instance,* uint8_t *watermark*) [inline], [static]

While the value in Rx FIFO is larger or equal to the watermark , it generates an interrupt request or a DMA request. The watermark value cannot be greater than the depth of FIFO.

Parameters

instance	The SAI peripheral instance number.
watermark	Watermark value of a FIFO.

17.1.2.20 void sai_hal_set_tx_master_slave (uint8_t *instance,* sai_master_slave_t *master_slave_mode*)

The function sets the Tx mode to either master or slave. The master mode provides its own clock and slave mode uses the external clock.

Parameters

instance	The SAI peripheral instance number.
master_slave_mode	Mater or slave mode.

17.1.2.21 void sai_hal_set_rx_master_slave (uint8_t *instance,* sai_master_slave_t *master_slave_mode*)

The function sets the Rx mode to either master or slave. Master mode provides its own clock and slave mode uses the external clock.

Parameters

instance	The SAI peripheral instance number.
master_slave_mode	Mater or slave mode.

17.1.2.22 void sai_hal_set_tx_sync_mode (uint8_t *instance,* sai_sync_mode_t *sync_mode*)

The mode can be asynchronous mode, synchronous, or synchronous with another SAI device. When configured for a synchronous mode of operation, the receiver must be configured for the asynchronous operation.

Parameters

instance	The SAI peripheral instance number.
sync_mode	Synchronous mode or Asynchronous mode.

17.1.2.23 void sai_hal_set_rx_sync_mode (uint8_t *instance,* sai_sync_mode_t *sync_mode*)

The mode can be asynchronous mode, synchronous, synchronous with another SAI device. When configured for a synchronous mode of operation, the receiver must be configured for the asynchronous operation.

Parameters

instance	The SAI peripheral instance number.
sync_mode	Synchronous mode or Asynchronous mode.

17.1.2.24 uint8_t sai_hal_get_fifo_read_pointer (uint8_t *instance,* sai_io_mode_t *io_mode,* uint8_t *fifo_channel*)

It is used to judge whether the FIFO is full or empty and know how much space there is for FIFO. If read_ptr == write_ptr, the FIFO is empty. While the bit of the read_ptr and the write_ptr are equal except for the MSB, the FIFO is full.

Parameters

instance	The SAI peripheral instance number.
io_mode	Transmit or receive data.
fifo_channel	FIFO channel selected.

Returns

FIFO read pointer value.

17.1.2.25 uint8_t sai_hal_get_fifo_write_pointer (uint8_t *instance,* sai_io_mode_t *io_mode,* uint8_t *fifo_channel*)

It is used to judge whether the FIFO is full or empty and know how much space there is for FIFO. If the read_ptr == write_ptr, the FIFO is empty. While the bit of the read_ptr and write_ptr are equal except for the MSB, the FIFO is full.

Parameters

instance	The SAI peripheral instance number.
io_mode	Transmit or receive data.
fifo_channel	FIFO channel selected.

Returns

FIFO write pointer value

17.1.2.26 uint32_t* sai_hal_get_fifo_address (uint8_t *instance,* sai_io_mode_t *io_mode,* uint8_t *fifo_channel*)

This function is for DMA transfer because it needs to know the dest/src address of the DMA transfer.

Parameters

instance	The SAI peripheral instance number.
io_mode	Transmit or receive data.
fifo_channel	FIFO channel selected.

Returns

TDR register or RDR register address

17.1.2.27 static void sai_hal_enable_tx (uint8_t *instance*) [inline], [static]

Enables the transmitter. This function enables both the bit clock and the transfer channel.

Parameters

instance	The SAI peripheral instance number.

17.1.2.28 static void sai_hal_enable_rx (uint8_t *instance*) [inline], [static]

Enables the receiver. This function enables both the bit clock and the receive channel.

Parameters

instance	The SAI peripheral instance number.

17.1.2.29 static void sai_hal_disable_tx (uint8_t *instance*) [inline], [static]

Disables the transmitter. This function disables both the bit clock and the transfer channel. When software clears this field, the transmitter remains enabled, and this bit remains set, until the end of the current frame.

Parameters

instance	The SAI peripheral instance number.

17.1.2.30 static void sai_hal_disable_rx (uint8_t *instance*) [inline], [static]

Disables the receiver. This function disables both the bit clock and the transfer channel. When software clears this field, the receiver remains enabled, and this bit remains set, until the end of the current frame.

Parameters

instance	The SAI peripheral instance number.

17.1.2.31 void sai_hal_enable_tx_interrupt (uint8_t *instance,* sai_interrupt_request_t *source*)

The interrupt source can be : Word start flag, Sync error flag, FIFO error flag, FIFO warning flag, FIFO request flag. This function sets which flag causes an interrupt request.

Parameters

instance	The SAI peripheral instance number.
source	SAI interrupt request source.

17.1.2.32 void sai_hal_enable_rx_interrupt (uint8_t *instance,* sai_interrupt_request_t *source*)

The interrupt source can be : Word start flag, Sync error flag, FIFO error flag, FIFO warning flag, FIFO request flag. This function sets which flag causes an interrupt request.

Parameters

instance	The SAI peripheral instance number.
source	SAI interrupt request source.

17.1.2.33 void sai_hal_disable_tx_interrupt (uint8_t *instance,* sai_interrupt_request_t *source*)

This function disables the interrupt requests from the interrupt request source of SAI.

Parameters

instance	The SAI peripheral instance number.
source	SAI interrupt request source.

17.1.2.34 void sai_hal_disable_rx_interrupt (uint8_t *instance,* sai_interrupt_request_t *source*)

This function disables the interrupt requests from interrupt request source of SAI.

Parameters

instance	The SAI peripheral instance number.
source	SAI interrupt request source.

17.1.2.35 void sai_hal_enable_tx_dma (uint8_t *instance,* sai_dma_request_t *request*)

The DMA sources can be FIFO warning and FIFO request. This function enables the DMA request from different DMA request sources.

Parameters

instance	The SAI peripheral instance number.
source	SAI DMA request source.

17.1.2.36 void sai_hal_enable_rx_dma (uint8_t *instance,* sai_dma_request_t *request*)

The DMA sources can be: FIFO warning and FIFO request. This function enables the DMA request from different DMA request sources.

Parameters

instance	The SAI peripheral instance number.
source	SAI DMA request source.

17.1.2.37 void sai_hal_disable_tx_dma (uint8_t *instance,* sai_dma_request_t *request*)

The function disables the DMA request of Tx in SAI. DMA request can from FIFO warning or FIFO request which means FIFO is empty or reach the watermark.

Parameters

instance	The SAI peripheral instance number.
source	SAI DMA request source.

17.1.2.38 void sai_hal_disable_rx_dma (uint8_t *instance,* sai_dma_request_t *request*)

The function disables the DMA request of Tx in SAI. DMA request can from FIFO warning or FIFO request which means FIFO is empty or reach the watermark.

Parameters

instance	The SAI peripheral instance number.
source	SAI DMA request source.

17.1.2.39 void sai_hal_clear_tx_state_flag (uint8_t *instance,* sai_state_flag_t *flag*)

The function is used to clear the flags manually. It can clear word start, FIFO warning, FIFO error, and FIFO request flag.

Parameters

instance	The SAI peripheral instance number.
flag	SAI state flag type. The flag can be word start, sync error, FIFO error/warning.

17.1.2.40 void sai_hal_clear_rx_state_flag (uint8_t *instance,* sai_state_flag_t *flag*)

The function clears the flags manually. It can clear word start, FIFO warning, FIFO error, and FIFO request flag.

Parameters

instance	The SAI peripheral instance number.
flag	SAI state flag type. The flag can be word start, sync error, FIFO error/warning.

17.1.2.41 void sai_hal_reset_tx (uint8_t *instance,* sai_reset_type_t *mode*)

There are two kinds of reset: Software reset and FIFO reset. Software reset: resets all transmitter internal logic, including the bit clock generation, status flags and FIFO pointers. It does not reset the configuration registers. FIFO reset: synchronizes the FIFO write pointer to the same value as the FIFO read pointer. This empties the FIFO contents and is to be used after the Transmit FIFO Error Flag is set, and before the FIFO is re-initialized and the Error Flag is cleared.

Parameters

instance	The SAI peripheral instance number.

mode	SAI reset type.

17.1.2.42 void sai_hal_reset_rx (uint8_t *instance,* sai_reset_type_t *mode*)

Parameters

instance	The SAI peripheral instance number.
mode	SAI reset type.

17.1.2.43 static void sai_hal_set_tx_word_mask (uint8_t *instance,* uint32_t *mask*) [inline], [static]

Each bit number represent the mask word index. For example, 0 represents mask the 0th word, 3 represents mask 0th and 1st word. The TMR register can be different from frame to frame. If the user wants a mono audio, set the mask to 0/1.

Parameters

instance	The SAI peripheral instance number.
mask	Which bits need to be masked in a frame.

17.1.2.44 static void sai_hal_set_rx_word_mask (uint8_t *instance,* uint32_t *mask*) [inline], [static]

Parameters

instance	The SAI peripheral instance number.
mask	Which bits need to be masked in a frame.

17.1.2.45 static void sai_hal_set_tx_fifo_channel (uint8_t *instance,* uint8_t *fifo_channel*) [inline], [static]

A SAI instance includes a Tx and a Rx. Each has several channels according to different platforms. A channel means a path for the audio data input/output.

Parameters

instance	The SAI peripheral instance number.
fifo_channel	FIFO channel number.

17.1.2.46 static void sai_hal_set_rx_fifo_channel (uint8_t *instance,* uint8_t *fifo_channel*) [inline], [static]

Parameters

instance	The SAI peripheral instance number.
fifo_channel	FIFO channel number.

17.1.2.47 void sai_hal_set_tx_mode (uint8_t *instance,* sai_mode_t *mode*)

There is a debug mode, stop mode and a normal mode.

This function can set the working mode of the SAI instance. Stop mode is always used in low power cases, and the debug mode disables the SAI after the current transmit/receive is completed.

Parameters

instance	The SAI peripheral instance number.
mode	SAI running mode.

17.1.2.48 void sai_hal_set_rx_mode (uint8_t *instance,* sai_mode_t *mode*)

Parameters

instance	The SAI peripheral instance number.
mode	SAI running mode.

17.1.2.49 void sai_hal_set_tx_bclk_swap (uint8_t *instance,* bool *ifswap*)

While set in asynchronous mode, the transmitter is clocked by the receiver bit clock. When set in synchronous mode, the transmitter is clocked by the transmitter bit clock, but uses the receiver frame sync. This bit has no effect when synchronous with another SAI peripheral.

Parameters

instance	The SAI peripheral instance number.
ifswap	If swap bit clock.

17.1.2.50 void sai_hal_set_rx_bclk_swap (uint8_t *instance,* bool *ifswap*)

When set in asynchronous mode, the receiver is clocked by the transmitter bit clock. When set in synchronous mode, the receiver is clocked by the receiver bit clock, but uses the transmitter frame sync. This bit has no effect when synchronous with another SAI peripheral.

Parameters

instance	The SAI peripheral instance number.
ifswap	If swap bit clock.

17.1.2.51 static void sai_hal_set_tx_word_start_index (uint8_t *instance,* uint8_t *index*) [inline], [static]

Parameters

instance	The SAI peripheral instance number.
index	Which word triggers the word start flag.

17.1.2.52 static void sai_hal_set_rx_word_start_index (uint8_t *instance,* uint8_t *index*) [inline], [static]

Parameters

instance	The SAI peripheral instance number.
index	Which word would trigger word start flag.

17.1.2.53 static void sai_hal_set_tx_fbt (uint8_t *instance,* uint8_t *index*) [inline], [static]

The FIFO is 32-bit in SAI, but not all audio data is 32-bit. Mostly they are 16-bit. In this situation, the Codec needs to know which bit of the FIFO marks the valid audio data.

Parameters

instance	The SAI peripheral instance number.
index	First bit shifted in FIFO.

17.1.2.54 static void sai_hal_set_rx_fbt (uint8_t *instance,* uint8_t *index*) [inline], [static]

Parameters

instance	The SAI peripheral instance number.
index	First bit shifted in FIFO.

17.1.2.55 bool sai_hal_mclk_divider_is_update (uint8_t *instance*)

Parameters

instance	The SAI peripheral instance number.

Returns

True if the divider updated otherwise false.

17.1.2.56 bool sai_hal_word_start_is_detected (uint8_t *instance,* sai_io_mode_t *io_mode*)

Parameters

instance	The SAI peripheral instance number.
io_mode	Transmit or receive data.

Returns

True if detect word start otherwise false.

17.1.2.57 bool sai_hal_sync_error_is_detected (uint8_t *instance,* sai_io_mode_t *io_mode*)

Parameters

instance	The SAI peripheral instance number.
io_mode	Transmit or receive data.

Returns

True if detect sync error otherwise false.

17.1.2.58 bool sai_hal_fifo_warning_is_detected (uint8_t *instance,* sai_io_mode_t *io_mode*)

FIFO warning means that the FIFO is empty in Tx. While in Tx, FIFO warning means that the FIFO is empty and it needs data.

Parameters

instance	The SAI peripheral instance number.
io_mode	Transmit or receive data.

Returns

True if detect FIFO warning otherwise false.

17.1.2.59 bool sai_hal_fifo_error_is_detected (uint8_t *instance,* sai_io_mode_t *io_mode*)

FIFO error means that the FIFO has no data and the Codec is still transferring data. While in Rx, FIFO error means that the data is still in but the FIFO is full.

Parameters

instance	The SAI peripheral instance number.
io_mode	Transmit or receive data.

Returns

True if detects FIFO error otherwise false.

17.1.2.60 bool sai_hal_fifo_request_is_detected (uint8_t *instance,* sai_io_mode_t *io_mode*)

FIFO request means that the data in FIFO is less than the watermark in Tx and more than the watermark in Rx.

Parameters

instance	The SAI peripheral instance number.
io_mode	Transmit or receive data.

Returns

True if detects FIFO request otherwise false.

17.1.2.61 static void sai_hal_receive_data (uint8_t *instance*, uint8_t *rx_channel*, uint32_t * *data*) [inline], [static]

Parameters

instance	The SAI peripheral instance number.
rx_channel	Rx FIFO channel.
data	Pointer to the address to be written in.

17.1.2.62 static void sai_hal_transmit_data (uint8_t *instance*, uint8_t *tx_channel*, uint32_t *data*) [inline], [static]

Parameters

instance	The SAI peripheral instance number.
tx_channel	Tx FIFO channel.
data	Data value which needs to be written into FIFO.

17.2 SAI peripheral driver

The chapter describes the programming interface of the SAI Peripheral driver.

Data Structures

- struct sai_user_config_t
  The description structure for the SAI TX/RX module. More...
- struct sai_data_format_t
  Defines the PCM data format. More...
- struct sai_handler_t
  The SAI handler structure. More...

Macros

- #define USEDMA 1
  Use DMA mode or interrupt mode.

Typedefs

- typedef void(∗ sai_callback_t)(void ∗parameter)
  SAI callback function.

Enumerations

- enum sai_status_t
  Status structure for SAI.

Functions

- sai_status_t sai_init (sai_handler_t ∗handler, sai_user_config_t ∗config)
  Initializes the SAI module.
- sai_status_t sai_deinit (sai_handler_t ∗handler)
  De-initializes the SAI module.
- sai_status_t sai_configure_data_format (sai_handler_t ∗handler, sai_data_format_t ∗format)
  Configures the PCM data format.
- void sai_start_read_data (sai_handler_t ∗handler)
  Starts reading data from FIFO.
- void sai_start_write_data (sai_handler_t ∗handler)
  Starts writing data to FIFO.
- static void sai_stop_read_data (sai_handler_t ∗handler)
  Stops reading data from FIFO, mainly to disable the DMA or the interrupt request bit.
- static void sai_stop_write_data (sai_handler_t ∗handler)
  Stop write data to FIFO, mainly to disable the DMA or the interrupt request bit.
- static void sai_clear_tx_status (sai_handler_t ∗handler)

Clears the TX FIFO error flag.
- static void sai_clear_rx_status (sai_handler_t *handler)
  Clears the RX FIFO error flag.
- static uint32_t * sai_get_fifo_address (sai_handler_t *handler)
  Gets the FIFO address of the data channel.
- uint32_t sai_send_data (sai_handler_t *handler, uint8_t *addr, uint32_t len)
  Sends a certain length data.
- uint32_t sai_receive_data (sai_handler_t *handler, uint8_t *addr, uint32_t len)
  Receives a certain length data.
- void sai_register_callback (sai_handler_t *handler, sai_callback_t callback, void *callback_param)
  Registers the callback function after a transfer.

17.2.0.63 SAI Driver

Overview

The SAI driver initializes, configures, starts, and stop SAI. The SAI driver implements functions of configure, send and receive data.

Initialization

To initialize SAI, call the sai_init() function and pass the parameters needed. The parameter is the sai_handler_t structure and SAI configuration structure. The function opens the clock gate and initializes the SAI modules according to the structure information.

Configure

Configure is implemented by the sai_configure_data_format() function. To use this function, the users should transfer the audio data format to the SAI module.

Call diagram

To use the SAI driver, follow these steps:

1. Initialize the SAI module by calling the sai_init() function.
2. Configure the audio data features of SAI by calling the sai_configure_data_foramt() function.
3. Send/receive data by calling the sai_send_data() or the sai_receive_data() functions.
4. Start TX or RX by calling the sai_start_write_data() or the sai_start_read_data() function.
5. Shut down the SAI module by calling the sai_deinit() function.

This is example code to initialize and configure the SAI driver in the DMA mode:

```
// Initialize handler structure.
sai_handler_t handler;
```

```
handler.direction = AUDIO_TX;
handler.instance = 0;
handler.fifo_channel = 0;

//Initialize config structure.
sai_config_t tx_config;
tx_config.bus_type = kSaiBusI2SLeft;
tx_config.channel = 0;
tx_config.slave_master = kSaiMaster;
tx_config.sync_mode = kSaiModeAsync;
tx_config.bclk_source = kSaiBclkSourceMclkDiv;
tx_config.mclk_source = kSaiMclkSourceSysclk;
tx_config.mclk_divide_enable = true;

//Data format of audio data
audio_data_format_t format;
format.bits = 16;
format.sample_rate = 44100;
format.mclk = 384 * format->sample_rate;
format.words = 2;
format.watermark = 4;

sai_init(&handler, &tx_config);
sai_configure_data_format(&handler,format);

//option: register callback functions for finished transfer
sai_register_callback(&handler, callback, NULL);
//start send data
sai_send_data(&handler, addr, len);
```

17.2.1 Data Structure Documentation

17.2.1.1 struct sai_user_config_t

Data Fields

- sai_mclk_source_t mclk_source
  Master clock source.
- bool mclk_divide_enable
  Enable the divide of master clock to generate bit clock.
- sai_sync_mode_t sync_mode
  Synchronous or asynchronous.
- sai_bus_t bus_type
  I2S left, I2S right or I2S type.
- sai_master_slave_t slave_master
  Master or slave.
- sai_bclk_source_t bclk_source
  Bit clock from master clock or other modules.
- uint8_t channel
  Which FIFO is used to transfer.

17.2.1.1.0.32 Field Documentation

17.2.1.1.0.32.1 sai_mclk_source_t sai_user_config_t::mclk_source

17.2.1.1.0.32.2 bool sai_user_config_t::mclk_divide_enable

17.2.1.1.0.32.3 sai_sync_mode_t sai_user_config_t::sync_mode

17.2.1.1.0.32.4 sai_bus_t sai_user_config_t::bus_type

17.2.1.1.0.32.5 sai_master_slave_t sai_user_config_t::slave_master

17.2.1.1.0.32.6 sai_bclk_source_t sai_user_config_t::bclk_source

17.2.1.1.0.32.7 uint8_t sai_user_config_t::channel

17.2.1.2 struct sai_data_format_t

Data Fields

- uint32_t sample_rate
  Sample rate of the PCM file.
- uint32_t mclk
  Master clock frequency.
- uint8_t bits
  How many bits in a word.
- uint8_t words
  How many word in a frame.
- uint8_t watermark
  When to send interrupt/DMA request.

17.2.1.3 struct sai_handler_t

The structure is used in the SAI runtime.

Data Fields

- uint8_t instance
  SAI instance.
- bool direction
  RX or TX.
- uint8_t fifo_channel
  Which data channel is used.

17.2.2 Macro Definition Documentation

17.2.2.1 #define USEDMA 1

17.2.3 Function Documentation

17.2.3.1 sai_status_t sai_init (sai_handler_t * *handler*, sai_user_config_t * *config*)

This function initializes the SAI registers according to the configuration structure. This function initializes the basic settings for SAI, including some board relevant settings. Notice: This function does not initialize an entire SAI instance. This function only initializes the TX or RX according to the value in the handler.

Parameters

handler	SAI handler structure pointer.
config	The configuration structure of SAI.

Returns

Return kStatus_SAI_Success while the initialize success and kStatus_SAI_Fail if failed.

17.2.3.2 sai_status_t sai_deinit (sai_handler_t * *handler*)

This function closes the SAI device. It does not close the entire SAI instance. It only closes the TX or RX according to the value in the handler.

Parameters

handler	SAI handler structure pointer of the SAI module.

Returns

Return kStatus_SAI_Success while the process success and kStatus_SAI_Fail if failed.

17.2.3.3 sai_status_t sai_configure_data_format (sai_handler_t * *handler*, sai_data_format_t * *format*)

The function mainly configures an audio sample rate, data bits and a channel number.

Parameters

handler	SAI handler structure pointer of the SAI module.
format	PCM data format structure pointer.

Returns

Return kStatus_SAI_Success while the process success and kStatus_SAI_Fail if failed.

17.2.3.4 void sai_start_read_data (sai_handler_t * *handler*)

The function enables the interrupt/DMA request source and enables the transmit channel.

Parameters

handler	SAI handler structure pointer of the SAI module.

17.2.3.5 void sai_start_write_data (sai_handler_t * *handler*)

The function enables the interrupt/DMA request source and enables the receive channel.

Parameters

handler	SAI handler structure pointer of the SAI module.

17.2.3.6 static void sai_stop_read_data (sai_handler_t * *handler*) [inline], [static]

This function provides the method to pause receiving data.

Parameters

handler	SAI handler structure pointer of the SAI module.

17.2.3.7 static void sai_stop_write_data (sai_handler_t * *handler*) [inline], [static]

This function provides the method to pause writing data.

Parameters

handler	SAI handler structure pointer of the SAI module.

17.2.3.8 static void sai_clear_tx_status (sai_handler_t * *handler*) [inline], [static]

Parameters

handler	SAI handler structure pointer of the SAI module.

17.2.3.9 static void sai_clear_rx_status (sai_handler_t * *handler*) [inline], [static]

Parameters

handler	SAI handler structure pointer of the SAI module.

17.2.3.10 static uint32_t* sai_get_fifo_address (sai_handler_t * *handler*) [inline], [static]

This function is mainly used for the DMA settings, which the DMA configuration needs for the source/destination address of SAI.

Parameters

handler	SAI handler structure pointer of the SAI module.

Returns

Returns the address of the data channel FIFO.

17.2.3.11 uint32_t sai_send_data (sai_handler_t * *handler,* uint8_t * *addr,* uint32_t *len*)

This function sends the data to the TX FIFO. This function starts the transfer, and while finishing the transfer, it calls the callback function registered by users.

Parameters

handler	SAI handler structure pointer of the SAI module.
addr	Address of the data which needs to be transferred.
len	The data length which need to be sent.

Returns

Returns the length which was sent.

17.2.3.12 uint32_t sai_receive_data (sai_handler_t * *handler,* uint8_t * *addr,* uint32_t *len*)

This function receives the data from the RX FIFO. This function starts the transfer, and while finishing the transfer, it calls the callback function registered by users.

Parameters

handler	SAI handler structure pointer of the SAI module.
addr	Address of the data which needs to be transferred.
len	The data length which needs to be received.

Returns

Returns the length received.

17.2.3.13 void sai_register_callback (sai_handler_t * *handler,* sai_callback_t *callback,* void * *callback_param*)

This function tells SAI which function needs to be called after a period length transfer.

Parameters

handler	SAI handler structure pointer of the SAI module.
callback	Callback function defined by users.

callback_param	The parameter of the callback function.

Chapter 18
Secured Digital Host Controller (SDHC)

The Kinetis SDK provides both HAL and Peripheral drivers for the Secure Digital Host Controller (SDHC) block of Kinetis devices.

Modules

18.1 SDHC HAL

The chapter describes the programming interface of the SDHC HAL driver.

SDHC HAL FUNCTION

- static bool sdhc_hal_is_valid_instance (uint8_t instance)
  Checks whether the given instance is valid.
- static void sdhc_hal_set_dma_addr (uint8_t instance, uint32_t address)
  Configures the DMA address.
- static uint32_t sdhc_hal_get_dma_addr (uint8_t instance)
  Gets the DMA address.
- static uint32_t sdhc_hal_get_blksz (uint8_t instance)
  Gets the block size configured.
- static void sdhc_hal_set_blksz (uint8_t instance, uint32_t blockSize)
  Sets the block size.
- static void sdhc_hal_set_blkcnt (uint8_t instance, uint32_t blockCount)
  Sets the block count.
- static uint32_t sdhc_hal_get_blkcnt (uint8_t instance)
  Gets the block count configured.
- static void sdhc_hal_set_cmd_arg (uint8_t instance, uint32_t arg)
  Configures the command argument.
- static void sdhc_hal_send_cmd (uint8_t instance, uint32_t index, uint32_t flags)
  Sends a command.
- static void sdhc_hal_set_data (uint8_t instance, uint32_t data)
  Fills the the data port.
- static uint32_t sdhc_hal_get_data (uint8_t instance)
  Retrieves the data from the data port.
- static uint32_t sdhc_hal_is_cmd_inhibit (uint8_t instance)
  Checks whether the command inhibit bit is set or not.
- static uint32_t sdhc_hal_is_data_inhibit (uint8_t instance)
  Checks whether data inhibit bit is set or not.
- static uint32_t sdhc_hal_is_data_line_active (uint8_t instance)
  Checks whether data line is active.
- static uint32_t sdhc_hal_is_sd_clk_stable (uint8_t instance)
  Checks whether the SD clock is stable or not.
- static uint32_t sdhc_hal_is_ipg_clk_off (uint8_t instance)
  Checks whether the IPG clock is off or not.
- static uint32_t sdhc_hal_is_sys_clk_off (uint8_t instance)
  Checks whether the system clock is off or not.
- static uint32_t sdhc_hal_is_per_clk_off (uint8_t instance)
  Checks whether the peripheral clock is off or not.
- static uint32_t sdhc_hal_is_sd_clk_off (uint8_t instance)
  Checks whether the SD clock is off or not.
- static uint32_t sdhc_hal_is_write_trans_active (uint8_t instance)
  Checks whether the write transfer is active or not.
- static uint32_t sdhc_hal_is_read_trans_active (uint8_t instance)
  Checks whether the read transfer is active or not.
- static uint32_t sdhc_hal_is_buf_write_enabled (uint8_t instance)
  Check whether the buffer write is enabled or not.
- static uint32_t sdhc_hal_is_buf_read_enabled (uint8_t instance)

Checks whether the buffer read is enabled or not.
- static uint32_t sdhc_hal_is_card_inserted (uint8_t instance)
  Checks whether the card is inserted or not.
- static uint32_t sdhc_hal_is_cmd_line_level_high (uint8_t instance)
  Checks whether the command line signal is high or not.
- static uint32_t sdhc_hal_get_data_line_level (uint8_t instance)
  Gets the data line signal level or not.
- static void sdhc_hal_set_led_state (uint8_t instance, sdhc_hal_led_t state)
  Sets the LED state.
- static void sdhc_hal_set_data_trans_width (uint8_t instance, sdhc_hal_dtw_t dtw)
  Sets the data transfer width.
- static bool sdhc_hal_is_d3cd_enabled (uint8_t instance)
  Checks whether the DAT3 is taken as card detect pin.
- static void sdhc_hal_enable_d3cd (uint8_t instance, bool isEnabled)
  Enables the DAT3 as a card detect pin.
- static void sdhc_hal_set_endian (uint8_t instance, sdhc_hal_endian_t endianMode)
  Configures the endian mode.
- static uint32_t sdhc_hal_get_cd_test_level (uint8_t instance)
  Gets the card detect test level.
- static void sdhc_hal_enable_cd_test (uint8_t instance, bool isEnabled)
  Enables the card detect test.
- static void sdhc_hal_set_dma_mode (uint8_t instance, sdhc_hal_dma_mode_t dmaMode)
  Sets the DMA mode.
- static void sdhc_hal_enable_stop_at_blkgap (uint8_t instance, bool isEnabled)
  Enables stop at the block gap.
- static void sdhc_hal_continue_req (uint8_t instance)
  Restarts a transaction which has stopped at the block gap.
- static void sdhc_hal_enable_read_wait_ctrl (uint8_t instance, bool isEnabled)
  Enables teh read wait control for the SDIO cards.
- static void sdhc_hal_enable_intr_stop_at_blk_gap (uint8_t instance, bool isEnabled)
  Enables stop at the block gap requests.
- static void sdhc_hal_enable_wakeup_on_card_intr (uint8_t instance, bool isEnabled)
  Enables wakeup event on the card interrupt.
- static void sdhc_hal_enable_wakeup_on_card_ins (uint8_t instance, bool isEnabled)
  Enables wakeup event on the card insertion.
- static void sdhc_hal_enable_wakeup_on_card_rm (uint8_t instance, bool isEnabled)
  Enables wakeup event on card removal.
- static void sdhc_hal_enable_ipg_clk (uint8_t instance, bool isEnabled)
  Enables the IPG clock, then no automatic clock gating off.
- static void sdhc_hal_enable_sys_clk (uint8_t instance, bool isEnabled)
  Enables the system clock, then no automatic clock gating off.
- static void sdhc_hal_enable_per_clk (uint8_t instance, bool isEnabled)
  Enables the peripheral clock, then no automatic clock gating off.
- static void sdhc_hal_enable_sd_clk (uint8_t instance, bool isEnabled)
  Enables the SD clock.
- static void sdhc_hal_set_clk_div (uint8_t instance, uint32_t divisor)
  Sets the SD clock frequency divisor.
- static void sdhc_hal_set_clk_freq (uint8_t instance, uint32_t freq)
  Sets the SD clock frequency select.
- static void sdhc_hal_set_data_timeout (uint8_t instance, uint32_t timeout)
  Sets the data timeout counter value.

- static void sdhc_hal_reset (uint8_t instance, uint32_t type)
  Performs the kinds of SDHC reset.
- static uint32_t sdhc_hal_is_reset_done (uint8_t instance, uint32_t type)
  Checks whether the given SDHC reset is finished.
- static void sdhc_hal_init_card (uint8_t instance)
  Sends 80 SD clock cycles to the card.
- static uint32_t sdhc_hal_is_init_card_done (uint8_t instance)
  Checks whether sending 80 SD clock cycles to card is finished.
- static uint32_t sdhc_hal_get_intr_flags (uint8_t instance)
  Gets the current interrupt status.
- static void sdhc_hal_clear_intr_flags (uint8_t instance, uint32_t mask)
  Clears a specified interrupt status.
- static uint32_t sdhc_hal_get_intr_signal (uint8_t instance)
  Gets the currently enabled interrupt signal.
- static uint32_t sdhc_hal_get_intr_state (uint8_t instance)
  Gets the currently enabled interrupt state.
- static uint32_t sdhc_hal_get_ac12_error (uint8_t instance)
  Gets the auto cmd12 error.
- static uint32_t sdhc_hal_get_max_blklen (uint8_t instance)
  Gets the maximum block length supported.
- static uint32_t sdhc_hal_host_can_do_adma (uint8_t instance)
  Checks whether the ADMA is supported.
- static uint32_t sdhc_hal_host_can_do_highspeed (uint8_t instance)
  Checks whether the high speed is supported.
- static uint32_t sdhc_hal_host_can_do_dma (uint8_t instance)
  Checks whether the DMA is supported.
- static uint32_t sdhc_hal_host_can_do_suspend_resume (uint8_t instance)
  Checks whether the suspend/resume is supported.
- static uint32_t sdhc_hal_host_supports_v330 (uint8_t instance)
  Checks whether the voltage 3.3 is supported.
- static uint32_t sdhc_hal_host_supports_v300 (uint8_t instance)
  Checks whether the voltage 3.0 is supported.
- static uint32_t sdhc_hal_host_supports_v180 (uint8_t instance)
  Checks whether the voltage 1.8 is supported.
- static void sdhc_hal_set_write_watermark (uint8_t instance, uint32_t watermark)
  Sets the watermark for writing.
- static void sdhc_hal_set_read_watermark (uint8_t instance, uint32_t watermark)
  Sets the watermark for reading.
- static void sdhc_hal_set_force_event_flags (uint8_t instance, uint32_t mask)
  Sets the force events according to the given mask.
- static uint32_t sdhc_hal_is_adma_len_mismatch_err (uint8_t instance)
  Checks whether the ADMA error is length mismatch.
- static uint32_t sdhc_hal_get_adma_error_stat (uint8_t instance)
  Gets back the state of the ADMA error.
- static uint32_t sdhc_hal_is_adma_desc_err (uint8_t instance)
  Checks whether the ADMA error is a descriptor error.
- static void sdhc_hal_set_adma_addr (uint8_t instance, uint32_t address)
  Sets the ADMA address.
- static void sdhc_hal_enable_ext_dma_req (uint8_t instance, bool isEnabled)
  Enables the external DMA request.
- static void sdhc_hal_enable_exact_blk_num (uint8_t instance, bool isEnabled)

Enables the exact block number for the SDIO CMD53.
- static void sdhc_hal_set_boot_ack_timeout (uint8_t instance, uint32_t timeout)
  Sets the timeout value for the boot ACK.
- static void sdhc_hal_enable_boot_ack (uint8_t instance, bool isEnabled)
  Enables the boot ACK.
- static void sdhc_hal_set_boot_mode (uint8_t instance, sdhc_hal_mmcboot_t mode)
  Configures the boot mode.
- static void sdhc_hal_enable_fastboot (uint8_t instance, bool isEnabled)
  Enables the fast boot.
- static void sdhc_hal_enable_auto_stop_at_blkgap (uint8_t instance, bool isEnabled)
  Enables the automatic stop at the block gap.
- static void sdhc_hal_set_boot_blkcnt (uint8_t instance, uint32_t blockCount)
  Configures the the block count for the boot.
- static uint32_t sdhc_hal_get_spec_ver (uint8_t instance)
  Gets a specification version.
- static uint32_t sdhc_hal_get_vendor_ver (uint8_t instance)
  Gets the vendor version.
- void sdhc_hal_get_resp (uint8_t instance, uint32_t *resp)
  Gets the command response.
- void sdhc_hal_enable_intr_signal (uint8_t instance, bool isEnabled, uint32_t mask)
  Enables the specified interrupts.
- void sdhc_hal_enable_intr_state (uint8_t instance, bool isEnabled, uint32_t mask)
  Enables the specified interrupt state.

18.1.1 Function Documentation

18.1.1.1 static bool sdhc_hal_is_valid_instance (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

true if valid

18.1.1.2 static void sdhc_hal_set_dma_addr (uint8_t *instance,* uint32_t *address*) [inline], [static]

Parameters

instance	SDHC instance ID
address	the DMA address

18.1.1.3 static uint32_t sdhc_hal_get_dma_addr (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

the DMA address

18.1.1.4 static uint32_t sdhc_hal_get_blksz (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

the block size already configured

18.1.1.5 static void sdhc_hal_set_blksz (uint8_t *instance,* uint32_t *blockSize*) [inline], [static]

Parameters

instance	SDHC instance ID
blockSize	the block size

18.1.1.6 static void sdhc_hal_set_blkcnt (uint8_t *instance,* uint32_t *blockCount*) [inline], [static]

Parameters

instance	SDHC instance ID
blockCount	the block count

18.1.1.7 static uint32_t sdhc_hal_get_blkcnt (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance id

Returns

the block count already configured

18.1.1.8 static void sdhc_hal_set_cmd_arg (uint8_t *instance,* uint32_t *arg*) [inline], [static]

Parameters

instance	SDHC instance ID
arg	the command argument

18.1.1.9 static void sdhc_hal_send_cmd (uint8_t *instance,* uint32_t *index,* uint32_t *flags*) [inline], [static]

Parameters

instance	SDHC instance ID
index	command index
flags	transfer type flags

18.1.1.10 static void sdhc_hal_set_data (uint8_t *instance,* uint32_t *data*) [inline], [static]

Parameters

instance	SDHC instance ID
data	the data about to be sent

18.1.1.11 static uint32_t sdhc_hal_get_data (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

data the data read

18.1.1.12 static uint32_t sdhc_hal_is_cmd_inhibit (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if command inhibit, 0 if not.

18.1.1.13 static uint32_t sdhc_hal_is_data_inhibit (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if data inhibit, 0 if not.

18.1.1.14 static uint32_t sdhc_hal_is_data_line_active (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if it's active, 0 if not.

18.1.1.15 static uint32_t sdhc_hal_is_sd_clk_stable (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if it's stable, 0 if not.

18.1.1.16 static uint32_t sdhc_hal_is_ipg_clk_off (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if it's off, 0 if not.

18.1.1.17 static uint32_t sdhc_hal_is_sys_clk_off (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if it's off, 0 if not.

18.1.1.18 static uint32_t sdhc_hal_is_per_clk_off (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID.

Returns

1 if it's off, 0 if not.

18.1.1.19 static uint32_t sdhc_hal_is_sd_clk_off (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if it's off, 0 if not.

18.1.1.20 static uint32_t sdhc_hal_is_write_trans_active (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if it's active, 0 if not.

18.1.1.21 static uint32_t sdhc_hal_is_read_trans_active (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if it's off, 0 if not.

18.1.1.22 static uint32_t sdhc_hal_is_buf_write_enabled (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if it's isEnabledd, 0 if not.

18.1.1.23 static uint32_t sdhc_hal_is_buf_read_enabled (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if it's isEnabledd, 0 if not.

18.1.1.24 static uint32_t sdhc_hal_is_card_inserted (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID.

Returns

1 if it's inserted, 0 if not.

18.1.1.25 static uint32_t sdhc_hal_is_cmd_line_level_high (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

1 if it's high, 0 if not.

18.1.1.26 static uint32_t sdhc_hal_get_data_line_level (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

[7:0] data line signal level

18.1.1.27 static void sdhc_hal_set_led_state (uint8_t *instance,* sdhc_hal_led_t *state*) [inline], [static]

Parameters

instance	SDHC instance ID

state	the LED state

18.1.1.28 static void sdhc_hal_set_data_trans_width (uint8_t *instance,* sdhc_hal_dtw_t *dtw*) [inline], [static]

Parameters

instance	SDHC instance ID
dtw	data transfer width

18.1.1.29 static bool sdhc_hal_is_d3cd_enabled (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

18.1.1.30 static void sdhc_hal_enable_d3cd (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	isEnabled the feature

18.1.1.31 static void sdhc_hal_set_endian (uint8_t *instance,* sdhc_hal_endian_t *endianMode*) [inline], [static]

Parameters

instance	SDHC instance ID
endianMode	endian mode

18.1.1.32 static uint32_t sdhc_hal_get_cd_test_level (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

card detect test level

18.1.1.33 static void sdhc_hal_enable_cd_test (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.34 static void sdhc_hal_set_dma_mode (uint8_t *instance,* sdhc_hal_dma_mode_t *dmaMode*) [inline], [static]

Parameters

instance	SDHC instance ID
dmaMode	the DMA mode

18.1.1.35 static void sdhc_hal_enable_stop_at_blkgap (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.36 static void sdhc_hal_continue_req (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

18.1.1.37 static void sdhc_hal_enable_read_wait_ctrl (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.38 static void sdhc_hal_enable_intr_stop_at_blk_gap (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.39 static void sdhc_hal_enable_wakeup_on_card_intr (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.40 static void sdhc_hal_enable_wakeup_on_card_ins (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.41 static void sdhc_hal_enable_wakeup_on_card_rm (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.42 static void sdhc_hal_enable_ipg_clk (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.43 static void sdhc_hal_enable_sys_clk (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.44 static void sdhc_hal_enable_per_clk (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.45 static void sdhc_hal_enable_sd_clk (uint8_t *instance,* bool *isEnabled*) [inline], [static]

It should be disabled before changing SD clock frequency.

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.46 static void sdhc_hal_set_clk_div (uint8_t *instance,* uint32_t *divisor*) [inline], [static]

Parameters

instance	SDHC instance ID
divisor	the divisor

18.1.1.47 static void sdhc_hal_set_clk_freq (uint8_t *instance,* uint32_t *freq*) [inline], [static]

Parameters

instance	SDHC instance ID
freq	the frequency selector

18.1.1.48 static void sdhc_hal_set_data_timeout (uint8_t *instance,* uint32_t *timeout*) [inline], [static]

Parameters

instance	SDHC instance ID
timeout	

18.1.1.49 static void sdhc_hal_reset (uint8_t *instance,* uint32_t *type*) [inline], [static]

Parameters

instance	SDHC instance ID
type	the type of reset

18.1.1.50 static uint32_t sdhc_hal_is_reset_done (uint8_t *instance,* uint32_t *type*) [inline], [static]

Parameters

instance	SDHC instance ID
type	the type of reset

Returns

if the given reset is done

18.1.1.51 static void sdhc_hal_init_card (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

18.1.1.52 static uint32_t sdhc_hal_is_init_card_done (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

if sending 80 SD clock cycles is finished

18.1.1.53 static uint32_t sdhc_hal_get_intr_flags (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

current interrupt flags

18.1.1.54 static void sdhc_hal_clear_intr_flags (uint8_t *instance,* uint32_t *mask*) [inline], [static]

Parameters

instance	SDHC instance ID
mask	to specify interrupts' flags to be cleared

18.1.1.55 static uint32_t sdhc_hal_get_intr_signal (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

currently enabled interrupt signal

18.1.1.56 static uint32_t sdhc_hal_get_intr_state (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

currently enabled interrupts' state

18.1.1.57 static uint32_t sdhc_hal_get_ac12_error (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

auto cmd12 error status

18.1.1.58 static uint32_t sdhc_hal_get_max_blklen (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

the maximum block length support

18.1.1.59 static uint32_t sdhc_hal_host_can_do_adma (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

if ADMA is supported

18.1.1.60 static uint32_t sdhc_hal_host_can_do_highspeed (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

if high speed is supported

18.1.1.61 static uint32_t sdhc_hal_host_can_do_dma (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

if high speed is supported

18.1.1.62 static uint32_t sdhc_hal_host_can_do_suspend_resume (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

if suspend and resume is supported

18.1.1.63 static uint32_t sdhc_hal_host_supports_v330 (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

if voltage 3.3 is supported

18.1.1.64 static uint32_t sdhc_hal_host_supports_v300 (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

if voltage 3.0 is supported

18.1.1.65 static uint32_t sdhc_hal_host_supports_v180 (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

if voltage 1.8 is supported

18.1.1.66 static void sdhc_hal_set_write_watermark (uint8_t *instance,* uint32_t *watermark*) [inline], [static]

Parameters

instance	SDHC instance ID
watermark	for writing

18.1.1.67 static void sdhc_hal_set_read_watermark (uint8_t *instance,* uint32_t *watermark*) [inline], [static]

Parameters

instance	SDHC instance ID
watermark	for reading

18.1.1.68 static void sdhc_hal_set_force_event_flags (uint8_t *instance,* uint32_t *mask*) [inline], [static]

Parameters

instance	SDHC instance ID
mask	to specify the force events' flags to be set

18.1.1.69 static uint32_t sdhc_hal_is_adma_len_mismatch_err (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

if ADMA error is length mismatch

18.1.1.70 static uint32_t sdhc_hal_get_adma_error_stat (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

error state

18.1.1.71 static uint32_t sdhc_hal_is_adma_desc_err (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

if ADMA error is descriptor error

18.1.1.72 static void sdhc_hal_set_adma_addr (uint8_t *instance,* uint32_t *address*) [inline], [static]

Parameters

instance	SDHC instance ID
address	for ADMA transfer

18.1.1.73 static void sdhc_hal_enable_ext_dma_req (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	or not

18.1.1.74 static void sdhc_hal_enable_exact_blk_num (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	or not

18.1.1.75 static void sdhc_hal_set_boot_ack_timeout (uint8_t *instance,* uint32_t *timeout*) [inline], [static]

Parameters

instance	SDHC instance ID
timeout	

18.1.1.76 static void sdhc_hal_enable_boot_ack (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	

18.1.1.77 static void sdhc_hal_set_boot_mode (uint8_t *instance,* sdhc_hal_mmcboot_t *mode*) [inline], [static]

Parameters

instance	SDHC instance ID
mode	the boot mode

18.1.1.78 static void sdhc_hal_enable_fastboot (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	or not

18.1.1.79 static void sdhc_hal_enable_auto_stop_at_blkgap (uint8_t *instance,* bool *isEnabled*) [inline], [static]

Parameters

instance	SDHC instance ID
isEnabled	or not

18.1.1.80 static void sdhc_hal_set_boot_blkcnt (uint8_t *instance,* uint32_t *blockCount*) [inline], [static]

Parameters

instance	SDHC instance ID
blockCount	the block count for boot

18.1.1.81 static uint32_t sdhc_hal_get_spec_ver (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

specification version

18.1.1.82 static uint32_t sdhc_hal_get_vendor_ver (uint8_t *instance*) [inline], [static]

Parameters

instance	SDHC instance ID

Returns

vendor version

18.1.1.83 void sdhc_hal_get_resp (uint8_t *instance,* uint32_t * *resp*)

Parameters

instance	SDHC instance ID
resp	an array of response, 4 bytes

18.1.1.84 void sdhc_hal_enable_intr_signal (uint8_t *instance,* bool *isEnabled,* uint32_t *mask*)

Parameters

instance	SDHC instance ID
isEnabled	enable or disable
mask	to specify interrupts to be isEnabledd

18.1.1.85 void sdhc_hal_enable_intr_state (uint8_t *instance,* bool *isEnabled,* uint32_t *mask*)

Parameters

instance	SDHC instance ID
isEnabled	enable or disable
mask	to specify interrupts' state to be enabled

18.2 SDHC Peripheral Driver

The chapter describes the programming interface of the SDCH Peripheral driver.

SDHC PD FUNCTION

- sdhc_status_t sdhc_init (uint8_t instance, sdhc_host_t *host, const sdhc_user_config_t *config)
  Initializes the Host controller by a specific instance index.
- void sdhc_shutdown (sdhc_host_t *host)
  Destroy host controller.
- sdhc_status_t sdhc_check_card (sdhc_host_t *host, sdhc_card_t *card)
  check whether the card is present on specified host controller.
- sdhc_status_t sdhc_check_ro (sdhc_host_t *host)
  Checks the read only for the attached card.
- sdhc_status_t sdhc_config_host (sdhc_host_t *host, sdhc_host_config_t *config)
  Configures the specified host controller.
- sdhc_status_t sdhc_issue_request (sdhc_host_t *host, sdhc_request_t *req)
  Issues the request on a specific Host controller and returns on completion.

18.2.0.86 SDHC Peripheral Driver

Overview

The SDHC driver configures the secure digital host controller and provides an easy way to operate the SDHC module.

Initialization

To initialize the SDHC module, call the sdhc_init() function and pass in the configuration data structure.

This is an example code to initialize and configure the driver:

```
// Define device configuration.
sdhc_init_config_t config = {0};

config.busWidth = SD_BUS_WIDTH_1BIT;
config.clock = 0;

// Initialize
if (sdhc_init(instance, &host, &config) != kStatus_SDHC_NoError)
{
    /* error occurs */
}
else
{
    /* SDHC has been successfully initialized */
}
```

Issuing a request to the card

The SDHC driver provides a simple way to send commands to and retrieve response/data from the card.

This is a example to send the SD_SWITCH command to the card.

```
sdhc_request_t req = {0};
sdhc_command_t cmd = {0};
sdhc_data_t data = {0};

cmd.index = kSdSwitch;                                  /*  Set command index */
cmd.argument = mode << 31 | 0x00FFFFFF;                 /*  Set argument */
cmd.argument &= ~((0xF) << (group * 4));
cmd.argument |= value << (group * 4);
cmd.flags = SDMMC_RSP_R1 | SDMMC_CMD_ADTC;              /*  Set command flags */

data.blockSize = 64;                                    /*  Set data block size */
data.blockCount = 1;                                    /*  Set data block count */
data.flags = SDMMC_DATA_READ;                           /*  Set data direction */
data.buffer = response;                                 /*  Set data buffer */

/* link data, command with request */
data.req = &req;
data.cmd = &cmd;
req.cmd = &cmd;
req.data = &data;
cmd.data = &data;
if (kStatus_SDHC_NoError != sdhc_issue_request(host, &req))  /* issue
       reqeust */
{
    /* error occurs */
}
else
{
    /* request has been issued successfully */
}
```

18.2.1 Function Documentation

18.2.1.1 sdhc_status_t sdhc_init (uint8_t *instance*, sdhc_host_t * *host*, const sdhc_user_config_t * *config*)

This function initializes the SDHC module according to the given initialization configuration structure including the clock frequency, bus width, and card detect callback.

Parameters

instance	the specific instance index
host	the memory address allocated for the host handle

config	initialization configuration data

Returns

kStatus_SDHC_NoError if success

18.2.1.2 void sdhc_shutdown (sdhc_host_t * *host*)

Parameters

host	the pointer to the host controller about to be destroyed

18.2.1.3 sdhc_status_t sdhc_check_card (sdhc_host_t * *host*, sdhc_card_t * *card*)

This function checks if there's a card inserted to the SDHC.

Parameters

host	the pointer to the host controller
card	the pointer to retrieve card information

Returns

kStatus_SDHC_NoError on success

18.2.1.4 sdhc_status_t sdhc_check_ro (sdhc_host_t * *host*)

Parameters

host	the pointer to the host controller

Returns

kStatus_SDHC_NoError on success

18.2.1.5 sdhc_status_t sdhc_config_host (sdhc_host_t * *host*, sdhc_host_config_t * *config*)

With this function, a user can modify the specific SDHC configuration.

Parameters

host	the pointer to the host controller
config	the pointer to the configuration information

Returns

kStatus_SDHC_NoError on success

18.2.1.6 sdhc_status_t sdhc_issue_request (sdhc_host_t * *host*, sdhc_request_t * *req*)

This function issues the request to the card on a specific SDHC. The command is sent and is blocked as long as the response/data is sending back from the card.

Parameters

host	the pointer to the host controller
req	the pointer to the request

Returns

kStatus_SDHC_NoError on success

18.3 SDHC Data Types

The chapter describes the SDHC Data Types.

Data Structures

- struct sdhc_card_t
  SDHC Card Structure. More...
- struct sdhc_user_config_t
  SDHC Initialization Configuration Structure. More...
- struct sdhc_host_config_t
  SDHC Configure Structure. More...
- struct sdhc_host_t
  SDHC Host Device Structure. More...
- struct sdhc_data_t
  SDHC Data Structure. More...
- struct sdhc_command_t
  SDHC Command Structure. More...
- struct sdhc_request_t
  SDHC Request Structure. More...

Typedefs

- typedef void(∗ request_done)(struct SdhcHostDevice ∗host, struct SdhcRequest ∗req)
  SDHC Request Done Callback.

Enumerations

- enum sdhc_card_type_t {
  kCardTypeUnknown = 1,
  kCardTypeSDSC,
  kCardTypeSDHC,
  kCardTypeSDXC,
  kCardTypeMMC }
- enum sdhc_status_t {

kStatus_SDHC_NoError = 0,
kStatus_SDHC_WaitTimeoutError,
kStatus_SDHC_IoError,
kStatus_SDHC_CmdIoError,
kStatus_SDHC_DataIoError,
kStatus_SDHC_InvalidParameter,
kStatus_SDHC_RequestFailed,
kStatus_SDHC_SwitchFailed,
kStatus_SDHC_NotSupportYet,
kStatus_SDHC_TimeoutError,
kStatus_SDHC_CardNotSupport,
kStatus_SDHC_CmdError,
kStatus_SDHC_DataError,
kStatus_SDHC_Failed,
kStatus_SDHC_NoMedium }

- enum sdhc_power_mode_t {
  kSdhcPowerModeRunning = 0,
  kSdhcPowerModeSuspended,
  kSdhcPowerModeStopped }

18.3.1 Data Structure Documentation

18.3.1.1 struct sdhc_card_t

Defines the card structure including the necessary fields to identify and describe the card.

Data Fields

- struct SdhcHostDevice * host
  Associated host.
- uint32_t version
  Card version.
- uint32_t rca
  RCA.
- sdhc_card_type_t cardType
  Card type.
- uint32_t flags
  Flags.
- uint32_t caps
  Capability.
- uint32_t busMode
  Data width.
- uint32_t rawCid [4]
  Raw CID.
- uint32_t rawCsd [4]
  Raw CSD.

- uint32_t rawScr [2]
  Raw SCR.
- uint8_t * rawExtCsd
  Raw EXT_CSD.
- uint32_t capacity
  Card total size.

18.3.1.2 struct sdhc_user_config_t

Defines the configuration data structure to initialize the SDHC.

Data Fields

- uint32_t clock
  Clock rate.
- uint32_t busWidth
  Data bus width.
- void(* card_detect_callback)(void *param)
  Card detect callback function.

18.3.1.3 struct sdhc_host_config_t

SDHC Configuration Data Structure

Data Fields

- uint32_t clock
  Clock rate.
- sdhc_power_mode_t powerMode
  Power supply mode.
- uint32_t busWidth
  Data bus width.

18.3.1.4 struct sdhc_host_t

Defines the Host device structure which includes both the static and the runtime SDHC information.

Data Fields

- uint8_t instance
  Host instance index.
- uint32_t specVer
  Host specification version.
- uint32_t vendorVer
  Host vendor version.

- sdhc_hal_endian_t endian
  Endian mode the host's working at.
- IRQn_Type irq
  IRQ number.
- uint32_t flags
  Host flags.
- uint32_t busWidth
  Current busWidth.
- uint32_t caps
  Host capability.
- uint32_t ocrSupported
  Supported OCR.
- uint32_t clock
  Current clock frequency.
- sdhc_power_mode_t powerMode
  Current power mode.
- uint32_t maxClock
  Maximum clock supported.
- uint32_t maxBlockSize
  Maximum block size supported.
- struct SdhcHostConfig config
  Host configuration.
- struct SdhcRequest * currentReq
  Associated request.
- struct SdhcCommand * currentCmd
  Associated command.
- struct SdhcData * currentData
  Associated data.
- struct SdhcCard * card
  Associated card.
- sync_object_t host_lock
  Sync object.

18.3.1.5 struct sdhc_data_t

Defines the SDHC data structure including the block size/count and flags.

Data Fields

- struct SdhcRequest * req
  Associated request.
- struct SdhcCommand * cmd
  Associated command.
- uint32_t blockSize
  Block size.
- uint32_t blockCount
  Block count.
- uint32_t flags
  Data flags.

- uint32_t error
  Data error code.
- uint32_t bytesTransferred
  Transferred buffer.
- uint32_t length
  Data length.
- uint32_t * buffer
  Data buffer.

18.3.1.6 struct sdhc_command_t

Defines the SDHC command structure including the command index, argument, flags, and response.

Data Fields

- struct SdhcRequest * req
  Associated request.
- struct SdhcData * data
  Data associated with request.
- uint32_t index
  Command index.
- uint32_t argument
  Command argument.
- uint32_t flags
  Command flags.
- uint32_t response [4]
  Response for this command.
- uint32_t error
  Command error code.

18.3.1.7 struct sdhc_request_t

Defines the SDHC request structure. In most cases, it includes the related command and data and callback upon its completion.

Data Fields

- struct SdhcCommand * cmd
  Command associated with the request.
- struct SdhcData * data
  Data associated with request.
- sync_object_t done
  Sync object.
- request_done onDone
  Callback on request done.

18.3.2 Enumeration Type Documentation

18.3.2.1 enum sdhc_card_type_t

Enumerator

kCardTypeUnknown Unknown card type.
kCardTypeSDSC SDSC card type.
kCardTypeSDHC SDHC card type.
kCardTypeSDXC SDXC card type.
kCardTypeMMC MMC card type.

18.3.2.2 enum sdhc_status_t

Enumerator

kStatus_SDHC_NoError SDHC no error.
kStatus_SDHC_WaitTimeoutError SDHC wait timeout error.
kStatus_SDHC_IoError SDHC general IO error.
kStatus_SDHC_CmdIoError SDHC CMD I/O error.
kStatus_SDHC_DataIoError SDHC data I/O error.
kStatus_SDHC_InvalidParameter SDHC invalid parameter.
kStatus_SDHC_RequestFailed SDHC request failed.
kStatus_SDHC_SwitchFailed SDHC switch failed.
kStatus_SDHC_NotSupportYet SDHC not support.
kStatus_SDHC_TimeoutError SDHC timeout error.
kStatus_SDHC_CardNotSupport SDHC card does not support.
kStatus_SDHC_CmdError SDHC CMD error.
kStatus_SDHC_DataError SDHC data error.
kStatus_SDHC_Failed SDHC general failed.
kStatus_SDHC_NoMedium SDHC no medium error.

18.3.2.3 enum sdhc_power_mode_t

Enumerator

kSdhcPowerModeRunning SDHC is running.
kSdhcPowerModeSuspended SDHC is suspended.
kSdhcPowerModeStopped SDHC is stopped.

18.4 SDHC Card Related Standard Definition

The chapter describes the card standard definition.

Macros

- #define SDMMC_CARD_BUSY (uint32_t)(1 << 31)
  card initialization complete
- #define SDMMC_CARD_4BIT_MODE (1 << 18)
  card supports 4 bit mode
- #define SD_OCR_CCS (1 << 30)
  card capacity status
- #define SD_OCR_XPC (1 << 28)
  SDXC power control.
- #define SD_OCR_S18R (1 << 24)
  switch to 1.8V request
- #define SD_OCR_S18A SD_OCR_S18R
  switch to 1.8V accepted
- #define SD_BUS_WIDTH_1BIT (0U)
  SD data bus width 1-bit mode.
- #define SD_BUS_WIDTH_4BIT (2U)
  SD data bus width 4-bit mode.
- #define SD_HIGHSPEED_BUSY (0x00020000U)
  SD card high speed busy status bit in CMD6 response.
- #define SD_HIGHSPEED_SUPPORTED (0x00020000U)
  SD card high speed support bit in CMD6 response.
- #define SD_OCR_VDD_27_28 (1 << 15)
  VDD 2.7-2.8.
- #define SD_OCR_VDD_28_29 (1 << 16)
  VDD 2.8-2.9.
- #define SD_OCR_VDD_29_30 (1 << 17)
  VDD 2.9-3.0.
- #define SD_OCR_VDD_30_31 (1 << 18)
  VDD 3.0-3.1.
- #define SD_OCR_VDD_31_32 (1 << 19)
  VDD 3.1-3.2.
- #define SD_OCR_VDD_32_33 (1 << 20)
  VDD 3.2-3.3.
- #define SD_OCR_VDD_33_34 (1 << 21)
  VDD 3.3-3.4.
- #define SD_OCR_VDD_34_35 (1 << 22)
  VDD 3.4-3.5.
- #define SD_OCR_VDD_35_36 (1 << 23)
  VDD 3.5-3.6.
- #define SDMMC_R1_OUT_OF_RANGE (uint32_t)(1 << 31)
  R1: out of range status bit.
- #define SDMMC_R1_ADDRESS_ERROR (1 << 30)
  R1: address error status bit.
- #define SDMMC_R1_BLOCK_LEN_ERROR (1 << 29)
  R1: block length error status bit.
- #define SDMMC_R1_ERASE_SEQ_ERROR (1 << 28)

R1: erase sequence error status bit.

- #define SDMMC_R1_ERASE_PARAM (1 << 27)
  R1: erase parameter error status bit.
- #define SDMMC_R1_WP_VIOLATION (1 << 26)
  R1: write protection violation status bit.
- #define SDMMC_R1_CARD_IS_LOCKED (1 << 25)
  R1: card locked status bit.
- #define SDMMC_R1_LOCK_UNLOCK_FAILED (1 << 24)
  R1: lock/unlock error status bit.
- #define SDMMC_R1_COM_CRC_ERROR (1 << 23)
  R1: CRC error status bit.
- #define SDMMC_R1_ILLEGAL_COMMAND (1 << 22)
  R1: illegal command status bit.
- #define SDMMC_R1_CARD_ECC_FAILED (1 << 21)
  R1: card ecc error status bit.
- #define SDMMC_R1_CC_ERROR (1 << 20)
  R1: internal card controller status bit.
- #define SDMMC_R1_ERROR (1 << 19)
  R1: a general or an unknown error status bit.
- #define SDMMC_R1_UNDERRUN (1 << 18)
  R1: underrun status bit.
- #define SDMMC_R1_OVERRUN (1 << 17)
  R1: overrun status bit.
- #define SDMMC_R1_CID_CSD_OVERWRITE (1 << 16)
  R1: cid/csd overwrite status bit.
- #define SDMMC_R1_WP_ERASE_SKIP (1 << 15)
  R1: write protection erase skip status bit.
- #define SDMMC_R1_CARD_ECC_DISABLED (1 << 14)
  R1: card ecc disabled status bit.
- #define SDMMC_R1_ERASE_RESET (1 << 13)
  R1: erase reset status bit.
- #define SDMMC_R1_STATUS(x) do { ((x) & 0xFFFFE000U); } while(0)
  R1: status.
- #define SDMMC_R1_READY_FOR_DATA (1 << 8)
  R1: ready for data status bit.
- #define SDMMC_R1_SWITCH_ERROR (1 << 7)
  R1: switch error status bit.
- #define SDMMC_R1_APP_CMD (1 << 5)
  R1: application command enabled status bit.
- #define SDMMC_R1_CURRENT_STATE(x) do { (((x) & 0x00001E00U) >> 9); } while(0)
  R1: current state.
- #define SDMMC_R1_STATE_IDLE (0U)
  R1: current state: idle.
- #define SDMMC_R1_STATE_READY (1U)
  R1: current state: ready.
- #define SDMMC_R1_STATE_IDENT (2U)
  R1: current state: ident.
- #define SDMMC_R1_STATE_STBY (3U)
  R1: current state: stby.
- #define SDMMC_R1_STATE_TRAN (4U)
  R1: current state: tran.

- #define SDMMC_R1_STATE_DATA (5U)
  R1: current state: data.
- #define SDMMC_R1_STATE_RCV (6U)
  R1: current state: rcv.
- #define SDMMC_R1_STATE_PRG (7U)
  R1: current state: prg.
- #define SDMMC_R1_STATE_DIS (8U)
  R1: current state: dis.
- #define SDMMC_SD_VERSION_1_0 (1 << 0)
  SD card version 1.0.
- #define SDMMC_SD_VERSION_1_1 (1 << 1)
  SD card version 1.1.
- #define SDMMC_SD_VERSION_2_0 (1 << 2)
  SD card version 2.0.
- #define SDMMC_SD_VERSION_3_0 (1 << 3)
  SD card version 3.0.

Enumerations

- enum mmc_cmd_t {
  kMmcSetRelativeAddr = 3,
  kMmcSleepAwake = 5,
  kMmcSwitch = 6,
  kMmcSendExtCsd = 8,
  kMmcReadDataUntilStop = 11,
  kMmcBusTestRead = 14,
  kMmcWriteDataUntilStop = 20,
  kMmcProgramCid = 26,
  kMmcEraseGroupStart = 35,
  kMmcEraseGroupEnd = 36,
  kMmcFastIo = 39,
  kMmcGoIrqState = 40 }
  < type argument response
- enum sdmmc_cmd_t {

kGoIdleState = 0,
kSendOpCond = 1,
kAllSendCid = 2,
kSetDsr = 4,
kSelectCard = 7,
kSendCsd = 9,
kSendCid = 10,
kStopTransmission = 12,
kSendStatus = 13,
kGoInactiveState = 15,
kSetBlockLen = 16,
kReadSingleBlock = 17,
kReadMultipleBlock = 18,
kSendTuningBlock = 19,
kSetBlockCount = 23,
kWriteBlock = 24,
kWriteMultipleBlock = 25,
kProgramCsd = 27,
kSetWriteProt = 28,
kClrWriteProt = 29,
kSendWriteProt = 30,
kErase = 38,
kLockUnlock = 42,
kAppCmd = 55,
kGenCmd = 56 }

- enum sd_cmd_t {
  kSdSendRelativeAddr = 3,
  kSdSwitch = 6,
  kSdSendIfCond = 8,
  kSdVoltageSwitch = 11,
  kSdSpeedClassControl = 20,
  kSdEraseWrBlkStart = 32,
  kSdEraseWrBlkEnd = 33 }
- enum sd_acmd_t {
  kSdAppSetBusWdith = 6,
  kSdAppStatus = 13,
  kSdAppSendNumWrBlocks = 22,
  kSdAppSetWrBlkEraseCount = 23,
  kSdAppSendOpCond = 41,
  kSdAppSetClrCardDetect = 42,
  kSdAppSendScr = 51 }
- enum sd_switch_mode_t {
  kSdSwitchCheck = 0,
  kSdSwitchSet = 1 }

18.4.1 Enumeration Type Documentation

18.4.1.1 enum mmc_cmd_t

Enumerator

kMmcSetRelativeAddr ac [31:16] RCA R1
kMmcSleepAwake ac [31:16] RCA R1b [15] flag
kMmcSwitch ac [31:16] RCA R1b
kMmcSendExtCsd adtc R1
kMmcReadDataUntilStop adtc [31:0] data R1 address
kMmcBusTestRead adtc R1
kMmcWriteDataUntilStop ac [31:0] data R1 address
kMmcProgramCid adtc R1
kMmcEraseGroupStart ac [31:0] data R1 address
kMmcEraseGroupEnd ac [31:0] data R1 address
kMmcFastIo ac R4
kMmcGoIrqState bcr R5

18.4.1.2 enum sdmmc_cmd_t

Enumerator

kGoIdleState bc
kSendOpCond bcr [31:0] OCR R3
kAllSendCid bcr R2
kSetDsr bc [31:16] RCA
kSelectCard ac [31:16] RCA R1b
kSendCsd ac [31:16] RCA R2
kSendCid ac [31:16] RCA R2
kStopTransmission ac [31:16] RCA R1b
kSendStatus ac [31:16] RCA R1
kGoInactiveState ac [31:16] RCA
kSetBlockLen ac [31:0] block R1 length
kReadSingleBlock adtc [31:0] data R1 address
kReadMultipleBlock adtc [31:0] data R1 address
kSendTuningBlock adtc [31:0] all R1 zero
kSetBlockCount ac [31:0] block R1 count
kWriteBlock adtc [31:0] data R1 address
kWriteMultipleBlock adtc [31:0] data R1 address
kProgramCsd adtc R1
kSetWriteProt ac [31:0] data R1b address
kClrWriteProt ac [31:0] data R1b address
kSendWriteProt adtc [31:0] write R1b protect data address
kErase ac R1

kLockUnlock adtc all zero R1
kAppCmd ac [31:16] RCA R1
kGenCmd adtc [0] RD/WR R1

18.4.1.3 enum sd_cmd_t

Enumerator

kSdSendRelativeAddr bcr R6
kSdSwitch adtc [31] mode R1 [15:12] func group 4: current limit [11:8] func group 3: drive strength [7:4] func group 2: command system [3:0] func group 1: access mode
kSdSendIfCond bcr [11:8] supply R7 voltage [7:0] check pattern
kSdVoltageSwitch ac R1
kSdSpeedClassControl ac [31:28] speed R1b class control
kSdEraseWrBlkStart ac [31:0] data R1 address
kSdEraseWrBlkEnd ac [31:0] data R1 address

18.4.1.4 enum sd_acmd_t

Enumerator

kSdAppSetBusWdith ac [1:0] bus R1 width
kSdAppStatus adtc R1
kSdAppSendNumWrBlocks adtc R1
kSdAppSetWrBlkEraseCount ac [22:0] number R1 of blocks
kSdAppSendOpCond bcr [30] HCS R3 [28] XPC [24] S18R [23:0] VDD voltage window
kSdAppSetClrCardDetect ac [0] set cd R1
kSdAppSendScr adtc R1

18.4.1.5 enum sd_switch_mode_t

Enumerator

kSdSwitchCheck SD switch mode 0: check function.
kSdSwitchSet SD switch mode 1: set function.

18.5 SDHC Standard Definition

The chapter describes the SDHC standard definition.

Macros

- #define SDHC_DMA_ADDRESS (0x00U)
  SDHC DMA ADDRESS REG.
- #define SDHC_BLOCK_SIZE (0x04U)
  SDHC BLOCK SIZE REG.
- #define SDHC_BLOCK_COUNT (0x06U)
  SDHC BLOCK COUNT REG.
- #define SDHC_ARGUMENT (0x08U)
  SDHC ARGUMENT REG.
- #define SDHC_TRANSFER_MODE (0x0C)
  SDHC TRANSFER MODE REG.
- #define SDHC_TRNSM_DMA_EN (0x01U)
  SDHC TRANSFER MODE DMA ENABLE BIT.
- #define SDHC_TRNSM_BLKCNT_EN (0x02U)
  SDHC TRANSFER MODE BLOCK COUNT ENABLE BIT.
- #define SDHC_TRNSM_AUTOCMD12 (0x04U)
  SDHC TRANSFER MODE AUTO CMD12 BIT.
- #define SDHC_TRNSM_AUTOCMD23 (0x08U)
  SDHC TRANSFER MODE AUTO CMD23 BIT.
- #define SDHC_TRNSM_READ (0x10U)
  SDHC TRANSFER MODE READ DATA BIT.
- #define SDHC_TRNSM_MULTI (0x20U)
  SDHC TRANSFER MODE MULTIBLOCK BIT.
- #define SDHC_COMMAND (0x0E)
  SDHC COMMAND REG.
- #define SDHC_CMD_RESPTYPE_LSF (0U)
  SDHC COMMAND RESPONSE TYPE SHIFT.
- #define SDHC_CMD_RESPTYPE_MASK (0x03U)
  SDHC COMMAND RESPONSE MASK.
- #define SDHC_CMD_CRC_CHK (0x08U)
  SDHC COMMAND CRC CHEKCING BIT.
- #define SDHC_CMD_INDEX_CHK (0x10U)
  SDHC COMMAND INDEX CHECKING BIT.
- #define SDHC_CMD_DATA_PRSNT (0x20U)
  SDHC COMMAND DATA PRESENT BIT.
- #define SDHC_CMD_CMDTYPE_LSF (6U)
  SDHC COMMAND COMMAND TYPE SHIFT.
- #define SDHC_CMD_CMDTYPE_MASK (0xC0U)
  SDHC COMMAND COMMAND TYPE MASK.
- #define SDHC_CMD_CMDINDEX_LSF (8U)
  SDHC COMMAND COMMAND INDEX SHIFT.
- #define SDHC_CMD_CMDINDEX_MASK (0x3F)
  SDHC COMMAND COMMAND INDEX MASK.
- #define SDHC_RESPONSE (0x10U)
  SDHC RESPONSE REG.
- #define SDHC_BUFFER (0x20U)

SDHC BUFFER REG.

- #define SDHC_PRESENT_STATE (0x24U)
  SDHC PRESENT STATE REG.
- #define SDHC_PRST_CMD_INHIBIT (0x1U)
  SDHC PRESENT STATE CMD INHIBIT BIT.
- #define SDHC_PRST_DATA_INHIBIT (0x1 << 1)
  SDHC PRESENT STATE DATA INHIBIT BIT.
- #define SDHC_PRST_DLA (0x1 << 2)
  SDHC PRESENT STATE DATA LINE ACTIVE BIT.
- #define SDHC_PRST_RETUNE_REQ (0x1 << 3)
  SDHC PRESENT STATE RETUNE REQUEST BIT.
- #define SDHC_PRST_WR_TRANS_A (0x1 << 8)
  SDHC PRESENT STATE WRITE TRANSFER ACTIVE BIT.
- #define SDHC_PRST_RD_TRANS_A (0x1 << 9)
  SDHC PRESENT STATE READ TRANSFER ACTIVE BIT.
- #define SDHC_PRST_BUFF_WR (0x1 << 10)
  SDHC PRESENT STATE BUFFER WRITE ENABLE BIT.
- #define SDHC_PRST_BUFF_RD (0x1 << 11)
  SDHC PRESENT STATE BUFFER READ ENABLE BIT.
- #define SDHC_PRST_CARD_INSERTED (0x1 << 16)
  SDHC PRESENT STATE CARD INSERTED BIT.
- #define SDHC_PRST_CSS (0x1 << 17)
  SDHC PRESENT STATE CARD STATE STABLE BIT.
- #define SDHC_PRST_CD_LVL (0x1 << 18)
  SDHC PRESENT STATE CARD DETECT PIN LEVEL BIT.
- #define SDHC_PRST_WP_LVL (0x1 << 19)
  SDHC PRESENT STATE WRITE PROTECT PIN LEVEL BIT.
- #define SDHC_PRST_DLSL_0_3_LSF (20U)
  SDHC PRESENT STATE DAT[3:0] LINE LEVEL SHIFT.
- #define SDHC_PRST_DLSL_0_3_MASK (0x0F000000U)
  SDHC PRESENT STATE DAT[3:0] LINE LEVEL MASK.
- #define SDHC_PRST_CMD_LVL (0x1 << 24)
  SDHC PRESENT STATE CMD LINE LEVEL BIT.
- #define SDHC_HOST_CONTROL1 (0x28U)
  SDHC HOST CONTROL1 REG.
- #define SDHC_CTRL_LED (0x01U)
  SDHC HOST CONTROL1 LED CONTROL BIT.
- #define SDHC_CTRL_4BIT (0x02U)
  SDHC HOST CONTROL1 DATA TRANSFER WIDTH BIT.
- #define SDHC_CTRL_HISPD (0x04U)
  SDHC HOST CONTROL1 HIGH SPEED ENABLE BIT.
- #define SDHC_CTRL_DMA_LSF (0x3U)
  SDHC HOST CONTROL1 DMA SELECT SHIFT.
- #define SDHC_CTRL_DMA_MASK (0x18U)
  SDHC HOST CONTROL1 DMA SELECT MASK.
- #define SDHC_CTRL_DMA_SDMA (0x0U)
  SDHC HOST CONTROL1 DMA SELECT SDMA.
- #define SDHC_CTRL_DMA_ADMA32 (0x2U)
  SDHC HOST CONTROL1 DMA SELECT ADMA32.
- #define SDHC_CTRL_DMA_ADMA64 (0x3U)
  SDHC HOST CONTROL1 DMA SELECT ADMA64.

- #define SDHC_CTRL_8BIT (0x20U)
  SDHC HOST CONTROL1 EXTENDED DATA TRANSFER WIDTH BIT.
- #define SDHC_CTRL_CD_TEST_LVL (0x40U)
  SDHC HOST CONTROL1 CARD DETECT TEST LEVEL BIT.
- #define SDHC_CTRL_CD_SSELECT (0x80U)
  SDHC HOST CONTROL1 CARD DETECT SIGNAL SELECTION BIT.
- #define SDHC_POWER_CONTROL (0x29U)
  SDHC POWER CONTROL REG.
- #define SDHC_POWER_ON (0x01U)
  SDHC POWER CONTROL SD BUS POWER.
- #define SDHC_POWER_180 (0x0A)
  SDHC POWER CONTROL SD BUS POWER 1.8V.
- #define SDHC_POWER_300 (0x0C)
  SDHC POWER CONTROL SD BUS POWER 3.0V.
- #define SDHC_POWER_330 (0x0E)
  SDHC POWER CONTROL SD BUS POWER 3.3V.
- #define SDHC_BLOCK_GAP_CTRL (0x2A)
  SDHC BLOCK GAP CONTROL REG.
- #define SDHC_BGCTRL_STPATGAPREQ (0x01U)
  SDHC BLOCK GAP CONTROL STOP AT BLOCK GAP BIT.
- #define SDHC_BGCTRL_CNTNREQ (0x02U)
  SDHC BLOCK GAP CONTROL CONTINUE REQUEST BIT.
- #define SDHC_BGCTRL_READWAIT (0x04U)
  SDHC BLOCK GAP CONTROL READ WAIT CONTROL BIT.
- #define SDHC_BGCTRL_INTRATGAP (0x08U)
  SDHC BLOCK GAP CONTROL INTERRUPT AT BLOCK GAP BIT.
- #define SDHC_WAKEUP_CONTROL (0x2B)
  SDHC WAKEUP CONTROL REG.
- #define SDHC_WAKE_ON_INT (0x01U)
  SDHC WAKEUP CONTROL WAKEUP ON CARD INTERRUPT BIT.
- #define SDHC_WAKE_ON_INSERT (0x02U)
  SDHC WAKEUP CONTROL WAKEUP ON CARD INSERTION BIT.
- #define SDHC_WAKE_ON_REMOVE (0x04U)
  SDHC WAKEUP CONTROL WAKEUP ON CARD REMOVAL BIT.
- #define SDHC_CLOCK_CONTROL (0x2C)
  SDHC CLOCK CONTROL REG.
- #define SDHC_CLK_INTCLK_EN (0x0001U)
  SDHC CLOCK CONTROL INTERNAL CLOCK ENABLE BIT.
- #define SDHC_CLK_INTCLK_STB (0x0002U)
  SDHC CLOCK CONTROL INTERNAL CLOCK STABLE BIT.
- #define SDHC_CLK_SDCLK_EN (0x0004U)
  SDHC CLOCK CONTROL SD CLOCK ENABLE BIT.
- #define SDHC_CLK_CLKGEN_PRG_SEL (0x0020U)
  SDHC CLOCK CONTROL CLOCK GENERATOR SELECTOR BIT.
- #define SDHC_CLK_FREQ_U_LSF (6U)
  SDHC CLOCK CONTROL UPPER BITS OF FREQUENCY SELECTOR SHIFT.
- #define SDHC_CLK_FREQ_U_MASK (0x00C0U)
  SDHC CLOCK CONTROL UPPER BITS OF FREQUENCY SELECTOR MASK.
- #define SDHC_CLK_FREQ_SEL_LSF (8U)
  SDHC CLOCK CONTROL FREQUENCY SELECTOR SHIFT.
- #define SDHC_CLK_FREQ_SEL_MASK (0xFF00U)

SDHC CLOCK CONTROL FREQUENCY SELECTOR MASK.
- #define SDHC_TIMEOUT_CONTROL (0x2E)
  SDHC TIMEOUT CONTROL REG.
- #define SDHC_SOFTWARE_RESET (0x2F)
  SDHC SOFTWARE RESET REG.
- #define SDHC_RESET_ALL (0x01U)
  SDHC SOFTWARE RESET RESET FOR ALL.
- #define SDHC_RESET_CMD (0x02U)
  SDHC SOFTWARE RESET RESET FOR CMD LINE.
- #define SDHC_RESET_DATA (0x04U)
  SDHC SOFTWARE RESET RESET FOR DATA LINE.
- #define SDHC_INT_STATUS (0x30U)
  SDHC NORMAL INTERRUPT STATUS REG.
- #define SDHC_INT_ENABLE (0x34U)
  SDHC NORMAL INTERRUPT STATUS ENABLE REG.
- #define SDHC_SIGNAL_ENABLE (0x38U)
  SDHC NORMAL INTERRUPT SIGNAL REG.
- #define SDHC_INT_CMD_DONE (0x1U << 0)
  SDHC NORMAL INTERRUPT CMD COMPLETE EVENT BIT.
- #define SDHC_INT_TRANSFER_DONE (0x1U << 1)
  SDHC NORMAL INTERRUPT TRANSFER COMPLETE EVENT BIT.
- #define SDHC_INT_BLKGAP_EVENT (0x1U << 2)
  SDHC NORMAL INTERRUPT BLOCK GAP EVENT BIT.
- #define SDHC_INT_DMA (0x1U << 3)
  SDHC NORMAL INTERRUPT DMA EVENT BIT.
- #define SDHC_INT_WBUF_READY (0x1U << 4)
  SDHC NORMAL INTERRUPT WRITE BUFFER READY EVENT BIT.
- #define SDHC_INT_RBUF_READY (0x1U << 5)
  SDHC NORMAL INTERRUPT READ BUFFER READY EVENT BIT.
- #define SDHC_INT_CARD_INSERT (0x1U << 6)
  SDHC NORMAL INTERRUPT CARD INSERTION EVENT BIT.
- #define SDHC_INT_CARD_REMOVE (0x1U << 7)
  SDHC NORMAL INTERRUPT CARD REMOVAL EVENT BIT.
- #define SDHC_INT_CARD_INTR (0x1U << 8)
  SDHC NORMAL INTERRUPT CARD INTERRUPT BIT.
- #define SDHC_INT_INT_A (0x1U << 9)
  SDHC NORMAL INTERRUPT INT_A EVENT BIT.
- #define SDHC_INT_INT_B (0x1U << 10)
  SDHC NORMAL INTERRUPT INT_B EVENT BIT.
- #define SDHC_INT_INT_C (0x1U << 11)
  SDHC NORMAL INTERRUPT INT_C EVENT BIT.
- #define SDHC_INT_RETUNING (0x1U << 12)
  SDHC NORMAL INTERRUPT RETUNING EVENT BIT.
- #define SDHC_INT_ERROR_INTR (0x1U << 15)
  SDHC NORMAL INTERRUPT ERROR INTERRUPT BIT.
- #define SDHC_INT_E_CMD_TIMEOUT (0x1U << 16)
  SDHC NORMAL INTERRUPT CMD TIMEOUT ERROR BIT.
- #define SDHC_INT_E_CMD_CRC (0x1U << 17)
  SDHC NORMAL INTERRUPT CMD CRC ERROR BIT.
- #define SDHC_INT_E_CMD_END_BIT (0x1U << 18)
  SDHC NORMAL INTERRUPT CMD INDEX ERROR BIT.

- #define SDHC_INT_E_CMD_INDEX (0x1U << 19)
  SDHC NORMAL INTERRUPT CMD END BIT ERROR BIT.
- #define SDHC_INT_E_DATA_TIMEOUT (0x1U << 20)
  SDHC NORMAL INTERRUPT DATA TIMEOUT ERROR BIT.
- #define SDHC_INT_E_DATA_CRC (0x1U << 21)
  SDHC NORMAL INTERRUPT DATA CRC ERROR BIT.
- #define SDHC_INT_E_DATA_END_BIT (0x1U << 22)
  SDHC NORMAL INTERRUPT DATA END BIT ERROR BIT.
- #define SDHC_INT_E_CUR_LIMIT (0x1U << 23)
  SDHC NORMAL INTERRUPT CURRENT LIMIT ERROR BIT.
- #define SDHC_INT_E_AUTOCMD12 (0x1U << 24)
  SDHC NORMAL INTERRUPT AUTO CMD12 ERROR BIT.
- #define SDHC_INT_E_ADMA (0x1U << 25)
  SDHC NORMAL INTERRUPT ADMA ERROR BIT.
- #define SDHC_INT_E_TUNING (0x1U << 26)
  SDHC NORMAL INTERRUPT TUNING ERROR BIT.
- #define SDHC_ACMD12_ERROR (0x3CU)
  SDHC AUTO CMD12 ERROR REG.
- #define SDHC_HOST_CONTROL2 (0x3EU)
  SDHC HOST CONTROL2 REG.
- #define SDHC_CTRL2_UHS_MASK (0x0007U)
  SDHC HOST CONTROL2 UHS MODE MASK.
- #define SDHC_CTRL2_UHS_SDR12 (0x0000U)
  SDHC HOST CONTROL2 UHS-I SDR12.
- #define SDHC_CTRL2_UHS_SDR25 (0x0001U)
  SDHC HOST CONTROL2 UHS-I SDR25.
- #define SDHC_CTRL2_UHS_SDR50 (0x0002U)
  SDHC HOST CONTROL2 UHS-I SDR50.
- #define SDHC_CTRL2_UHS_SDR104 (0x0003U)
  SDHC HOST CONTROL2 UHS-I SDR104.
- #define SDHC_CTRL2_UHS_DDR50 (0x0004U)
  SDHC HOST CONTROL2 UHS-I DDR50.
- #define SDHC_CTRL2_HS_SDR200 (0x0005U)
  SDHC HOST CONTROL2 HS SDR2000.
- #define SDHC_CTRL2_VDD_180 (0x0008U)
  SDHC HOST CONTROL2 1.8V SINGALING ENABLE.
- #define SDHC_CTRL2_DRV_TYPE_MASK (0x0030U)
  SDHC HOST CONTROL2 DRIVE MASK.
- #define SDHC_CTRL2_DRV_TYPE_B (0x0000U)
  SDHC HOST CONTROL2 DRIVE TYPE B.
- #define SDHC_CTRL2_DRV_TYPE_A (0x0010U)
  SDHC HOST CONTROL2 DRIVE TYPE A.
- #define SDHC_CTRL2_DRV_TYPE_C (0x0020U)
  SDHC HOST CONTROL2 DRIVE TYPE C.
- #define SDHC_CTRL2_DRV_TYPE_D (0x0030U)
  SDHC HOST CONTROL2 DRIVE TYPE D.
- #define SDHC_CTRL2_EXEC_TUNING (0x0040U)
  SDHC HOST CONTROL2 EXECUTE TUNING.
- #define SDHC_CTRL2_TUNED_CLK (0x0080U)
  SDHC HOST CONTROL2 SAMPLING CLOCK SELECT.
- #define SDHC_CTRL2_ASNYC_INTR_EN (0x4000U)

SDHC HOST CONTROL2 ASYNC INTERRUPT ENABLE.

- #define SDHC_CTRL2_PRESET_VAL_EN (0x8000U)
  SDHC HOST CONTROL2 PRESET VALUE ENABLE.
- #define SDHC_HOST_CAPABILITIES (0x40U)
  SDHC CAPABILITIES REG.
- #define SDHC_HCAP_TOCLKFREQ_MASK (0x0000003F)
  SDHC CAPABILITIES TIMEOUT CLOCK FREQUENCY.
- #define SDHC_HCAP_TOCKLUINT_MHZ (0x00000080U)
  SDHC CAPABILITIES TIMEOUT CLOCK UNIT.
- #define SDHC_HCAP_CLK_BASE_MASK (0x00003F00U)
  SDHC CAPABILITIES BASE CLOCK FREQUENCY FOR SD CLOCK MASK.
- #define SDHC_HCAP_MAX_BLK_LSF (16U)
  SDHC CAPABILITIES MAX BLOCK LENGTH SHIFT.
- #define SDHC_HCAP_MAX_BLK_MASK (0x00030000U)
  SDHC CAPABILITIES MAX BLOCK LENGTH MASK.
- #define SDHC_HCAP_MAXBLK_512 (0x0U)
  SDHC CAPABILITIES MAX BLOCK LENGTH 512B.
- #define SDHC_HCAP_MAXBLK_1024 (0x1U)
  SDHC CAPABILITIES MAX BLOCK LENGTH 1024B.
- #define SDHC_HCAP_MAXBLK_2048 (0x2U)
  SDHC CAPABILITIES MAX BLOCK LENGTH 2048B.
- #define SDHC_HCAP_SUPPORT_8BIT (0x00040000U)
  SDHC CAPABILITIES SUPPORT 8 BIT.
- #define SDHC_HCAP_SUPPORT_ADMA2 (0x00080000U)
  SDHC CAPABILITIES SUPPORT ADMA2.
- #define SDHC_HCAP_SUPPORT_ADMA1 (0x00100000U)
  SDHC CAPABILITIES SUPPORT ADMA1.
- #define SDHC_HCAP_SUPPORT_HISPD (0x00200000U)
  SDHC CAPABILITIES SUPPORT HIGH SPEED.
- #define SDHC_HCAP_SUPPORT_SDMA (0x00400000U)
  SDHC CAPABILITIES SUPPORT SDMA.
- #define SDHC_HCAP_SUPPORT_SUSPEND (0x00800000U)
  SDHC CAPABILITIES SUPPORT SUSPEND RESUME.
- #define SDHC_HCAP_SUPPORT_V330 (0x01000000U)
  SDHC CAPABILITIES SUPPORT 3.3V.
- #define SDHC_HCAP_SUPPORT_V300 (0x02000000U)
  SDHC CAPABILITIES SUPPORT 3.0V.
- #define SDHC_HCAP_SUPPORT_V180 (0x04000000U)
  SDHC CAPABILITIES SUPPORT 1.8V.
- #define SDHC_HCAP_SUPPORT_64BIT (0x10000000U)
  SDHC CAPABILITIES SUPPORT 64-BIT.
- #define SDHC_HCAP_SUPPORT_ASYNC (0x20000000U)
  SDHC CAPABILITIES SUPPORT ASYNC INTERRUPT.
- #define SDHC_HCAP_SLOT_TYPE_LSF (30U)
  SDHC CAPABILITIES SLOT TYPE SHIFT.
- #define SDHC_HCAP_SLOT_TYPE_MASK (0xC0000000U)
  SDHC CAPABILITIES SLOT TYPE MASK.
- #define SDHC_HCAP_SLOT_REMOVABLE (0x0U)
  SDHC CAPABILITIES SLOT TYPE REMOVABLE.
- #define SDHC_HCAP_SLOT_EMBEDDED (0x1U)
  SDHC CAPABILITIES SLOT TYPE EMBEDDED SLOT FOR ONE DEVICE.

- #define SDHC_HCAP_SLOT_SHARED (0x2U)
  SDHC CAPABILITIES SLOT TYPE SHARED BUS SLOT.
- #define SDHC_HOST_CAPABILITIES_1 (0x44U)
  SDHC CAPABILITIES1 REG.
- #define SDHC_HCAP_SUPORT_SDR50 (0x00000001U)
  SDHC CAPABILITIES1 SUPPORT SDR50.
- #define SDHC_HCAP_SUPORT_SDR104 (0x00000002U)
  SDHC CAPABILITIES1 SUPPORT SDR104.
- #define SDHC_HCAP_SUPORT_DDR50 (0x00000004U)
  SDHC CAPABILITIES1 SUPPORT DDR50.
- #define SDHC_HCAP_DRIVER_TYPE_A (0x00000010U)
  SDHC CAPABILITIES1 SUPPORT DRIVER TYPE A.
- #define SDHC_HCAP_DRIVER_TYPE_C (0x00000020U)
  SDHC CAPABILITIES1 SUPPORT DRIVER TYPE C.
- #define SDHC_HCAP_DRIVER_TYPE_D (0x00000040U)
  SDHC CAPABILITIES1 SUPPORT DRIVER TYPE D.
- #define SDHC_HCAP_RT_TMCNT_LSF (8U)
  SDHC CAPABILITIES1 TIMER COUNT FOR RETUNING SHIFT.
- #define SDHC_HCAP_RT_TMCNT_MASK (0x00000F00U)
  SDHC CAPABILITIES1 TIMER COUNT FOR RETUNING MASK.
- #define SDHC_HCAP_USE_SDR50_TUNE (0x00002000U)
  SDHC CAPABILITIES1 USE TUNING FOR SDR50.
- #define SDHC_HCAP_RT_MODE_LSF (14U)
  SDHC CAPABILITIES1 RETUNE MODE SHIFT.
- #define SDHC_HCAP_RT_MODE_MASK (0x0000C000U)
  SDHC CAPABILITIES1 RETUNE MODE MASK.
- #define SDHC_HCAP_CLK_MUL_LSF (16U)
  SDHC CAPABILITIES1 CLOCK MULTIPLIER SHIFT.
- #define SDHC_HCAP_CLK_MUL_MASK (0x00FF0000U)
  SDHC CAPABILITIES1 CLOCK MULTIPLIER MASK.
- #define SDHC_MAX_CURRENT (0x48U)
  SDHC MAX CURRENT REG.
- #define SDHC_MC_330_LSF (0U)
  SDHC MAX CURRENT MAXIMUM CURRENT FOR 3.3V SHIFT.
- #define SDHC_MC_330_MASK (0x0000FF)
  SDHC MAX CURRENT MAXIMUM CURRENT FOR 3.3V MASK.
- #define SDHC_MC_300_LSF (8U)
  SDHC MAX CURRENT MAXIMUM CURRENT FOR 3.0V SHIFT.
- #define SDHC_MC_300_MASK (0x00FF00U)
  SDHC MAX CURRENT MAXIMUM CURRENT FOR 3.0V MASK.
- #define SDHC_MC_180_LSF (16U)
  SDHC MAX CURRENT MAXIMUM CURRENT FOR 1.8V SHIFT.
- #define SDHC_MC_180_MASK (0xFF0000U)
  SDHC MAX CURRENT MAXIMUM CURRENT FOR 1.8V MASK.
- #define SDHC_FRC_EVENT_AUTOCMD (0x50U)
  SDHC FORCE EVENT FOR AUTOCMD REG.
- #define SDHC_FEA_E_NO_ACMD12_EXEC (0x0001U)
  SDHC FORCE EVENT AUTO CMD12 NOT EXECUTED.
- #define SDHC_FEA_E_ACMD_TIMEOUT (0x002U)
  SDHC FORCE EVENT AUTO CMD TIMEOUT ERROR.
- #define SDHC_FEA_E_ACMD_CRC (0x0004U)

SDHC FORCE EVENT AUTO CMD CRC ERROR.

- #define SDHC_FEA_E_ACMD_END (0x0008U)
  SDHC FORCE EVENT AUTO CMD END BIT ERROR.
- #define SDHC_FEA_E_ACMD_INDEX (0x0010U)
  SDHC FORCE EVENT AUTO CMD INDEX ERROR.
- #define SDHC_FEA_E_CMD_NOT_BY_ACMD12 (0x0080U)
  SDHC FORCE EVENT AUTO CMD NOT ISSUED ERROR.
- #define SDHC_FRC_EVENT_ERROR_INTR (0x52U)
  SDHC FORCE EVENT FOR ERROR REG.
- #define SDHC_FEI_E_CMD_TIMEOUT (0x0001U)
  SDHC FORCE CMD TIMEOUT ERROR.
- #define SDHC_FEI_E_CMD_CRC (0x0002U)
  SDHC FORCE CMD CRC ERROR.
- #define SDHC_FEI_E_CMD_END_BIT (0x0004U)
  SDHC FORCE CMD END BIT ERROR.
- #define SDHC_FEI_E_DATA_TIMEOUT (0x0008U)
  SDHC FORCE DATA TIMEOUT ERROR.
- #define SDHC_FEI_E_DATA_CRC (0x0010U)
  SDHC FORCE DATA CRC ERROR.
- #define SDHC_FEI_E_DATA_END_BIT (0x0020U)
  SDHC FORCE DATA END BIT ERROR.
- #define SDHC_FEI_E_CURRENT_LIMIT (0x0040U)
  SDHC FORCE CURRENT LIMIT ERROR.
- #define SDHC_FEI_E_AUTO_CMD (0x0080U)
  SDHC FORCE AUTOCMD ERROR.
- #define SDHC_FEI_E_ADMA (0x0100U)
  SDHC FORCE ADMA ERROR.
- #define SDHC_ADMA_ERROR (0x54U)
  SDHC ADMA ERROR REG.
- #define SDHC_ADMA_ADDRESS (0x58U)
  SDHC ADMA ADDRESS REG.
- #define SDHC_SLOT_INT_STATUS (0xFCU)
  SDHC SLOT INTERRUPT STATUS REG.
- #define SDHC_HOST_VERSION (0xFEU)
  SDHC HOST CONTROLLER VERSION REG.
- #define SDHC_VENDOR_VER_LSF (8U)
  SDHC HOST CONTROLLER VERSION VENDOR VERSION SHIFT.
- #define SDHC_VENDOR_VER_MASK (0xFF00U)
  SDHC HOST CONTROLLER VERSION VENDOR VERSION MASK.
- #define SDHC_SPEC_VER_LSF (0U)
  SDHC HOST CONTROLLER VERSION SPEC VERSION SHIFT.
- #define SDHC_SPEC_VER_MASK (0x00FFU)
  SDHC HOST CONTROLLER VERSION SPEC VERSION MASK.
- #define SDHC_SPEC_100 (0U)
  SDHC HOST CONTROLLER VERSION SPEC VERSION 1.00.
- #define SDHC_SPEC_200 (1U)
  SDHC HOST CONTROLLER VERSION SPEC VERSION 2.00.
- #define SDHC_SPEC_300 (2U)
  SDHC HOST CONTROLLER VERSION SPEC VERSION 3.00.

Chapter 19
System Mode Controller (SMC)

The Kinetis SDK provides both HAL and Peripheral drivers for the System Mode Controller (SMC) block of Kinetis devices.

Modules

- SMC HAL driver
  The part describes the programming interface of the SMC HAL driver.
- SMC Peripheral Driver
  The part describes the programming interface of the SMC Peripheral driver.

19.1 SMC HAL driver

The chapter describes the programming interface of the SMC HAL driver.

Data Structures

- struct smc_power_mode_protection_config_t
  Power mode protection configuration. More...

Enumerations

- enum power_mode_stat_t {
  kStatRun = 0x01,
  kStatStop = 0x02,
  kStatVlpr = 0x04,
  kStatVlpw = 0x08,
  kStatVlps = 0x10,
  kStatLls = 0x20,
  kStatVlls = 0x40,
  kStatHsrun = 0x80 }
  Power Modes in PMSTAT.
- enum power_modes_protect_t {
  kAllowHsrun,
  kAllowVlp,
  kAllowLls,
  kAllowVlls }
  Power Modes Protection.
- enum smc_run_mode_t {
  kSmcRun ,
  kSmcVlpr,
  kSmcHsrun }
  Run mode definition.
- enum smc_stop_mode_t {
  kSmcStop,
  kSmcReservedStop1,
  kSmcVlps,
  kSmcLls,
  kSmcVlls }
  Stop mode definition.
- enum smc_stop_submode_t
  VLLS/LLS stop sub mode definition.
- enum smc_lpwui_option_t {
  kSmcLpwuiEnabled,
  kSmcLpwuiDisabled }
  Low Power Wake Up on Interrupt option.

- enum smc_pstop_option_t {
  kSmcPstopStop,
  kSmcPstopStop1,
  kSmcPstopStop2 }
  Partial STOP option.
- enum smc_por_option_t {
  kSmcPorEnabled,
  kSmcPorDisabled }
  POR option.
- enum smc_lpo_option_t {
  kSmcLpoEnabled,
  kSmcLpoDisabled }
  LPO power option.
- enum smc_power_options_t {
  kSmcOptionLpwui,
  kSmcOptionPropo }
  Power mode control options.

System mode controller APIs

- void smc_hal_config_power_mode_protection (smc_power_mode_protection_config_t *protectConfig)
  Configures all power mode protection settings.
- void smc_hal_set_power_mode_protection (power_modes_protect_t protect, bool allow)
  Configures the individual power mode protection settings.
- bool smc_hal_get_power_mode_protection (power_modes_protect_t protect)
  Gets the the current power mode protection setting.
- void smc_hal_power_mode_config_run (smc_run_mode_t runMode)
  Configures the the RUN mode control setting.
- smc_run_mode_t smc_hal_power_mode_get_run_config (void)
  Gets the current RUN mode configuration setting.
- void smc_hal_power_mode_config_stop (smc_stop_mode_t stopMode)
  Configures the STOP mode control setting.
- smc_stop_mode_t smc_hal_power_mode_get_stop_config (void)
  Gets the current STOP mode control settings.
- void smc_hal_power_mode_config_stop_submode (smc_stop_submode_t stopSubMode)
  Configures the stop sub mode control setting.
- smc_stop_submode_t smc_hal_power_mode_get_stop_submode_config (void)
  Gets the current stop submode configuration settings.
- uint8_t smc_hal_get_power_mode_stat (void)
  Gets the current power mode stat.

19.1.0.6 SMC Hal Driver

Overview

The System Mode Controller (SMC) sequences the system in and out of all low-power stop and run modes.

Specifically, it monitors events to trigger transitions between the power modes while controlling the power, clocks, and memories of the system to achieve the power consumption and functionality of that mode.

Control register access APIs

- Power mode configurations register access
- VLLS mode configurations register access
- Power Mode Status access

This is an example of the SMC HAL access APIs.

```
#include "fsl_smc_hal.h"

// Enable the clock gate for specific module. (SMC_PMCTL)
uint32_t *pmctrl = HW_SMC_PMCTRL_ADDR;          // System Mode Power Control config register address
uint32_t mask = SMC_PMCTRL_RUNM_MASK;           // Run mode bit mask
uint32_t shift = SMC_PMCTRL_RUNM_SHIFT;         // Run mode bit shift
uint32_t settings = 0x2;                        // Set to Very Low Power Run mode (VLPR)

// calling smc control API to enable the clock
smc_hal_config_power_mode(pmctrl, mask, shift, settings);
```

19.1.1 Data Structure Documentation

19.1.1.1 struct smc_power_mode_protection_config_t

Data Fields

- bool vlpProt
  VLP protect.
- bool llsProt
  LLS protect.
- bool vllsProt
  VLLS protect.

19.1.2 Enumeration Type Documentation

19.1.2.1 enum power_mode_stat_t

Enumerator

kStatRun 0000_0001 - Current power mode is RUN

kStatStop 0000_0010 - Current power mode is STOP
kStatVlpr 0000_0100 - Current power mode is VLPR
kStatVlpw 0000_1000 - Current power mode is VLPW
kStatVlps 0001_0000 - Current power mode is VLPS
kStatLls 0010_0000 - Current power mode is LLS
kStatVlls 0100_0000 - Current power mode is VLLS
kStatHsrun 1000_0000 - Current power mode is HSRUN

19.1.2.2 enum power_modes_protect_t

Enumerator

kAllowHsrun Allow High Speed Run mode.
kAllowVlp Allow Very-Low-Power Modes.
kAllowLls Allow Low-Leakage Stop Mode.
kAllowVlls Allow Very-Low-Leakage Stop Mode.

19.1.2.3 enum smc_run_mode_t

Enumerator

kSmcRun normal RUN mode
kSmcVlpr Very-Low-Power RUN mode.
kSmcHsrun High Speed Run mode (HSRUN)

19.1.2.4 enum smc_stop_mode_t

Enumerator

kSmcStop Normal STOP mode.
kSmcReservedStop1 Reserved.
kSmcVlps Very-Low-Power STOP mode.
kSmcLls Low-Leakage Stop mode.
kSmcVlls Very-Low-Leakage Stop mode.

19.1.2.5 enum smc_lpwui_option_t

Enumerator

kSmcLpwuiEnabled Low Power Wake Up on Interrupt enabled.
kSmcLpwuiDisabled Low Power Wake Up on Interrupt disabled.

19.1.2.6 enum smc_pstop_option_t

Enumerator

kSmcPstopStop STOP - Normal Stop mode.
kSmcPstopStop1 Partial Stop with both system and bus clocks disabled.
kSmcPstopStop2 Partial Stop with system clock disabled and bus clock enabled.

19.1.2.7 enum smc_por_option_t

Enumerator

kSmcPorEnabled POR detect circuit is enabled in VLLS0.
kSmcPorDisabled POR detect circuit is disabled in VLLS0.

19.1.2.8 enum smc_lpo_option_t

Enumerator

kSmcLpoEnabled LPO clock is enabled in LLS/VLLSx.
kSmcLpoDisabled LPO clock is disabled in LLS/VLLSx.

19.1.2.9 enum smc_power_options_t

Enumerator

kSmcOptionLpwui Low Power Wake Up on Interrupt.
kSmcOptionPropo POR option.

19.1.3 Function Documentation

19.1.3.1 void smc_hal_config_power_mode_protection (smc_power_mode_protection_config_t * *protectConfig*)

This function configures the power mode protection settings for supported power modes in the specified chip family. The available power modes are defined in the smc_power_mode_protection_config_t. An application should provide the protect settings for all supported power modes on the chip. This should be done at an early system level initialization stage. See the reference manual for details. This register can only write once after the power reset. If the user has only a single option to set, either use this function or use the individual set function.

Parameters

protectConfig	Configurations for the supported power mode protect settings • See smc_power_mode_protection_config_t for details.

19.1.3.2 void smc_hal_set_power_mode_protection (power_modes_protect_t *protect*, bool *allow*)

This function only configures the power mode protection settings for a specified power mode on the specified chip family. The available power modes are defined in the smc_power_mode_protection_config_t. See the reference manual for details. This register can only write once after the power reset.

Parameters

protect	Power mode to set for protection
allow	Allow or not allow the power mode protection

19.1.3.3 bool smc_hal_get_power_mode_protection (power_modes_protect_t *protect*)

This function gets the current power mode protection settings for a specified power mode.

Parameters

protect	Power mode to set for protection

Returns

state Status of the protection setting
- true: Allowed
- false: Not allowed

19.1.3.4 void smc_hal_power_mode_config_run (smc_run_mode_t *runMode*)

This function sets the run mode settings, for example, normal run mode, very lower power run mode, etc. See the smc_run_mode_t for supported run mode on the chip family and the reference manual for details about the run mode.

Parameters

runMode	Run mode setting defined in smc_run_mode_t

19.1.3.5 smc_run_mode_t smc_hal_power_mode_get_run_config (void)

This function gets the run mode settings. See the smc_run_mode_t for a supported run mode on the chip family and the reference manual for details about the run mode.

Returns

setting Run mode configuration setting

19.1.3.6 void smc_hal_power_mode_config_stop (smc_stop_mode_t *stopMode*)

This function sets the stop mode settings, for example, normal stop mode, very lower power stop mode, etc. See the smc_stop_mode_t for supported stop mode on the chip family and the reference manual for details about the stop mode.

Parameters

stopMode	Stop mode defined in smc_stop_mode_t

19.1.3.7 smc_stop_mode_t smc_hal_power_mode_get_stop_config (void)

This function gets the stop mode settings, for example, normal stop mode, very lower power stop mode, etc. See the smc_stop_mode_t for supported stop mode on the chip family and the reference manual for details about the stop mode.

Returns

setting Current stop mode configuration setting

19.1.3.8 void smc_hal_power_mode_config_stop_submode (smc_stop_submode_t *stopSubMode*)

This function sets the stop submode settings. Some of the stop mode further supports submodes. See the smc_stop_submode_t for supported stop submodes and the reference manual for details about the submodes for a specific stop mode.

Parameters

stopSubMode	Stop submode setting defined in smc_stop_submode_t

Returns

none

19.1.3.9 smc_stop_submode_t smc_hal_power_mode_get_stop_submode_config (void)

This function gets the stop submode settings. Some of the stop mode further support submodes. See the smc_stop_submode_t for supported stop submodes and the reference manual for details about the submode for a specific stop mode.

Returns

setting Current stop submode setting

19.1.3.10 uint8_t smc_hal_get_power_mode_stat (void)

This function returns the current power mode stat. Once application switches the power mode, it should always check the stat to check whether it runs into the specified mode or not. An application should check this mode before switching to a different mode. The system requires that only certain modes can switch to other specific modes. See the reference manual for details and the _power_mode_stat for information about the power stat.

Returns

stat Current power mode stat

19.2 SMC Peripheral Driver

The chapter describes the programming interface of the SMC Peripheral driver.

Data Structures

- struct smc_power_mode_config_t
  Power mode control configuration used for calling the smc_set_power_mode API. More...

Enumerations

- enum power_modes_t
  Power Modes.
- enum smc_manager_error_code_t {
  kSmcManagerSuccess,
  kSmcManagerNoSuchModeName,
  kSmcManagerAlreadyInTheState,
  kSmcManagerFailed }
  Error code definition for the system mode controller manager APIs.

Power Mode Configuration

- smc_manager_error_code_t smc_set_power_mode (const smc_power_mode_config_t *power-ModeConfig)
  Configures the power mode.

19.2.0.11 SMC Peripheral Driver

Overview

The System Mode Controller (SMC) manager provides wrapper functions to access SMC HAL layer APIs.

It also provides a set of functions to configure the power mode protection, the power mode, and other configuration settings.

Power Mode Configuration APIs

- Power Mode Protection configurations APIs Allow the Very Low Power mode / Read the current mode Allow the Low Leakage Stop mode / Read the current mode Allow the Very Low Leakage Stop mode / Read the current mode
- Power Mode Control configurations APIs
- VLLS Mode Control configurations APIs
- Power Mode Status APIs

This is an example of the SMC manager APIs.

```
#include "fsl_smc_manager.h"

// set power mode to normal Run mode
power_mode_config(kRun, false);

// set power mode to VLPS Stop mode
power_mode_config(kVlps, false);
```

19.2.1 Data Structure Documentation

19.2.1.1 struct smc_power_mode_config_t

Data Fields

- power_modes_t powerModeName
  Power mode(enum), see power_modes_t.
- smc_stop_submode_t stopSubMode
  Stop submode(enum), see smc_stop_submode_t.
- bool lpwuiOption
  If LPWUI option is needed.
- smc_lpwui_option_t lpwuiOptionValue
  LPWUI option(enum), see smc_lpwui_option_t.
- bool porOption
  If POR option is needed.
- smc_por_option_t porOptionValue
  POR option(enum), see smc_por_option_t.

19.2.2 Enumeration Type Documentation

19.2.2.1 enum smc_manager_error_code_t

Enumerator

kSmcManagerSuccess Success.
kSmcManagerNoSuchModeName Cannot find the mode name specified.
kSmcManagerAlreadyInTheState Already in the required state.
kSmcManagerFailed Unknown error, operation failed.

19.2.3 Function Documentation

19.2.3.1 smc_manager_error_code_t smc_set_power_mode (const smc_power_mode_config_t * *powerModeConfig*)

This function configures the power mode control for both run, stop, and stop sub mode if needed. Also it configures the power options for a specific power mode. An application should follow the proper procedure

to configure and switch power modes between different run and stop modes. For proper procedures and supported power modes, see an appropriate chip reference manual. See the smc_power_mode_config_t for required parameters to configure the power mode and the supported options. Other options may need to be individually configured through the HAL driver. See the HAL driver header file for details.

Parameters

powerModeConfig	Power mode configuration structure smc_power_mode_config_t

Chapter 20
Soundcard (SND)

The Kinetis SDK provides both HAL and Peripheral drivers for the Soundcard (SND) block of Kinetis devices.

Data Structures

- struct snd_state_t
  Soundcard status includes the information which the application can see. More...
- struct audio_ctrl_operation_t
  The operations of an audio controller, for example SAI, SSI and so on. More...
- struct audio_codec_operation_t
  Audio codec operation structure. More...
- struct audio_controller_t
  The definition of the audio device which may be a controller. More...
- struct audio_codec_t
  The Codec structure. More...
- struct audio_buffer_t
  Audio buffer structure. More...
- struct sound_card_t
  A sound card includes the audio controller and a Codec. More...

Macros

- #define AUDIO_CONTROLLER AUDIO_CONTROLLER_SAI
  Define audio controller sai.
- #define AUDIO_CODEC AUDIO_CODEC_SGTL5000
  Define audio codec sgtl5000.
- #define AUDIO_BUFFER_BLOCK_SIZE 128
  Buffer block size setting.
- #define AUDIO_BUFFER_BLOCK 2
  Buffer block number setting.
- #define AUDIO_TX 1
  Audio transfer direction Tx.
- #define AUDIO_RX 0
  Audio transfer direction Rx.

Enumerations

- enum snd_status_t {
  kStatus_SND_Success = 0U,
  kStatus_SND_Fail = 1U,
  kStatus_SND_DmaFail = 2U,
  kStatus_SND_CtrlFail = 3U,
  kStatus_SND_CodecFail = 4U,

kStatus_SND_BufferAllocateFail = 5U }
Soundcard return status.

Functions

- snd_status_t snd_init (sound_card_t *card, void *ctrl_config, void *codec_config)
  Initializes the Soundcard.
- snd_status_t snd_deinit (sound_card_t *card)
  Deinitializes the sound card instance.
- snd_status_t snd_data_format_configure (sound_card_t *card, ctrl_data_format_t *format)
  Configures the audio data format running in the Soundcard.
- uint32_t snd_update_tx_status (sound_card_t *card, uint32_t len)
  Updates the status of the TX.
- uint32_t snd_update_rx_status (sound_card_t *card, uint32_t len)
  Updates status of the RX.
- void snd_get_status (snd_state_t *status, sound_card_t *card)
  Gets the status of the Soundcard.
- void snd_start_tx (sound_card_t *card)
  Starts the Soundcard TX process.
- void snd_start_rx (sound_card_t *card)
  Starts the Soundcard RX process.
- static void snd_stop_tx (sound_card_t *card)
  Stops Soundcard TX process.
- static void snd_stop_rx (sound_card_t *card)
  Stops the Soundcard RX process.
- void snd_wait_event (sound_card_t *card)
  Waits for the semaphore of the write/read data from the internal buffer.

20.0.4 Soundcard Driver

Overview

The Soundcard is used to handle both audio controller and audio Codec. It provides an easy way to initialize, configure and handle(read/write) the Soundcard.

Initialization

A Soundcard includes a controller and a Codec (SAI + sgtl5000). Application needs to prepare the configuration structure for SAI and SGTL5000, and then call the snd_init() to finish the initialization.

Configuration

The configuration includes the static configuration and dynamic configuration. Static configuration is configuring the SAI features. Dynamic configuration is configuring the audio data format (sample rate and bit depth).

Call diagram

To use the SAI driver, user should follow these steps:

1. Initialize the Soundcard by calling the snd_init() function.
2. Configure the audio data format information of the Soundcard by calling the snd_data_format_configure() function.
3. Update the status of the Soundcard after copying into/output from the buffer by calling the snd_update_status() function.
4. Start the TX/RX by calling the snd_start_tx() or the snd_start_rx() function.
5. Stop by calling the snd_stop_tx() or snd_stop_rx() functions.
6. Close the devices by calling the snd_deinit() function.

This is an example code to initialize and configure the SAI driver in the DMA mode:

```
    sound_card_t g_card;
    static audio_data_format_t *format;
    static sai_config_t tx_config;

    void snd_card_config(void)
    {
        /* SAI configuration */
        tx_config.bus_type = kSaiBusI2SLeft;
        tx_config.channel = 0;
        tx_config.slave_master = kSaiMaster;
        tx_config.sync_mode = kSaiModeAsync;
        tx_config.bclk_source = kSaiBclkSourceMclkDiv;
        tx_config.mclk_source = kSaiMclkSourceSysclk;
        tx_config.mclk_divide_enable = true;

        sai_handler_t *tx_handler = (sai_handler_t *)
      mem_allocate_zero(sizeof(sai_handler_t));
        g_card.controller.handler = tx_handler;
        tx_handler->direction = AUDIO_TX;
        tx_handler->instance = 0;
        tx_handler->fifo_channel = 0;
        g_card.controller.ops = &g_sai_ops;
#if USEDMA
        g_card.controller.dma_source = kDmaRequestMux0I2S0Tx;
#endif
        sgtl_handler_t *codec_handler = (sgtl_handler_t *)
      mem_allocate_zero(sizeof(sgtl_handler_t));
        g_card.codec.handler = codec_handler;
        g_card.codec.ops = &g_sgtl_ops;
    }


    // Initialize the Soundcard basically
    snd_init(&g_card);

    // Configure the audio data format for TX
    snd_data_format_configure(&g_card,format);

    // Copy data to SAI buffer
    snd_wait_event(&g_card);
    snd_get_status(&tx_status, &g_card);
    memcpy(tx_status.input_address, pData, tx_status.size);
    snd_update_tx_status(&g_card, tx_status.size);
```

20.1 Data Structure Documentation

20.1.1 struct snd_state_t

This structure is the interface between the driver and the application. The application can get the information where and when to input/output data.

Data Fields

- uint32_t size
  The size of a block.
- uint32_t empty_block
  How many blocks are empty.
- uint32_t full_block
  How many blocks are full.
- uint8_t * input_address
  The input address.
- uint8_t * output_address
  The output address.

20.1.2 struct audio_ctrl_operation_t

The operation includes the basic initialize, configure, send, receive and so on.

Data Fields

- ctrl_status_t(* ctrl_init)(ctrl_handler_t *param, ctrl_config_t *config)
  Initializes the controller.
- ctrl_status_t(* ctrl_deinit)(ctrl_handler_t *param)
  Deinitializes the controller.
- ctrl_status_t(* ctrl_config_data_format)(ctrl_handler_t *param, ctrl_data_format_t *format)
  Configures the audio data format.
- void(* ctrl_start_write)(ctrl_handler_t *param)
  Used in a start transfer or a resume transfer.
- void(* ctrl_start_read)(ctrl_handler_t *param)
  Used in a start receive or a resume receive.
- void(* ctrl_stop_write)(ctrl_handler_t *param)
  Used in the stop transfer.
- void(* ctrl_stop_read)(ctrl_handler_t *param)
  Used in the stop receive.
- uint32_t(* ctrl_send_data)(ctrl_handler_t *param, uint8_t *addr, uint32_t len)
  Sends data function.
- uint32_t(* ctrl_receive_data)(ctrl_handler_t *param, uint8_t *addr, uint32_t len)
  Receives data.
- void(* ctrl_register_callback)(ctrl_handler_t *param, ctrl_callback_t callback, void *callback_param)
  Registers a callback function.
- uint32_t *(* ctrl_get_fifo_address)(ctrl_handler_t *param)
  Gets the FIFO address.

20.1.2.0.0.33 Field Documentation

20.1.2.0.0.33.1 ctrl_status_t(∗ audio_ctrl_operation_t::ctrl_init)(ctrl_handler_t ∗param, ctrl_config_t ∗config)

20.1.2.0.0.33.2 ctrl_status_t(∗ audio_ctrl_operation_t::ctrl_deinit)(ctrl_handler_t ∗param)

20.1.2.0.0.33.3 ctrl_status_t(∗ audio_ctrl_operation_t::ctrl_config_data_format)(ctrl_handler_t ∗param, ctrl_data_format_t ∗format)

20.1.2.0.0.33.4 void(∗ audio_ctrl_operation_t::ctrl_register_callback)(ctrl_handler_t ∗param, ctrl_callback_t callback, void ∗callback_param)

20.1.3 struct audio_codec_operation_t

Data Fields

- codec_status_t(∗ codec_init)(codec_handler_t ∗param, void ∗config)
  Codec initialize function.
- codec_status_t(∗ codec_deinit)(codec_handler_t ∗param)
  Codec deinitialize function.
- codec_status_t(∗ codec_config_data_format)(codec_handler_t ∗param, uint32_t mclk, uint32_t sample_rate, uint8_t bits)
  Configures data format.

20.1.3.0.0.34 Field Documentation

20.1.3.0.0.34.1 codec_status_t(∗ audio_codec_operation_t::codec_config_data_format)(codec_handler_t ∗param, uint32_t mclk, uint32_t sample_rate, uint8_t bits)

20.1.4 struct audio_controller_t

Data Fields

- audio_ctrl_operation_t ∗ ops
  Operations including the initialize, configure, etc.

20.1.4.0.0.35 Field Documentation

20.1.4.0.0.35.1 audio_ctrl_operation_t∗ audio_controller_t::ops

20.1.5 struct audio_codec_t

Data Fields

- void ∗ handler

Codec instance.

- audio_codec_operation_t * ops
  Operations.

20.1.5.0.0.36 Field Documentation

20.1.5.0.0.36.1 audio_codec_operation_t* audio_codec_t::ops

20.1.6 struct audio_buffer_t

Data Fields

- uint8_t * buff
  Buffer address.
- uint8_t blocks
  Block number of the buffer.
- uint16_t size
  The size of a block.
- uint32_t requested
  The request data number to transfer.
- uint32_t queued
  Data which is in buffer, but not processed.
- uint32_t processed
  Data which is put into the FIFO.
- uint8_t input_index
  Buffer input block index.
- uint8_t output_index
  Buffer output block index.
- uint8_t * input_curbuff
  Buffer input address.
- uint8_t * output_curbuff
  Buffer output address.
- uint32_t empty_block
  Empty block number.
- uint32_t full_block
  Full block numbers.
- uint32_t fifo_error
  FIFO error numbers.
- uint32_t buffer_error
  Buffer error numbers.
- sync_object_t sem
  Semaphores to control the data flow.
- bool first_io
  Means the first time the transfer.

20.1.6.0.0.37 Field Documentation

20.1.6.0.0.37.1 uint8_t audio_buffer_t::blocks

20.1.6.0.0.37.2 uint32_t audio_buffer_t::queued

20.1.6.0.0.37.3 uint32_t audio_buffer_t::processed

This is used to judge if the SAI is under run.

20.1.6.0.0.37.4 uint8_t audio_buffer_t::input_index

20.1.6.0.0.37.5 uint8_t audio_buffer_t::output_index

20.1.6.0.0.37.6 uint8_t* audio_buffer_t::input_curbuff

20.1.6.0.0.37.7 uint8_t* audio_buffer_t::output_curbuff

20.1.6.0.0.37.8 uint32_t audio_buffer_t::empty_block

20.1.6.0.0.37.9 uint32_t audio_buffer_t::full_block

20.1.6.0.0.37.10 uint32_t audio_buffer_t::fifo_error

20.1.6.0.0.37.11 uint32_t audio_buffer_t::buffer_error

20.1.6.0.0.37.12 sync_object_t audio_buffer_t::sem

20.1.7 struct sound_card_t

Data Fields

- audio_controller_t controller
  Controller.
- audio_codec_t codec
  Codec.
- audio_buffer_t buffer
  Audio buffer managed by Soundcard.
- bool direction
  TX or RX.

20.1.7.0.0.38 Field Documentation

20.1.7.0.0.38.1 audio_buffer_t sound_card_t::buffer

20.2 Enumeration Type Documentation

20.2.1 enum snd_status_t

Enumerator

kStatus_SND_Success Execute successfully.
kStatus_SND_Fail Execute fail.
kStatus_SND_DmaFail DMA operation fail.
kStatus_SND_CtrlFail Audio controller operation fail.
kStatus_SND_CodecFail Audio codec operation fail.
kStatus_SND_BufferAllocateFail Buffer allocate failure.

20.3 Function Documentation

20.3.1 snd_status_t snd_init (sound_card_t * *card,* void * *ctrl_config,* void * *codec_config*)

The function initializes the controller and the Codec.

The function initializes the generic layer structure, for example the buffer and the status structure, then the function calls the initialize functions of the controller and the Codec.

Parameters

card	Soundcard pointer
ctrl_config	The configuration structure of the audio controller
codec_config	The configuration structure of the audio Codec

Returns

Return kStatus_SND_Success while the initialize success and kStatus_SND_fail if failed.

20.3.2 snd_status_t snd_deinit (sound_card_t * *card*)

The function calls the Codec and controller deinitialization function and frees the buffer controlled by the sound card. The function should be used at the end of the application. If the playback/record is paused, you shouldn't use the function. Instead, you should use the snd_stop_tx/snd_stop_rx instead.

Parameters

card	Soundcard pointer

Returns

Return kStatus_SND_Success while the initialize success and kStatus_SND_fail if failed.

20.3.3 snd_status_t snd_data_format_configure (sound_card_t * *card*, ctrl_data_format_t * *format*)

This function can make the application change the data format during the run time. This function cannot be called while TCSR.TE or RCSR.RE is enabled. This function can change the sample rate, bit depth(i.e. 16-bit).

Parameters

card	Soundcard pointer
format	Data format used in the sound card

Returns

Return kStatus_SND_Success while the initialize success and kStatus_SND_fail if failed.

20.3.4 uint32_t snd_update_tx_status (sound_card_t * *card*, uint32_t *len*)

This function should be called after the application copied data into the buffer provided by the Soundcard. The Soundcard does not help users copy data to the internal buffer. This operation should be done by the applications. Soundcard provides an interface ,snd_get_status(), for an application to get the information about the internal buffer status, including the starting address and empty blocks and so on.

Parameters

card	Soundcard pointer
len	Data size of the data to write

Returns

The size which has been written.

20.3.5 uint32_t snd_update_rx_status (sound_card_t * *card*, uint32_t *len*)

This function should be called after the application copied data from the buffer provided by the Soundcard. The Soundcard does not help applications to copy data from the internal buffer. This operation should be done by applications. The Soundcard provides an interface snd_get_status() for applications to get the information about the internal buffer status, including the starting address and full blocks and so on.

Parameters

card	Soundcard pointer
len	Data size of data to read

Returns

The data size which has been read.

20.3.6 void snd_get_status (snd_state_t * *status*, sound_card_t * *card*)

Each time the application wants to write/read data from the internal buffer, it should call this function to get the status of the internal buffer. This function copies data to the

Parameters

status,from	the structure. The user can get the information about where to write/read data and how much data can be read/written.
card	Soundcard pointer
status	Pointer of the audio_status_t structure
card	Soundcard pointer

20.3.7 void snd_start_tx (sound_card_t * *card*)

This function starts the TX process of the Soundcard. This function enables the DMA/interrupt request source and enables the TX and the bit clock of the TX. Note that this function can be used both in the beginning of the SAI transfer and also resume the transfer.

Parameters

card	Soundcard pointer

20.3.8 void snd_start_rx (sound_card_t * *card*)

This function starts the RX process of the Soundcard. This function is the same as the snd_start_tx. It is used in the beginning of the Soundcard RX transfer and also resume RX.

Parameters

card	Soundcard pointer

20.3.9 static void snd_stop_tx (sound_card_t * *card*) [inline], [static]

This function stops the transfer of the Soundcard TX. Note that this function does not close the audio controller. It disables the DMA/interrupt request source. Therefore, this function can be used to pause the audio play.

Parameters

card	Soundcard pointer

20.3.10 static void snd_stop_rx (sound_card_t * *card*) [inline], [static]

This function is the same as the snd_stop_tx, used to stop the RX process. This function also disables the DMA/interrupt source.

Parameters

card	Soundcard pointer

20.3.11 void snd_wait_event (sound_card_t * *card*)

Application should call this function before write/read data from the Soundcard buffer. Before application writes data to the Soundcard buffer, the buffer must have free space for the new data, or it causes data loss. This function waits for the semaphore which represents free space in the Soundcard buffer. Similarly to the reading data from the Soundcard buffer, effective data must be in the buffer. This function waits for that semaphore.

Parameters

card	Soundcard pointer

Chapter 21
Serial Peripheral Interface (SPI)

The Kinetis SDK provides both HAL and Peripheral drivers for the Serial Peripheral Interface (SPI) block of Kinetis devices.

Modules

- SPI Classes
  This part describes the SPI driver C++ classes.
- SPI HAL driver
  The part describes the programming interface of the SPI HAL driver.
- SPI Master Peripheral Driver
  The part describes the programming interface of the SPI master mode Peripheral driver.
- SPI Slave Peripheral Driver
  The part describes the programming interface of the SPI slave mode Peripheral driver.
- Shared SPI Types
  This part describes SPI driver shared types.

21.1 SPI HAL driver

The chapter describes the programming interface of the SPI HAL driver.

Data Structures

- struct spi_config_t
  SPI hardware configuration settings. More...

Enumerations

- enum spi_status_t { ,
  kStatus_SPI_SlaveTxUnderrun,
  kStatus_SPI_SlaveRxOverrun,
  kStatus_SPI_Timeout,
  kStatus_SPI_Busy,
  kStatus_SPI_NoTransferInProgress }
  Error codes for the SPI driver.
- enum spi_master_slave_mode_t {
  kSpiMaster = 1,
  kSpiSlave = 0 }
  SPI master or slave configuration.
- enum spi_clock_polarity_t {
  kSpiClockPolarity_ActiveHigh = 0,
  kSpiClockPolarity_ActiveLow = 1 }
  SPI clock polarity configuration.
- enum spi_clock_phase_t {
  kSpiClockPhase_FirstEdge = 0,
  kSpiClockPhase_SecondEdge = 1 }
  SPI clock phase configuration.
- enum spi_shift_direction_t {
  kSpiMsbFirst = 0,
  kSpiLsbFirst = 1 }
  SPI data shifter direction options.
- enum spi_ss_output_mode_t {
  kSpiSlaveSelect_AsGpio = 0,
  kSpiSlaveSelect_FaultInput = 2,
  kSpiSlaveSelect_AutomaticOutput = 3 }
  SPI slave select output mode options.
- enum spi_pin_mode_t {
  kSpiPinMode_Normal = 0,
  kSpiPinMode_Input = 1,
  kSpiPinMode_Output = 3 }
  SPI pin mode options.

Configuration

- void spi_hal_init (uint32_t instance, const spi_config_t *config)
  Configures the SPI peripheral.
- void spi_hal_reset (uint32_t instance)
  Restores the SPI to reset the configuration.
- static void spi_hal_enable (uint32_t instance)
  Enables the SPI peripheral.
- static void spi_hal_disable (uint32_t instance)
  Disables the SPI peripheral.
- void spi_hal_set_baud (uint32_t instance, uint32_t kbitsPerSec)
  Sets the SPI baud rate in kilobits per second.
- static void spi_hal_set_baud_divisors (uint32_t instance, uint32_t prescaleDivisor, uint32_t rateDivisor)
  Configures the baud rate divisors manually.
- static void spi_hal_set_master_slave (uint32_t instance, spi_master_slave_mode_t mode)
  Configures the SPI for master or slave.
- void spi_hal_set_slave_select_output_mode (uint32_t instance, spi_ss_output_mode_t mode)
  Sets how the slave select output operates.
- void spi_hal_set_data_format (uint32_t instance, spi_clock_polarity_t polarity, spi_clock_phase_t phase, spi_shift_direction_t direction)
  Sets the polarity, phase, and shift direction.
- void spi_hal_set_pin_mode (uint32_t instance, spi_pin_mode_t mode)
  Sets the SPI pin mode.

DMA

- void spi_hal_configure_dma (uint32_t instance, bool enableTransmit, bool enableReceive)
  Configures the transmit and receive DMA requests.

Low power

- static void spi_hal_configure_stop_in_wait_mode (uint32_t instance, bool enable)
  Enables or disables the SPI clock to stop when the CPU enters wait mode.

Interrupts

- static void spi_hal_enable_receive_and_fault_interrupt (uint32_t instance)
  Enables the receive buffer full and mode fault interrupt.
- static void spi_hal_disable_receive_and_fault_interrupt (uint32_t instance)
  Disables the receive buffer full and mode fault interrupt.
- static void spi_hal_enable_transmit_interrupt (uint32_t instance)
  Enables the transmit buffer empty interrupt.
- static void spi_hal_disable_transmit_interrupt (uint32_t instance)
  Disables the transmit buffer empty interrupt.
- static void spi_hal_enable_match_interrupt (uint32_t instance)
  Enables the match interrupt.

- static void spi_hal_disable_match_interrupt (uint32_t instance)
  Disables the match interrupt.

Status

- static bool spi_hal_is_read_buffer_full (uint32_t instance)
  Checks whether the read buffer is full.
- static bool spi_hal_is_transmit_buffer_empty (uint32_t instance)
  Checks whether the transmit buffer is empty.
- static bool spi_hal_is_mode_fault (uint32_t instance)
  Checks whether a mode fault occurred.
- void spi_hal_clear_mode_fault (uint32_t instance)
  Clears the mode fault flag.
- static bool spi_hal_is_match (uint32_t instance)
  Checks whether the data received matches the previously-set match value.
- void spi_hal_clear_match (uint32_t instance)
  Clears the match flag.

Data transfer

- static uint8_t spi_hal_read_data (uint32_t instance)
  Reads a byte from the data buffer.
- static void spi_hal_write_data (uint32_t instance, uint8_t data)
  Writes a byte into the data buffer.

Match byte

- static void spi_hal_set_match_value (uint32_t instance, uint8_t matchByte)
  Sets the value which triggers the match interrupt.

21.1.1 Data Structure Documentation

21.1.1.1 struct spi_config_t

Use an instance of this structure with spi_hal_init(). This allows you to configure the most common settings of the SPI peripheral with a single function call.

The `kbitsPerSec` member is handled separately. If this value is set to 0, then the baud is not set by spi_hal_init(), and must be set with a separate call to either the spi_hal_set_baud() or the spi_hal_set_baud_divisors(). This can be useful if you know the divisors in advance and don't want to spend the time to compute them for the provided rate in kilobits/sec.

Data Fields

- bool isEnabled
  Set to true to enable the SPI peripheral.
- uint32_t kbitsPerSec
  Baud rate in kilobits per second.
- spi_master_slave_mode_t masterOrSlave
  Whether to put the peripheral in master or slave mode.
- spi_clock_polarity_t polarity
  Clock polarity setting.
- spi_clock_phase_t phase
  Clock phase setting.
- spi_shift_direction_t shiftDirection
  Direction in which data is shifted out.
- spi_ss_output_mode_t ssOutputMode
  Output mode for the slave select signal.
- spi_pin_mode_t pinMode
  Pin mode with bidirectional option.
- bool enableReceiveAndFaultInterrupt
  Enable for the receive and fault interrupt.
- bool enableTransmitInterrupt
  Enable for the transmit interrupt.
- bool enableMatchInterrupt
  Enable for the match interrupt.

21.1.1.1.0.39 Field Documentation

21.1.1.1.0.39.1 bool spi_config_t::isEnabled

21.1.1.1.0.39.2 uint32_t spi_config_t::kbitsPerSec

21.1.1.1.0.39.3 spi_master_slave_mode_t spi_config_t::masterOrSlave

21.1.1.1.0.39.4 spi_clock_polarity_t spi_config_t::polarity

21.1.1.1.0.39.5 spi_clock_phase_t spi_config_t::phase

21.1.1.1.0.39.6 spi_shift_direction_t spi_config_t::shiftDirection

21.1.1.1.0.39.7 spi_ss_output_mode_t spi_config_t::ssOutputMode

21.1.1.1.0.39.8 spi_pin_mode_t spi_config_t::pinMode

21.1.1.1.0.39.9 bool spi_config_t::enableReceiveAndFaultInterrupt

21.1.1.1.0.39.10 bool spi_config_t::enableTransmitInterrupt

21.1.1.1.0.39.11 bool spi_config_t::enableMatchInterrupt

21.1.2 Enumeration Type Documentation

21.1.2.1 enum spi_status_t

Enumerator

kStatus_SPI_SlaveTxUnderrun SPI Slave TX Underrun error.
kStatus_SPI_SlaveRxOverrun SPI Slave RX Overrun error.
kStatus_SPI_Timeout SPI transfer timed out.
kStatus_SPI_Busy SPI instance is already busy performing a transfer.
kStatus_SPI_NoTransferInProgress Attempt to abort a transfer when no transfer was in progress.

21.1.2.2 enum spi_master_slave_mode_t

Enumerator

kSpiMaster SPI peripheral operates in master mode.
kSpiSlave SPI peripheral operates in slave mode.

21.1.2.3 enum spi_clock_polarity_t

Enumerator

kSpiClockPolarity_ActiveHigh Active-high SPI clock (idles low).

kSpiClockPolarity_ActiveLow Active-low SPI clock (idles high).

21.1.2.4 enum spi_clock_phase_t

Enumerator

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

21.1.2.5 enum spi_shift_direction_t

Enumerator

kSpiMsbFirst Data transfers start with most significant bit.
kSpiLsbFirst Data transfers start with least significant bit.

21.1.2.6 enum spi_ss_output_mode_t

Enumerator

kSpiSlaveSelect_AsGpio Slave select pin configured as GPIO.
kSpiSlaveSelect_FaultInput Slave select pin configured for fault detection.
kSpiSlaveSelect_AutomaticOutput Slave select pin configured for automatic SPI output.

21.1.2.7 enum spi_pin_mode_t

Enumerator

kSpiPinMode_Normal Pins operate in normal, single-direction mode.
kSpiPinMode_Input Bidirectional mode. Master: MOSI pin is input; Slave: MISO pin is input
kSpiPinMode_Output Bidirectional mode. Master: MOSI pin is output; Slave: MISO pin is output

21.1.3 Function Documentation

21.1.3.1 static bool spi_hal_is_read_buffer_full (uint32_t *instance*) [inline], [static]

The read buffer full flag is only cleared by reading it when it is set, then reading the data register by calling the spi_hal_read_data(). This example code demonstrates how to check the flag, read data, and clear the flag.

```
*      // Check read buffer flag.
*      if (spi_hal_is_read_buffer_full(0))
*      {
*          // Read the data in the buffer, which also clears the flag.
*          byte = spi_hal_read_data(0);
*      }
*
```

21.1.3.2 static bool spi_hal_is_transmit_buffer_empty (uint32_t *instance*) [inline], [static]

To clear the transmit buffer empty flag, you must first read the flag when it is set. Then write a new data value into the transmit buffer with a call to the spi_hal_write_data(). The example code shows how to do this.

```
*      // Check if transmit buffer is empty.
*      if (spi_hal_is_transmit_buffer_empty(0))
*      {
*          // Buffer has room, so write the next data value.
*          spi_hal_write_data(0, byte);
*      }
*
```

21.2 SPI Master Peripheral Driver

The chapter describes the programming interface of the SPI master mode Peripheral driver.

Data Structures

- struct spi_user_config_t
  Information about a device on the SPI bus. More...

Enumerations

- enum _spi_timeouts { kSpiWaitForever = 0xffffffffU }

Initialization and shutdown

- void spi_master_init (uint32_t instance)
  Initializes an SPI instance for master mode operation.
- void spi_master_shutdown (uint32_t instance)
  Shuts down an SPI instance.

Bus configuration

- void spi_master_configure_bus (uint32_t instance, const spi_user_config_t *device)
  Configures the SPI port to access a device on the bus.

Blocking transfers

- spi_status_t spi_master_transfer (uint32_t instance, const spi_user_config_t *restrict device, const uint8_t *restrict sendBuffer, uint8_t *restrict receiveBuffer, size_t transferByteCount, uint32_t timeout)
  Performs a blocking SPI master mode transfer.

Non-blocking transfers

- spi_status_t spi_master_transfer_async (uint32_t instance, const spi_user_config_t *restrict device, const uint8_t *restrict sendBuffer, uint8_t *restrict receiveBuffer, size_t transferByteCount)
  Performs a non-blocking SPI master mode transfer.
- spi_status_t spi_master_get_tranfer_status (uint32_t instance, uint32_t *bytesTransferred)
  Returns whether the previous transfer finished.
- spi_status_t spi_master_abort_transfer (uint32_t instance)
  Terminates an asynchronous transfer early.

21.2.0.3 SPI Master Peripheral Driver

Overview

The SPI master mode peripheral driver transfers data to and from the external devices on the SPI bus in master mode. It provides an easy way to transfer buffers of data with a single function call.

Device structures

The driver uses instances of the #spi_device_t structure to represent the SPI bus configuration required to communicate the bus connectivity to an external device.

The device structures can be passed to the data transfer functions and the bus is reconfigured before the transfer is started. Alternatively, you can manually configure the SPI bus for a device. For example, if there is only one device connected to the bus, you might configure it only once.

Initialization

To initialize the SPI master driver, call the spi_master_init() function and pass the instance number of the SPI peripheral you want to use. For example, to use the SPI1, pass a value of 1 to the initialization function.

This is an example code to initialize and configure the driver:

```
// Define bus configuration.
spi_device_t device = {
        .busFrequencyKHz = 1000,
        .polarity = kSpiClockPolarity_ActiveHigh,
        .phase = kSpiClockPhase_SecondEdge,
        .direction = kSpiMsbFirst
    };

// Init driver.
spi_master_init(0);

// Configure bus.
spi_master_configure_bus(0, &device);
```

Transfers

The driver supports two different modes to transfer data: blocking and non-blocking. The blocking transfer function is the spi_master_transfer().

This is an example of a blocking transfer:

```
uint8_t sendBuf[] = { 0, 1, 2, 3 };
uint8_t receiveBuf[sizeof(sendBuf)];
```

```
spi_master_transfer(0,  // SPI0
    &device,            // Device SPI bus configuration
    sendBuf,            // Data to send
    receiveBuf,         // Buffer to hold received data
    sizeof(sendBuf),    // Number of bytes to transfer
    kSpiWaitForever);   // No timeout
```

This is an example of a non-blocking transfer:

```
uint8_t sendBuf[] = { 0, 1, 2, 3 };
uint8_t receiveBuf[sizeof(sendBuf)];

// Start the transfer.
spi_master_transfer_async(0,  // SPI0
    &device,            // Device SPI bus configuration
    sendBuf,            // Data to send
    receiveBuf,         // Buffer to hold received data
    sizeof(sendBuf));   // Number of bytes to transfer

// Wait for the transfer to complete.
while (spi_master_get_transfer_status(0, NULL) == kStatus_SPI_Busy)
{
}
```

21.2.1 Data Structure Documentation

21.2.1.1 struct spi_user_config_t

21.2.2 Enumeration Type Documentation

21.2.2.1 enum _spi_timeouts

Enumerator

kSpiWaitForever Waits forever for a transfer to complete.

21.2.3 Function Documentation

21.2.3.1 void spi_master_init (uint32_t *instance*)

Parameters

instance	The instance number of the SPI peripheral.

21.2.3.2 void spi_master_shutdown (uint32_t *instance*)

Resets the SPI peripheral and gates its clock.

Parameters

instance	The instance number of the SPI peripheral.

21.2.3.3 void spi_master_configure_bus (uint32_t *instance*, const spi_user_config_t * *device*)

Parameters

instance	The instance number of the SPI peripheral.
device	Pointer to the device information structure. This structure contains the settings for SPI bus configurations.

21.2.3.4 spi_status_t spi_master_transfer (uint32_t *instance*, const spi_user_config_t *restrict *device*, const uint8_t *restrict *sendBuffer*, uint8_t *restrict *receiveBuffer*, size_t *transferByteCount*, uint32_t *timeout*)

This function simultaneously sends and receives data on the SPI bus, as SPI is naturally a full-duplex bus.

Parameters

instance	The instance number of the SPI peripheral.
device	Pointer to the device information structure. This structure contains the settings for the SPI bus configuration for this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified.
sendBuffer	Buffer of data to send. You may pass NULL for this parameter, in which case bytes with a value of 0 (zero) are sent.
receiveBuffer	Buffer where received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.
transferByte-Count	The number of bytes to send and receive.
timeout	A timeout for the transfer in microseconds. If the transfer takes longer than this amount of time, the transfer is aborted and a kStatus_SPI_Timeout error is returned.

Return values

#kStatus_Success	The transfer was successful.
kStatus_SPI_Busy	Cannot perform another transfer because one is already in progress.
kStatus_SPI_Timeout	The transfer timed out and was aborted.

21.2.3.5 spi_status_t spi_master_transfer_async (uint32_t *instance*, const spi_user_config_t *restrict *device*, const uint8_t *restrict *sendBuffer*, uint8_t *restrict *receiveBuffer*, size_t *transferByteCount*)

Parameters

instance	The instance number of the SPI peripheral.
device	Pointer to the device information structure. This structure contains the settings for the SPI bus configuration for this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified.
sendBuffer	Buffer of data to send. You may pass NULL for this parameter, in which case bytes with a value of 0 (zero) will be sent.
receiveBuffer	Buffer where received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.
transferByte-Count	The number of bytes to send and receive.

Return values

#kStatus_Success	The transfer was successful.
kStatus_SPI_Busy	Cannot perform another transfer because one is already in progress.
kStatus_SPI_Timeout	The transfer timed out and was aborted.

21.2.3.6 spi_status_t spi_master_get_tranfer_status (uint32_t *instance*, uint32_t * *bytesTransferred*)

Parameters

instance	The instance number of the SPI peripheral.

bytes-Transferred	Pointer to a value that is filled in with the number of bytes that were sent in the active transfer

Return values

kStatus_Success	The transfer has completed successfully.
kStatus_SPI_Busy	The transfer is still in progress. *bytesTransferred* is filled with the number of bytes that have been transferred so far.

21.2.3.7 spi_status_t spi_master_abort_transfer (uint32_t *instance*)

Parameters

instance	The instance number of the SPI peripheral.

Return values

#kStatus_Success	The transfer was successful.

21.3 SPI Slave Peripheral Driver

The chapter describes the programming interface of the SPI slave mode Peripheral driver.

Data Structures

- struct spi_slave_callbacks_t
  The set of callbacks used for SPI slave mode. More...
- struct spi_slave_user_config_t
  Definition of application implemented configuration and callback. More...

SPI Slave

- void spi_slave_init (uint32_t instance, const spi_slave_user_config_t ∗config)
  Initializes the SPI module.
- void spi_slave_shutdown (uint32_t instance)
  De-initializes the device.

21.3.0.8 SPI Slave Peripheral Driver

Overview

The SPI slave peripheral driver provides an easy way to use an SPI peripheral in slave mode.

Data transfer is performed through callback functions.

Callbacks

To use this driver, first define several application callbacks. These are the callback functions that are set in the spi_slave_callbacks_t structure.

The three callbacks are:

- Data source
- Data sink
- Error notification

The first two callbacks are used to send and receive data. The third callback is invoked if an error occurs.

The prototypes for the three callbacks are:

```
status_t dataSource(uint8_t * sourceByte, uint16_t instance);

status_t dataSink(uint8_t sinkByte, uint16_t instance);

void onError(status_t error);
```

All callbacks are invoked from IRQ state.

Setup

To initialize the SPI slave driver, first create and fill in the #spi_slave_config_t structure. This structure defines the callbacks and data format settings for the SPI peripheral. The structure is not required after the driver is initialized and can be allocated on the stack.

This is an example of a config struct definition:

```
spi_slave_config_t config = {
        {
            appDataSource,
            appDataSink,
            appErrorHandler,
        },
        kSpiClockPhase_FirstEdge,
        kSpiClockPolarity_ActiveHigh,
        kSpiLsbFirst
    };
```

21.3.1 Data Structure Documentation

21.3.1.1 struct spi_slave_callbacks_t

Data Fields

- spi_status_t(∗ dataSource)(uint8_t ∗sourceByte, uint16_t instance)
  Callback used to get byte to transmit.
- spi_status_t(∗ dataSink)(uint8_t sinkByte, uint16_t instance)
  Callback used to put received byte.
- void(∗ onError)(spi_status_t error)
  Callback used to report an SPI error.

21.3.1.1.0.40 Field Documentation

21.3.1.1.0.40.1 spi_status_t(∗ spi_slave_callbacks_t::dataSource)(uint8_t ∗sourceByte, uint16_t instance)

21.3.1.1.0.40.2 spi_status_t(∗ spi_slave_callbacks_t::dataSink)(uint8_t sinkByte, uint16_t instance)

21.3.1.1.0.40.3 void(∗ spi_slave_callbacks_t::onError)(spi_status_t error)

21.3.1.2 struct spi_slave_user_config_t

functions used by the SPI slave driver.

Data Fields

- spi_slave_callbacks_t callbacks
  Application callbacks.

- spi_clock_phase_t phase
  Clock phase setting.
- spi_clock_polarity_t polarity
  Clock polarity setting.
- spi_shift_direction_t direction
  Either LSB or MSB first.

21.3.1.2.0.41 Field Documentation

21.3.1.2.0.41.1 spi_slave_callbacks_t spi_slave_user_config_t::callbacks

21.3.1.2.0.41.2 spi_clock_phase_t spi_slave_user_config_t::phase

21.3.1.2.0.41.3 spi_clock_polarity_t spi_slave_user_config_t::polarity

21.3.1.2.0.41.4 spi_shift_direction_t spi_slave_user_config_t::direction

21.3.2 Function Documentation

21.3.2.1 void spi_slave_init (uint32_t *instance,* const spi_slave_user_config_t * *config*)

Saves the application callback info, turns on the clock to the module, enables the device, and enables interrupts. Sets the SPI to a slave mode.

Parameters

instance	Instance number of the SPI module.
config	Pointer to slave mode configuration.

21.3.2.2 void spi_slave_shutdown (uint32_t *instance*)

Clears the control register and turns off the clock to the module.

Parameters

instance	Instance number of the SPI module.

21.4 Shared SPI Types

This chapter describes SPI driver shared types.

21.5 SPI Classes

This chapter describes the SPI driver C++ classes.

Chapter 22
Universal Asynchronous Receiver/Transmitter (UART)

The Kinetis SDK provides both HAL and Peripheral drivers for the Universal Asynchronous Receiver/-Transmitter (UART) block of Kinetis devices.

Modules

- UART HAL driver
  The part describes the programming interface of the UART HAL driver.
- UART Peripheral Driver
  The part describes the programming interface of the UART Peripheral driver.

22.1 UART HAL driver

The chapter describes the programming interface of the UART HAL driver.

Data Structures

- struct uart_idle_line_config_t
  Structure for idle line configuration settings. More...
- struct uart_status_flag_all_t
  Structure for all UART status flags. More...
- struct uart_interrupt_config_t
  UART interrupt configuration structure, default settings are 0 (disabled). More...
- struct uart_config_t
  UART configuration structure. More...

Enumerations

- enum uart_status_t { ,
  kStatus_UART_BaudRateCalculationError,
  kStatus_UART_BaudRatePercentDiffExceeded,
  kStatus_UART_BitCountNotSupported,
  kStatus_UART_StopBitCountNotSupported,
  kStatus_UART_RxStandbyModeError,
  kStatus_UART_ClearStatusFlagError,
  kStatus_UART_MSBFirstNotSupported,
  kStatus_UART_ResyncNotSupported,
  kStatus_UART_TxNotDisabled,
  kStatus_UART_RxNotDisabled,
  kStatus_UART_TxOrRxNotDisabled,
  kStatus_UART_TxBusy,
  kStatus_UART_RxBusy,
  kStatus_UART_NoTransmitInProgress,
  kStatus_UART_NoReceiveInProgress,
  kStatus_UART_InvalidInstanceNumber,
  kStatus_UART_InvalidBitSetting,
  kStatus_UART_OverSamplingNotSupported,
  kStatus_UART_BothEdgeNotSupported,
  kStatus_UART_Timeout }
  Error codes for the UART driver.
- enum uart_stop_bit_count_t {
  kUartOneStopBit = 0,
  kUartTwoStopBit = 1 }
  UART number of stop bits.
- enum uart_parity_mode_t {
  kUartParityDisabled = 0x0,
  kUartParityEven = 0x2,

kUartParityOdd = 0x3 }
UART parity mode.

- enum uart_bit_count_per_char_t {
kUart8BitsPerChar = 0,
kUart9BitsPerChar = 1,
kUart10BitsPerChar = 2 }
UART number of bits in a character.
- enum uart_operation_config_t {
kUartOperates = 0,
kUartStops = 1 }
UART operation configuration constants.
- enum uart_wakeup_method_t {
kUartIdleLineWake = 0,
kUartAddrMarkWake = 1 }
UART wakeup from standby method constants.
- enum uart_idle_line_select_t {
kUartIdleLineAfterStartBit = 0,
kUartIdleLineAfterStopBit = 1 }
UART idle-line detect selection types.
- enum uart_break_char_length_t {
kUartBreakChar10BitMinimum = 0,
kUartBreakChar13BitMinimum = 1 }
UART break character length settings for transmit/detect.
- enum uart_singlewire_txdir_t {
kUartSinglewireTxdirIn = 0,
kUartSinglewireTxdirOut = 1 }
UART single-wire mode transmit direction.
- enum uart_status_flag_t {
kUartTransmitDataRegisterEmpty,
kUartTransmissionComplete,
kUartReceiveDataRegisterFull,
kUartIdleLineDetect,
kUartReceiveOverrun,
kUartNoiseDetect,
kUartFrameError,
kUartParityError,
kUartLineBreakDetect,
kUartReceiveActiveEdgeDetect,
kUartReceiverActive }
UART status flags.
- enum uart_ir_tx_pulsewidth_t {
kUartIrThreeSixteenthsWidth = 0,
kUartIrOneSixteenthWidth = 1,
kUartIrOneThirtysecondsWidth = 2,
kUartIrOneFourthWidth = 3 }
UART infrared transmitter pulse width options.

UART Common Configurations

- uart_status_t uart_hal_init (uint32_t uartInstance, const uart_config_t *config)
  Initialize the UART controller.
- uart_status_t uart_hal_set_baud_rate (uint32_t uartInstance, uint32_t sourceClockInHz, uint32_t desiredBaudRate)
  Configure the UART baud rate.
- uart_status_t uart_hal_set_baud_rate_divisor (uint32_t uartInstance, uint32_t baudRateDivisor)
  Set the UART baud rate modulo divisor value.
- uart_status_t uart_hal_configure_bit_count_per_char (uint32_t uartInstance, uart_bit_count_per_char_t bitCountPerChar)
  Configure number of bits per character in the UART controller.
- void uart_hal_configure_parity_mode (uint32_t uartInstance, uart_parity_mode_t parityModeType)
  Configure the parity mode in the UART controller.
- uart_status_t uart_hal_configure_stop_bit_count (uint32_t uartInstance, uart_stop_bit_count_t stopBitCount)
  Configure the number of stop bits in the UART controller.
- void uart_hal_configure_tx_rx_inversion (uint32_t uartInstance, uint32_t rxInvert, uint32_t txInvert)
  Configure the transmit and receive inversion control in UART controller.
- void uart_hal_enable_transmitter (uint32_t uartInstance)
  Enable the UART transmitter.
- void uart_hal_disable_transmitter (uint32_t uartInstance)
  Disable the UART transmitter.
- bool uart_hal_is_transmitter_enabled (uint32_t uartInstance)
  Get the UART transmitter enabled/disabled configuration setting.
- void uart_hal_enable_receiver (uint32_t uartInstance)
  Enable the UART receiver.
- void uart_hal_disable_receiver (uint32_t uartInstance)
  Disable the UART receiver.
- bool uart_hal_is_receiver_enabled (uint32_t uartInstance)
  Get the UART receiver enabled/disabled configuration setting.

UART Interrupts and DMA

- void uart_hal_configure_interrupts (uint32_t uartInstance, const uart_interrupt_config_t *interruptConfig)
  Configure the UART module interrupts to enable/disable various interrupt sources.
- void uart_hal_enable_break_detect_interrupt (uint32_t uartInstance)
  Enable the break_detect_interrupt.
- void uart_hal_disable_break_detect_interrupt (uint32_t uartInstance)
  Disable the break_detect_interrupt.
- bool uart_hal_is_break_detect_interrupt_enabled (uint32_t uartInstance)
  Get the configuration of the break_detect_interrupt enable setting.
- void uart_hal_enable_rx_active_edge_interrupt (uint32_t uartInstance)
  Enable the rx_active_edge_interrupt.
- void uart_hal_disable_rx_active_edge_interrupt (uint32_t uartInstance)
  Disable the rx_active_edge_interrupt.
- bool uart_hal_is_rx_active_edge_interrupt_enabled (uint32_t uartInstance)

Get the configuration of the rx_active_edge_interrupt enable setting.
- void uart_hal_enable_tx_data_register_empty_interrupt (uint32_t uartInstance)
  Enable the tx_data_register_empty_interrupt.
- void uart_hal_disable_tx_data_register_empty_interrupt (uint32_t uartInstance)
  Disable the tx_data_register_empty_interrupt.
- bool uart_hal_is_tx_data_register_empty_interrupt_enabled (uint32_t uartInstance)
  Get the configuration of the tx_data_register_empty_interrupt enable setting.
- void uart_hal_enable_transmission_complete_interrupt (uint32_t uartInstance)
  Enable the transmission_complete_interrupt.
- void uart_hal_disable_transmission_complete_interrupt (uint32_t uartInstance)
  Disable the transmission_complete_interrupt.
- bool uart_hal_is_transmission_complete_interrupt_enabled (uint32_t uartInstance)
  Get the configuration of the transmission_complete_interrupt enable setting.
- void uart_hal_enable_rx_data_register_full_interrupt (uint32_t uartInstance)
  Enable the rx_data_register_full_interrupt.
- void uart_hal_disable_rx_data_register_full_interrupt (uint32_t uartInstance)
  Disable the rx_data_register_full_interrupt.
- bool uart_hal_is_receive_data_full_interrupt_enabled (uint32_t uartInstance)
  Get the configuration of the rx_data_register_full_interrupt enable setting.
- void uart_hal_enable_idle_line_interrupt (uint32_t uartInstance)
  Enable the idle_line_interrupt.
- void uart_hal_disable_idle_line_interrupt (uint32_t uartInstance)
  Disable the idle_line_interrupt.
- bool uart_hal_is_idle_line_interrupt_enabled (uint32_t uartInstance)
  Get the configuration of the idle_line_interrupt enable setting.
- void uart_hal_enable_rx_overrun_interrupt (uint32_t uartInstance)
  Enable the rx_overrun_interrupt.
- void uart_hal_disable_rx_overrun_interrupt (uint32_t uartInstance)
  Disable the rx_overrun_interrupt.
- bool uart_hal_is_rx_overrun_interrupt_enabled (uint32_t uartInstance)
  Get the configuration of the rx_overrun_interrupt enable setting.
- void uart_hal_enable_noise_error_interrupt (uint32_t uartInstance)
  Enable the noise_error_interrupt.
- void uart_hal_disable_noise_error_interrupt (uint32_t uartInstance)
  Disable the noise_error_interrupt.
- bool uart_hal_is_noise_error_interrupt_enabled (uint32_t uartInstance)
  Get the configuration of the noise_error_interrupt enable setting.
- void uart_hal_enable_framing_error_interrupt (uint32_t uartInstance)
  Enable the framing_error_interrupt.
- void uart_hal_disable_framing_error_interrupt (uint32_t uartInstance)
  Disable the framing_error_interrupt.
- bool uart_hal_is_framing_error_interrupt_enabled (uint32_t uartInstance)
  Get the configuration of the framing_error_interrupt enable setting.
- void uart_hal_enable_parity_error_interrupt (uint32_t uartInstance)
  Enable the parity_error_interrupt.
- void uart_hal_disable_parity_error_interrupt (uint32_t uartInstance)
  Disable the parity_error_interrupt.
- bool uart_hal_is_parity_error_interrupt_enabled (uint32_t uartInstance)
  Get the configuration of the parity_error_interrupt enable setting.
- void uart_hal_configure_dma (uint32_t uartInstance, bool txDmaConfig, bool rxDmaConfig)
  Configure the UART DMA requests for the Transmitter and Receiver.

- bool uart_hal_is_txdma_enabled (uint32_t uartInstance)
  Get the UART Transmit DMA request configuration setting.
- bool uart_hal_is_rxdma_enabled (uint32_t uartInstance)
  Get the UART Receive DMA request configuration setting.

UART Transfer Functions

- void uart_hal_putchar (uint32_t uartInstance, uint8_t data)
  This function allows the user to send an 8-bit character from the UART data register.
- void uart_hal_putchar9 (uint32_t uartInstance, uint16_t data)
  This function allows the user to send a 9-bit character from the UART data register.
- uart_status_t uart_hal_putchar10 (uint32_t uartInstance, uint16_t data)
  This function allows the user to send a 10-bit character from the UART data register.
- void uart_hal_getchar (uint32_t uartInstance, uint8_t *readData)
  This function gets a received 8-bit character from the UART data register.
- void uart_hal_getchar9 (uint32_t uartInstance, uint16_t *readData)
  This function gets a received 9-bit character from the UART data register.
- uart_status_t uart_hal_getchar10 (uint32_t uartInstance, uint16_t *readData)
  This function gets a received 10-bit character from the UART data register.

UART Special Feature Configurations

- void uart_hal_configure_wait_mode_operation (uint32_t uartInstance, uart_operation_config_t mode)
  Configure the UART to either operate or cease to operate in WAIT mode.
- uart_operation_config_t uart_hal_get_wait_mode_operation_config (uint32_t uartInstance)
  Determine if the UART operates or ceases to operate in WAIT mode.
- void uart_hal_configure_loopback_mode (uint32_t uartInstance, bool enable)
  Configure the UART loopback operation.
- void uart_hal_configure_singlewire_mode (uint32_t uartInstance, bool enable)
  Configure the UART single-wire operation.
- void uart_hal_configure_txdir_in_singlewire_mode (uint32_t uartInstance, uart_singlewire_txdir_t direction)
  Configure the UART transmit direction while in single-wire mode.
- uart_status_t uart_hal_put_receiver_in_standby_mode (uint32_t uartInstance)
  Place the UART receiver in standby mode.
- void uart_hal_put_receiver_in_normal_mode (uint32_t uartInstance)
  Place the UART receiver in normal mode (disable standby mode operation).
- bool uart_hal_is_receiver_in_standby (uint32_t uartInstance)
  Determine if the UART receiver is currently in standby mode.
- void uart_hal_select_receiver_wakeup_method (uint32_t uartInstance, uart_wakeup_method_-t method)
  Select the UART receiver wakeup method (idle-line or address-mark) from standby mode.
- uart_wakeup_method_t uart_hal_get_receiver_wakeup_method (uint32_t uartInstance)
  Get the UART receiver wakeup method (idle-line or address-mark) from standby mode.
- void uart_hal_configure_idle_line_detect (uint32_t uartInstance, const uart_idle_line_config_t *config)
  Configure the operation options of the UART idle line detect.

- void uart_hal_set_break_char_transmit_length (uint32_t uartInstance, uart_break_char_length_t length)
  Configure the UART break character transmit length.
- void uart_hal_set_break_char_detect_length (uint32_t uartInstance, uart_break_char_length_t length)
  Configure the UART break character detect length.
- void uart_hal_queue_break_char_to_send (uint32_t uartInstance, bool enable)
  Configure the UART transmit send break character operation.
- uart_status_t uart_hal_configure_match_address_operation (uint32_t uartInstance, bool matchAddrMode1, bool matchAddrMode2, uint8_t matchAddrValue1, uint8_t matchAddrValue2)
  Configure the UART match address mode control operation.
- uart_status_t uart_hal_configure_send_msb_first_operation (uint32_t uartInstance, bool enable)
  Configure the UART to send data MSB first (Note: Feature available on select UART instances)
- uart_status_t uart_hal_configure_receive_resync_disable_operation (uint32_t uartInstance, bool enable)
  Configuration option to disable the UART resynchronization during received data.

UART Status Flags

- void uart_hal_get_all_status_flag (uint32_t uartInstance, uart_status_flag_all_t *allStatusFlag)
  Get all of the UART status flag states.
- bool uart_hal_is_transmit_data_register_empty (uint32_t uartInstance)
  Get the UART Transmit data register empty flag.
- bool uart_hal_is_transmission_complete (uint32_t uartInstance)
  Get the UART Transmission complete flag.
- bool uart_hal_is_receive_data_register_full (uint32_t uartInstance)
  Get the UART Receive data register full flag.
- bool uart_hal_is_idle_line_detected (uint32_t uartInstance)
  Get the UART Idle-line detect flag.
- bool uart_hal_is_receive_overrun_detected (uint32_t uartInstance)
  Get the UART Receiver Overrun status flag.
- bool uart_hal_is_noise_detected (uint32_t uartInstance)
  Get the UART noise status flag.
- bool uart_hal_is_frame_error_detected (uint32_t uartInstance)
  Get the UART Frame error status flag.
- bool uart_hal_is_parity_error_detected (uint32_t uartInstance)
  Get the UART parity error status flag.
- bool uart_hal_is_line_break_detected (uint32_t uartInstance)
  Get the UART LIN break detect interrupt status flag.
- bool uart_hal_is_receive_active_edge_detected (uint32_t uartInstance)
  Get the UART Receive pin active edge interrupt status flag.
- bool uart_hal_is_receiver_active (uint32_t uartInstance)
  Get the UART Receiver Active Flag (RAF) state.
- uart_status_t uart_hal_clear_status_flag (uint32_t uartInstance, uart_status_flag_t statusFlag)
  Clear an individual and specific UART status flag.
- void uart_hal_clear_all_non_autoclear_status_flags (uint32_t uartInstance)
  Clear ALL of the UART status flags.

22.1.1 Data Structure Documentation

22.1.1.1 struct uart_idle_line_config_t

This structure contains settings for the UART idle line configuration such as the Idle Line Type (ILT) and the Receiver Wake Up Idle Detect (RWUID).

Data Fields

- unsigned idleLineType: 1
  ILT, Idle bit count start: 0 - after start bit (default), 1 - after stop bit.
- unsigned rxWakeIdleDetect: 1
  RWUID, Receiver Wake Up Idle Detect.

22.1.1.1.0.42 Field Documentation

22.1.1.1.0.42.1 unsigned uart_idle_line_config_t::rxWakeIdleDetect

IDLE status bit operation during receive standby. Controls whether idle character that wakes up receiver will also set IDLE status bit 0 - IDLE status bit doesn't get set (default), 1 - IDLE status bit gets set

22.1.1.2 struct uart_status_flag_all_t

This structure contains the settings for all of the UART status flags.

Data Fields

- unsigned transmitDataRegisterEmpty: 1
  Transmit data register empty flag, sets when transmit buffer is empty.
- unsigned transmissionComplete: 1
  Transmission complete flag, sets when transmitter is idle (transmission activity complete)
- unsigned receiveDataRegisterFull: 1
  Receive data register full flag, sets when the receive data buffer is full.
- unsigned idleLineDetect: 1
  Idle line detect flag, sets when idle line detected.
- unsigned receiveOverrun: 1
  Receiver Overrun, sets when new data is received before data is read from receive register.
- unsigned noiseDetect: 1
  Receiver takes 3 samples of each received bit.
- unsigned frameError: 1
  Frame error flag, sets if logic 0 was detected where stop bit expected.
- unsigned parityError: 1
  If parity enabled, will set upon parity error detection.
- unsigned lineBreakDetect: 1
  LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.
- unsigned receiveActiveEdgeDetect: 1
  Receive pin active edge interrupt flag, sets when active edge detected.

- unsigned receiverActive: 1
  Receiver Active Flag (RAF), sets at beginning of valid start bit.

22.1.1.2.0.43 Field Documentation

22.1.1.2.0.43.1 unsigned uart_status_flag_all_t::noiseDetect

If any of these samples differ, noise flag sets

22.1.1.3 struct uart_interrupt_config_t

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

Data Fields

- unsigned linBreakDetect: 1
  LIN break detect: 0 - disable interrupt, 1 - enable interrupt.
- unsigned rxActiveEdge: 1
  RX Active Edge: 0 - disable interrupt, 1 - enable interrupt.
- unsigned transmitDataRegisterEmpty: 1
  Transmit data register empty: 0 - disable interrupt, 1 - enable interrupt.
- unsigned transmitComplete: 1
  Transmission complete: 0 - disable interrupt, 1 - enable interrupt.
- unsigned receiverDataRegisterFull: 1
  Receiver data register full: 0 - disable interrupt, 1 - enable interrupt.
- unsigned idleLine: 1
  Idle line: 0 - disable interrupt, 1 - enable interrupt.
- unsigned receiverOverrun: 1
  Receiver Overrun: 0 - disable interrupt, 1 - enable interrupt.
- unsigned noiseErrorFlag: 1
  Noise error flag: 0 - disable interrupt, 1 - enable interrupt.
- unsigned frameErrorFlag: 1
  Framing error flag: 0 - disable interrupt, 1 - enable interrupt.
- unsigned parityErrorFlag: 1
  Parity error flag: 0 - disable interrupt, 1 - enable interrupt.

22.1.1.4 struct uart_config_t

This structure contains the settings for the most common UART configurations including the UART module source clock, baud rate, parity mode, stop bit count, data bit count per character, and tx/rx inversion options (which is the least common of the configurations).

Data Fields

- uint32_t uartSourceClockInHz
  UART module source clock in Hz.
- uint32_t baudRate

UART baud rate.

- uart_parity_mode_t parityMode
  Parity mode, disabled (default), even, or odd.
- uart_stop_bit_count_t stopBitCount
  Number of stop bits, 1 stop bit (default) or 2 stop bits.
- uart_bit_count_per_char_t bitCountPerChar
  Number of bits, 8-bit (default) or 9-bit in a word (up to 10-bits in some UART instances)
- unsigned rxDataInvert: 1
  Receive Data Inversion: 0 - not inverted (default), 1 - inverted.
- unsigned txDataInvert: 1
  Transmit Data Inversion: 0 - not inverted (default), 1 - inverted.

22.1.2 Enumeration Type Documentation

22.1.2.1 enum uart_status_t

Enumerator

kStatus_UART_BaudRateCalculationError UART Baud Rate calculation error out of range.
kStatus_UART_BaudRatePercentDiffExceeded UART Baud Rate exceeds percentage difference.
kStatus_UART_BitCountNotSupported UART bit count config not supported.
kStatus_UART_StopBitCountNotSupported UART stop bit count config not supported.
kStatus_UART_RxStandbyModeError UART unable to place receiver in standby mode.
kStatus_UART_ClearStatusFlagError UART clear status flag error.
kStatus_UART_MSBFirstNotSupported UART MSB first feature not supported.
kStatus_UART_ResyncNotSupported UART resync disable operation not supported.
kStatus_UART_TxNotDisabled UART Transmitter not disabled before enabling feature.
kStatus_UART_RxNotDisabled UART Receiver not disabled before enabling feature.
kStatus_UART_TxOrRxNotDisabled UART Transmitter or Receiver not disabled.
kStatus_UART_TxBusy UART transmit still in progress.
kStatus_UART_RxBusy UART receive still in progress.
kStatus_UART_NoTransmitInProgress UART no transmit in progress.
kStatus_UART_NoReceiveInProgress UART no receive in progress.
kStatus_UART_InvalidInstanceNumber Invalid UART instance number.
kStatus_UART_InvalidBitSetting Invalid setting for desired UART register bit field.
kStatus_UART_OverSamplingNotSupported UART oversampling not supported.
kStatus_UART_BothEdgeNotSupported UART both edge sampling not supported.
kStatus_UART_Timeout UART transfer timed out.

22.1.2.2 enum uart_stop_bit_count_t

These constants define the number of allowable stop bits to configure in a UART instance.

Enumerator

kUartOneStopBit one stop bit

kUartTwoStopBit two stop bits

22.1.2.3 enum uart_parity_mode_t

These constants define the UART parity mode options: disabled or enabled of type even or odd.

Enumerator

kUartParityDisabled parity disabled
kUartParityEven parity enabled, type even, bit setting: PE|PT = 10
kUartParityOdd parity enabled, type odd, bit setting: PE|PT = 11

22.1.2.4 enum uart_bit_count_per_char_t

These constants define the number of allowable data bits per UART character. Note, check the UART documentation to determine if the desired UART instance supports the desired number of data bits per UART character.

Enumerator

kUart8BitsPerChar 8-bit data characters
kUart9BitsPerChar 9-bit data characters
kUart10BitsPerChar 10-bit data characters

22.1.2.5 enum uart_operation_config_t

This provides constants for UART operational states: "operates normally" or "stops/ceases operation"

Enumerator

kUartOperates UART continues to operate normally.
kUartStops UART ceases operation.

22.1.2.6 enum uart_wakeup_method_t

This provides constants for the two UART wakeup methods: idle-line or address-mark.

Enumerator

kUartIdleLineWake The idle-line wakes UART receiver from standby.
kUartAddrMarkWake The address-mark wakes UART receiver from standby.

22.1.2.7 enum uart_idle_line_select_t

This provides constants for the UART idle character bit-count start: either after start or stop bit.

Enumerator

kUartIdleLineAfterStartBit UART idle character bit count start after start bit.
kUartIdleLineAfterStopBit UART idle character bit count start after stop bit.

22.1.2.8 enum uart_break_char_length_t

This provides constants for the UART break character length for both transmission and detection purposes. Note that the actual maximum bit times may vary depending on the UART instance.

Enumerator

kUartBreakChar10BitMinimum UART break char length 10 bit times (if M = 0, SBNS = 0) or 11 (if M = 1, SBNS = 0 or M = 0, SBNS = 1) or 12 (if M = 1, SBNS = 1 or M10 = 1, SNBS = 0) or 13 (if M10 = 1, SNBS = 1)
kUartBreakChar13BitMinimum UART break char length 13 bit times (if M = 0, SBNS = 0) or 14 (if M = 1, SBNS = 0 or M = 0, SBNS = 1) or 15 (if M = 1, SBNS = 1 or M10 = 1, SNBS = 0) or 16 (if M10 = 1, SNBS = 1)

22.1.2.9 enum uart_singlewire_txdir_t

This provides constants for the UART transmit direction when configured for single-wire mode. The transmit line TXDIR is either an input or output.

Enumerator

kUartSinglewireTxdirIn UART Single-Wire mode TXDIR input.
kUartSinglewireTxdirOut UART Single-Wire mode TXDIR output.

22.1.2.10 enum uart_status_flag_t

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

Enumerator

kUartTransmitDataRegisterEmpty Transmit data register empty flag, sets when transmit buffer is empty.
kUartTransmissionComplete Transmission complete flag, sets when transmitter is idle (transmission activity complete)

kUartReceiveDataRegisterFull Receive data register full flag, sets when the receive data buffer is full.
kUartIdleLineDetect Idle line detect flag, sets when idle line detected.
kUartReceiveOverrun Receiver Overrun, sets when new data is received before data is read from receive register.
kUartNoiseDetect Receiver takes 3 samples of each received bit. If any of these samples differ, noise flag sets
kUartFrameError Frame error flag, sets if logic 0 was detected where stop bit expected.
kUartParityError If parity enabled, sets upon parity error detection.
kUartLineBreakDetect LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.
kUartReceiveActiveEdgeDetect Receive pin active edge interrupt flag, sets when active edge detected.
kUartReceiverActive Receiver Active Flag (RAF), sets at beginning of valid start bit.

22.1.2.11 enum uart_ir_tx_pulsewidth_t

This provides constants for the UART infrared (IR) pulse widths. Options include 3/16, 1/16 1/32, and 1/4 pulse widths.

Enumerator

kUartIrThreeSixteenthsWidth 3/16 pulse
kUartIrOneSixteenthWidth 1/16 pulse
kUartIrOneThirtysecondsWidth 1/32 pulse
kUartIrOneFourthWidth 1/4 pulse

22.1.3 Function Documentation

22.1.3.1 uart_status_t uart_hal_init (uint32_t *uartInstance,* const uart_config_t * *config*)

This function initializes the module to user defined settings and default settings. Here is an example demonstrating how to define the uart_config_t structure and call the uart_hal_init function:

```
uart_config_t uartConfig;
uartConfig.uartSourceClockInHz = uartSourceClock;
uartConfig.baudRate = baudRate;
uartConfig.bitCountPerChar = kUart8BitsPerChar;
uartConfig.parityMode = kUartParityDisabled;
uartConfig.stopBitCount = kUartOneStopBit;
uartConfig.txDataInvert = 0;
uartConfig.rxDataInvert = 0;
uart_hal_init(uartInstance, &uartConfig);
```

Parameters

uartInstance	UART module instance number.
config	UART configuration data.

Returns

An error code or kStatus_UART_Success.

22.1.3.2 uart_status_t uart_hal_set_baud_rate (uint32_t *uartInstance,* uint32_t *sourceClockInHz,* uint32_t *desiredBaudRate*)

This function programs the UART baud rate to the desired value passed in by the user. The user must also pass in the module source clock so that the function can calculate the baud rate divisors to their appropriate values. In some UART instances it is required that the transmitter/receiver be disabled before calling this function. Generally this is applied to all UARTs to ensure safe operation.

Parameters

uartInstance	UART module instance number.
sourceClockInHz	UART source input clock in Hz.
desiredBaudRate	UART desired baud rate.

Returns

An error code or kStatus_UART_Success

22.1.3.3 uart_status_t uart_hal_set_baud_rate_divisor (uint32_t *uartInstance,* uint32_t *baudRateDivisor*)

This function allows the user to program the baud rate divisor directly in situations where the divisor value is known. In this case, the user may not want to call the uart_hal_set_baud_rate() function, as the divisor is already known.

Parameters

uartInstance	UART module instance number.
baudRate-Divisor	The baud rate modulo division "SBR" value.

Returns

An error code or kStatus_UART_Success.

22.1.3.4 uart_status_t uart_hal_configure_bit_count_per_char (uint32_t *uartInstance,* uart_bit_count_per_char_t *bitCountPerChar*)

This function allows the user to configure the number of bits per character according to the typedef uart-_bit_count_per_char_t. In some UART instances it is required that the transmitter/receiver be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

uartInstance	UART module instance number.
bitCountPer-Char	Number of bits per char (8, 9, or 10, depending on the UART instance).

Returns

An error code or kStatus_UART_Success.

22.1.3.5 void uart_hal_configure_parity_mode (uint32_t *uartInstance,* uart_parity_mode_t *parityModeType*)

This function allows the user to configure the parity mode of the UART controller to disable it or enable it for even parity or for odd parity. In some UART instances it is required that the transmitter/receiver be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

uartInstance	UART module instance number.
parityMode-Type	Parity mode setting (enabled, disable, odd, even - see parity_mode_t struct).

22.1.3.6 uart_status_t uart_hal_configure_stop_bit_count (uint32_t *uartInstance,* uart_stop_bit_count_t *stopBitCount*)

This function allows the user to configure the number of stop bits in the UART controller to be one or two stop bits. In some UART instances it is required that the transmitter/receiver be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

uartInstance	UART module instance number.
stopBitCount	Number of stop bits setting (1 or 2 - see uart_stop_bit_count_t struct).

Returns

An error code (an unsupported setting in some UARTs) or kStatus_UART_Success.

22.1.3.7 void uart_hal_configure_tx_rx_inversion (uint32_t *uartInstance,* uint32_t *rxInvert,* uint32_t *txInvert*)

This function allows the user to invert the transmit and receive signals, independently. This function should only be called when the UART is between transmit and receive packets.

Parameters

uartInstance	UART module instance number.
rxInvert	Enable (1) or disable (0) receive inversion.
txInvert	Enable (1) or disable (0) transmit inversion.

22.1.3.8 void uart_hal_enable_transmitter (uint32_t *uartInstance*)

This function allows the user to enable the UART transmitter.

Parameters

uartInstance	UART module instance number.

22.1.3.9 void uart_hal_disable_transmitter (uint32_t *uartInstance*)

This function allows the user to disable the UART transmitter.

Parameters

uartInstance	UART module instance number.

22.1.3.10 bool uart_hal_is_transmitter_enabled (uint32_t *uartInstance*)

This function allows the user to get the setting of the UART transmitter.

Parameters

uartInstance	UART module instance number.

Returns

The state of UART transmitter enable(true)/disable(false) setting.

22.1.3.11 void uart_hal_enable_receiver (uint32_t *uartInstance*)

This function allows the user to enable the UART receiver.

Parameters

uartInstance	UART module instance number.

22.1.3.12 void uart_hal_disable_receiver (uint32_t *uartInstance*)

This function allows the user to disable the UART receiver.

Parameters

uartInstance	UART module instance number.

22.1.3.13 bool uart_hal_is_receiver_enabled (uint32_t *uartInstance*)

This function allows the user to get the setting of the UART receiver.

Parameters

uartInstance	UART module instance number.

Returns

The state of UART receiver enable(true)/disable(false) setting.

22.1.3.14 void uart_hal_configure_interrupts (uint32_t *uartInstance,* const uart_interrupt_config_t * *interruptConfig*)

This function allows the user to configure all of the UART interrupts with one function call. The user will first need to initialize and pass in a structure of type uart_interrupt_config_t which sets the configuration of each interrupt.

Parameters

uartInstance	UART module instance number.
interruptConfig	UART interrupt configuration data.

22.1.3.15 void uart_hal_enable_break_detect_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.16 void uart_hal_disable_break_detect_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.17 bool uart_hal_is_break_detect_interrupt_enabled (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

Returns

The bit setting of the interrupt enable bit.

22.1.3.18 void uart_hal_enable_rx_active_edge_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.19 void uart_hal_disable_rx_active_edge_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.20 bool uart_hal_is_rx_active_edge_interrupt_enabled (uint32_t *uartInstance*)

Parameters

uartInstance	UART instance number.

Returns

Bit setting of the interrupt enable bit.

22.1.3.21 void uart_hal_enable_tx_data_register_empty_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.22 void uart_hal_disable_tx_data_register_empty_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.23 bool uart_hal_is_tx_data_register_empty_interrupt_enabled (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

Returns

Bit setting of the interrupt enable bit.

22.1.3.24 void uart_hal_enable_transmission_complete_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.25 void uart_hal_disable_transmission_complete_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.26 bool uart_hal_is_transmission_complete_interrupt_enabled (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

Returns

Bit setting of the interrupt enable bit.

22.1.3.27 void uart_hal_enable_rx_data_register_full_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.28 void uart_hal_disable_rx_data_register_full_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.29 bool uart_hal_is_receive_data_full_interrupt_enabled (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

Returns

Bit setting of the interrupt enable bit.

22.1.3.30 void uart_hal_enable_idle_line_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.31 void uart_hal_disable_idle_line_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.32 bool uart_hal_is_idle_line_interrupt_enabled (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

Returns

Bit setting of the interrupt enable bit.

22.1.3.33 void uart_hal_enable_rx_overrun_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.34 void uart_hal_disable_rx_overrun_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.35 bool uart_hal_is_rx_overrun_interrupt_enabled (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

Returns

Bit setting of the interrupt enable bit .

22.1.3.36 void uart_hal_enable_noise_error_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.37 void uart_hal_disable_noise_error_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.38 bool uart_hal_is_noise_error_interrupt_enabled (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

Returns

Bit setting of the interrupt enable bit.

22.1.3.39 void uart_hal_enable_framing_error_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.40 void uart_hal_disable_framing_error_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.41 bool uart_hal_is_framing_error_interrupt_enabled (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

Returns

Bit setting of the interrupt enable bit.

22.1.3.42 void uart_hal_enable_parity_error_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.43 void uart_hal_disable_parity_error_interrupt (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

22.1.3.44 bool uart_hal_is_parity_error_interrupt_enabled (uint32_t *uartInstance*)

Parameters

uartInstance	UART module instance number.

Returns

Bit setting of the interrupt enable bit.

22.1.3.45 void uart_hal_configure_dma (uint32_t *uartInstance,* bool *txDmaConfig,* bool *rxDmaConfig*)

This function allows the user to configure the transmit data register empty flag to generate an interrupt request (default) or a DMA request. Similarly, this function allows the user to configure the receive data register full flag to generate an interrupt request (default) or a DMA request.

Parameters

uartInstance	UART module instance number.
txDmaConfig	Transmit DMA request configuration setting (enable: true /disable: false).
rxDmaConfig	Receive DMA request configuration setting (enable: true/disable: false).

22.1.3.46 bool uart_hal_is_txdma_enabled (uint32_t *uartInstance*)

This function returns the configuration setting of the Transmit DMA request.

Parameters

uartInstance	UART module instance number.

Returns

Transmit DMA request configuration setting (enable: true /disable: false).

22.1.3.47 bool uart_hal_is_rxdma_enabled (uint32_t *uartInstance*)

This function returns the configuration setting of the Receive DMA request.

Parameters

uartInstance	UART module instance number.

Returns

Receive DMA request configuration setting (enable: true /disable: false).

22.1.3.48 void uart_hal_putchar (uint32_t *uartInstance,* uint8_t *data*)

Parameters

uartInstance	UART module instance number.
data	The data to send of size 8-bit.

22.1.3.49 void uart_hal_putchar9 (uint32_t *uartInstance,* uint16_t *data*)

Parameters

uartInstance	UART module instance number.
data	The data to send of size 9-bit.

22.1.3.50 uart_status_t uart_hal_putchar10 (uint32_t *uartInstance,* uint16_t *data*)

```
(Note: Feature available on select UART instances)
```

Parameters

uartInstance	UART module instance number.
data	The data to send of size 10-bit.

Returns

An error code or kStatus_UART_Success.

22.1.3.51 void uart_hal_getchar (uint32_t *uartInstance,* uint8_t * *readData*)

Parameters

uartInstance	UART module instance number.
readData	The received data read from data register of size 8-bit.

22.1.3.52 void uart_hal_getchar9 (uint32_t *uartInstance,* uint16_t * *readData*)

Parameters

uartInstance	UART module instance number.
readData	The received data read from data register of size 9-bit.

22.1.3.53 uart_status_t uart_hal_getchar10 (uint32_t *uartInstance,* uint16_t * *readData*)

```
(Note: Feature available on select UART instances)
```

Parameters

uartInstance	UART module instance number.
readData	The received data read from data register of size 10-bit.

Returns

An error code or kStatus_UART_Success.

22.1.3.54 void uart_hal_configure_wait_mode_operation (uint32_t *uartInstance,* uart_operation_config_t *mode*)

The function configures the UART to either operate or cease to operate when WAIT mode is entered. In some UART instances it is required that the transmitter/receiver be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

uartInstance	UART module instance number.

mode	The UART WAIT mode operation - operates or ceases to operate in WAIT mode.

22.1.3.55 uart_operation_config_t uart_hal_get_wait_mode_operation_config (uint32_t *uartInstance*)

This function returns kUartOperates if the UART has been configured to operate in WAIT mode. Else it returns KUartStops if the UART has been configured to cease-to-operate in WAIT mode.

Parameters

uartInstance	UART module instance number.

Returns

The UART WAIT mode operation configuration, returns either kUartOperates or KUartStops.

22.1.3.56 void uart_hal_configure_loopback_mode (uint32_t *uartInstance,* bool *enable*)

This function enables or disables the UART loopback operation. In some UART instances it is required that the transmitter/receiver be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

uartInstance	UART module instance number.
enable	The UART loopback mode configuration, either disabled (false) or enabled (true).

22.1.3.57 void uart_hal_configure_singlewire_mode (uint32_t *uartInstance,* bool *enable*)

This function enables or disables the UART single-wire operation. In some UART instances it is required that the transmitter/receiver be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

uartInstance	UART module instance number.
enable	The UART single-wire mode configuration, either disabled (false) or enabled (true).

22.1.3.58 void uart_hal_configure_txdir_in_singlewire_mode (uint32_t *uartInstance,* uart_singlewire_txdir_t *direction*)

This function configures the transmitter direction when the UART is configured for single-wire operation.

Parameters

uartInstance	UART module instance number.
direction	The UART single-wire mode transmit direction configuration of type uart_-singlewire_txdir_t (either kUartSinglewireTxdirIn or kUartSinglewireTxdirOut.

22.1.3.59 uart_status_t uart_hal_put_receiver_in_standby_mode (uint32_t *uartInstance*)

This function, when called, places the UART receiver into standby mode. In some UART instances, there are conditions that must be met before placing rx in standby mode. Before placing UART in standby, determine if receiver is set to wake on idle, and if receiver is already in idle state. NOTE: RWU should only be set with C1[WAKE] = 0 (wakeup on idle) if the channel is currently not idle. This can be determined by the S2[RAF] flag. If set to wake up FROM an IDLE event and the channel is already idle, it is possible that the UART will discard data because data must be received (or a LIN break detect) after an IDLE is detected before IDLE is allowed to be reasserted.

Parameters

uartInstance	UART module instance number.

Returns

Error code or kStatus_UART_Success.

22.1.3.60 void uart_hal_put_receiver_in_normal_mode (uint32_t *uartInstance*)

This function, when called, places the UART receiver into normal mode and out of standby mode.

Parameters

uartInstance	UART module instance number.

22.1.3.61 bool uart_hal_is_receiver_in_standby (uint32_t *uartInstance*)

This function determines the state of the UART receiver. If it returns true, this means that the UART receiver is in standby mode; if it returns false, the UART receiver is in normal mode.

Parameters

uartInstance	UART module instance number.

Returns

The UART receiver is in normal mode (false) or standby mode (true).

22.1.3.62 void uart_hal_select_receiver_wakeup_method (uint32_t *uartInstance,* uart_wakeup_method_t *method*)

This function configures the wakeup method of the UART receiver from standby mode. The options are idle-line wake or address-mark wake.

Parameters

uartInstance	UART module instance number.
method	The UART receiver wakeup method options: kUartIdleLineWake - Idle-line wake or kUartAddrMarkWake - address-mark wake.

22.1.3.63 uart_wakeup_method_t uart_hal_get_receiver_wakeup_method (uint32_t *uartInstance*)

This function returns how the UART receiver is configured to wake from standby mode. The wake method options that can be returned are kUartIdleLineWake or kUartAddrMarkWake.

Parameters

uartInstance	UART module instance number.

Returns

The UART receiver wakeup from standby method, false: kUartIdleLineWake (idle-line wake) or true: kUartAddrMarkWake (address-mark wake).

22.1.3.64 void uart_hal_configure_idle_line_detect (uint32_t *uartInstance,* const uart_idle_line_config_t * *config*)

This function allows the user to configure the UART idle-line detect operation. There are two separate operations for the user to configure, the idle line bit-count start and the receive wake up affect on IDLE status bit. The user will pass in a structure of type uart_idle_line_config_t. In some UART instances it is required that the transmitter/receiver be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

uartInstance	UART module instance number.
config	The UART configuration pointer to the structure for idle-line detect operation of type uart_idle_line_config_t.

22.1.3.65 void uart_hal_set_break_char_transmit_length (uint32_t *uartInstance,* uart_break_char_length_t *length*)

This function allows the user to configure the UART break character transmit length. Refer to the typedef uart_break_char_length_t for setting options. In some UART instances it is required that the transmitter be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

uartInstance	UART module instance number.
length	The UART break character length setting of type uart_break_char_length_t, either a minimum 10-bit times or a minimum 13-bit times.

22.1.3.66 void uart_hal_set_break_char_detect_length (uint32_t *uartInstance,* uart_break_char_length_t *length*)

This function allows the user to configure the UART break character detect length. Refer to the typedef uart_break_char_length_t for setting options.

Parameters

uartInstance	UART module instance number.
length	The UART break character length setting of type uart_break_char_length_t, either a minimum 10-bit times or a minimum 13-bit times.

22.1.3.67 void uart_hal_queue_break_char_to_send (uint32_t *uartInstance,* bool *enable*)

This function allows the user to queue a UART break character to send. If true is passed into the function, then a break character is queued for transmission. A break character will continuously be queued until this function is called again when a false is passed into this function.

Parameters

uartInstance	UART module instance number.
enable	If false, the UART normal/queue break character setting is disabled, which configures the UART for normal transmitter operation. If true, a break character is queued for transmission.

22.1.3.68 uart_status_t uart_hal_configure_match_address_operation (uint32_t *uartInstance,* bool *matchAddrMode1,* bool *matchAddrMode2,* uint8_t *matchAddrValue1,* uint8_t *matchAddrValue2*)

(Note: Feature available on select UART instances)

The function allows the user to configure the UART match address control operation. The user has the option to enable the match address mode and to program the match address value. There are two match address modes, each with its own enable and programmable match address value.

Parameters

uartInstance	UART module instance number.
matchAddr-Mode1	If true, this enables match address mode 1 (MAEN1), where false disables.
matchAddr-Mode2	If true, this enables match address mode 2 (MAEN2), where false disables.

matchAddr-Value1	The match address value to program for match address mode 1.
matchAddr-Value2	The match address value to program for match address mode 2.

Returns

An error code or kStatus_UART_Success.

22.1.3.69 uart_status_t uart_hal_configure_send_msb_first_operation (uint32_t *uartInstance,* bool *enable*)

The function allows the user to configure the UART to send data MSB first or LSB first. In some UART instances it is required that the transmitter/receiver be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

uartInstance	UART module instance number.
enable	This configures send MSB first mode configuration. If true, the data is sent MSB first; if false, it is sent LSB first.

Returns

An error code or kStatus_UART_Success.

22.1.3.70 uart_status_t uart_hal_configure_receive_resync_disable_operation (uint32_t *uartInstance,* bool *enable*)

```
(Note: Feature available on select UART instances)
```

This function allows the user to disable the UART resync of received data. The default setting is false, meaning that resynchronization during the received data word is supported. If the user passes in true, this disables resynchronization during the received data word.

Parameters

uartInstance	UART module instance number.
enable	If false, then resynchronization during the received data word is supported. If true, resynchronization during the received data word is disabled.

Returns

An error code or kStatus_UART_Success.

22.1.3.71 void uart_hal_get_all_status_flag (uint32_t *uartInstance,* uart_status_flag_all_t * *allStatusFlag*)

This function gets all of the UART status flag states and places into a structure of type uart_status_flag_-all_t. The user must pass in a pointer to this structure.

Parameters

uartInstance	UART module instance number.
allStatusFlag	Pointer to the structure of all the UART status flags states.

22.1.3.72 bool uart_hal_is_transmit_data_register_empty (uint32_t *uartInstance*)

This function returns the state of the UART Transmit data register empty flag.

Parameters

uartInstance	UART module instance number.

Returns

The status of Transmit data register empty flag, which is set when transmit buffer is empty.

22.1.3.73 bool uart_hal_is_transmission_complete (uint32_t *uartInstance*)

This function returns the state of the UART Transmission complete flag.

Parameters

uartInstance	UART module instance number.

Returns

The status of Transmission complete flag, which is set when the transmitter is idle (transmission activity complete).

22.1.3.74 bool uart_hal_is_receive_data_register_full (uint32_t *uartInstance*)

This function returns the state of the UART Receive data register full flag.

Parameters

uartInstance	UART module instance number.

Returns

The status of Receive data register full flag, which is set when the receive data buffer is full.

22.1.3.75 bool uart_hal_is_idle_line_detected (uint32_t *uartInstance*)

This function returns the state of the UART Idle-line detect flag.

Parameters

uartInstance	UART module instance number.

Returns

The status of Idle-line detect flag which is set when an idle-line detected.

22.1.3.76 bool uart_hal_is_receive_overrun_detected (uint32_t *uartInstance*)

This function returns the state of the the UART Receiver Overrun status flag.

Parameters

uartInstance	UART module instance number.

Returns

The status of Receiver Overrun, which is set when new data is received before data is read from receive register.

22.1.3.77 bool uart_hal_is_noise_detected (uint32_t *uartInstance*)

This function returns the state of the UART noise status flag.

Parameters

uartInstance	UART module instance number.

Returns

The status of the noise flag, which is set if any of the 3 samples taken on receive differ.

22.1.3.78 bool uart_hal_is_frame_error_detected (uint32_t *uartInstance*)

This function returns the state of the UART Frame error status flag.

Parameters

uartInstance	UART module instance number.

Returns

The status of Frame error flag, which is set if a logic 0 was detected where a stop bit was expected.

22.1.3.79 bool uart_hal_is_parity_error_detected (uint32_t *uartInstance*)

This function returns the state of the UART parity error status flag.

Parameters

uartInstance	UART module instance number.

Returns

The status of parity error detection flag, which is set if parity mode enabled and the parity bit received does not match what was expected.

22.1.3.80 bool uart_hal_is_line_break_detected (uint32_t *uartInstance*)

This function returns the state of the UART LIN break detect interrupt status flag.

Parameters

uartInstance	UART module instance number.

Returns

The status of LIN break detect interrupt flag, which is set when the LIN break char is detected assuming the LIN circuit is enabled.

22.1.3.81 bool uart_hal_is_receive_active_edge_detected (uint32_t *uartInstance*)

This function returns the state of the UART Receive pin active edge interrupt status flag.

Parameters

uartInstance	UART module instance number.

Returns

The status of Receive pin active edge interrupt flag, which is set when active edge detected on the receive pin.

22.1.3.82 bool uart_hal_is_receiver_active (uint32_t *uartInstance*)

This function returns the state of the UART Receiver Active Flag (RAF).

Parameters

uartInstance	UART module instance number.

Returns

The status of the Receiver Active Flag (RAF), which is set at the beginning of a received valid start bit.

22.1.3.83 uart_status_t uart_hal_clear_status_flag (uint32_t *uartInstance,* uart_status_flag_t *statusFlag*)

This function allows the user to clear an individual and specific UART status flag. Refer to structure definition uart_status_flag_t for list of status bits.

Parameters

uartInstance	UART module instance number.
statusFlag	The desired UART status flag to clear.

Returns

An error code or kStatus_UART_Success.

22.1.3.84 void uart_hal_clear_all_non_autoclear_status_flags (uint32_t *uartInstance*)

This function tries to clear all of the UART status flags. In some cases, some of the status flags may not get cleared because the condition that set the flag may still exist.

Parameters

uartInstance	UART module instance number.

22.2 UART Peripheral Driver

The chapter describes the programming interface of the UART Peripheral driver.

Data Structures

- struct uart_state_t
  Runtime state of the UART driver. More...
- struct uart_user_config_t
  User configuration structure for UART driver. More...

UART Driver

- uart_status_t uart_init (uint32_t uartInstance, uart_state_t *uartState, const uart_user_config_t *uartUserConfig)
  This function initializes a UART instance for operation.
- uart_status_t uart_send_data (uart_state_t *uartState, const uint8_t *sendBuffer, uint32_t txByteCount, uint32_t timeout)
  This function sends (transmits) data out through the UART module using a blocking method.
- uart_status_t uart_send_data_async (uart_state_t *uartState, const uint8_t *sendBuffer, uint32_t txByteCount)
  This function sends (transmits) data through the UART module using a non-blocking method.
- uart_status_t uart_get_transmit_status (uart_state_t *uartState, uint32_t *bytesTransmitted)
  This function returns whether the previous UART transmit has finished.
- uart_status_t uart_get_receive_status (uart_state_t *uartState, uint32_t *bytesReceived)
  This function returns whether the previous UART receive is complete.
- void uart_shutdown (uart_state_t *uartState)
  This function shuts down the UART by disabling interrupts and the transmitter/receiver.
- uart_status_t uart_receive_data (uart_state_t *uartState, uint8_t *rxBuffer, uint32_t requestedByteCount, uint32_t timeout)
  This function gets (receives) data from the UART module using a blocking method.
- uart_status_t uart_receive_data_async (uart_state_t *uartState, uint8_t *rxBuffer, uint32_t requestedByteCount)
  This function gets (receives) data from the UART module using a non-blocking method.
- uart_status_t uart_abort_sending_data (uart_state_t *uartState)
  This function terminates an asynchronous UART transmission early.
- uart_status_t uart_abort_receiving_data (uart_state_t *uartState)
  This function terminates an asynchronous UART receive early.

22.2.0.85 UART Peripheral Driver

Overview

The UART peripheral driver is used to transfer data to and from external devices on the Universal Asynchronous Receiver/Transmitter (UART) serial bus. It provides a way to transmit and receive buffers of data with a single function call.

Device structs

The driver uses an instantiation of the uart_state_t structure to maintain the current state of a particular UART instance module driver. This struct holds data that are used by the UART peripheral driver to communicate between the transmit and receive transfer functions and the interrupt handler. The interrupt handler also uses this information to keep track of its progress. Because the driver itself does not statically allocate memory, the caller provides memory for the driver state structure during init. The user is only responsible to pass in the memory for this run-time state structure where the UART driver will take care of filling out the members.

User config structs

The UART driver uses instances of the user configuration structure uart_user_config_t for the UART driver. This allows you to configure the most common settings of the UART with a single function call. Settings include: UART baud rate; UART parity mode: disabled (default), or even or odd; the number of stop bits; the number of bits per data word.

Initialization

To initialize the UART driver, call uart_init() and pass the instance number of the UART peripheral you want to use, memory for the run-time state structure, and a pointer to the user configuration structure. For example, to use UART0 pass a value of 0 to the init function. Then, memory for the run-time state structure. Finally, pass a user configuration structure of the type uart_user_config_t as shown here:

```
// UART configuration structure for user
typedef struct UartUserConfig {
    uint32_t baudRate;
    uart_parity_mode_t parityMode;
    uart_stop_bit_count_t stopBitCount;
    uart_bit_count_per_char_t bitCountPerChar;
} uart_user_config_t;
```

Typically, uart_user_config_t instantiation is configured as 8-bit-char, no-parity, 1-stop-bit (8-n-1) with a 9600 bps baud rate. uart_user_config_t instantiation can be easily modified to configure the UART peripheral either to a different baud rate or character transfer features. This is an example code to set up a user UART configuration instantiation:

```
uart_user_config_t uartConfig;
uartConfig.baudRate = 9600;
uartConfig.bitCountPerChar = kUart8BitsPerChar;
uartConfig.parityMode = kUartParityDisabled;
uartConfig.stopBitCount = kUartOneStopBit;
```

This is an example of how to make the uart_init() function call given the user configuration structure previously given, as well as for UART instance 0.

```
uint32_t uartInstance = 0;
uart_state_t uartState; // user provides memory for the driver state structure

uart_init(uartInstance, &uartConfig, &uartState);
```

Transfers

The driver implements transmit and receive functions to transfer buffers of data. The driver also supports two different modes for transferring data: blocking and non-blocking.

The blocking transmit and receive functions are uart_send_data() and uart_receive_data().

The non-blocking (async) transmit and receive functions are uart_send_data_async() and uart_receive_-data_async().

In all cases mentioned here, the functions are interrupt driven.

The following code examples show how to use the aforementioned functions. First, it is assumed that the UART module has been initialized as described previously in the Initialization chapter.

For blocking transfer functions transmit and receive:

```
uint8_t sourceBuff[26] ={0}; // sourceBuff can be filled out with desired data
uint8_t readBuffer[10] = {0}; // readBuffer gets filled with uart_receive_data function

uint32_t byteCount = sizeof(sourceBuff);
uint32_t readByteCount = sizeof(readBuffer);

// for each use there, set timeout as "1"
// uartState is the run-time state. Pass in memory for this
// declared previously in the initialization chapter
uart_send_data(&uartState, sourceBuff, byteCount, 1); // function won't return until transmit
       is complete
uart_receive_data(&uartState, readBuffer, 1, timeoutValue); // function won't return until
       it receives all data
```

For non-blocking (async) transfer functions transmit and receive:

```
uint8_t *pTxBuff;
uint8_t rxBuff[10];
uint32_t txCurrentByteCount, readByteCount;

// assume pTxBuff and txCurrentByteCount have been initialized
uart_send_data_async(&uartState, pTxBuff, txCurrentByteCount);

// now check on status of transmit and wait until done, the code can do something else and
// check back later, this is just an example
while (uart_get_transmit_status(&uartState, &bytesTransmittedCount) ==
      kStatus_UART_TxBusy);

// for receive, assume rxBuff is set up to receive data and readByteCount is initialized
uart_receive_data_async(&uartState, rxBuff, readByteCount);

// now check on status of receive and wait until done, the code can do something else and
// check back later, this is just an example
while (uart_get_receive_status(&uartState, &bytesReceivedCount) ==
      kStatus_UART_RxBusy);
```

22.2.1 Data Structure Documentation

22.2.1.1 struct uart_state_t

This struct holds data that are used by the UART peripheral driver to communicate between the transfer function and the interrupt handler. The interrupt handler also uses this information to keep track of its progress. The user is only responsible to pass in the memory for this run-time state structure where the UART driver will take care of filling out the members.

Data Fields

- uint32_t instance
  UART module instance number.
- volatile bool isTransmitInProgress
  True if there is an active transmit.
- bool isTransmitAsync
  Whether the transmit is asynchronous or not.
- const uint8_t * sendBuffer
  The buffer of data being sent.
- volatile size_t remainingSendByteCount
  The remaining number of bytes to be transmitted.
- volatile size_t transmittedByteCount
  Number of bytes transmitted so far.
- volatile bool isReceiveInProgress
  True if there is an active receive.
- bool isReceiveAsync
  Whether the receive is asynchronous or not.
- uint8_t * receiveBuffer
  The buffer of received data.
- volatile size_t remainingReceiveByteCount
  The remaining number of bytes to be received.
- volatile size_t receivedByteCount
  Number of bytes received so far.
- sync_object_t txIrqSync
  Used to wait for ISR to complete its TX business.
- sync_object_t rxIrqSync
  Used to wait for ISR to complete its RX business.
- uint8_t txFifoEntryCount
  Number of data word entries in TX FIFO.

22.2.1.1.0.44 Field Documentation

22.2.1.1.0.44.1 uint32_t uart_state_t::instance

22.2.1.1.0.44.2 volatile bool uart_state_t::isTransmitInProgress

22.2.1.1.0.44.3 bool uart_state_t::isTransmitAsync

22.2.1.1.0.44.4 const uint8_t* uart_state_t::sendBuffer

22.2.1.1.0.44.5 volatile size_t uart_state_t::remainingSendByteCount

22.2.1.1.0.44.6 volatile size_t uart_state_t::transmittedByteCount

22.2.1.1.0.44.7 volatile bool uart_state_t::isReceiveInProgress

22.2.1.1.0.44.8 bool uart_state_t::isReceiveAsync

22.2.1.1.0.44.9 uint8_t* uart_state_t::receiveBuffer

22.2.1.1.0.44.10 volatile size_t uart_state_t::remainingReceiveByteCount

22.2.1.1.0.44.11 volatile size_t uart_state_t::receivedByteCount

22.2.1.1.0.44.12 sync_object_t uart_state_t::txIrqSync

22.2.1.1.0.44.13 sync_object_t uart_state_t::rxIrqSync

22.2.1.1.0.44.14 uint8_t uart_state_t::txFifoEntryCount

22.2.1.2 struct uart_user_config_t

Use an instance of this struct with uart_init(). This allows you to configure the most common settings of the UART peripheral with a single function call. Settings include: UART baud rate; UART parity mode: disabled (default), or even or odd; the number of stop bits; the number of bits per data word.

Data Fields

- uint32_t baudRate
  UART baud rate.
- uart_parity_mode_t parityMode
  parity mode, disabled (default), even, odd
- uart_stop_bit_count_t stopBitCount
  number of stop bits, 1 stop bit (default) or 2 stop bits
- uart_bit_count_per_char_t bitCountPerChar
  number of bits, 8-bit (default) or 9-bit in a word (up to 10-bits in some UART instances)

22.2.2 Function Documentation

22.2.2.1 uart_status_t uart_init (uint32_t *uartInstance,* uart_state_t * *uartState,* const uart_user_config_t * *uartUserConfig*)

This function will initialize the run-time state structure to keep track of the on-going transfers, ungate the clock to the UART module, initialize the module to user defined settings and default settings, configure the IRQ state structure and enable the module-level interrupt to the core, and enable the UART module transmitter and receiver. The following is an example of how to set up the uart_state_t and the uart_user_config_t parameters and how to call the uart_init function by passing in these parameters:

```
uart_user_config_t uartConfig;
uartConfig.baudRate = 9600;
uartConfig.bitCountPerChar = kUart8BitsPerChar;
uartConfig.parityMode = kUartParityDisabled;
uartConfig.stopBitCount = kUartOneStopBit;
uart_state_t uartState;
uart_init(uartInstance, &uartState, &uartConfig);
```

Parameters

uartInstance	The UART module instance number.
uartState	A pointer to the UART driver state structure memory. The user is only responsible to pass in the memory for this run-time state structure where the UART driver will take care of filling out the members. This run-time state structure keeps track of the current transfer in progress.
uartUserConfig	The user configuration structure of type uart_user_config_t. The user is responsbile to fill out the members of this structure and to pass the pointer of this struct into this function.

Returns

An error code or kStatus_UART_Success.

22.2.2.2 uart_status_t uart_send_data (uart_state_t * *uartState,* const uint8_t * *sendBuffer,* uint32_t *txByteCount,* uint32_t *timeout*)

A blocking (also known as synchronous) function means that the function does not return until the transmit is complete. This blocking function is used to send data through the UART port.

Parameters

uartInstance	The UART module instance number.
sendBuffer	A pointer to the source buffer containing 8-bit data chars to send.
txByteCount	The number of bytes to send.
timeout	A timeout value for RTOS abstraction sync control in milli-seconds (ms).

Returns

An error code or kStatus_UART_Success.

22.2.2.3 uart_status_t uart_send_data_async (uart_state_t * *uartState,* const uint8_t * *sendBuffer,* uint32_t *txByteCount*)

A non-blocking (also known as synchronous) function means that the function returns immediately after initiating the transmit function. The application has to get the transmit status to see when the transmit is complete. In other words, after calling non-blocking (asynchronous) send function, the application must get the transmit status to check if transmit is completed or not. The asynchronous method of transmitting and receiving allows the UART to perform a full duplex operation (simultaneously transmit and receive).

Parameters

uartState	A pointer to the UART driver state structure.
sendBuffer	A pointer to the source buffer containing 8-bit data chars to send.
txByteCount	The number of bytes to send.

Returns

An error code or kStatus_UART_Success.

22.2.2.4 uart_status_t uart_get_transmit_status (uart_state_t * *uartState,* uint32_t * *bytesTransmitted*)

When performing an async transmit, the user can call this function to ascertain the state of the current transmission: in progress (or busy) or complete (success). In addition, if the transmission is still in progress, the user can obtain the number of words that have been currently transferred.

Parameters

uartState	A pointer to the UART driver state structure.
bytes-Transmitted	A pointer to a value that is filled in with the number of bytes that are sent in the active transfer.

Return values

kStatus_UART_Success	The transmit has completed successfully.
kStatus_UART_TxBusy	The transmit is still in progress. *bytesTransmitted* is filled with the number of bytes which are transmitted up to that point.

22.2.2.5 uart_status_t uart_get_receive_status (uart_state_t * *uartState,* uint32_t * *bytesReceived*)

When performing an async receive, the user can call this function to ascertain the state of the current receive progress: in progress (or busy) or complete (success). In addition, if the receive is still in progress, the user can obtain the number of words that have been currently received.

Parameters

uartState	A pointer to the UART driver state structure.
bytesReceived	A pointer to a value that is filled in with the number of bytes which are received in the active transfer.

Return values

kStatus_UART_Success	The receive has completed successfully.
kStatus_UART_RxBusy	The receive is still in progress. *bytesReceived* is filled with the number of bytes which are received up to that point.

22.2.2.6 void uart_shutdown (uart_state_t * *uartState*)

This function disables the UART interrupts, disables the transmitter and receiver, and flushes the FIFOs (for modules that support FIFOs).

Parameters

uartState	A pointer to the UART driver state structure.

22.2.2.7 uart_status_t uart_receive_data (uart_state_t * *uartState*, uint8_t * *rxBuffer*, uint32_t *requestedByteCount*, uint32_t *timeout*)

A blocking (also known as synchronous) function means that the function does not return until the receive is complete. This blocking function is used to send data through the UART port.

Parameters

uartState	A pointer to the UART driver state structure.
rxBuffer	A pointer to the buffer containing 8-bit read data chars received.
requestedByte-Count	The number of bytes to receive.
timeout	A timeout value for RTOS abstraction sync control in milli-seconds (ms).

Returns

An error code or kStatus_UART_Success.

22.2.2.8 uart_status_t uart_receive_data_async (uart_state_t * *uartState*, uint8_t * *rxBuffer*, uint32_t *requestedByteCount*)

A non-blocking (also known as synchronous) function means that the function returns immediately after initiating the receive function. The application has to get the receive status to see when the receive is complete. In other words, after calling non-blocking (asynchronous) get function, the application must get the receive status to check if receive is completed or not. The asynchronous method of transmitting and receiving allows the UART to perform a full duplex operation (simultaneously transmit and receive).

Parameters

uartState	A pointer to the UART driver state structure.
rxBuffer	A pointer to the buffer containing 8-bit read data chars received.
requestedByte-Count	The number of bytes to receive.

Returns

An error code or kStatus_UART_Success.

22.2.2.9 uart_status_t uart_abort_sending_data (uart_state_t * *uartState*)

During an async UART tranmission, the user has the option to terminate the transmission early if the transmission is still in progress.

Parameters

uartState	A pointer to the UART driver state structure.

Return values

kStatus_UART_Success	The transmit was succesful.
kStatus_UART_No-TransmitInProgress	No transmission is currently in progress.

22.2.2.10 uart_status_t uart_abort_receiving_data (uart_state_t * *uartState*)

During an async UART receive, the user has the option to terminate the receive early if the receive is still in progress.

Parameters

uartState	A pointer to the UART driver state structure.

Return values

kStatus_UART_Success	The receive was succesful.
kStatus_UART_No-TransmitInProgress	No receive is currently in progress.

Chapter 23
Watchdog Timer (WDOG)

The Kinetis SDK provides both HAL and Peripheral drivers for the Watchdog Timer (WDOG) block of Kinetis devices.

Modules

- WDOG HAL driver
  The part describes the programming interface of the WDOG HAL driver.
- WDOG Peripheral Driver
  The part describes the programming interface of the WDOG Peripheral driver.

23.1 WDOG HAL driver

The chapter describes the programming interface of the WDOG HAL driver.

Typedefs

- typedef void(∗ wdog_isr_callback_t)(void)
  Watchdog ISR callback function type.

Enumerations

- enum wdog_clock_source_t {
  kWdogDedicatedClock = 0,
  kWdogAlternateClock = 1 }
  Watchdog clock source selection.
- enum wdog_clock_prescaler_t {
  kWdogClockPrescaler1 = 0,
  kWdogClockPrescaler2 = 1,
  kWdogClockPrescaler3 = 2,
  kWdogClockPrescaler4 = 3,
  kWdogClockPrescaler5 = 4,
  kWdogClockPrescaler6 = 5,
  kWdogClockPrescaler7 = 6,
  kWdogClockPrescaler8 = 7 }
  Define the selection of the clock prescaler.

Watchdog HAL.

- void wdog_hal_enable (void)
  Enable watchdog module.
- void wdog_hal_disable (void)
  Disable watchdog module.
- static bool wdog_hal_is_enabled (void)
  Check whether WDOG is enabled.
- void wdog_hal_configure_interrupt (bool isEnabled)
  Enable and disable watchdog interrupt.
- static bool wdog_hal_is_interrupt_enabled (void)
  Check whether WDOG interrupt is enabled.
- void wdog_hal_set_clock_source (wdog_clock_source_t clockSource)
  set watchdog clock Source.
- static wdog_clock_source_t wdog_hal_get_clock_source (void)
  Get watchdog clock Source.
- void wdog_hal_configure_window_mode (bool isEnabled)
  Enable and disable watchdog window mode.
- static bool wdog_hal_is_window_mode_enabled (void)
  Check whether window mode is enabled.

- void wdog_hal_configure_register_update (bool isEnabled)
  Enable and disable watchdog write-once-only register update.
- static bool wdog_hal_is_register_update_enabled (void)
  Check whether register update is enabled.
- void wdog_hal_configure_enabled_in_cpu_debug_mode (bool isEnabled)
  Set whether watchdog is working while cpu is in debug mode.
- static bool wdog_hal_is_cpu_debug_mode_enabled (void)
  Check whether WDOG works while in CPU debug mode.
- void wdog_hal_configure_enabled_in_cpu_stop_mode (bool isEnabled)
  Set whether watchdog is working while cpu is in stop mode.
- static bool wdog_hal_is_cpu_stop_mode_enabled (void)
  Check whether WDOG works while in CPU stop mode.
- void wdog_hal_configure_enabled_in_cpu_wait_mode (bool isEnabled)
  Set whether watchdog is working while CPU is in wait mode.
- static bool wdog_hal_is_cpu_wait_mode_enabled (void)
  Check whether WDOG works while in CPU wait mode.
- static bool wdog_hal_is_interrupt_asserted (void)
  Get watchdog interrupt status.
- static void wdog_hal_clear_interrupt_flag (void)
  Clear watchdog interrupt flag.
- static void wdog_hal_set_timeout_value (uint32_t timeoutCount)
  set watchdog timeout value.
- static uint32_t wdog_hal_get_timeout_value (void)
  Get watchdog timeout value.
- static uint32_t wdog_hal_get_timer_output (void)
  Get watchdog timer output.
- static void wdog_hal_set_clock_prescaler (wdog_clock_prescaler_t clockPrescaler)
  Set watchdog clock prescaler.
- static wdog_clock_prescaler_t wdog_hal_get_clock_prescaler (void)
  Get watchdog clock prescaler.
- static void wdog_hal_set_window_value (uint32_t windowValue)
  Set watchdog window value.
- static uint32_t wdog_hal_get_window_value (void)
  Get watchdog window value.
- static void wdog_hal_unlock (void)
  Unlock watchdog register written.
- static void wdog_hal_refresh (void)
  Refresh watchdog timer.
- static void wdog_hal_reset_chip (void)
  Reset chip using watchdog.
- static uint32_t wdog_hal_get_reset_count (void)
  Get chip reset count that was reset by watchdog.
- static void wdog_hal_clear_reset_count (void)
  Clear chip reset count that was reset by watchdog.

23.1.1 Enumeration Type Documentation

23.1.1.1 enum wdog_clock_source_t

Enumerator

kWdogDedicatedClock Dedicated clock source (LPO Oscillator),1K HZ.
kWdogAlternateClock Alternate clock source, Bus clock.

23.1.1.2 enum wdog_clock_prescaler_t

Enumerator

kWdogClockPrescaler1 Divide 1, default.
kWdogClockPrescaler2 Divide 2.
kWdogClockPrescaler3 Divide 3.
kWdogClockPrescaler4 Divide 4.
kWdogClockPrescaler5 Divide 5.
kWdogClockPrescaler6 Divide 6.
kWdogClockPrescaler7 Divide 7.
kWdogClockPrescaler8 Divide 8.

23.1.2 Function Documentation

23.1.2.1 void wdog_hal_enable (void)

This function is used to enable the WDOG and must be called after all necessary configure have been set.

23.1.2.2 void wdog_hal_disable (void)

This function is used to disable the WDOG.

23.1.2.3 static bool wdog_hal_is_enabled (void) [inline], [static]

This function is used check whether WDOG is enabled.

Returns

0 means WDOG is disabled, 1 means WODG is enabled.

23.1.2.4 void wdog_hal_configure_interrupt (bool *isEnabled*)

This function is used to configure the WDOG interrupt. The configuration is saved in an internal configure buffer and written back to the register in wdog_hal_enable function, so this function must be called before wdog_hal_enable is called.

Parameters

isEnabled	0 means disable watchdog interrupt. 1 means enable watchdog interrupt.

23.1.2.5 static bool wdog_hal_is_interrupt_enabled (void) [inline], [static]

This function is used to check whether the WDOG interrupt is enabled.

Returns

0 means interrupt is disabled, 1 means interrupt is enabled.

23.1.2.6 void wdog_hal_set_clock_source (wdog_clock_source_t *clockSource*)

This function is used to set the WDOG clock source. There are two clock sources that can be used, the LPO clock and the bus clock. The configuration is saved in an internal configure buffer and written back to the register in wdog_hal_enable function, so this function must be called before wdog_hal_enable is called.

Parameters

clockSource	watchdog clock source, see wdog_clock_source_t.

23.1.2.7 static wdog_clock_source_t wdog_hal_get_clock_source (void) [inline], [static]

This function is used to get the WDOG clock source. There are two clock sources that can be used, the LPO clock and the bus clock.

Returns

watchdog clock source, see wdog_clock_source_t.

23.1.2.8 void wdog_hal_configure_window_mode (bool *isEnabled*)

This function is used to configure the WDOG window mode. The configuration is saved in an internal configure buffer and written back to the register in wdog_hal_enable function, so this function must be called before wdog_hal_enable is called.

Parameters

isEnabled	0 means disable watchdog window mode. 1 means enable watchdog window mode.

23.1.2.9 static bool wdog_hal_is_window_mode_enabled (void) [inline], [static]

This function is used to check whether the WDOG window mode is enabled.

Returns

0 means window mode is disabled, 1 means window mode is enabled.

23.1.2.10 void wdog_hal_configure_register_update (bool *isEnabled*)

This function is used to configure the WDOG register update feature. If disabled, it means that all WDOG registers will never be written again unless Power On Reset. The configuration is saved in an internal configure buffer and written back to the register in wdog_hal_enable function, so this function must be called before wdog_hal_enable is called.

Parameters

isEnabled	0 means disable watchdog write-once-only register update. 1 means enable watchdog write-once-only register update.

23.1.2.11 static bool wdog_hal_is_register_update_enabled (void) [inline], [static]

This function is used to check whether the WDOG register update is enabled.

Returns

0 means register update is disabled, 1 means register update is enabled.

23.1.2.12 void wdog_hal_configure_enabled_in_cpu_debug_mode (bool *isEnabled*)

This function is used to configure whether the WDOG is enabled in CPU debug mode. The configuration is saved in an internal configure buffer and written back to the register in wdog_hal_enable function, so this function must be called before wdog_hal_enable is called.

Parameters

isEnabled	0 means watchdog is disabled in CPU debug mode. 1 means watchdog is enabled in CPU debug mode.

23.1.2.13 static bool wdog_hal_is_cpu_debug_mode_enabled (void) [inline], [static]

This function is used to check whether WDOG works in CPU debug mode.

Returns

0 means not work while in cpu debug mode, 1 means works while in cpu debug mode.

23.1.2.14 void wdog_hal_configure_enabled_in_cpu_stop_mode (bool *isEnabled*)

This function is used to configure whether the WDOG is enabled in CPU stop mode. The configuration is saved in an internal configure buffer and written back to the register in wdog_hal_enable function, so this function must be called before wdog_hal_enable is called.

Parameters

isEnabled	0 means watchdog is disabled in CPU stop mode. 1 means watchdog is enabled in CPU stop mode.

23.1.2.15 static bool wdog_hal_is_cpu_stop_mode_enabled (void) [inline], [static]

This function is used to check whether WDOG works in CPU stop mode.

Returns

0 means not work while in CPU stop mode, 1 means works while in CPU stop mode.

23.1.2.16 void wdog_hal_configure_enabled_in_cpu_wait_mode (bool *isEnabled*)

This function is used to configure whether the WDOG is enabled in CPU wait mode. The configuration is saved in an internal configure buffer and written back to the register in wdog_hal_enable function, so this function must be called before wdog_hal_enable is called.

Parameters

isEnabled	0 means watchdog is disabled in CPU wait mode. 1 means watchdog is enabled in CPU wait mode.

23.1.2.17 static bool wdog_hal_is_cpu_wait_mode_enabled (void) [inline], [static]

This function is used to check whether WDOG works in CPU wait mode.

Returns

0 means not work while in CPU wait mode, 1 means works while in CPU wait mode.

23.1.2.18 static bool wdog_hal_is_interrupt_asserted (void) [inline], [static]

This function is used to get the WDOG interrupt flag.

Returns

Watchdog interrupt status, 0 means interrupt not asserted, 1 means interrupt asserted.

23.1.2.19 static void wdog_hal_clear_interrupt_flag (void) [inline], [static]

This function is used to clear the WDOG interrupt flag.

23.1.2.20 static void wdog_hal_set_timeout_value (uint32_t *timeoutCount*) [inline], [static]

This function is used to set the WDOG_TOVAL value.

Parameters

timeoutCount	watchdog timeout value, count of watchdog clock tick.

23.1.2.21 static uint32_t wdog_hal_get_timeout_value (void) [inline], [static]

This function is used to Get the WDOG_TOVAL value.

Returns

value of register WDOG_TOVAL.

23.1.2.22 static uint32_t wdog_hal_get_timer_output (void) [inline], [static]

This function is used to get the WDOG_TMROUT value.

Returns

Current value of watchdog timer counter.

23.1.2.23 static void wdog_hal_set_clock_prescaler (wdog_clock_prescaler_t *clockPrescaler*) [inline], [static]

This function is used to set the WDOG clock proscaler.

Parameters

clockPrescaler	watchdog clock prescaler, see wdog_clock_prescaler_t.

23.1.2.24 static wdog_clock_prescaler_t wdog_hal_get_clock_prescaler (void) [inline], [static]

This function is used to get the WDOG clock prescaler.

Returns

WDOG clock prescaler.

23.1.2.25 static void wdog_hal_set_window_value (uint32_t *windowValue*) [inline], [static]

This function is used to set the WDOG_WIN value.

Parameters

windowValue	watchdog window value.

23.1.2.26 static uint32_t wdog_hal_get_window_value (void) [inline], [static]

This function is used to Get the WDOG_WIN value.

Returns

watchdog window value.

23.1.2.27 static void wdog_hal_unlock (void) [inline], [static]

This function is used to unlock the WDOG register written. This function must be called before any configuration is set because watchdog register will be locked automatically after a WCT(256 bus cycles).

23.1.2.28 static void wdog_hal_refresh (void) [inline], [static]

This function is used to feed the WDOG. This function should be called before watchdog timer is in timeout, otherwise a RESET will assert.

23.1.2.29 static void wdog_hal_reset_chip (void) [inline], [static]

This function is used to reset chip using WDOG.

23.1.2.30 static uint32_t wdog_hal_get_reset_count (void) [inline], [static]

This function is used to get the value of WDOG_RSTCNT.

Returns

Chip reset count that was reset by watchdog.

23.1.2.31 static void wdog_hal_clear_reset_count (void) [inline], [static]

This function is used to clear the WDOG_RSTCNT.

23.2 WDOG Peripheral Driver

The chapter describes the programming interface of the WDOG Peripheral driver.

Data Structures

- struct wdog_user_config_t
  Data struct for watchdog initialize. More...

Watchdog Driver

- void wdog_init (const wdog_user_config_t *init_ptr)
  Initialize watchdog.
- void wdog_shutdown (void)
  Shutdown watchdog.
- void wdog_refresh (void)
  Refresh watchdog.
- void wdog_clear_reset_count (void)
  Clear watchdog reset count.
- bool wdog_is_running (void)
  Get watchdog running status.
- void wdog_reset_chip (void)
  Reset chip by watchdog.
- uint32_t wdog_get_reset_count (void)
  Get chip reset count that reset by watchdog.

23.2.0.32 WDOG Peripheral Driver

Overview

The WDOG driver is used to configure WDOG. It provides an easy way to make necessary module initializations and configure WDOG.

Initialization

To initialize the WDOG module, call wdog_init() and pass in the initialization data structure. This function enables the WDOG module and clock automatically.

After wdog_init() is called, the WDOG is enabled and its count is working, so wdog_refresh() should be called before WDOG times out.

This is example code for initializing and configuring the driver:

```
// Define device configuration.
const wdog_init_t wdogInit =
{
```

```
    .clockSource = kWdogDedicatedClock,          // Use dedicated clock source,LPO,1K HZ.
    .clockPrescaler = kWdogClockPrescaler1,      // Clock prescaler is 1.
    .timeOutValue = 2048,                        // Timeout count is 2048.
    .wdogCallbackFunc = NULL,                    // WDOG interrupt callback function is NULL, interrupt is
       disabled.
    .updateRegisterEnable = true,                // Enable WDOG register update after first configure.
    .cpuWaitModeEnable = true,                   // Enable WDOG while CPU is in Wait mode.
    .cpuStopModeEnable = true,                   // Enable WDOG while CPU is in Stop mode.
};

// Initialize WDOG.
WDOG_init(&wdogInit);
```

WDOG Refresh

After WDOG is enabled, the wdog_refresh() should be called periodically to prevent the WDOG from timing out, or a RESET will assert. This is called "Feed Dog".

WDOG Reset Count

The WDOG can record the reset count caused by WDOG timeout.

1. wdog_get_reset_count() gets the reset count caused by WDOG timeout.
2. wdog_clear_reset_count() clears the reset count caused by WDOG timeout.

Reset Chip

The WDOG can be used to reset the chip.

1. wdog_reset_chip() is used to reset the chip whether the WDOG is enabled or not.

23.2.1 Data Structure Documentation

23.2.1.1 struct wdog_user_config_t

This structure is used when initialize the WDOG while wdog_init function is called. It contains all configure of the WDOG.

Data Fields

- wdog_clock_source_t clockSource
  Clock source select.
- bool updateRegisterEnable
  Update write-once register enable.
- bool cpuDebugModeEnable
  Enable watchdog ini cpu debug mode.

- bool cpuWaitModeEnable
  Enable watchdog ini cpu wait mode.
- bool cpuStopModeEnable
  Enable watchdog ini cpu stop mode.
- uint32_t windowValue
  Window value.
- uint32_t timeOutValue
  Timeout value.
- wdog_clock_prescaler_t clockPrescaler
  Clock prescaler.
- wdog_isr_callback_t wdogCallbackFunc
  Isr callback function.

23.2.1.1.0.45 Field Documentation

23.2.1.1.0.45.1 wdog_isr_callback_t wdog_user_config_t::wdogCallbackFunc

must in 256 bus cycles.

23.2.2 Function Documentation

23.2.2.1 void wdog_init (const wdog_user_config_t * *init_ptr*)

This function is used to initialize the WDOG. When called, the WDOG runs immediately according to the configuration.

Parameters

init_ptr	Watchdog Initialize data structure.

23.2.2.2 void wdog_shutdown (void)

This function is used to shut down the WDOG.

23.2.2.3 void wdog_refresh (void)

This function is used to feed the WDOG. It sets the WDOG timer count to zero and should be called before watchdog timer times out, otherwise a RESET will assert.

23.2.2.4 void wdog_clear_reset_count (void)

This function sets the WDOG reset count to zero. The WDOG_RSTCNT register will only clear on Power On Reset or clear by this function.

23.2.2.5 bool wdog_is_running (void)

This function gets the WDOG running status.

Returns

watchdog running status, 0 means not running, 1 means running

23.2.2.6 void wdog_reset_chip (void)

This function is used to reset chip using WDOG.

23.2.2.7 uint32_t wdog_get_reset_count (void)

This function gets the WDOG_RSTCNT value.

Returns

Chip reset count that reset by watchdog.

Chapter 24
Multipurpose Clock Generator (MCG)

The Kinetis SDK provides a HAL driver for the Multipurpose Clock Generator (MCG) block of Kinetis devices.

Modules

- MCG HAL driver
  The part describes the programming interface of the MCG Hal driver.

24.1 MCG HAL driver

The chapter describes the programming interface of the MCG Hal driver.

Enumerations

- enum _mcg_constant
  MCG constant definitions.
- enum mcg_clock_select_t
  MCG clock source select.
- enum mcg_iref_clock_source_t
  MCG internal reference clock source select.
- enum mcg_freq_range_select_t
  MCG frequency range select.
- enum mcg_hgo_select_t
  MCG high gain oscillator select.
- enum mcg_eref_clock_select_t
  MCG high gain oscillator select.
- enum mcg_lp_select_t
  MCG low power select.
- enum mcg_iref_clock_select_t
  MCG internal reference clock select.
- enum mcg_dmx32_select_t
  MCG DCO Maximum Frequency with 32.768 kHz Reference.
- enum mcg_dco_range_select_t
  MCG DCO range select.
- enum mcg_pll_eref_clock_select_t
  MCG PLL external reference clock select.
- enum mcg_pll_select_t
  MCG PLL select.
- enum mcg_lols_status_t
  MCG loss of lock status.
- enum mcg_lock_status_t
  MCG lock status.
- enum mcg_pllst_status_t
  MCG clock status.
- enum mcg_irefst_status_t
  MCG iref status.
- enum mcg_clkst_status_t
  MCG clock mode status.
- enum mcg_ircst_status_t
  MCG ircst status.
- enum mcg_atmf_status_t
  MCG auto trim fail status.
- enum mcg_locs0_status_t
  MCG loss of clock status.
- enum mcg_atms_select_t
  MCG Automatic Trim Machine Select.
- enum mcg_oscsel_select_t
  MCG OSC Clock Select.
- enum mcg_locs1_status_t

MCG loss of clock status.
- enum mcg_pllcs_select_t
  MCG PLLCS select.
- enum mcg_locs2_status_t
  MCG loss of clock status.

MCG out clock access API

- uint32_t clock_hal_get_fllclk (void)
  Gets the current MCG FLL clock.
- uint32_t clock_hal_get_pll0clk (void)
  Gets the current MCG PLL/PLL0 clock.
- uint32_t clock_hal_get_irclk (void)
  Gets the current MCG IR clock.
- uint32_t clock_hal_get_outclk (void)
  Gets the current MCG out clock.

MCG control register access API

- static void clock_set_clks (mcg_clock_select_t select)
  Sets the Clock Source Select.
- static mcg_clock_select_t clock_get_clks (void)
  Gets the Clock Source Select.
- static void clock_set_frdiv (uint8_t setting)
  Sets the FLL External Reference Divider.
- static uint8_t clock_get_frdiv (void)
  Gets the FLL External Reference Divider.
- static void clock_set_irefs (mcg_iref_clock_source_t select)
  Sets the Internal Reference Select.
- static mcg_iref_clock_source_t clock_get_irefs (void)
  Gets the Internal Reference Select.
- static void clock_set_clks_frdiv_irefs (mcg_clock_select_t clks, uint8_t frdiv, mcg_iref_clock_source_t irefs)
  Sets the CLKS, FRDIV and IREFS at the same time.
- static void clock_set_irclken (bool enable)
  Sets the Enable Internal Reference Clock setting.
- static bool clock_get_irclken (void)
  Gets the enable Internal Reference Clock setting.
- static void clock_set_irefsten (bool enable)
  Sets the Internal Reference Clock Stop Enable setting.
- static bool clock_get_irefsten (void)
  Gets the Enable Internal Reference Clock setting.
- static void clock_set_locre0 (bool enable)
  Sets the Loss of Clock Reset Enable setting.
- static bool clock_get_locre0 (void)
  Gets the Loss of Clock Reset Enable setting.
- static void clock_set_range0 (mcg_freq_range_select_t select)
  Sets the Frequency Range Select.

- static mcg_freq_range_select_t clock_get_range0 (void)
  Gets the Frequency Range Select.
- static void clock_set_hgo0 (mcg_hgo_select_t select)
  Sets the High Gain Oscillator Select.
- static mcg_hgo_select_t clock_get_hgo0 (void)
  Gets the High Gain Oscillator Select.
- static void clock_set_erefs0 (mcg_eref_clock_select_t select)
  Sets the External Reference Select.
- static mcg_eref_clock_select_t clock_get_erefs0 (void)
  Gets the External Reference Select.
- static void clock_set_lp (mcg_lp_select_t select)
  Sets the Low Power Select.
- static mcg_lp_select_t clock_get_lp (void)
  Gets the Low Power Select.
- static void clock_set_ircs (mcg_iref_clock_select_t select)
  Sets the Internal Reference Clock Select.
- static mcg_iref_clock_select_t clock_get_ircs (void)
  Gets the Internal Reference Clock Select.
- static void clock_set_sctrim (uint8_t setting)
  Sets the Slow Internal Reference Clock Trim Setting.
- static uint8_t clock_get_sctrim (void)
  Gets the Slow Internal Reference Clock Trim Setting.
- static void clock_set_dmx32 (mcg_dmx32_select_t setting)
  Sets the DCO Maximum Frequency with 32.768 kHz Reference.
- static mcg_dmx32_select_t clock_get_dmx32 (void)
  Gets the DCO Maximum Frequency with the 32.768 kHz Reference Setting.
- static void clock_set_drst_drs (mcg_dco_range_select_t setting)
  Sets the DCO Range Select.
- static mcg_dco_range_select_t clock_get_drst_drs (void)
  Gets the DCO Range Select Setting.
- static void clock_set_fctrim (uint8_t setting)
  Sets the Fast Internal Reference Clock Trim Setting.
- static uint8_t clock_get_fctrim (void)
  Gets the Fast Internal Reference Clock Trim Setting.
- static void clock_set_scftrim (uint8_t setting)
  Sets the Slow Internal Reference Clock Fine Trim Setting.
- static uint8_t clock_get_scftrim (void)
  Gets the Slow Internal Reference Clock Fine Trim Setting.
- static void clock_set_pllclken0 (bool enable)
  Sets the PLL Clock Enable Setting.
- static bool clock_get_pllclken0 (void)
  Gets the PLL Clock Enable Setting.
- static void clock_set_pllsten0 (bool enable)
  Sets the PLL0 Stop Enable Setting.
- static bool clock_get_pllsten0 (void)
  Gets the PLL0 Stop Enable Setting.
- static void clock_set_prdiv0 (uint8_t setting)
  Sets the PLL0 External Reference Divider Setting.
- static uint8_t clock_get_prdiv0 (void)
  Gets the PLL0 External Reference Divider Setting.
- static void clock_set_lolie0 (bool enable)

Sets the Loss of Lock Interrupt Enable Setting.
- static bool clock_get_lolie0 (void)
  Gets the Loss of the Lock Interrupt Enable Setting.
- static void clock_set_plls (mcg_pll_select_t setting)
  Sets the PLL Select Setting.
- static mcg_pll_select_t clock_get_plls (void)
  Gets the PLL Select Setting.
- static void clock_set_cme0 (bool enable)
  Sets the Clock Monitor Enable Setting.
- static bool clock_get_cme0 (void)
  Gets the Clock Monitor Enable Setting.
- static void clock_set_vdiv0 (uint8_t setting)
  Sets the VCO0 Divider Setting.
- static uint8_t clock_get_vdiv0 (void)
  Gets the VCO0 Divider Setting.
- static mcg_lols_status_t clock_get_lols0 (void)
  Gets the Loss of the Lock Status.
- static mcg_lock_status_t clock_get_lock0 (void)
  Gets the Lock Status.
- static mcg_pllst_status_t clock_get_pllst (void)
  Gets the PLL Select Status.
- static mcg_irefst_status_t clock_get_irefst (void)
  Gets the Internal Reference Status.
- static mcg_clkst_status_t clock_get_clkst (void)
  Gets the Clock Mode Status.
- static uint8_t clock_get_oscinit0 (void)
  Gets the OSC Initialization Status.
- static mcg_ircst_status_t clock_get_ircst (void)
  Gets the Internal Reference Clock Status.
- static mcg_atmf_status_t clock_get_atmf (void)
  Gets the Automatic Trim machine Fail Flag.
- static void clock_set_atmf (void)
  Sets the Automatic Trim machine Fail Flag.
- static mcg_locs0_status_t clock_get_locs0 (void)
  Gets the OSC0 Loss of Clock Status.
- static void clock_set_atme (bool enable)
  Sets the Automatic Trim Machine Enable Setting.
- static bool clock_get_atme (void)
  Gets the Automatic Trim Machine Enable Setting.
- static void clock_set_atms (mcg_atms_select_t setting)
  Sets the Automatic Trim Machine Select Setting.
- static mcg_atms_select_t clock_get_atms (void)
  Gets the Automatic Trim Machine Select Setting.
- static void clock_set_fltprsrv (bool enable)
  Sets the FLL Filter Preserve Enable Setting.
- static bool clock_get_fltprsrv (void)
  Gets the FLL Filter Preserve Enable Setting.
- static void clock_set_fcrdiv (uint8_t setting)
  Sets the Fast Clock Internal Reference Divider Setting.
- static uint8_t clock_get_fcrdiv (void)
  Gets the Fast Clock Internal Reference Divider Setting.

- static void clock_set_atcvh (uint8_t setting)
  Sets the ATM Compare Value High Setting.
- static uint8_t clock_get_atcvh (void)
  Gets the ATM Compare Value High Setting.
- static void clock_set_atcvl (uint8_t setting)
  Sets the ATM Compare Value Low Setting.
- static uint8_t clock_get_atcvl (void)
  Gets the ATM Compare Value Low Setting.

24.1.0.8 MCG Hal Driver

Overview

The multipurpose clock generator (MCG) module provides a several clock source choices for the MCU. The MCG HAL provides a set of APIs to accessing these registers.

Get current default reference clock frequency

The MCG HAL provides a set of APIs dedicated to get the system default clock frequency. For example, MCG out clock, FLL clock, PLL clock, etc.

Example to get a default system reference clock:

```
#include "mcg/hal/fsl_mcg_hal.h"

// return frequncy value for specified clock name
uint32_t frequency = 0;

// get the current system default reference clock frequncy
frequency = clock_hal_get_outclk();
```

24.1.1 Function Documentation

24.1.1.1 uint32_t clock_hal_get_fllclk (void)

This function returns the mcgfllclk value in frequency(Hertz) based on the current MCG configurations and settings. FLL should be properly configured in order to get the valid value.

Parameters

none	

Returns

value Frequency value in Hertz of the mcgpllclk.

24.1.1.2 uint32_t clock_hal_get_pll0clk (void)

This function returns the mcgpllclk/mcgpll0 value in frequency(Hertz) based on the current MCG configurations and settings. PLL/PLL0 should be properly configured in order to get the valid value.

Parameters

none	

Returns

value Frequency value in Hertz of the mcgpllclk or the mcgpll0clk.

24.1.1.3 uint32_t clock_hal_get_irclk (void)

This function returns the mcgirclk value in frequency (Hertz) based on the current MCG configurations and settings. It does not check if the mcgirclk is enabled or not, just calculate and return the value.

Parameters

none	

Returns

value Frequency value in Hertz of the mcgirclk.

24.1.1.4 uint32_t clock_hal_get_outclk (void)

This function returns the mcgoutclk value in frequency (Hertz) based on the current MCG configurations and settings. The configuration should be properly done in order to get the valid value.

Parameters

none	

Returns

value Frequency value in Hertz of mcgoutclk.

24.1.1.5 static void clock_set_clks (mcg_clock_select_t *select*) [inline], [static]

This function selects the clock source for the MCGOUTCLK.

Parameters

select	Clock source selection • 00: Output of FLL or PLLCS is selected(depends on PLLS control bit) • 01: Internal reference clock is selected. • 10: External reference clock is selected. • 11: Reserved.

24.1.1.6 static mcg_clock_select_t clock_get_clks (void) [inline], [static]

This function gets the select of the clock source for the MCGOUTCLK.

Returns

select Clock source selection

24.1.1.7 static void clock_set_frdiv (uint8_t *setting*) [inline], [static]

This function sets the FLL External Reference Divider.

Parameters

setting	Divider setting

24.1.1.8 static uint8_t clock_get_frdiv (void) [inline], [static]

This function gets the FLL External Reference Divider.

Returns

setting Divider setting

24.1.1.9 static void clock_set_irefs (mcg_iref_clock_source_t *select*) [inline], [static]

This function selects the reference clock source for the FLL.

Parameters

select	Clock source select • 0: External reference clock is selected • 1: The slow internal reference clock is selected

24.1.1.10 static mcg_iref_clock_source_t clock_get_irefs (void) [inline], [static]

This function gets the reference clock source for the FLL.

Returns

select Clock source select

24.1.1.11 static void clock_set_clks_frdiv_irefs (mcg_clock_select_t *clks,* uint8_t *frdiv,* mcg_iref_clock_source_t *irefs*) [inline], [static]

This function sets the CLKS, FRDIV, and IREFS settings at the same time in order keep the integrity of the clock switching.

Parameters

clks	Clock source select
frdiv	FLL external reference divider select
irefs	Internal reference select

24.1.1.12 static void clock_set_irclken (bool *enable*) [inline], [static]

This function enables/disables the internal reference clock to use as the MCGIRCLK.

enable Enable or disable internal reference clock.

- true: MCGIRCLK active
- false: MCGIRCLK inactive

24.1.1.13 static bool clock_get_irclken (void) [inline], [static]

This function gets the reference clock enable setting.

Returns

enabled True if the internal reference clock is enabled.

24.1.1.14 static void clock_set_irefsten (bool *enable*) [inline], [static]

This function controls whether or not the internal reference clock remains enabled when the MCG enters Stop mode.

enable Enable or disable the internal reference clock stop setting.

- true: Internal reference clock is enabled in Stop mode if IRCLKEN is set or if MCG is in FEI, FBI, or BLPI modes before entering Stop mode.
- false: Internal reference clock is disabled in Stop mode

24.1.1.15 static bool clock_get_irefsten (void) [inline], [static]

This function gets the Internal Reference Clock Stop Enable setting.

Returns

enabled True if internal reference clock stop is enabled.

24.1.1.16 static void clock_set_locre0 (bool *enable*) [inline], [static]

This function determines whether an interrupt or a reset request is made following a loss of the OSC0 external reference clock. The LOCRE0 only has an affect when CME0 is set.

enable Loss of Clock Reset Enable setting

- true: Generate a reset request on a loss of OSC0 external reference clock
- false: Interrupt request is generated on a loss of OSC0 external reference clock

24.1.1.17 static bool clock_get_locre0 (void) [inline], [static]

This function gets the Loss of Clock Reset Enable setting.

Returns

enabled True if Loss of Clock Reset is enabled.

24.1.1.18 static void clock_set_range0 (mcg_freq_range_select_t *select*) [inline], [static]

This function selects the frequency range for the crystal oscillator or an external clock source. See the Oscillator (OSC) chapter for more details and the device data sheet for the frequency ranges used.

select Frequency Range Select

- 00: Low frequency range selected for the crystal oscillator
- 01: High frequency range selected for the crystal oscillator
- 1X: Very high frequency range selected for the crystal oscillator

24.1.1.19 static mcg_freq_range_select_t clock_get_range0 (void) [inline], [static]

This function gets the Frequency Range Select.

Returns

select Frequency Range Select

24.1.1.20 static void clock_set_hgo0 (mcg_hgo_select_t *select*) [inline], [static]

This function controls the crystal oscillator mode of operation. See the Oscillator (OSC) chapter for more details.

select High Gain Oscillator Select.

- 0: Configure crystal oscillator for low-power operation
- 1: Configure crystal oscillator for high-gain operation

24.1.1.21 static mcg_hgo_select_t clock_get_hgo0 (void) [inline], [static]

This function gets the High Gain Oscillator Select.

Returns

select High Gain Oscillator Select

24.1.1.22 static void clock_set_erefs0 (mcg_eref_clock_select_t *select*) [inline], [static]

This function selects the source for the external reference clock. See the Oscillator (OSC) chapter for more details.

select External Reference Select

- 0: External reference clock requested
- 1: Oscillator requested

24.1.1.23 static mcg_eref_clock_select_t clock_get_erefs0 (void) [inline], [static]

This function gets the External Reference Select.

Returns

select External Reference Select

24.1.1.24 static void clock_set_lp (mcg_lp_select_t *select*) [inline], [static]

This function controls whether the FLL (or PLL) is disabled in the BLPI and the BLPE modes. In the FBE or the PBE modes, setting this bit to 1 transitions the MCG into the BLPE mode; in the FBI mode, setting this bit to 1 transitions the MCG into the BLPI mode. In any other MCG mode, the LP bit has no affect..

select Low Power Select

- 0: FLL (or PLL) is not disabled in bypass modes
- 1: FLL (or PLL) is disabled in bypass modes (lower power)

24.1.1.25 static mcg_lp_select_t clock_get_lp (void) [inline], [static]

This function gets the Low Power Select.

Returns

select Low Power Select

24.1.1.26 static void clock_set_ircs (mcg_iref_clock_select_t *select*) [inline], [static]

This function selects between the fast or slow internal reference clock source.

select Low Power Select

- 0: Slow internal reference clock selected.
- 1: Fast internal reference clock selected.

24.1.1.27 static mcg_iref_clock_select_t clock_get_ircs (void) [inline], [static]

This function gets the Internal Reference Clock Select.

Returns

select Internal Reference Clock Select

24.1.1.28 static void clock_set_sctrim (uint8_t *setting*) [inline], [static]

This function controls the slow internal reference clock frequency by controlling the slow internal reference clock period. The SCTRIM bits are binary weighted (that is, bit 1 adjusts twice as much as bit 0). Increasing the binary value increases the period, and decreasing the value decreases the period. An additional fine trim bit is available in the C4 register as the SCFTRIM bit. Upon reset, this value is loaded with a factory trim value. If an SCTRIM value stored in non-volatile memory is to be used, it is the user's responsibility to copy that value from the non-volatile memory location to this register.

setting Slow Internal Reference Clock Trim Setting

24.1.1.29 static uint8_t clock_get_sctrim (void) [inline], [static]

This function gets the Slow Internal Reference Clock Trim Setting.

Returns

setting Slow Internal Reference Clock Trim Setting

24.1.1.30 static void clock_set_dmx32 (mcg_dmx32_select_t *setting*) [inline], [static]

This function controls whether or not the DCO frequency range is narrowed to its maximum frequency with a 32.768 kHz reference.

setting DCO Maximum Frequency with 32.768 kHz Reference Setting

- 0: DCO has a default range of 25%.
- 1: DCO is fine-tuned for maximum frequency with 32.768 kHz reference.

24.1.1.31 static mcg_dmx32_select_t clock_get_dmx32 (void) [inline], [static]

This function gets the DCO Maximum Frequency with 32.768 kHz Reference Setting.

Returns

setting DCO Maximum Frequency with 32.768 kHz Reference Setting

24.1.1.32 static void clock_set_drst_drs (mcg_dco_range_select_t *setting*) [inline], [static]

This function selects the frequency range for the FLL output, DCOOUT. When the LP bit is set, the writes to the DRS bits are ignored. The DRST read field indicates the current frequency range for the DCOOUT. The DRST field does not update immediately after a write to the DRS field due to internal synchronization between the clock domains. See the DCO Frequency Range table for more details.

setting DCO Range Select Setting

- 00: Low range (reset default).
- 01: Mid range.
- 10: Mid-high range.
- 11: High range.

24.1.1.33 static mcg_dco_range_select_t clock_get_drst_drs (void) [inline], [static]

This function gets the DCO Range Select Setting.

Returns

setting DCO Range Select Setting

24.1.1.34 static void clock_set_fctrim (uint8_t *setting*) [inline], [static]

This function controls the fast internal reference clock frequency by controlling the fast internal reference clock period. The FCTRIM bits are binary weighted (that is, bit 1 adjusts twice as much as bit 0). Increasing the binary value increases the period, and decreasing the value decreases the period. If an FCTRIM[3:0] value stored in non-volatile memory is to be used, it is the user's responsibility to copy that value from the non-volatile memory location to this register.

setting Fast Internal Reference Clock Trim Setting.

24.1.1.35 static uint8_t clock_get_fctrim (void) [inline], [static]

This function gets the Fast Internal Reference Clock Trim Setting.

Returns

setting Fast Internal Reference Clock Trim Setting

24.1.1.36 static void clock_set_scftrim (uint8_t *setting*) [inline], [static]

This function controls the smallest adjustment of the slow internal reference clock frequency. Setting the SCFTRIM increases the period and clearing the SCFTRIM decreases the period by the smallest amount possible. If an SCFTRIM value, stored in non-volatile memory, is to be used, it is the user's responsibility to copy that value from the non-volatile memory location to this bit.

setting Slow Internal Reference Clock Fine Trim Setting

24.1.1.37 static uint8_t clock_get_scftrim (void) [inline], [static]

This function gets the Slow Internal Reference Clock Fine Trim Setting.

Returns

setting Slow Internal Reference Clock Fine Trim Setting

24.1.1.38 static void clock_set_pllclken0 (bool *enable*) [inline], [static]

This function enables/disables the PLL0 independent of the PLLS and enables the PLL0 clock to use as the MCGPLL0CLK and the MCGPLL0CLK2X. (PRDIV0 needs to be programmed to the correct divider to generate a PLL1 reference clock in a valid reference range prior to setting the PLLCLKEN0 bit). Setting PLLCLKEN0 enables the external oscillator selected by REFSEL if not already enabled. Whenever the PLL0 is being enabled with the PLLCLKEN0 bit, and the external oscillator is being used as the reference clock, the OSCINIT 0 bit should be checked to make sure it is set.

enable PLL Clock Enable Setting

- true: MCGPLL0CLK and MCGPLL0CLK2X are active
- false: MCGPLL0CLK and MCGPLL0CLK2X are inactive

24.1.1.39 static bool clock_get_pllclken0 (void) [inline], [static]

This function gets the PLL Clock Enable Setting.

Returns

enabled True if PLL0 PLL Clock is enabled.

24.1.1.40 static void clock_set_pllsten0 (bool *enable*) [inline], [static]

This function enables/disables the PLL0 Clock during a Normal Stop (In Low Power Stop mode, the PLL0 clock gets disabled even if PLLSTEN0=1). In all other power modes, the PLLSTEN0 bit has no affect and does not enable the PLL0 Clock to run if it is written to 1.

enable PLL0 Stop Enable Setting

- true: MCGPLL0CLK and MCGPLL0CLK2X are enabled if system is in Normal Stop mode.
- false: MCGPLL0CLK and MCGPLL0CLK2X are disabled in any of the Stop modes.

24.1.1.41 static bool clock_get_pllsten0 (void) [inline], [static]

This function gets the PLL0 Stop Enable Setting.

Returns

enabled True if the PLL0 Stop is enabled.

24.1.1.42 static void clock_set_prdiv0 (uint8_t *setting*) [inline], [static]

This function selects the amount to divide down the external reference clock for the PLL0. The resulting frequency must be in a valid reference range. After the PLL0 is enabled, (by setting either PLLCLKEN0 or PLLS), the PRDIV0 value must not be changed when LOCK0 is zero.

setting PLL0 External Reference Divider Setting

24.1.1.43 static uint8_t clock_get_prdiv0 (void) [inline], [static]

This function gets the PLL0 External Reference Divider Setting.

Returns

setting PLL0 External Reference Divider Setting

24.1.1.44 static void clock_set_lolie0 (bool *enable*) [inline], [static]

This function determine whether an interrupt request is made following a loss of lock indication. This bit only has an effect when LOLS 0 is set.

enable Loss of Lock Interrupt Enable Setting

- true: Generate an interrupt request on loss of lock.
- false: No interrupt request is generated on loss of lock.

24.1.1.45 static bool clock_get_lolie0 (void) [inline], [static]

This function gets the Loss of the Lock Interrupt Enable Setting.

Returns

enabled True if the Loss of Lock Interrupt is enabled.

24.1.1.46 static void clock_set_plls (mcg_pll_select_t *setting*) [inline], [static]

This function controls whether the PLLCS or FLL output is selected as the MCG source when CLKS[1-:0]=00. If the PLLS bit is cleared and PLLCLKEN0 and PLLCLKEN1 is not set, the PLLCS output clock is disabled in all modes. If the PLLS is set, the FLL is disabled in all modes.

setting PLL Select Setting

- 0: FLL is selected.
- 1: PLLCS output clock is selected (PRDIV0 bits of PLL in control need to be programmed to the correct divider to generate a PLL reference clock in the range of 1 - 32 MHz prior to setting the PLLS bit).

24.1.1.47 static mcg_pll_select_t clock_get_plls (void) [inline], [static]

This function gets the PLL Select Setting.

Returns

setting PLL Select Setting

24.1.1.48 static void clock_set_cme0 (bool *enable*) [inline], [static]

This function enables/disables the loss of clock monitoring circuit for the OSC0 external reference mux select. The LOCRE0 bit determines whether an interrupt or a reset request is generated following a loss of the OSC0 indication. The CME0 bit should only be set to a logic 1 when the MCG is in an operational mode that uses the external clock (FEE, FBE, PEE, PBE, or BLPE). Whenever the CME0 bit is set to a logic 1, the value of the RANGE0 bits in the C2 register should not be changed. CME0 bit should be set to a logic 0 before the MCG enters any Stop mode. Otherwise, a reset request may occur while in Stop mode. CME0 should also be set to a logic 0 before entering VLPR or VLPW power modes if the MCG is in BLPE mode.

enable Clock Monitor Enable Setting

- true: External clock monitor is enabled for OSC0.
- false: External clock monitor is disabled for OSC0.

24.1.1.49 static bool clock_get_cme0 (void) [inline], [static]

This function gets the Clock Monitor Enable Setting.

Returns

enabled True if Clock Monitor is enabled

24.1.1.50 static void clock_set_vdiv0 (uint8_t *setting*) [inline], [static]

This function selects the amount to divide the VCO output of the PLL0. The VDIV0 bits establish the multiplication factor (M) applied to the reference clock frequency. After the PLL0 is enabled (by setting either PLLCLKEN0 or PLLS), the VDIV0 value must not be changed when LOCK0 is zero.

setting VCO0 Divider Setting

24.1.1.51 static uint8_t clock_get_vdiv0 (void) [inline], [static]

This function gets the VCO0 Divider Setting.

Returns

setting VCO0 Divider Setting

24.1.1.52 static mcg_lols_status_t clock_get_lols0 (void) [inline], [static]

This function gets the Loss of Lock Status. This bit is a sticky bit indicating the lock status for the PLL. LOLS 0 is set if after acquiring lock, the PLL output frequency has fallen outside the lock exit frequency tolerance, D unl . LOLIE 0 determines whether an interrupt request is made when LOLS 0 is set. This bit is cleared by reset or by writing a logic 1 to it when set. Writing a logic 0 to this bit has no effect.

Returns

status Loss of Lock Status
- 0: PLL has not lost lock since LOLS 0 was last cleared
- 1: PLL has lost lock since LOLS 0 was last cleared

24.1.1.53 static mcg_lock_status_t clock_get_lock0 (void) [inline], [static]

This function gets the Lock Status. This bit indicates whether the PLL0 has acquired the lock. Lock detection is disabled when not operating in either the PBE or the PEE mode unless PLLCLKEN0=1 and the MCG is not configured in the BLPI or the BLPE mode. While the PLL0 clock is locking to the desired frequency, MCGPLL0CLK and MCGPLL0CLK2X are gated off until the LOCK0 bit gets asserted. If the lock status bit is set, changing the value of the PRDIV0[2:0] bits in the C5 register or the VDIV0[4:0] bits in the C6 register causes the lock status bit to clear and stay cleared until the PLL0 has reacquired the lock. The loss of the PLL0 reference clock also causes the LOCK0 bit to clear until the PLL0 has an entry into the LLS, VLPS, or a regular Stop with PLLSTEN0=0 also causes the lock status bit to clear and stay cleared until the stop mode is exited and the PLL0 has reacquired the lock. Any time the PLL0 is enabled and the LOCK0 bit is cleared, the MCGPLL0CLK and MCGPLL0CLK2X are gated off until the LOCK0 bit is reasserted.

Returns

status Lock Status
- 0: PLL is currently unlocked
- 1: PLL is currently locked

24.1.1.54 static mcg_pllst_status_t clock_get_pllst (void) [inline], [static]

This function gets the PLL Select Status. This bit indicates the clock source selected by PLLS . The PLLST bit does not update immediately after a write to the PLLS bit due to the internal synchronization between the clock domains.

Returns

status PLL Select Status
- 0: Source of PLLS clock is FLL clock.
- 1: Source of PLLS clock is PLLCS output clock.

24.1.1.55 static mcg_irefst_status_t clock_get_irefst (void) [inline], [static]

This function gets the Internal Reference Status. This bit indicates the current source for the FLL reference clock. The IREFST bit does not update immediately after a write to the IREFS bit due to internal synchronization between the clock domains.

Returns

status Internal Reference Status
- 0: Source of FLL reference clock is the external reference clock.
- 1: Source of FLL reference clock is the internal reference clock.

24.1.1.56 static mcg_clkst_status_t clock_get_clkst (void) [inline], [static]

This function gets the Clock Mode Status. These bits indicate the current clock mode. The CLKST bits do not update immediately after a write to the CLKS bits due to internal synchronization between clock domains.

Returns

status Clock Mode Status
- 00: Output of the FLL is selected (reset default).
- 01: Internal reference clock is selected.
- 10: External reference clock is selected.
- 11: Output of the PLL is selected.

24.1.1.57 static uint8_t clock_get_oscinit0 (void) [inline], [static]

This function gets the OSC Initialization Status. This bit, which resets to 0, is set to 1 after the initialization cycles of the crystal oscillator clock have completed. After being set, the bit is cleared to 0 if the OSC is subsequently disabled. See the OSC module's detailed description for more information.

Returns

status OSC Initialization Status

24.1.1.58 static mcg_ircst_status_t clock_get_ircst (void) [inline], [static]

This function gets the Internal Reference Clock Status. The IRCST bit indicates the current source for the internal reference clock select clock (IRCSCLK). The IRCST bit does not update immediately after a write to the IRCS bit due to the internal synchronization between clock domains. The IRCST bit is only updated if the internal reference clock is enabled, either by the MCG being in a mode that uses the IRC or by setting the C1[IRCLKEN] bit.

Returns

status Internal Reference Clock Status
- 0: Source of internal reference clock is the slow clock (32 kHz IRC).
- 1: Source of internal reference clock is the fast clock (2 MHz IRC).

24.1.1.59 static mcg_atmf_status_t clock_get_atmf (void) [inline], [static]

This function gets the Automatic Trim machine Fail Flag. This Fail flag for the Automatic Trim Machine (ATM). This bit asserts when the Automatic Trim Machine is enabled (ATME=1) and a write to the C1, C3, C4, and SC registers is detected or the MCG enters into any Stop mode. A write to ATMF clears the flag.

Returns

flag Automatic Trim machine Fail Flag
- 0: Automatic Trim Machine completed normally.
- 1: Automatic Trim Machine failed.

24.1.1.60 static void clock_set_atmf (void) [inline], [static]

This function clears the ATMF flag.

24.1.1.61 static mcg_locs0_status_t clock_get_locs0 (void) [inline], [static]

This function gets the OSC0 Loss of Clock Status. The LOCS0 indicates when a loss of OSC0 reference clock has occurred. The LOCS0 bit only has an effect when CME0 is set. This bit is cleared by writing a logic 1 to it when set.

Returns

status OSC0 Loss of Clock Status
- 0: Loss of OSC0 has not occurred.
- 1: Loss of OSC0 has occurred.

24.1.1.62 static void clock_set_atme (bool *enable*) [inline], [static]

This function enables/disables the Auto Trim Machine to start automatically trimming the selected Internal Reference Clock. ATME de-asserts after the Auto Trim Machine has completed trimming all trim bits of the IRCS clock selected by the ATMS bit. Writing to C1, C3, C4, and SC registers or entering Stop mode aborts the auto trim operation and clears this bit.

enable Automatic Trim Machine Enable Setting

- true: Auto Trim Machine enabled
- false: Auto Trim Machine disabled

24.1.1.63 static bool clock_get_atme (void) [inline], [static]

This function gets the Automatic Trim Machine Enable Setting.

Returns

enabled True if Automatic Trim Machine is enabled

24.1.1.64 static void clock_set_atms (mcg_atms_select_t *setting*) [inline], [static]

This function selects the IRCS clock for Auto Trim Test.

setting Automatic Trim Machine Select Setting

- 0: 32 kHz Internal Reference Clock selected
- 1: 4 MHz Internal Reference Clock selected

24.1.1.65 static mcg_atms_select_t clock_get_atms (void) [inline], [static]

This function gets the Automatic Trim Machine Select Setting.

Returns

setting Automatic Trim Machine Select Setting

24.1.1.66 static void clock_set_fltprsrv (bool *enable*) [inline], [static]

This function sets the FLL Filter Preserve Enable. This bit prevents the FLL filter values from resetting allowing the FLL output frequency to remain the same during the clock mode changes where the FLL/-DCO output is still valid. (Note: This requires that the FLL reference frequency remain the same as the value prior to the new clock mode switch. Otherwise, the FLL filter and the frequency values change.)

enable FLL Filter Preserve Enable Setting

- true: FLL filter and FLL frequency retain their previous values during new clock mode change
- false: FLL filter and FLL frequency will reset on changes to correct clock mode

24.1.1.67 static bool clock_get_fltprsrv (void) [inline], [static]

This function gets the FLL Filter Preserve Enable Setting.

Returns

enabled True if FLL Filter Preserve is enabled.

24.1.1.68 static void clock_set_fcrdiv (uint8_t *setting*) [inline], [static]

This function selects the amount to divide down the fast internal reference clock. The resulting frequency is in the range 31.25 kHz to 4 MHz. (Note: Changing the divider when the Fast IRC is enabled is not supported).

setting Fast Clock Internal Reference Divider Setting

24.1.1.69 static uint8_t clock_get_fcrdiv (void) [inline], [static]

This function gets the Fast Clock Internal Reference Divider Setting.

Returns

setting Fast Clock Internal Reference Divider Setting

24.1.1.70 static void clock_set_atcvh (uint8_t *setting*) [inline], [static]

This function sets the ATM compare value high setting. The values are used by the Auto Trim Machine to compare and adjust the Internal Reference trim values during the ATM SAR conversion.

setting ATM Compare Value High Setting

24.1.1.71 static uint8_t clock_get_atcvh (void) [inline], [static]

This function gets the ATM Compare Value High Setting.

Returns

setting ATM Compare Value High Setting

24.1.1.72 static void clock_set_atcvl (uint8_t *setting*) [inline], [static]

This function sets the ATM compare value low setting. The values are used by the Auto Trim Machine to compare and adjust Internal Reference trim values during the ATM SAR conversion.

setting ATM Compare Value Low Setting

24.1.1.73 static uint8_t clock_get_atcvl (void) [inline], [static]

This function gets the ATM Compare Value Low Setting.

Returns

setting ATM Compare Value Low Setting

Chapter 25
Oscillator (OSC)

The Kinetis SDK provides a HAL driver for the Oscillator (OSC) block of Kinetis devices.

Modules

- OSC HAL driver
  The part describes the programming interface of the OSC HAL driver.
- Shared OSC Types
  The part describes the OSC HAL driver shared types.

25.1 OSC HAL driver

The chapter describes the programming interface of the OSC HAL driver.

Macros

- #define kOscCapacitorMask (OSC_CR_SC2P_MASK | OSC_CR_SC4P_MASK | OSC_CR_SC8P_MASK | OSC_CR_SC16P_MASK)
  Oscillator capacitor load configurations mask.

Enumerations

- enum osc_instance_t { kOsc0 = 0 }
  Oscillator instance.
- enum osc_capacitor_config_t {
  kOscCapacitor2p = OSC_CR_SC2P_MASK,
  kOscCapacitor4p = OSC_CR_SC4P_MASK,
  kOscCapacitor8p = OSC_CR_SC8P_MASK,
  kOscCapacitor16p = OSC_CR_SC16P_MASK }
  Oscillator capacitor load configurations.

oscillator control APIs

- void osc_hal_enable_external_reference_clock (osc_instance_t instance)
  Enables the external reference clock for oscillator.
- void osc_hal_disable_external_reference_clock (osc_instance_t instance)
  Disables the external reference clock for oscillator.
- void osc_hal_enable_external_reference_clock_in_stop_mode (osc_instance_t instance)
  Enables the external reference clock in stop mode.
- void osc_hal_disable_external_reference_clock_in_stop_mode (osc_instance_t instance)
  Disables the external reference clock in stop mode.
- void osc_hal_enable_capacitor_config (osc_instance_t instance, uint32_t capacitorConfigs)
  Enables the capacitor configuration for oscillator.
- void osc_hal_disable_capacitor_config (osc_instance_t instance, uint32_t capacitorConfigs)
  Disables the capacitor configuration for specific oscillator.

25.1.0.74 OSC Hal Driver

Overview

The OSC module is a crystal oscillator. The module and the external crystal or resonator generate a reference clock for the MCU.

The osc_hal provides a set of APIs used to access these SIM registers including enable/disable external reference clock and set the capacitor configurations.

Oscillator Control register access APIs

Each Oscillator only has one control register, OSCx_CR. It provides an external reference clock enable, external reference clock stop enable and the capacitor load configuration settings.

Example of OSC HAL access APIs

```
#include "osc/hal/fsl_osc_hal.h"

// Enable the external reference clock
osc_hal_enable_external_reference_clock(
      kOsc0);

// Disable the external reference clock
osc_hal_disable_external_reference_clock(
      kOsc0);


#include "osc/hal/fsl_osc_hal.h"

// Enable the osc0 capacitor 2p and 8p
osc_hal_enable_capacitor_config(kOsc0,
      kOscCapacitor2p | kOscCapacitor8p);

// Disable the osc0 capacitor 4p and 16p
osc_hal_disable_capacitor_config(kOsc0,
      kOscCapacitor4p | kOscCapacitor16p);
```

25.1.1 Enumeration Type Documentation

25.1.1.1 enum osc_instance_t

Enumerator

kOsc0 Oscillator 0.

25.1.1.2 enum osc_capacitor_config_t

Enumerator

kOscCapacitor2p 2 pF capacitor load
kOscCapacitor4p 4 pF capacitor load
kOscCapacitor8p 8 pF capacitor load
kOscCapacitor16p 16 pF capacitor load

25.1.2 Function Documentation

25.1.2.1 void osc_hal_enable_external_reference_clock (osc_instance_t *instance*)

This function enables the external reference clock output for the oscillator - that is the OSCERCLK. This clock is used by many peripherals. It should be enabled at an early system initialization stage to ensure

the peripherals could select it and use it.

Parameters

instance	Oscillator instance

25.1.2.2 void osc_hal_disable_external_reference_clock (osc_instance_t *instance*)

This function disables the external reference clock output for oscillator - that is the OSCERCLK. This clock is used by many peripherals. It should be enabled at an early system initialization stage to ensure the peripherals could select and use it.

Parameters

instance	Oscillator instance

25.1.2.3 void osc_hal_enable_external_reference_clock_in_stop_mode (osc_instance_t *instance*)

This function enables the external reference clock (OSCERCLK) when MCU enters Stop mode.

Parameters

instance	Oscillator instance

25.1.2.4 void osc_hal_disable_external_reference_clock_in_stop_mode (osc_instance_t *instance*)

This function disables the external reference clock (OSCERCLK) when MCU enters Stop mode.

Parameters

instance	Oscillator instance

25.1.2.5 void osc_hal_enable_capacitor_config (osc_instance_t *instance,* uint32_t *capacitorConfigs*)

This function enables the specified capacitors configuration for the oscillator. This should be done in the early system level initialization function call based on system configuration.

Parameters

instance	Oscillator instance
capacitor-Config	Capacitor configurations. Combination of OSC_CR_SCxP_MASK

25.1.2.6 void osc_hal_disable_capacitor_config (osc_instance_t *instance,* uint32_t *capacitorConfigs*)

This function enables the specified capacitors configuration for the oscillator.

Parameters

instance	Oscillator instance
capacitor-Config	Capacitor configurations. Combination of OSC_CR_SCxP_MASK

25.2 Shared OSC Types

The chapter describes the OSC HAL driver shared types.

Chapter 26
Power Management Controller (PMC)

The Kinetis SDK provides a HAL driver for the Power Management Controller (PMC) block of Kinetis devices.

Enumerations

- enum pmc_lvwv_select_t {
  kPmcLvwvLowTrip,
  kPmcLvwvMid1Trip,
  kPmcLvwvMid2Trip,
  kPmcLvwvHighTrip }
  Low-Voltage Warning Voltage Select.
- enum pmc_lvdv_select_t {
  kPmcLvdvLowTrip,
  kPmcLvdvHighTrip }
  Low-Voltage Detect Voltage Select.

Power Management Controller Control APIs

- static void pmc_hal_enable_low_voltage_detect_interrupt (void)
  Low-Voltage Detect Interrupt Enable.
- static void pmc_hal_disable_low_voltage_detect_interrupt (void)
  Low-Voltage Detect Interrupt Disable (use polling)
- static void pmc_hal_enable_low_voltage_detect_reset (void)
  Low-Voltage Detect Hardware Reset Enable (write once)
- static void pmc_hal_disable_low_voltage_detect_reset (void)
  Low-Voltage Detect Hardware Reset Disable.
- static void pmc_hal_low_voltage_detect_ack (void)
  Low-Voltage Detect Acknowledge.
- static bool pmc_hal_get_low_voltage_detect_flag (void)
  Low-Voltage Detect Flag Read.
- static void pmc_hal_set_low_voltage_detect_voltage_select (pmc_lvdv_select_t select)
  Sets the Low-Voltage Detect Voltage Select.
- static pmc_lvdv_select_t pmc_hal_get_low_voltage_detect_voltage_select (void)
  Gets the Low-Voltage Detect Voltage Select.
- static void pmc_hal_enable_low_voltage_warning_interrupt (void)
  Low-Voltage Warning Interrupt Enable.
- static void pmc_hal_disable_low_voltage_warning_interrupt (void)
  Low-Voltage Warning Interrupt Disable (use polling)
- static void pmc_hal_low_voltage_warning_ack (void)
  Low-Voltage Warning Acknowledge.
- static bool pmc_hal_get_low_voltage_warning_flag (void)
  Low-Voltage Warning Flag Read.
- static void pmc_hal_set_low_voltage_warning_voltage_select (pmc_lvwv_select_t select)

Sets the Low-Voltage Warning Voltage Select.
- static pmc_lvwv_select_t pmc_hal_get_low_voltage_warning_voltage_select (void)
  Gets the Low-Voltage Warning Voltage Select.
- static void pmc_hal_enable_bandgap_buffer (void)
  Enables the Bandgap Buffer.
- static void pmc_hal_disable_bandgap_buffer (void)
  Disables the Bandgap Buffer.
- static uint8_t pmc_hal_get_ack_isolation (void)
  Gets the Acknowledge Isolation.
- static void pmc_hal_clear_ack_isolation (void)
  Clears an Acknowledge Isolation.
- static uint8_t pmc_hal_get_regulator_status (void)
  Gets the Regulator regulation status.

26.0.1 PMC HAL driver

Overview

The power management controller (PMC) contains the internal voltage regulator, power on reset (POR), and low voltage detect system.

The pmc_hal provides a set of APIs used to access the control registers including enable/disable features provided by this module.

Power Management Controller feature access APIs

1. Internal voltage regulator
2. Active POR providing brown-out detect
3. Low-voltage detect supporting two low-voltage trip points with four warning levels per trip point

This is an example of PMC HAL access APIs

```
#include "pmc/hal/fsl_pmc_hal.h"

// Low-Voltage Detect Interrupt Enable
pmc_hal_enable_low_voltage_detect_interrupt();

// Low-Voltage Detect Interrupt Disable (use polling)
pmc_hal_disable_low_voltage_detect_interrupt();

// Set Low-Voltage Detect Voltage Select to Low trip point (V LVD = V LVDL )
pmc_hal_set_low_voltage_detect_voltage_select(
      kPmcLvdvLowTrip);
```

26.1 Enumeration Type Documentation

26.1.1 enum pmc_lvwv_select_t

Enumerator

kPmcLvwvLowTrip Low trip point selected (VLVW = VLVW1)
kPmcLvwvMid1Trip Mid 1 trip point selected (VLVW = VLVW2)

kPmcLvwvMid2Trip Mid 2 trip point selected (VLVW = VLVW3)
kPmcLvwvHighTrip High trip point selected (VLVW = VLVW4)

26.1.2 enum pmc_lvdv_select_t

Enumerator

kPmcLvdvLowTrip Low trip point selected (V LVD = V LVDL)
kPmcLvdvHighTrip High trip point selected (V LVD = V LVDH)

26.2 Function Documentation

26.2.1 static void pmc_hal_enable_low_voltage_detect_interrupt (void) [inline], [static]

This function enables the interrupt for the low voltage detection. When enabled, if the LVDF (Low Voltage Detect Flag) is set, a hardware interrupt occurs.

26.2.2 static void pmc_hal_disable_low_voltage_detect_interrupt (void) [inline], [static]

This function disables the the interrupt for low voltage detection. When disabled, an application can only check the low voltage through polling the LVDF (Low Voltage Detect Flag).

26.2.3 static void pmc_hal_enable_low_voltage_detect_reset (void) [inline], [static]

This function enables the hardware reset for the low voltage detection. When enabled, if the LVDF (Low Voltage Detect Flag) is set, a hardware reset occurs. This setting is a write-once-only; Additional writes are ignored.

26.2.4 static void pmc_hal_disable_low_voltage_detect_reset (void) [inline], [static]

This function disables the the hardware reset for low voltage detection. When disabled, if the LVDF (Low Voltage Detect Flag) is set, a hardware reset does not occur. This setting is a write-once-only; Additional writes are ignored.

26.2.5 static void pmc_hal_low_voltage_detect_ack (void) [inline], [static]

This function acknowledges the low voltage detection errors (write 1 to clear LVDF).

26.2.6 static bool pmc_hal_get_low_voltage_detect_flag (void) [inline], [static]

This function reads the current LVDF status. If it returns 1, low voltage event is detected.

Returns

status Current low voltage detect flag
- true: Low-Voltage detected
- false: Low-Voltage not detected

26.2.7 static void pmc_hal_set_low_voltage_detect_voltage_select (pmc_lvdv_select_t *select*) [inline], [static]

This function sets the low voltage detect voltage select. It sets the low voltage detect trip point voltage (Vlvd). An application can select either a low-trip or a high-trip point. See a chip reference manual for details.

Parameters

select	Voltage select setting defined in pmc_lvdv_select_t

26.2.8 static pmc_lvdv_select_t pmc_hal_get_low_voltage_detect_voltage_select (void) [inline], [static]

This function gets the low voltage detect voltage select. It gets the low voltage detect trip point voltage (Vlvd). An application can select either a low-trip or a high-trip point. See a chip reference manual for details.

Returns

select Current voltage select setting

26.2.9 static void pmc_hal_enable_low_voltage_warning_interrupt (void) [inline], [static]

This function enables the interrupt for the low voltage warning detection. When enabled, if the LVWF (Low Voltage Warning Flag) is set, a hardware interrupt occurs.

26.2.10 static void pmc_hal_disable_low_voltage_warning_interrupt (void) [inline], [static]

This function disables the interrupt for the low voltage warning detection. When disabled, if the LVWF (Low Voltage Warning Flag) is set, a hardware interrupt does not occur.

26.2.11 static void pmc_hal_low_voltage_warning_ack (void) [inline], [static]

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

26.2.12 static bool pmc_hal_get_low_voltage_warning_flag (void) [inline], [static]

This function polls the current LVWF status. When 1 is returned, it indicates a low-voltage warning event. LVWF is set when V Supply transitions below the trip point or after reset and V Supply is already below the V LVW.

Returns

status Current LVWF status
- true: Low-Voltage Warning Flag is set.
- false: the Low-Voltage Warning does not happen.

26.2.13 static void pmc_hal_set_low_voltage_warning_voltage_select (pmc_lvwv_select_t *select*) [inline], [static]

This function sets the low voltage warning voltage select. It sets the low voltage warning trip point voltage (Vlvw). An application can select either a low, mid1, mid2 and a high-trip point. See a chip reference manual for details and the pmc_lvwv_select_t for supported settings.

Parameters

select	Low voltage warning select setting

26.2.14 static pmc_lvwv_select_t pmc_hal_get_low_voltage_warning_voltage_-select (void) [inline], [static]

This function gets the low voltage warning voltage select. It gets the low voltage warning trip point voltage (Vlvw). See the pmc_lvwv_select_t for supported settings.

Returns

select Current low voltage warning select setting

26.2.15 static void pmc_hal_enable_bandgap_buffer (void) [inline], [static]

This function enables the Bandgap buffer.

26.2.16 static void pmc_hal_disable_bandgap_buffer (void) [inline], [static]

This function disables the Bandgap buffer.

26.2.17 static uint8_t pmc_hal_get_ack_isolation (void) [inline], [static]

This function reads the Acknowledge Isolation setting that indicates whether certain peripherals and the I/O pads are in a latched state as a result of having been in the VLLS mode.

Returns

value ACK isolation 0 - Peripherals and I/O pads are in a normal run state. 1 - Certain peripherals and I/O pads are in an isolated and latched state.

26.2.18 static void pmc_hal_clear_ack_isolation (void) [inline], [static]

This function clears the ACK Isolation flag. Writing one to this setting when it is set releases the I/O pads and certain peripherals to their normal run mode state.

26.2.19 static uint8_t pmc_hal_get_regulator_status (void) [inline], [static]

This function returns the regulator to a run regulation status. It provides the current status of the internal voltage regulator.

Returns

value Regulation status 0 - Regulator is in a stop regulation or in transition to/from it. 1 - Regulator is in a run regulation.

Chapter 27
Port Control and Interrupts (PORT)

The Kinetis SDK provides a HAL driver for the Port Control and Interrupts (PORT) block of Kinetis devices.

Modules

- PORT HAL driver
  The part describes the programming interface of the PORT HAL driver.

27.1 PORT HAL driver

The chapter describes the programming interface of the PORT HAL driver.

Enumerations

- enum port_pull_t {
  kPortPullDown = 0U,
  kPortPullUp = 1U }
  Internal resistor pull feature selection.
- enum port_slew_rate_t {
  kPortFastSlewRate = 0U,
  kPortSlowSlewRate = 1U }
  Slew rate selection.
- enum port_drive_strength_t {
  kPortLowDriveStrength = 0U,
  kPortHighDriveStrength = 1U }
  Configures the drive strength.
- enum port_mux_t {
  kPortPinDisabled = 0U,
  kPortMuxAsGpio = 1U,
  kPortMuxAlt2 = 2U,
  kPortMuxAlt3 = 3U,
  kPortMuxAlt4 = 4U,
  kPortMuxAlt5 = 5U,
  kPortMuxAlt6 = 6U,
  kPortMuxAlt7 = 7U }
  Pin mux selection.
- enum port_interrupt_config_t {
  kPortIntDisabled = 0x0U,
  kPortDmaRisingEdge = 0x1U,
  kPortDmaFallingEdge = 0x2U,
  kPortDmaEitherEdge = 0x3U,
  kPortIntLogicZero = 0x8U,
  kPortIntRisingEdge = 0x9U,
  kPortIntFallingEdge = 0xAU,
  kPortIntEitherEdge = 0xBU,
  kPortIntLogicOne = 0xCU }
  Digital filter clock source selection.

Configuration

- static void port_hal_pull_select (uint32_t instance, uint32_t pin, port_pull_t pullSelect)
  Selects the internal resistor as pull-down or pull-up.
- static void port_hal_configure_pull (uint32_t instance, uint32_t pin, bool isPullEnabled)

Enables or disables the internal pull resistor.

- static void port_hal_configure_slew_rate (uint32_t instance, uint32_t pin, port_slew_rate_t rateSelect)
  Configures the fast/slow slew rate if the pin is used as a digital output.
- static void port_hal_configure_passive_filter (uint32_t instance, uint32_t pin, bool isPassiveFilterEnabled)
  Configures the passive filter if the pin is used as a digital input.
- static void port_hal_configure_drive_strength (uint32_t instance, uint32_t pin, port_drive_strength_t driveSelect)
  Configures the drive strength if the pin is used as a digital output.
- static void port_hal_mux_control (uint32_t instance, uint32_t pin, port_mux_t mux)
  Configures the pin muxing.
- void port_hal_global_pin_control_low (uint32_t instance, uint16_t lowPinSelect, uint16_t config)
  Configures the low half of pin control register for the same settings.
- void port_hal_global_pin_control_high (uint32_t instance, uint16_t highPinSelect, uint16_t config)
  Configures the high half of pin control register for the same settings.

Interrupt

- static void port_hal_configure_pin_interrupt (uint32_t instance, uint32_t pin, port_interrupt_config_t intConfig)
  Configures the port pin interrupt/DMA request.
- static port_interrupt_config_t port_hal_get_pin_interrupt_config (uint32_t instance, uint32_t pin)
  Gets the current port pin interrupt/DMA request configuration.
- static bool port_hal_read_pin_interrupt_flag (uint32_t instance, uint32_t pin)
  Reads the individual pin-interrupt status flag.
- static void port_hal_clear_pin_interrupt_flag (uint32_t instance, uint32_t pin)
  Clears the individual pin-interrupt status flag.
- static uint32_t port_hal_read_port_interrupt_flag (uint32_t instance)
  Reads the entire port interrupt status flag.
- static void port_hal_clear_port_interrupt_flag (uint32_t instance)
  Clears the entire port interrupt status flag.

27.1.0.1 PORT HAL Driver

Overview

Port control and interrupts hardware driver configuration. Use these functions to set port control and external interrupt functions. Most functions can be configured independently for each pin in the 32-bit port and affect the pin regardless of its pin muxing state. To use these functions, pass into instance number (HW_PORTA, HW_PORTB, HW_PORTC, etc).

27.1.1 Enumeration Type Documentation

27.1.1.1 enum port_pull_t

Enumerator

kPortPullDown internal pull-down resistor is enabled.
kPortPullUp internal pull-up resistor is enabled.

27.1.1.2 enum port_slew_rate_t

Enumerator

kPortFastSlewRate fast slew rate is configured.
kPortSlowSlewRate slow slew rate is configured.

27.1.1.3 enum port_drive_strength_t

Enumerator

kPortLowDriveStrength low drive strength is configured.
kPortHighDriveStrength high drive strength is configured.

27.1.1.4 enum port_mux_t

Enumerator

kPortPinDisabled corresponding pin is disabled as analog.
kPortMuxAsGpio corresponding pin is configured as GPIO.
kPortMuxAlt2 chip-specific
kPortMuxAlt3 chip-specific
kPortMuxAlt4 chip-specific
kPortMuxAlt5 chip-specific
kPortMuxAlt6 chip-specific
kPortMuxAlt7 chip-specific

27.1.1.5 enum port_interrupt_config_t

Configures the interrupt generation condition.

Enumerator

kPortIntDisabled Interrupt/DMA request is disabled.

kPortDmaRisingEdge DMA request on rising edge.
kPortDmaFallingEdge DMA request on falling edge.
kPortDmaEitherEdge DMA request on either edge.
kPortIntLogicZero Interrupt when logic zero.
kPortIntRisingEdge Interrupt on rising edge.
kPortIntFallingEdge Interrupt on falling edge.
kPortIntEitherEdge Interrupt on either edge.
kPortIntLogicOne Interrupt when logic one.

27.1.2 Function Documentation

27.1.2.1 static void port_hal_pull_select (uint32_t *instance,* uint32_t *pin,* port_pull_t *pullSelect*) [inline], [static]

Pull configuration is valid in all digital pin muxing modes.

Parameters

instance	port instance number.
pin	port pin number
pullSelect	internal resistor pull feature selection • kPortPullDown: internal pull-down resistor is enabled. • kPortPullUp : internal pull-up resistor is enabled.

27.1.2.2 static void port_hal_configure_pull (uint32_t *instance,* uint32_t *pin,* bool *isPullEnabled*) [inline], [static]

Parameters

instance	port instance number
pin	port pin number
isPullEnabled	internal pull resistor enable or disable • true : internal pull resistor is enabled. • false: internal pull resistor is disabled.

27.1.2.3 static void port_hal_configure_slew_rate (uint32_t *instance,* uint32_t *pin,* port_slew_rate_t *rateSelect*) [inline], [static]

Parameters

instance	port instance number
pin	port pin number
rateSelect	slew rate selection • kPortFastSlewRate: fast slew rate is configured. • kPortSlowSlewRate: slow slew rate is configured.

27.1.2.4 static void port_hal_configure_passive_filter (uint32_t *instance,* uint32_t *pin,* bool *isPassiveFilterEnabled*) [inline], [static]

If enabled, a low pass filter (10 MHz to 30 MHz bandwidth) is enabled on the digital input path. Disable the Passive Input Filter when supporting high speed interfaces (> 2 MHz) on the pin.

Parameters

instance	port instance number
pin	port pin number
isPassiveFilter-Enabled	passive filter configuration • false: passive filter is disabled. • true : passive filter is enabled.

27.1.2.5 static void port_hal_configure_drive_strength (uint32_t *instance,* uint32_t *pin,* port_drive_strength_t *driveSelect*) [inline], [static]

Parameters

instance	port instance number
pin	port pin number
driveSelect	drive strength selection • kLowDriveStrength : low drive strength is configured. • kHighDriveStrength: high drive strength is configured.

27.1.2.6 static void port_hal_mux_control (uint32_t *instance,* uint32_t *pin,* port_mux_t *mux*) [inline], [static]

Parameters

instance	port instance number
pin	port pin number
mux	pin muxing slot selection • kPinDisabled: Pin disabled. • kMuxAsGpio : Set as GPIO. • others : chip-specific.

27.1.2.7 void port_hal_global_pin_control_low (uint32_t *instance,* uint16_t *lowPinSelect,* uint16_t *config*)

```
This function operates pin 0 -15 of one specific port.
```

Parameters

instance	port instance number
pinSelect	update corresponding pin control register or not. For a specific bit: • 0: corresponding low half of pin control register won't be updated according to configuration. • 1: corresponding low half of pin control register will be updated according to configuration.
config	value is written to a low half port control register bits[15:0].

27.1.2.8 void port_hal_global_pin_control_high (uint32_t *instance,* uint16_t *highPinSelect,* uint16_t *config*)

```
This function operates pin 16 -31 of one specific port.
```

Parameters

instance	port instance number
pinSelect	update corresponding pin control register or not. For a specific bit: • 0: corresponding high half of pin control register won't be updated according to configuration. • 1: corresponding high half of pin control register will be updated according to configuration.
config	value is written to a high half port control register bits[15:0].

27.1.2.9 static void port_hal_configure_pin_interrupt (uint32_t *instance,* uint32_t *pin,* port_interrupt_config_t *intConfig*) [inline], [static]

Parameters

instance	port instance number.
pin	port pin number
intConfig	interrupt configuration • kIntDisabled : Interrupt/DMA request disabled. • kDmaRisingEdge : DMA request on rising edge. • kDmaFallingEdge: DMA request on falling edge. • kDmaEitherEdge : DMA request on either edge. • KIntLogicZero : Interrupt when logic zero. • KIntRisingEdge : Interrupt on rising edge. • KIntFallingEdge: Interrupt on falling edge. • KIntEitherEdge : Interrupt on either edge. • KIntLogicOne : Interrupt when logic one.

27.1.2.10 static port_interrupt_config_t port_hal_get_pin_interrupt_config (uint32_t *instance,* uint32_t *pin*) [inline], [static]

Parameters

instance	port instance number
pin	port pin number

Returns

interrupt configuration
- kIntDisabled : Interrupt/DMA request disabled.
- kDmaRisingEdge : DMA request on rising edge.
- kDmaFallingEdge: DMA request on falling edge.
- kDmaEitherEdge : DMA request on either edge.
- KIntLogicZero : Interrupt when logic zero.
- KIntRisingEdge : Interrupt on rising edge.
- KIntFallingEdge: Interrupt on falling edge.
- KIntEitherEdge : Interrupt on either edge.
- KIntLogicOne : Interrupt when logic one.

27.1.2.11 static bool port_hal_read_pin_interrupt_flag (uint32_t *instance,* uint32_t *pin*) [inline], [static]

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

instance	port instance number
pin	port pin number

Returns

current pin interrupt status flag
- 0: interrupt is not detected.
- 1: interrupt is detected.

27.1.2.12 static void port_hal_clear_pin_interrupt_flag (uint32_t *instance,* uint32_t *pin*) [inline], [static]

Parameters

instance	port instance number
pin	port pin number

27.1.2.13 static uint32_t port_hal_read_port_interrupt_flag (uint32_t *instance*) [inline], [static]

Parameters

instance	port instance number

Returns

all 32 pin interrupt status flags. For specific bit:
- 0: interrupt is not detected.
- 1: interrupt is detected.

27.1.2.14 static void port_hal_clear_port_interrupt_flag (uint32_t *instance*) [inline], [static]

Parameters

instance	port instance number

Chapter 28
System Integration Module (SIM)

The Kinetis SDK provides a HAL driver for the System Integration Module (SIM) block of Kinetis devices.

Modules

- SIM HAL driver
  The part describes the programming interface of the SIM HAL driver.

28.1 SIM HAL driver

The chapter describes the programming interface of the SIM HAL driver.

Data Structures

- struct sim_clock_gate_module_config_t
  Clock gate module configuration table structure. More...
- struct sim_clock_source_value_t
  clock source value table structure More...
- struct sim_clock_name_config_t
  Clock name configuration table structure. More...

Enumerations

- enum sim_clock_gate_module_names_t
  Clock gate module names.
- enum sim_clock_source_names_t
  Clock source and sel names.
- enum sim_clock_divider_names_t
  Clock Divider names.
- enum sim_usbsstby_stop_t
  SIM USB voltage regulator in standby mode setting during stop modes.
- enum sim_usbvstby_stop_t
  SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.
- enum sim_cmtuartpad_strengh_t
  SIM CMT/UART pad drive strength.
- enum sim_ptd7pad_strengh_t
  SIM PTD7 pad drive strength.
- enum sim_flexbus_security_level_t
  SIM FlexBus security level.
- enum sim_pretrgsel_t
  SIM ADCx pre-trigger select.
- enum sim_trgsel_t
  SIM ADCx trigger select.
- enum sim_uart_rxsrc_t
  SIM receive data source select.
- enum sim_uart_txsrc_t
  SIM transmit data source select.
- enum sim_ftm_trg_src_t
  SIM FlexTimer x trigger y select.
- enum sim_ftm_clk_sel_t
  SIM FlexTimer external clock select.
- enum sim_ftm_ch_src_t
  SIM FlexTimer x channel y input capture source select.
- enum sim_ftm_flt_sel_t
  SIM FlexTimer x Fault y select.
- enum sim_tpm_clk_sel_t
  SIM Timer/PWM external clock select.

- enum sim_tpm_ch_src_t
  SIM Timer/PWM x channel y input capture source select.
- enum sim_hal_status_t
  SIM HAL API return status.

Variables

- const sim_clock_name_config_t kSimClockNameConfigTable [ ]
  clock name configuration table for specified CPU defined in fsl_clock_module_names_Kxxx.h
- const sim_clock_gate_module_config_t kSimClockGateModuleConfigTable [ ]
  SIM configuration table for clock module names.
- const sim_clock_source_value_t * kClockSourceValueTable [ ]
  clock source value table for specified CPU

clock related feature APIs

- sim_hal_status_t clock_hal_set_gate (sim_clock_gate_module_names_t clockModule, uint8_t instance, bool enable)
  Enables or disables the clock for a specified clock module.
- sim_hal_status_t clock_hal_get_gate (sim_clock_gate_module_names_t clockModule, uint8_t instance, bool *isEnabled)
  Gets the clock enabled or disabled state.
- sim_hal_status_t clock_hal_set_clock_source (sim_clock_source_names_t clockSource, uint8_t setting)
  Sets the clock source setting.
- sim_hal_status_t clock_hal_get_clock_source (sim_clock_source_names_t clockSource, uint8_t *setting)
  Gets the clock source setting.
- sim_hal_status_t clock_hal_set_clock_divider (sim_clock_divider_names_t clockDivider, uint32_t setting)
  Sets the clock divider setting.
- void clock_hal_set_clock_out_dividers (uint32_t outdiv1, uint32_t outdiv2, uint32_t outdiv3, uint32_t outdiv4)
  Sets the clock out dividers setting.
- sim_hal_status_t clock_hal_get_clock_divider (sim_clock_divider_names_t clockDivider, uint32_t *setting)
  Gets the clock divider setting.

individual field access APIs

- static void sim_set_usbregen (bool enable)
  Sets the USB voltage regulator enabled setting.
- static bool sim_get_usbregen (void)
  Gets the USB voltage regulator enabled setting.
- static void sim_set_usbsstby (sim_usbsstby_stop_t setting)

Sets the USB voltage regulator in a standby mode setting during Stop, VLPS, LLS, and VLLS.

- static sim_usbsstby_stop_t sim_get_usbsstby (void)
  Gets the USB voltage regulator in a standby mode setting.
- static void sim_set_usbvstby (sim_usbvstby_stop_t setting)
  Sets the USB voltage regulator in a standby mode during the VLPR or the VLPW.
- static sim_usbvstby_stop_t sim_get_usbvstby (void)
  Gets the USB voltage regulator in a standby mode during the VLPR or the VLPW.
- static void sim_set_usswe (bool enable)
  Sets the USB voltage regulator stop standby write enable setting.
- static bool sim_get_usswe (void)
  Gets the USB voltage regulator stop standby write enable setting.
- static void sim_set_uvswe (bool enable)
  Sets the USB voltage regulator VLP standby write enable setting.
- static bool sim_get_uvswe (void)
  Gets the USB voltage regulator VLP standby write enable setting.
- static void sim_set_urwe (bool enable)
  Sets the USB voltage regulator enable write enable setting.
- static bool sim_get_urwe (void)
  Gets the USB voltage regulator enable write enable setting.
- void sim_set_alttrgen (uint8_t instance, bool enable)
  Sets the ADCx alternate trigger enable setting.
- bool sim_get_alttrgen (uint8_t instance)
  Gets the ADCx alternate trigger enable setting.
- void sim_set_pretrgsel (uint8_t instance, sim_pretrgsel_t select)
  Sets the ADCx pre-trigger select setting.
- sim_pretrgsel_t sim_get_pretrgsel (uint8_t instance)
  Gets the ADCx pre-trigger select setting.
- void sim_set_trgsel (uint8_t instance, sim_trgsel_t select)
  Sets the ADCx trigger select setting.
- sim_pretrgsel_t sims_get_trgsel (uint8_t instance)
  Gets the ADCx trigger select setting.
- void sim_set_uart_rxsrc (uint8_t instance, sim_uart_rxsrc_t select)
  Sets the UARTx receive data source select setting.
- sim_uart_rxsrc_t sim_get_uart_rxsrc (uint8_t instance)
  Gets the UARTx receive data source select setting.
- void sim_set_uart_txsrc (uint8_t instance, sim_uart_txsrc_t select)
  Sets the UARTx transmit data source select setting.
- sim_uart_txsrc_t sim_get_uart_txsrc (uint8_t instance)
  Gets the UARTx transmit data source select setting.
- static uint32_t sim_get_fam_id (void)
  Gets the Kinetis Fam ID in System Device ID register (SIM_SDID).
- static uint32_t sim_get_pin_id (void)
  Gets the Kinetis Pincount ID in System Device ID register (SIM_SDID).
- static uint32_t sim_get_rev_id (void)
  Gets the Kinetis Revision ID in the System Device ID register (SIM_SDID).
- static uint32_t sim_get_pf_size (void)
  Gets the program flash size in the Flash Configuration Register 1 (SIM_FCFG).

28.1.0.15 SIM Hal Driver

Overview

The system integration module (SIM) provides the system control and chip configuration registers. The sim_hal provides a set of APIs used to access the SIM registers including clock gate control and other configuration settings.

Clock Gate Control register access APIs

Clock gate control is based on the module. Each chip has a sub-set of modules that can be gated through gate control registers in SIM. The gate control module names are defined in the fsl_clock_names.h. Each chip also provides a configuration table that defines the supported module names in that chip. Users can access the gate control through the clock manager. See the clock manager for examples to enable a clock module.

Clock Source Control access APIs

Clock source control is also based on the module. Only certain modules have the clock source control in SIM. See the fsl_sim_features.h for available clock sources controlled by the SIM registers. Others are controlled by the device registers. See device drivers for those that are not supported by the SIM registers.

Clock Divider access APIs

Certain clocks use dividers configured in SIM. Only certain dividers are available from SIM. See details in the fsl_sim_features.h for available dividers provided by SIM.

This is an example of SIM HAL access APIs

```
#include "sim/fsl_sim_types.h"
#include "sim/hal/fsl_sim_hal.h"

// calling sim clock gate control API to enable the clock for DMA
clock_hal_set_gate(kSimClockModuleDMA, 0, true);


#include "sim/fsl_sim_types.h"
#include "sim/hal/fsl_sim_hal.h"

uint8_t setting;

// calling sim clock source control API to get the setting for USBSRC
clock_hal_get_clock_source(kSimClockUsbSrc, &setting);


#include "sim/fsl_sim_types.h"
#include "sim/hal/fsl_sim_hal.h"

uint32_t divider;
```

```
// calling sim divider access API to get the setting for divider outdiv1
clock_hal_get_clock_divider(kSimClockDividerOutdiv1, &divider);
```

28.1.1 Data Structure Documentation

28.1.1.1 struct sim_clock_gate_module_config_t

Data Fields

- sim_clock_gate_module_names_t clockGateModuleName
  clock module name
- uint8_t deviceInstance
  device instance
- uint32_t scgcRegAddress
  clock gate control register address
- uint32_t deviceMask
  device mask in control register

28.1.1.2 struct sim_clock_source_value_t

Data Fields

- bool isSel
  clock sel flag
- bool hasDivider
  has divider
- sim_clock_divider_names_t dividerName
  divider name
- sim_clock_names_t clockName
  clock name

28.1.1.3 struct sim_clock_name_config_t

Data Fields

- sim_clock_names_t clockName
  clock name
- bool useOtherRefClock
  if it uses the other ref clock
- sim_clock_names_t otherRefClockName
  other ref clock name
- sim_clock_divider_names_t dividerName
  clock divider name

28.1.2 Function Documentation

28.1.2.1 sim_hal_status_t clock_hal_set_gate (sim_clock_gate_module_names_t *clockModule,* uint8_t *instance,* bool *enable*)

This function enables/disables the clock for a specified clock module and instance.

Parameters

clockModule	Clock module name defined in the sim_clock_gate_module_names_t
instance	Module instance
enable	Enable or disable the clock • true: enable the clock • false: disable the clock

Returns

status If the clock module name doesn't exist, it returns an error.

28.1.2.2 sim_hal_status_t clock_hal_get_gate (sim_clock_gate_module_names_t *clockModule*, uint8_t *instance*, bool * *isEnabled*)

This function gets the current clock gate status of the specified clock module and instance.

Parameters

clockModule	Clock module name defined in sim_clock_gate_module_names_t
instance	Module instance
isEnabled	returned status, clock is enabled or disabled for the module. • true: enabled • false: disabled

Returns

status if the clock module name doesn't exist, it returns an error.

28.1.2.3 sim_hal_status_t clock_hal_set_clock_source (sim_clock_source_names_t *clockSource*, uint8_t *setting*)

This function sets the settings for a specified clock source. Each clock source has its own clock selection settings. See the chip reference manual for clock source detailed settings and the sim_clock_source_-names_t for clock sources.

Parameters

clockSource	Clock source name defined in sim_clock_source_names_t
setting	Setting value

Returns

status If the clock source doesn't exist, it returns an error.

28.1.2.4 sim_hal_status_t clock_hal_get_clock_source (sim_clock_source_names_t *clockSource,* uint8_t * *setting*)

This function gets the settings for a specified clock source. Each clock source has its own clock selection settings. See the reference manual for clock source detailed settings and the sim_clock_source_names_t for clock sources.

Parameters

clockSource	Clock source name
setting	Current setting for the clock source

Returns

status If the clock source doesn't exist, it returns an error.

28.1.2.5 sim_hal_status_t clock_hal_set_clock_divider (sim_clock_divider_names_t *clockDivider,* uint32_t *setting*)

This function sets the setting for a specified clock divider. See the reference manual for a supported clock divider and value range and the sim_clock_divider_names_t for dividers.

Parameters

clockDivider	Clock divider name
divider	Divider setting

Returns

status If the clock divider doesn't exist, it returns an error.

28.1.2.6 void clock_hal_set_clock_out_dividers (uint32_t *outdiv1,* uint32_t *outdiv2,* uint32_t *outdiv3,* uint32_t *outdiv4*)

This function sets the setting for all clock out dividers at the same time. See the reference manual for a supported clock divider and value range and the sim_clock_divider_names_t for clock out dividers.

Parameters

outdiv1	Outdivider1 setting
outdiv2	Outdivider2 setting
outdiv3	Outdivider3 setting
outdiv4	Outdivider4 setting

28.1.2.7 sim_hal_status_t clock_hal_get_clock_divider (sim_clock_divider_names_t *clockDivider,* uint32_t * *setting*)

This function gets the setting for a specified clock divider. See the reference manual for a supported clock divider and value range and the sim_clock_divider_names_t for dividers.

Parameters

clockDivider	Clock divider name
divider	Divider value pointer

Returns

status If the clock divider doesn't exist, it returns an error.

28.1.2.8 static void sim_set_usbregen (bool *enable*) [inline], [static]

This function controls whether the USB voltage regulator is enabled. This bit can only be written when the SOPT1CFG[URWE] bit is set.

Parameters

enable	USB voltage regulator enable setting • true: USB voltage regulator is enabled. • false: USB voltage regulator is disabled.

28.1.2.9 static bool sim_get_usbregen (void) [inline], [static]

This function gets the USB voltage regulator enabled setting.

Returns

enabled True if the USB voltage regulator is enabled.

28.1.2.10 static void sim_set_usbsstby (sim_usbsstby_stop_t *setting*) [inline], [static]

This function controls whether the USB voltage regulator is placed in a standby mode during Stop, VLPS, LLS, and VLLS modes. This bit can only be written when the SOPT1CFG[USSWE] bit is set.

Parameters

setting	USB voltage regulator in standby mode setting • 0: USB voltage regulator not in standby during Stop, VLPS, LLS and VLLS modes. • 1: USB voltage regulator in standby during Stop, VLPS, LLS and VLLS modes.

28.1.2.11 static sim_usbsstby_stop_t sim_get_usbsstby (void) [inline], [static]

This function gets the USB voltage regulator in a standby mode setting.

Returns

setting USB voltage regulator in a standby mode setting

28.1.2.12 static void sim_set_usbvstby (sim_usbvstby_stop_t *setting*) [inline], [static]

This function controls whether the USB voltage regulator is placed in a standby mode during the VLPR and the VLPW modes. This bit can only be written when the SOPT1CFG[UVSWE] bit is set.

Parameters

setting	USB voltage regulator in standby mode setting • 0: USB voltage regulator not in standby during VLPR and VLPW modes. • 1: USB voltage regulator in standby during VLPR and VLPW modes.

28.1.2.13 static sim_usbvstby_stop_t sim_get_usbvstby (void) [inline], [static]

This function gets the USB voltage regulator in a standby mode during the VLPR or the VLPW.

Returns

setting USB voltage regulator in a standby mode during the VLPR or the VLPW

28.1.2.14 static void sim_set_usswe (bool *enable*) [inline], [static]

This function controls whether the USB voltage regulator stop standby write feature is enabled. Writing one to this bit allows the SOPT1[USBSSTBY] bit to be written. This register bit clears after a write to SOPT1[USBSSTBY].

Parameters

enable	USB voltage regulator stop standby write enable setting • true: SOPT1[USBSSTBY] can be written. • false: SOPT1[USBSSTBY] cannot be written.

28.1.2.15 static bool sim_get_usswe (void) [inline], [static]

This function gets the USB voltage regulator stop standby write enable setting.

Returns

enabled True if the USB voltage regulator stop standby write is enabled.

28.1.2.16 static void sim_set_uvswe (bool *enable*) [inline], [static]

This function controls whether USB voltage regulator VLP standby write feature is enabled. Writing one to this bit allows the SOPT1[USBVSTBY] bit to be written. This register bit clears after a write to SOPT1[USBVSTBY].

Parameters

enable	USB voltage regulator VLP standby write enable setting • true: SOPT1[USBSSTBY] can be written. • false: SOPT1[USBSSTBY] cannot be written.

28.1.2.17 static bool sim_get_uvswe (void) [inline], [static]

This function gets the USB voltage regulator VLP standby write enable setting.

Returns

enabled True if the USB voltage regulator VLP standby write is enabled.

28.1.2.18 static void sim_set_urwe (bool *enable*) [inline], [static]

This function controls whether the USB voltage regulator write enable feature is enabled. Writing one to this bit allows the SOPT1[USBREGEN] bit to be written. This register bit clears after a write to SOPT1[USBREGEN].

Parameters

enable	USB voltage regulator enable write enable setting • true: SOPT1[USBSSTBY] can be written. • false: SOPT1[USBSSTBY] cannot be written.

28.1.2.19 static bool sim_get_urwe (void) [inline], [static]

This function gets the USB voltage regulator enable write enable setting.

Returns

enabled True if USB voltage regulator enable write is enabled.

28.1.2.20 void sim_set_alttrgen (uint8_t *instance,* bool *enable*)

This function enables/disables the alternative conversion triggers for ADCx.

Parameters

enable	Enable alternative conversion triggers for ADCx • true: Select alternative conversion trigger. • false: Select PDB trigger.

28.1.2.21 bool sim_get_alttrgen (uint8_t *instance*)

This function gets the ADCx alternate trigger enable setting.

Returns

enabled True if ADCx alternate trigger is enabled

28.1.2.22 void sim_set_pretrgsel (uint8_t *instance,* sim_pretrgsel_t *select*)

This function selects the ADCx pre-trigger source when the alternative triggers are enabled through ADCxALTTRGEN.

Parameters

select	pre-trigger select setting for ADCx • 0: Pre-trigger A selected for ADCx. • 1: Pre-trigger B selected for ADCx.

28.1.2.23 sim_pretrgsel_t sim_get_pretrgsel (uint8_t *instance*)

This function gets the ADCx pre-trigger select setting.

Returns

select ADCx pre-trigger select setting

28.1.2.24 void sim_set_trgsel (uint8_t *instance,* sim_trgsel_t *select*)

This function selects the ADCx trigger source when alternative triggers are enabled through ADCxALTTRGEN.

Parameters

select	trigger select setting for ADCx • 0000: External trigger • 0001: High speed comparator 0 asynchronous interrupt • 0010: High speed comparator 1 asynchronous interrupt • 0011: High speed comparator 2 asynchronous interrupt • 0100: PIT trigger 0 • 0101: PIT trigger 1 • 0110: PIT trigger 2 • 0111: PIT trigger 3 • 1000: FTM0 trigger • 1001: FTM1 trigger • 1010: FTM2 trigger • 1011: FTM3 trigger • 1100: RTC alarm • 1101: RTC seconds • 1110: Low-power timer trigger • 1111: High speed comparator 3 asynchronous interrupt

28.1.2.25 sim_pretrgsel_t sims_get_trgsel (uint8_t *instance*)

This function gets the ADCx trigger select setting.

Returns

select ADCx trigger select setting

28.1.2.26 void sim_set_uart_rxsrc (uint8_t *instance,* sim_uart_rxsrc_t *select*)

This function selects the source for the UARTx receive data.

Parameters

select	the source for the UARTx receive data • 00: UARTx_RX pin. • 01: CMP0. • 10: CMP1. • 11: Reserved.

28.1.2.27 sim_uart_rxsrc_t sim_get_uart_rxsrc (uint8_t *instance*)

This function gets the UARTx receive data source select setting.

Returns

select UARTx receive data source select setting

28.1.2.28 void sim_set_uart_txsrc (uint8_t *instance,* sim_uart_txsrc_t *select*)

This function selects the source for the UARTx transmit data.

Parameters

select	the source for the UARTx transmit data • 00: UARTx_TX pin. • 01: UARTx_TX pin modulated with FTM1 channel 0 output. • 10: UARTx_TX pin modulated with FTM2 channel 0 output. • 11: Reserved.

28.1.2.29 sim_uart_txsrc_t sim_get_uart_txsrc (uint8_t *instance*)

This function gets the UARTx transmit data source select setting.

Returns

select UARTx transmit data source select setting

28.1.2.30 static uint32_t sim_get_fam_id (void) [inline], [static]

This function gets the Kinetis Fam ID in System Device ID register.

Returns

id Kinetis Fam ID

28.1.2.31 static uint32_t sim_get_pin_id (void) [inline], [static]

This function gets the Kinetis Pincount ID in System Device ID register.

Returns

id Kinetis Pincount ID

28.1.2.32 static uint32_t sim_get_rev_id (void) [inline], [static]

This function gets the Kinetis Revision ID in System Device ID register.

Returns

id Kinetis Revision ID

28.1.2.33 static uint32_t sim_get_pf_size (void) [inline], [static]

This function gets the program flash size in the Flash Configuration Register 1.

Returns

size Program flash Size

Chapter 29
Software Timer (SWT)

The Kinetis SDK provides a Software Timer (SWT) for timeout and delay.

Typedefs

- typedef int32_t time_counter_t
  Data type of the counter of each timer channel.
- typedef uint32_t time_free_counter_t
  Data type of the free running counter.

Enumerations

- enum sw_timer_channel_status_t {
  kSwTimerChannelExpired = 0x00,
  kSwTimerChannelStillCounting = 0x01,
  kSwTimerChannelIsDisable = 0x02,
  kSwTimerChannelNotAvailable = 0xFF }
  Definition of the possible status of a software channel timer.
- enum _sw_timer_errors {
  kSwTimerStatusSuccess,
  kSwTimerStatusFail,
  kSwTimerStatusInvalidChannel }
  List of status and errors.
- enum sw_timer_timeouts
  Max timeout value according to size of the time counter.

Functions

- uint32_t sw_timer_init_service (void)
  Initializes the software timer module.
- void sw_timer_shutdown_service (void)
  Deinitializes the software timer module.
- uint8_t sw_timer_reserve_channel (void)
  Reserves a free timer channel to be used by any module and returns its identifier.
- sw_timer_channel_status_t sw_timer_get_channel_status (uint8_t timerChannel)
  Returns the actual status of the given timer channel.
- uint32_t sw_timer_start_channel (uint8_t timerChannel, time_counter_t timeout)
  Starts the count down of the given timer channel.
- uint32_t sw_timer_release_channel (uint8_t timerChannel)
  Releases the given timer channel, so it can be used by someone else.
- time_free_counter_t sw_timer_get_free_counter (void)
  Gets the current value of the free running counter.
- void sw_timer_update_counters (void)
  This function is called every 1ms by the interruption and update count down values of all timer channels.

29.1 Typedef Documentation

29.1.1 typedef int32_t time_counter_t

If it is an int8_t the counter will count up to 127ms, int16_t up to 32767ms and int32_t up to 2147483647ms.

29.1.2 typedef uint32_t time_free_counter_t

This data type should be unsigned and will count up to 255ms if it is uint8_t, 65535ms for uint16_t and 4294967295ms for uint32_t.

29.2 Enumeration Type Documentation

29.2.1 enum sw_timer_channel_status_t

Enumerator

kSwTimerChannelExpired Indicates the timer channel has counted the given ms.

kSwTimerChannelStillCounting Indicates the timeout of the channel has not expired and the timer is still counting.

kSwTimerChannelIsDisable Indicates the timer channel is not reserved.

kSwTimerChannelNotAvailable Indicates there are not available channels to reserve or the requested channel is not available.

29.2.2 enum _sw_timer_errors

Enumerator

kSwTimerStatusSuccess The execution was successful.

kSwTimerStatusFail The execution failed.

kSwTimerStatusInvalidChannel The given channel is not valid. Valid channels are 0 to (SW_TIMER_NUMBER_CHANNELS - 1).

29.3 Function Documentation

29.3.1 uint32_t sw_timer_init_service (void)

Prepares variables and HAL layer to provide timer services. Starts the free running counter which will be available to get its value any time while the service is running; it is useful whenever a module wants to keep track of time, but do not wants to reserve a channel.

Returns

status_t Returns software timer status after initialization.

Return values

kSwTimerStatusSuccess	The initialization was successful and the software timer is ready to provide services.
kSwTimerStatusFail	The initialization failed.

29.3.2 void sw_timer_shutdown_service (void)

Shutdown HAL layer, so no timer service can be provided after the execution of this function.

Returns

void

29.3.3 uint8_t sw_timer_reserve_channel (void)

Returns

uint8_t Returns the number of the channel that was reserved.

Return values

Any	value between 0 and SW_TIMER_NUMBER_CHANNELS is a valid channel. It indicates the channel was reserved and can be used.
kSwTimerChannelNotAvailable	If there is not any available channel, because all channels are already reserved.

29.3.4 sw_timer_channel_status_t sw_timer_get_channel_status (uint8_t *timerChannel*)

The timer has to be previously started to return a valid status.

Parameters

timerChannel	[in] Indicates the timer channel which status is going to be returned.

Returns

sw_timer_channel_status_t Current status of the given timer channel.

Return values

kSwTimerChannelExpired	Indicates the timer channel has counted the given ms.
kSwTimerChannelStill-Counting	Indicates the timeout of the channel has not expired and the timer is still counting.
kSwTimerChannelIs-Disable	Indicates the timer channel is not reserved.
kSwTimerChannelNot-Available	Indicates the timer channel is invalid.

29.3.5 uint32_t sw_timer_start_channel (uint8_t *timerChannel,* time_counter_t *timeout*)

The timer channel has to be previously reserved.

Parameters

timerChannel	[in] Indicates the timer channel that is going to be started.
timeout	[in] Time in ms that the timer channel will count. The timeout should be a multiple of count unit of the timer, otherwise it will be taken the integer part of the division and the exact count will not be achieved

Returns

status_t Reports failures in the execution of the function.

Return values

kSwTimerStatusSuccess	A channel was started successfully.
kSwTimerStatusInvalid-Channel	The timer channel is invalid, it does not exist.

29.3.6 uint32_t sw_timer_release_channel (uint8_t *timerChannel*)

Parameters

timerChannel	[in] Identifier of the timer channel.

Returns

status_t Reports failures in the execution of the function.

Return values

kSwTimerStatusSuccess	A channel was released successfully.
kSwTimerStatusInvalid-Channel	The timer channel is invalid, it does not exist.

29.3.7 time_free_counter_t sw_timer_get_free_counter (void)

Any module can keep track of the time by reading this counter and calculates time difference. No reservation of timer channel is needed. Consider for calculations that when the counter overflows it will start from 0 again.

Returns

time_free_counter_t Returns current count of the free running counter.

29.3.8 void sw_timer_update_counters (void)

Returns

void

Chapter 30
OS Abstraction Layer (OSA)

The Kinetis SDK provides the abstraction layer for real-time operating systems.

Modules

- Bare Metal Abstraction Layer
  The Kinetis SDK provides the bare metal abstraction layer for synchronization, mutual exclusion, message queue, etc.
- FreeRTOS Abstraction Layer
  The Kinetis SDK provides the FreeRTOS Abstraction Layer for synchronization, mutual exclusion, message queue, etc.
- Freescale MQX™ RTOS Abstraction Layer
  The Freescale MQX™ RTOS OSA layer provides the OSA services mapped to the MQX software services.
- µC/OS-II Abstraction Layer
  The Kinetis SDK provides the µC/OS-II Abstraction Layer for synchronization, mutual exclusion, message queue, etc.
- µC/OS-III Abstraction Layer
  The Kinetis SDK provides the µC/OS-III for synchronization, mutual exclusion, message queue, etc.

Enumerations

- enum fsl_rtos_status {
  kSuccess = 0,
  kError,
  kTimeout,
  kIdle }
  Status values to be returned by functions.
- enum event_status {
  kFlagNotSet = 0,
  kFlagSet }
  The event flags are set or not.
- enum event_clear_type {
  kEventAutoClr = 0,
  kEventManualClr }
  The event flags are cleared automatically or manually.

Synchronization

- fsl_rtos_status sync_create (sync_object_t *obj, uint8_t initValue)
  Initialize a synchronization object to a given state.
- fsl_rtos_status sync_wait (sync_object_t *obj, uint32_t timeout)
  Wait for the synchronization object.
- fsl_rtos_status sync_poll (sync_object_t *obj)

Checks a synchronization object's status.

- fsl_rtos_status sync_signal (sync_object_t *obj)
  Signal for someone waiting on the synchronization object to wake up.
- fsl_rtos_status sync_signal_from_isr (sync_object_t *obj)
  Signal for someone waiting on the synchronization object to wake up.
- fsl_rtos_status sync_destroy (sync_object_t *obj)
  Destroy a previously created synchronization object.

Resource locking

- fsl_rtos_status lock_create (lock_object_t *obj)
  Initialize a locking object.
- fsl_rtos_status lock_wait (lock_object_t *obj, uint32_t timeout)
  Wait for the object to be unlocked and lock it.
- fsl_rtos_status lock_poll (lock_object_t *obj)
  Checks if a locking object can be locked and locks it if possible.
- fsl_rtos_status lock_release (lock_object_t *obj)
  Unlock a previously locked object.
- fsl_rtos_status lock_destroy (lock_object_t *obj)
  Destroy a previously created locking object.

Event signaling

- fsl_rtos_status event_create (event_object_t *obj, event_clear_type clearType)
  Initializes the event object.
- fsl_rtos_status event_wait (event_object_t *obj, uint32_t timeout, event_group_t *setFlags)
  Wait for any event flags to be set.
- fsl_rtos_status event_set (event_object_t *obj, event_group_t flags)
  Set one or more event flags of an event object.
- fsl_rtos_status event_set_from_isr (event_object_t *obj, event_group_t flags)
  Set one or more event flags of an event object.
- fsl_rtos_status event_clear (event_object_t *obj, event_group_t flags)
  Clear one or more events of an event object.
- event_status event_check_flags (event_object_t *obj, event_group_t flag)
  Check the flags are set or not.
- fsl_rtos_status event_destroy (event_object_t *obj)
  Destroy a previously created event object.

Thread management

- fsl_rtos_status __task_create (task_t task, uint8_t *name, uint16_t stackSize, task_stack_t *stackMem, uint16_t priority, void *param, bool usesFloat, task_handler_t *handler)
  Create a task.
- fsl_rtos_status task_destroy (task_handler_t handler)
  Destroy a previously created task.

Message queues

- msg_queue_handler_t msg_queue_create (msg_queue_t *queue, uint16_t number, uint16_t size)
  Initialize the message queue.

- fsl_rtos_status msg_queue_put (msg_queue_handler_t handler, msg_queue_item_t item)
  Introduce an element at the tail of the queue.
- fsl_rtos_status msg_queue_get (msg_queue_handler_t handler, msg_queue_item_t *item, uint32_t timeout)
  Read and remove an element at the head of the queue.
- fsl_rtos_status msg_queue_flush (msg_queue_handler_t handler)
  Discards all elements in the queue and leaves the queue empty.
- fsl_rtos_status msg_queue_destroy (msg_queue_handler_t handler)
  Destroy a previously created queue.

Memory Management

- void * mem_allocate (size_t size)
  Reserves the requested amount of memory in bytes.
- void * mem_allocate_zero (size_t size)
  Reserves the requested amount of memory in bytes and initializes it to 0.
- fsl_rtos_status mem_free (void *ptr)
  Releases the memory previously reserved.

Time management

- void time_delay (uint32_t delay)
  Delays execution for a number of milliseconds.

Interrupt management

- fsl_rtos_status interrupt_handler_register (int32_t irqNumber, void(*handler)(void))
  Install interrupt handler.

30.0.9 OS Abstraction Layer

Overview

Operating System Abstraction layer (OSA) provides a common set of services for drivers and applications so that they can work with or without the operating system. OSA provides services that abstract most of the OS kernel functionality. These services can either be mapped to the target OS functions directly, or implemented by OSA when no OS is used (bare metal) or when the service does not exist in the target OS. Freescale Kinetis SDK implements the OS abstraction layer for the Freescale MQX RTOS, FreeRTOS, µC/OS, and for no OS usage (bare metal). The bare metal OS abstraction implementation is selected as the default option.

OSA provides these services: task management, semaphore, mutex, event, message queue, memory allocator, critical part, and time delay.

Task Management

With OSA, applications can create and destroy tasks dynamically. These services are mapped to the task functions of RTOSes. For bare metal, a function poll mechanism is used to simulate a task.

To create a task, use the FSL_RTOS_TASK_DEFINE() to declare the task statically. Then, use the task_-create() to create the task.

This is an example code to creating and destroying a task:

```
// Define the task with entry function task_func.
FSL_RTOS_TASK_DEFINE(task_func, stackSize, taskName, FALSE);

void main(void)
{
    // Define the task handler.
    task_handler_t handler;
    // Create the task.
    task_create(task_func, priority, param, &handler);
    task_destroy(&handler);
    // ...
}
```

Semaphore

OSA provides the drivers and applications with a counting semaphore. It can be used either to synchronize tasks or to synchronize a task and an ISR.

To use the semaphore, use the sync_object_declare() and the sync_create() to create it. The object can be created with an initial value.

Depending on the target OS, the sync_object_declare() may either define or declare the object to be used by the sync_create(). When the sync_object_declare() defines the object with the memory allocated, the memory has be to available for the entire lifespan of the object. It is recommended to declare objects in places where the memory allocated has the desired scope. When the sync object is not used any more, use the sync_destroy() to destroy it.

This is an example code to create and destroy a sync object:

```
// Declare the object.
sync_object_declare(my_sync);
// Create the object with initial value.
sync_create(&my_sync, initValue);
// Destroy the object.
sync_destory(&my_sync);
```

There are two ways to wait for the sync object.

1. The first way to wait for the sync object involves using the function, fsl_rtos_status sync_wait,(sync-_object_t ∗obj, uint32_t timeout). This function waits for a timeout in milliseconds. Setting a parameter timeout as the kSyncWaitForever causes the thread to wait infinitely if the object can't be obtained. Note that the parameter timeout must not be 0.
2. The second way to wait for the sync object involves using the function, fsl_rtos_status sync_-poll(sync_object_t ∗obj), as a non-blocking interface.

To post the object, use either the sync_signal() or the sync_signal_from_isr().

Mutex

A mutex is used for mutual exclusion of the tasks when they access a shared resource.

Depending on the target OS, the lock_object_declare() may either define or declare the object to be used by the lock_create(). When the lock_object_declare() defines the object with memory allocated, the memory has be to available for the entire lifespan of the object. It is recommended to declare objects in places where the memory allocated has the desired scope. You should use the lock_object_declare() and the lock_create() to create the object, and the lock_destroy() to destroy it. When the lock object is created, it is in an unlocked state.

This is example code to create and destroy a lock object:

```
// Declare the object.
lock_object_declare(my_lock);
// Create the object.
lock_create(&my_lock);
// Destroy the object.
lock_destroy(&my_lock);
```

There are two ways to wait for the lock object.

1. The first way to wait for the lock object involves the function fsl_rtos_status lock_wait(lock_object_t *obj, uint32_t timeout). This function waits for the timeout in milliseconds. Setting a parameter timeout as the kSyncWaitForever makes the thread wait infinitely if the lock can't be obtained. Note that the parameter timeout must not be 0.
2. The other way to wait for the lock object involves the function, fsl_rtos_status lock_poll(lock_object_t *obj), as a non-blocking interface.

To unlock the object, use the lock_release().

Event

An event is supported with a single consumer and multi-producers.

The function fsl_rtos_status event_create(event_object_t *obj, event_clear_type clearType) creates two types of event objects, auto-clear and manual-clear.

1. An auto-clear event means that the event flags are cleared automatically.
2. A manual-clear event means that applications must clear the flags manually.

The function fsl_rtos_status event_wait(event_object_t *obj, uint32_t timeout, event_group_t *setFlags) waits for a timeout in milliseconds if the event flags are not set. Passing the kSyncWaitForever causes it to wait forever. The event flags which are set can be obtained by the parameter setFlags. This interface does not have the parameter to specify which flags to wait for and will wait for any flag.

The functions fsl_rtos_status event_set(event_object_t *obj, event_group_t flags) and fsl_rtos_status event_set_from_isr(event_object_t *obj, event_group_t flags) set the event flags.

The function fsl_rtos_status event_clear(event_object_t *obj, event_group_t flags) clears specified flags manually.

The function event_status event_check_flags(event_object_t *obj, event_group_t flag) checks whether any of the specified flags are set.

When the event object is not used any more, use the event_destory to destroy it.

Message Queue

A message queue transfers messages between tasks. OSA provides two types of message queue.

1. The first message queue occurs when only the pointer to the message is stored in the queue. The senders put pointer to the message in the queue and the receivers get the message through the pointer. This type of message queue is more efficient because it does not need to copy memories. Applications and drivers must ensure that the message is always valid until receivers get it. If not, use the second message queue type.
2. The second message queue type occurs when entities are copied to the internal memory of the queue. Even though this is a safer approach, it slows down the performance.

Configure the macro __FSL_RTOS_MSGQ_COPY_MSG__ to determine which mode is used.

To create a message queue, use the MSG_QUEUE_DECLARE() and the msg_queue_create() like this:

```
// Declare the message queue.
MSG_QUEUE_DECLARE(my_message, msg_num, msg_size);

void main(void)
{
    msg_queue_handler_t handler;
    handler = msg_queue_create(&my_message, msg_num, msg_size);
    // ...
}
```

The function fsl_rtos_status msg_queue_put(msg_queue_handler_t handler, msg_queue_item_t item) puts the message to the queue. Depending on the target OS, the MSG_QUEUE_DECLARE() may either define or declare the message queue to be used by the msg_queue_create(). When the MSG_QUEUE_DECLARE() defines the message queue with memory allocated, the memory has be to available for the entire lifespan of the message queue. It is recommended to declare the message queue in places where the memory allocated has the desired scope. If the queue is full, an error returns.

The function fsl_rtos_status msg_queue_get(msg_queue_handler_t handler, msg_queue_item_t *item, uint32_t timeout) waits for a timeout to get the message if the queue is empty. Passing 0 causes it to return immediately and passing the kSyncWaitForever causes it to wait forever.

The function fsl_rtos_status msg_queue_flush(msg_queue_handler_t handler) cleans all messages in the queue.

If the queue is not used any more, use the msg_queue_destroy() to destroy it.

Critical Section

The rtos_enter_critical() and rtos_exit_critical() functions ensure that code is not pre-empted.

Dynamic Memory Allocation

The void *mem_allocate(size_t size) function allocates memory with specified size. The void *mem_allocate_zero(size_t size) function allocates and cleans the memory. The fsl_rtos_status mem_free(void *ptr) function frees the memory. For RTOSes that have internal memory manager, such as the Freescale MQX RTOS, OSA maps these functions directly. For other RTOSes or for bare metal, the standard functions malloc/calloc/free are used.

Time Delay

The function void time_delay(uint32_t delay) delays specified time in milliseconds.

30.1 Enumeration Type Documentation

30.1.1 enum fsl_rtos_status

Enumerator

kSuccess Functions work correctly.
kError Functions work failed.
kTimeout Timeout occurs while waiting for an object.
kIdle Can not get the object in non-blocking mode.

30.1.2 enum event_status

Enumerator

kFlagNotSet The flags checked are set.
kFlagSet The flags checked are not set.

30.1.3 enum event_clear_type

Enumerator

kEventAutoClr The flags of the event will be cleared automatically.
kEventManualClr The flags of the event will be cleared manually.

30.2 Function Documentation

30.2.1 fsl_rtos_status sync_create (sync_object_t * *obj*, uint8_t *initValue*)

Parameters

obj	The sync object to initialize.
initValue	The initial value the object will be set to.

Return values

kSuccess	The object was successfully created.
kError	Invalid parameter or no more objects can be created.

30.2.2 fsl_rtos_status sync_wait (sync_object_t * *obj*, uint32_t *timeout*)

This function checks the sync object's counting value, if it is positive, decreases it and returns kSuccess, otherwise, timeout will be used for wait.

Parameters

obj	Pointer to the synchronization object.
timeout	The maximum number of milliseconds to wait for the object to be signalled. Pass the kSyncWaitForever constant to wait indefinitely for someone to signal the object. A value of 0 should not be passed to this function. Instead, use sync_poll for a non blocking check.

Return values

kSuccess	The object was signalled.
kTimeout	A timeout occurred.
kError	An incorrect parameter was passed.
kIdle	The object has not been signalled.

Note

There could be only one process waiting for the object at the same time.

30.2.3 fsl_rtos_status sync_poll (sync_object_t * *obj*)

This function is used to poll a sync object's status. If the sync object's counting value is positive, decrease it and return kSuccess. If the object's counting value is 0, the function will return kIdle immediately

Parameters

obj	The synchronization object.

Return values

kSuccess	The object was signalled.
kIdle	The object was not signalled.
kError	An incorrect parameter was passed.

30.2.4 fsl_rtos_status sync_signal (sync_object_t * *obj*)

This function should not be called from an ISR.

Parameters

obj	The synchronization object to signal.

Return values

kSuccess	The object was successfully signaled.
kError	The object can not be signaled or invalid parameter.

30.2.5 fsl_rtos_status sync_signal_from_isr (sync_object_t * *obj*)

This function should only be called from an ISR.

Parameters

obj	The synchronization object to signal.

Return values

kSuccess	The object was successfully signaled.
kError	The object can not be signaled or invalid parameter.

30.2.6 fsl_rtos_status sync_destroy (sync_object_t * *obj*)

Parameters

obj	The synchronization object to destroy.

Return values

kSuccess	The object was successfully destroyed.
kError	Object destruction failed.

30.2.7 fsl_rtos_status lock_create (lock_object_t * *obj*)

Parameters

obj	The lock object to initialize.

Return values

kSuccess	The lock is created successfully.
kError	Tke lock creation failed.

30.2.8 fsl_rtos_status lock_wait (lock_object_t * *obj,* uint32_t *timeout*)

This function will wait for some time or wait forever if could not get the lock.

Parameters

obj	The locking object.
timeout	The maximum number of milliseconds to wait for the mutex. Pass the kSyncWaitForever constant to wait indefinitely for someone to unlock the object. A value of 0 should not be passed to this function. Instead, use lock_poll for a non blocking check.

Return values

kSuccess	The lock was obtained.
kTimeout	A timeout occurred.

kError	An incorrect parameter was passed.

30.2.9 fsl_rtos_status lock_poll (lock_object_t * *obj*)

This function returns instantly if could not get the lock.

Parameters

obj	The locking object.

Return values

kSuccess	The lock was obtained.
kIdle	The lock could not be obtained.
kError	An incorrect parameter was passed.

Note

There could be only one process waiting for the object at the same time. For RTOSes, wait for a lock recursively by one task is not supported.

30.2.10 fsl_rtos_status lock_release (lock_object_t * *obj*)

Parameters

obj	The locking object to unlock.

Return values

kSuccess	The object was successfully unlocked.
kError	The object can not be unlocked or invalid parameter.

30.2.11 fsl_rtos_status lock_destroy (lock_object_t * *obj*)

Parameters

obj	The locking object to destroy.

Return values

kSuccess	The object was successfully destroyed.
kError	Object destruction failed.

30.2.12 fsl_rtos_status event_create (event_object_t * *obj*, event_clear_type *clearType*)

When the object is created, the flags is 0.

Parameters

obj	Pointer to the event object to initialize.
clearType	The event is auto-clear or manual-clear.

Return values

kSuccess	The object was successfully created.
kError	Incorrect parameter or no more objects can be created.

30.2.13 fsl_rtos_status event_wait (event_object_t * *obj*, uint32_t *timeout*, event_group_t * *setFlags*)

This function will wait for some time or wait forever if no flags are set. Any flags set will wake up the function.

Parameters

obj	The event object.
timeout	The maximum number of milliseconds to wait for the event. Pass the kSyncWaitForever constant to wait indefinitely. A value of 0 should not be passed to this function.

setFlags	Pointer to receive the flags that were set.

Return values

kSuccess	An event was set.
kTimeout	A timeout occurred.
kError	An incorrect parameter was passed.

30.2.14 fsl_rtos_status event_set (event_object_t * *obj,* event_group_t *flags*)

This function should not be called from an ISR.

Parameters

obj	The event object.
flags	Event flags to be set.

Return values

kSuccess	The flags were successfully set.
kError	An incorrect parameter was passed.

Note

There could be only one process waiting for the event.

30.2.15 fsl_rtos_status event_set_from_isr (event_object_t * *obj,* event_group_t *flags*)

This function should only be called from an ISR.

Parameters

obj	The event object.
flags	Event flags to be set.

Return values

kSuccess	The flags were successfully set.
kError	An incorrect parameter was passed.

30.2.16 fsl_rtos_status event_clear (event_object_t * *obj*, event_group_t *flags*)

This function should not be called from an ISR.

Parameters

obj	The event object.
flags	Event flags to be clear.

Return values

kSuccess	The flags were successfully cleared.
kError	An incorrect parameter was passed.

30.2.17 event_status event_check_flags (event_object_t * *obj*, event_group_t *flag*)

Parameters

obj	The event object.
flag	The flag to check.

Return values

kFlagsSet	The flags checked are set.
kFlagsNotSet	The flags checked are not set or got an error.

30.2.18 fsl_rtos_status event_destroy (event_object_t * *obj*)

Parameters

obj	The event object to destroy.

Return values

kSuccess	The object was successfully destroyed.
kError	Event destruction failed.

30.2.19 fsl_rtos_status __task_create (task_t *task*, uint8_t * *name*, uint16_t *stackSize*, task_stack_t * *stackMem*, uint16_t *priority*, void * *param*, bool *usesFloat*, task_handler_t * *handler*)

This function is wrapped by the macro task_create. Generally, this function is for internal use only, applications must use FSL_RTOS_TASK_DEFINE to define resources for task statically then use task_create to create task. If applications have prepare the resouces for task dynamically, they can use this function to create the task.

Parameters

task	The task function.
name	The name of this task.
stackSize	The stack size in byte.
stackMem	Pointer to the stack. For bare metal, Freescale MQX RTOS, and FreeRTOS, this could be NULL.
priority	Initial priority of the task.
param	Pointer to be passed to the task when it is created.
usesFloat	This task will use float register or not.
handler	Pointer to the task handler.

Return values

kSuccess	The task was successfully created.
kError	The task could not be created.

Note

Different tasks can not use the same task function.

30.2.20 fsl_rtos_status task_destroy (task_handler_t *handler*)

Note

Depending on the RTOS, task resources may or may not be automatically freed, and this function may not return if the current task is destroyed.

Parameters

handler	The handler of the task to destroy. Returned by the task_create function.

Return values

kSuccess	The task was successfully destroyed.
kError	Task destruction failed or invalid parameter.

30.2.21 msg_queue_handler_t msg_queue_create (msg_queue_t * *queue,* uint16_t *number,* uint16_t *size*)

This function will initialize the message queue that declared previously. Here is an example demonstrating how to use:

```
msg_queue_handler_t handler;
MSG_QUEUE_DECLARE(my_message, msg_num, msg_size);
handler = msg_queue_create(&my_message, msg_num, msg_size);
```

Parameters

queue	The queue declared through the MSG_QUEUE_DECLARE macro.
number	The number of elements in the queue.
size	Size of every elements in words.

Return values

Handler	to access the queue for put and get operations. If message queue created failed, return 0.

30.2.22 fsl_rtos_status msg_queue_put (msg_queue_handler_t *handler,* msg_queue_item_t *item*)

Parameters

handler	Queue handler returned by the msg_queue_create function.
item	Pointer to the element to be introduced in the queue.

Return values

kSuccess	Element successfully introduced in the queue.
kError	The queue was full or an invalid parameter was passed.

30.2.23 fsl_rtos_status msg_queue_get (msg_queue_handler_t *handler*, msg_queue_item_t * *item*, uint32_t *timeout*)

Parameters

handler	Queue handler returned by the msg_queue_create function.
item	Pointer to store a pointer to the element of the queue.
timeout	In case the queue is empty, the number of milliseconds to wait for an element to be introduced into the queue. Use 0 to return immediately or kSyncWaitForever to wait indefinitely.

Return values

kSuccess	Element successfully obtained from the queue.
kTimeout	If a timeout was specified, the queue remained empty after timeout.
kError	The queue was empty or the handler was invalid.
kIdle	The queue was empty and the timeout has not expired.

Note

There should be only one process waiting on the queue.

30.2.24 fsl_rtos_status msg_queue_flush (msg_queue_handler_t *handler*)

Parameters

handler	Queue handler returned by the msg_queue_create function.

Return values

kSuccess	Queue successfully emptied.
kError	Emptying queue failed.

30.2.25 fsl_rtos_status msg_queue_destroy (msg_queue_handler_t *handler*)

Parameters

handler	Queue handler returned by the msg_queue_create function.

Return values

kSuccess	The queue was successfully destroyed.
kError	Message queue destruction failed.

30.2.26 void* mem_allocate (size_t *size*)

Parameters

size	Amount of bytes to reserve.

Return values

Pointer	to the reserved memory. NULL if memory could not be allocated.

30.2.27 void* mem_allocate_zero (size_t *size*)

Parameters

size	Amount of bytes to reserve.

Return values

Pointer	to the reserved memory. NULL if memory could not be allocated.

30.2.28 fsl_rtos_status mem_free (void * *ptr*)

Parameters

ptr	Pointer to the start of the memory block previously reserved.

Return values

kSuccess	Memory correctly released.

30.2.29 void time_delay (uint32_t *delay*)

Parameters

delay	The time in milliseconds to wait.

30.2.30 fsl_rtos_status interrupt_handler_register (int32_t *irqNumber*, void(*)(void) *handler*)

Parameters

irqNumber	IRQ number of the interrupt.
handler	The interrupt handler to install.

Return values

kSuccess	Handler is installed successfully.

kSuccess	Handler could not be installed.

30.3 Bare Metal Abstraction Layer

The Kinetis SDK provides the bare metal abstraction layer for synchronization, mutual exclusion, message queue, etc.

Data Structures

- struct sync_object_t
  Type for an synchronization object. More...
- struct lock_object_t
  Type for a resource locking object. More...
- struct event_object_t
  Type for an event group object. More...
- struct msg_queue_t
  Type for a message queue declaration and creation. More...
- struct POLL_SLOT_STRUCT
  Poll function slot. More...
- struct POLL_STRUCT
  Poll structure. More...

Macros

- #define kSyncWaitForever kSwTimerMaxTimeout
  Constant to pass as timeout value in order to wait indefinitely.
- #define FSL_RTOS_CURRENT_TASK ((task_handler_t)0)
  Macro passed to the task_destroy function to destroy the current task.

Typedefs

- typedef uint32_t event_group_t
  Type for an event flags group, bit 32 is reserved.
- typedef void(∗ task_t)(void ∗param)
  Type for a task pointer.
- typedef task_t task_handler_t
  Type for a task handler, returned by the task_create function.
- typedef uint32_t task_stack_t
  Type for a task stack.
- typedef msg_queue_t ∗ msg_queue_handler_t
  Type for a message queue declaration and creation.
- typedef void ∗ msg_queue_item_t
  Type for a message queue item.

Synchronization

- #define sync_object_declare(obj) sync_object_t obj
  Create the synchronization object.

Resource locking

- #define lock_object_declare(obj) lock_object_t obj
  Create the locking object.

Thread management

- #define FSL_RTOS_TASK_DEFINE(task, stackSize, name, usesFloat) uint8_t fslTaskName_-##task[ ] = name
  Define a task.
- #define task_create(task, priority, param, handler)
  Creates and sets the task to active.

Message queues

- #define MSG_QUEUE_DECLARE(name, number, size)
  This macro statically reserves the memory required for the queue.

Critical Sections

- #define rtos_enter_critical interrupt_disable_global
  Ensures the following code will not be preempted.
- #define rtos_exit_critical interrupt_enable_global
  Allows preemption.

Task poll

- void Poll (void)
  Poll every function registered in the POLL_STRUCT.
- void POLL_init (void)
  Initialize the POLL_STRUCT.
- #define POLL_MAX_NUM 5
  Maximum number of functions called every time Poll function is invoked.

30.3.0.1 Bare Metal Abstraction Layer

Overview

When RTOSes are not used, a bare metal abstraction layer provides synchronization, mutual exclusion, message queue, and so on.

Task Management

Bare metal abstraction layer provides a poll mechanism to simulate a task. The task functions should not be infinite. When tasks are created, the task functions are registered to a global array and called one by one.

This is an example code, which shows how to use a task with a bare metal abstraction layer.

```
void main(void)
{
    task_handler_t handler;
    POLL_init();

    task_create(task_func, priority, param, &handler);

    for(;;)
    {
        Poll();
    }
}
```

30.3.1 Data Structure Documentation

30.3.1.1 struct sync_object_t

Data Fields

- volatile bool isWaiting
  Is any task waiting for a timeout on this object.
- volatile uint8_t semCount
  The count value of the object.
- uint8_t timerId
  The software timer channel this object bind to.

30.3.1.2 struct lock_object_t

Data Fields

- volatile bool isWaiting
  Is any task waiting for a timeout on this lock.
- volatile bool isLocked
  Is the object locked or not.
- uint8_t timerId
  The software timer channel this lock bind to.

30.3.1.3 struct event_object_t

Data Fields

- volatile bool isWaiting

Is any task waiting for a timeout on this event.
- uint8_t timerId
  The software timer channel this event bind to.
- volatile event_group_t flags
  The flags status.
- event_clear_type clearType
  Auto clear or manual clear.

30.3.1.4 struct msg_queue_t

Data Fields

- void ** queueMem
  Points to the queue memory.
- uint16_t number
  Stores the elements in the queue.
- uint16_t size
  Stores the size in words of each element.
- uint16_t head
  Index of the next element to be read.
- uint16_t tail
  Index of the next place to write to.
- sync_object_t queueSync
  Sync object wakeup tasks waiting for msg.
- volatile bool isEmpty
  Whether queue is empty.

30.3.1.5 struct POLL_SLOT_STRUCT

Data Fields

- task_t p_func
  Task's entry.
- void * param
  Task's parameter.

30.3.1.6 struct POLL_STRUCT

Data Fields

- POLL_SLOT_STRUCT p_slot [POLL_MAX_NUM]
  polling function pointer array
- uint32_t registered_no
  number of registered function

30.3.2 Macro Definition Documentation

30.3.2.1 #define kSyncWaitForever kSwTimerMaxTimeout

30.3.2.2 #define sync_object_declare(*obj*) sync_object_t obj

To be used instead of a standard declaration.

Parameters

obj	The sync object to create.

30.3.2.3 #define lock_object_declare(*obj*) lock_object_t obj

To be used instead of a standard declaration.

Parameters

obj	The lock object to create.

30.3.2.4 #define FSL_RTOS_TASK_DEFINE(*task, stackSize, name, usesFloat*) uint8_t fslTaskName_##task[] = name

This macro is used to define resources for a task statically, then task_create will create task based-on these resources.

Parameters

task	The task function.
stackSize	Number of elements in the stack for this task.
name	String to assign to the task.
usesFloat	Boolean that indicates whether the task uses the floating point unit.

30.3.2.5 #define task_create(*task, priority, param, handler*)

Value:

```
__task_create(task,                                             \
                  fslTaskName_##task,                               \
                  0,                                                \
                  NULL,                                             \
                  priority,                                         \
                  param,                                            \
                  false,                                            \
                  handler)
```

This macro is used with FSL_RTOS_TASK_DEFINE to create a task. Here is an example demonstrating how to use:

```
FSL_RTOS_TASK_DEFINE(task_func, stackSize, taskName, FALSE);

void main(void)
{
```

```
    task_handler_t handler;
    task_create(task_func, priority, param, &handler);
}
```

Parameters

task	The task function.
priority	Initial priority of the task.
param	Parameter to be passed to the task when it is created.
handler	Returns the identifier to be used afterwards to destroy the task.

Return values

kSuccess	The task was successfully created.
kError	The task creation failed.

30.3.2.6 #define MSG_QUEUE_DECLARE(*name, number, size*)

Value:

```
void * queueMem##name[number];          \
                                            msg_queue_t
       name = {                     \
                                                .queueMem = queueMem##name          \
                                            }
```

Parameters

name	Identifier for the memory region.
number	Number of elements in the queue.
size	Size of every element in words.

30.3.3 Function Documentation

30.3.3.1 void Poll (void)

Every task function registered in the POLL_STRUCT will be called in turn. This function should be used in a infinite loop in main.

Here is an example demonstrating how to use:

```
int main(void)
{
```

```
    // System initialize functions.
    POLL_init();

    for(;;)
    {
        Poll();
    }
    return 0;
}
```

30.3.3.2 void POLL_init (void)

The struct POLL_STRUCT must be initialized before task functions are registered.

30.4 Freescale MQX™ RTOS Abstraction Layer

The Freescale MQX™ RTOS OSA layer provides the OSA services mapped to the MQX software services.

Macros

- #define FSL_RTOS_CURRENT_TASK ((task_handler_t)MQX_NULL_TASK_ID)
  Macro passed to the task_destroy function to destroy the current task.

Typedefs

- typedef LWSEM_STRUCT sync_object_t
  Type for an interrupt synchronization object.
- typedef MUTEX_STRUCT lock_object_t
  Type for a resource locking object.
- typedef LWEVENT_STRUCT event_object_t
  Type for an event group object.
- typedef _mqx_uint event_group_t
  Type for an event flags group.
- typedef TASK_FPTR task_t
  Type for a task pointer.
- typedef _task_id task_handler_t
  Type for a task handler, returned by the task_create function.
- typedef uint32_t task_stack_t
  Type for a task stack.
- typedef MQX_OSA_QUEUE msg_queue_t
  Type for a message queue declaration and creation.
- typedef void * msg_queue_handler_t
  Type for a message queue declaration and creation.
- typedef void * msg_queue_item_t
  Type for a message queue item.

Enumerations

- enum sync_timeouts { kSyncWaitForever = UINT32_MAX }

Interrupt handler synchronization

Definition of how MQX handles the message queues

- #define sync_object_declare(obj) sync_object_t obj
  Create the synchronization object.

Thread management

- #define FSL_RTOS_TASK_DEFINE(task, stackSize, name, usesFloat)
  Creates a task descriptor that is used to create the task with task_create.
- #define task_create(task, priority, param, handler)
  Creates and sets the task to active.

Message queues

- #define **SIZE_IN_MMT_UNITS**(size) ((size + sizeof(_mqx_max_type) - 1) / sizeof(_mqx_max_-type))
- #define MSG_QUEUE_DECLARE(name, number, size)
  This macro statically reserves the memory required for the queue.

Critical Sections

- void rtos_enter_critical (void)
  Ensures the following code will not be preempted.
- void rtos_exit_critical (void)
  Allows preemption.

30.4.1 Macro Definition Documentation

30.4.1.1 #define FSL_RTOS_CURRENT_TASK ((task_handler_t)MQX_NULL_TASK_ID)

30.4.1.2 #define sync_object_declare(*obj*) sync_object_t obj

To be used instead of a standard declaration.

Parameters

obj	The sync object to create.

30.4.1.3 #define FSL_RTOS_TASK_DEFINE(*task, stackSize, name, usesFloat*)

Value:

```
const uint16_t fslTaskStackSize_##task = stackSize; \
                                                                const uint8_t fslTaskName_##task[] = name; \
                                                                const bool fslTaskFloatUse_##task =
      usesFloat
```

Parameters

task	The task function.
stackSize	Number of elements in the stack for this task.
name	String to assign to the task.
usesFloat	Boolean that indicates whether the task uses the floating point unit.

30.4.1.4 #define task_create(*task, priority, param, handler*)

Value:

```
__task_create(task, \
                                                    (uint8_t *)fslTaskName_##task, \
                                                    fslTaskStackSize_##task, \
                                                    NULL, \
                                                    priority, \
                                                    param, \
                                                    fslTaskFloatUse_##task, \
                                                    handler)
```

Parameters

task	The task function.
priority	Initial priority of the task.
param	Pointer to be passed to the task when it is created.
handler	Returns the identifier to be used afterwards to destroy the task.

Return values

kSuccess	The task was successfully created.
kError	Creation of task failed.

30.4.1.5 #define MSG_QUEUE_DECLARE(*name, number, size*)

Value:

```
_mqx_max_type mqxq_##name[SIZE_IN_MMT_UNITS(sizeof(LWMSGQ_STRUCT)) + SIZE_IN_MMT_UNITS(size * 4) * number];
       \
    _mqx_max_type mqxm_##name[SIZE_IN_MMT_UNITS(size * 4)]; \
    MQX_OSA_QUEUE name = { \
        .one_msg = mqxm_##name, \
        .msgq = mqxq_##name, \
    }
```

Note

The queue will store pointers to the elements, and no the elements themselves. The element must continue to exist in memory for the receiving end to properly get the contents.

Parameters

name	Identifier for the memory region.
number	Number of elements in the queue.
size	Size of element in 4B units.

30.4.2 Typedef Documentation

30.4.2.1 typedef LWSEM_STRUCT sync_object_t

30.4.2.2 typedef MUTEX_STRUCT lock_object_t

30.4.2.3 typedef LWEVENT_STRUCT event_object_t

30.4.2.4 typedef _mqx_uint event_group_t

Bit 32 is reserved.

30.4.2.5 typedef TASK_FPTR task_t

30.4.2.6 typedef _task_id task_handler_t

30.4.2.7 typedef uint32_t task_stack_t

30.4.2.8 typedef MQX_OSA_QUEUE msg_queue_t

30.4.2.9 typedef void* msg_queue_handler_t

30.4.2.10 typedef void* msg_queue_item_t

30.4.3 Enumeration Type Documentation

30.4.3.1 enum sync_timeouts

Enumerator

kSyncWaitForever Constant to pass for the sync_wait() timeout in order to wait indefinitely.

30.4.4 Function Documentation

30.4.4.1 void rtos_enter_critical (void)

Required attention to other system events may be delayed.

30.5 µC/OS-II Abstraction Layer

The Kinetis SDK provides the µC/OS-II Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

Data Structures

- struct EVENT_UCOSII
  Type for an event group object in uCOS-II. More...
- struct MSGQ_STRUCT_UCOSII
  Type for message queue in uCOS-II. More...

Macros

- #define kSyncWaitForever 0xFFFFFFFFU
  Constant to pass as timeout value in order to wait indefinitely.
- #define FSL_RTOS_CURRENT_TASK ((task_handler_t)OS_PRIO_SELF)
  Macro passed to the task_destroy function to destroy the current task.

Typedefs

- typedef INT8U task_handler_t
  Type for a task handler, returned by the task_create function.
- typedef OS_STK task_stack_t
  Type for a task stack.
- typedef OS_FLAGS event_group_t
  Type for an event flags group.
- typedef OS_EVENT * sync_object_t
  Type for an synchronization object.
- typedef OS_EVENT * lock_object_t
  Type for a resource locking object.
- typedef EVENT_UCOSII event_object_t
  Type for an event group object.
- typedef MSGQ_STRUCT_UCOSII msg_queue_t
  Type for a message queue declaration and creation.
- typedef MSGQ_STRUCT_UCOSII * msg_queue_handler_t
  Type for a message queue handler.
- typedef void * msg_queue_item_t
  Type for a message queue item.
- typedef void(* task_t)(void *start)
  Type for a task pointer.

Thread management

- #define FSL_RTOS_TASK_DEFINE(task, stackSize, name, usesFloat)
  Creates a task descriptor that is used to create the task with task_create.

- #define task_create(task, priority, param, handler)
  Creates and sets the task to active.

Synchronization

- #define sync_object_declare(obj) sync_object_t obj = (sync_object_t)0
  Create the synchronization object.

Resource locking

- #define lock_object_declare(obj) lock_object_t obj = (lock_object_t)0
  Create the locking object.

Message queues

- #define MSG_QUEUE_DECLARE(name, number, size)
  This macro statically reserves the memory required for the queue.

Critical Sections

- #define rtos_enter_critical OS_ENTER_CRITICAL
  Ensures the following code will not be preempted.
- #define rtos_exit_critical OS_EXIT_CRITICAL
  Allows preemption.

30.5.1 Data Structure Documentation

30.5.1.1 struct EVENT_UCOSII

Data Fields

- OS_FLAG_GRP * pGrp
  Pointer to uCOS-II's event entity.
- event_clear_type clearType
  Auto clear or manual clear.

30.5.1.2 struct MSGQ_STRUCT_UCOSII

Data Fields

- OS_EVENT * pq
  pointer to the queue

- void ** msgTbl
  pointer to the array that saves the pointers to messages
- uint16_t size
  size of the message in words

30.5.2 Macro Definition Documentation

30.5.2.1 #define kSyncWaitForever 0xFFFFFFFFU

30.5.2.2 #define FSL_RTOS_TASK_DEFINE(*task, stackSize, name, usesFloat*)

Value:

```
task_stack_t fslTaskStack_##task[(stackSize)/sizeof(task_stack_t)]; \
    uint8_t fslTaskName_##task[] = name;                             \
    bool fslTaskFloatUse_##task = usesFloat
```

Parameters

task	The task function.
stackSize	Number of elements in the stack for this task.
name	String to assign to the task.
usesFloat	Boolean that indicates whether the task uses the floating point unit.

30.5.2.3 #define task_create(*task, priority, param, handler*)

Value:

```
__task_create(task,                                          \
                  fslTaskName_##task,                            \
                  sizeof(fslTaskStack_##task),                   \
                  fslTaskStack_##task,                           \
                  priority,                                      \
                  param,                                         \
                  fslTaskFloatUse_##task,                        \
                  handler)
```

Parameters

task	The task function.
priority	Initial priority of the task.
param	Pointer to be passed to the task when it is created.
handler	Returns the identifier to be used afterwards to destroy the task.

Return values

kSuccess	The task was successfully created.
kError	Creation of task failed.

30.5.2.4 #define sync_object_declare(*obj*) sync_object_t obj = (sync_object_t)0

To be used instead of a standard declaration.

Parameters

obj	The sync object to create.

30.5.2.5 #define lock_object_declare(*obj*) lock_object_t obj = (lock_object_t)0

To be used instead of a standard declaration.

Parameters

obj	The lock object to create.

30.5.2.6 #define MSG_QUEUE_DECLARE(*name, number, size*)

Value:

```
void* msgTbl##name[number];                \
    msg_queue_t name = {                       \
        .msgTbl = msgTbl##name                 \
    }
```

Parameters

name	Identifier for the memory region.
number	Number of elements in the queue.
size	Size of every elements in words.

30.6 µC/OS-III Abstraction Layer

The Kinetis SDK provides the µC/OS-III for synchronization, mutual exclusion, message queue, etc.

Data Structures

- struct EVENT_UCOSIII
  Type for an event group object in uCOS-III. More...

Macros

- #define kSyncWaitForever 0xFFFFFFFFU
  Constant to pass as timeout value in order to wait indefinitely.
- #define FSL_RTOS_CURRENT_TASK ((task_handler_t)OSTCBCurPtr)
  Macro passed to the task_destroy function to destroy the current task.

Typedefs

- typedef OS_TCB * task_handler_t
  Type for a task handler, returned by the task_create function.
- typedef CPU_STK task_stack_t
  Type for a task stack.
- typedef void(* task_t)(void *start)
  Type for a task pointer.
- typedef OS_SEM sync_object_t
  Type for an synchronization object.
- typedef OS_MUTEX lock_object_t
  Type for a resource locking object.
- typedef OS_FLAGS event_group_t
  Type for an event flags group, bit 32 is reserved.
- typedef EVENT_UCOSIII event_object_t
  Type for an event group object.
- typedef OS_Q msg_queue_t
  Type for a message queue declaration and creation.
- typedef msg_queue_t * msg_queue_handler_t
  Type for a message queue handler.
- typedef void * msg_queue_item_t
  Type for a message queue item.

Thread management

- #define FSL_RTOS_TASK_DEFINE(task, stackSize, name, usesFloat)
  Creates a task descriptor that is used to create the task with task_create.
- #define task_create(task, priority, param, handler)
  Creates and sets the task to active.

Synchronization

- #define sync_object_declare(obj) sync_object_t obj
  Create the synchronization object.

Resource locking

- #define lock_object_declare(obj) lock_object_t obj
  Create the locking object.

Message queues

- #define MSG_QUEUE_DECLARE(name, number, size) msg_queue_t name
  This macro statically reserves the memory required for the queue.

Critical Sections

- #define rtos_enter_critical OS_CRITICAL_ENTER
  Ensures the following code will not be preempted.
- #define rtos_exit_critical OS_CRITICAL_EXIT
  Allows preemption.

30.6.1 Data Structure Documentation

30.6.1.1 struct EVENT_UCOSIII

Data Fields

- OS_FLAG_GRP group
  uCOS-III's event entity
- event_clear_type clearType
  Auto clear or manual clear.

30.6.2 Macro Definition Documentation

30.6.2.1 #define kSyncWaitForever 0xFFFFFFFFU

30.6.2.2 #define FSL_RTOS_TASK_DEFINE(*task, stackSize, name, usesFloat*)

Value:

```
OS_TCB TCB_##task;                                      \
```

```
task_stack_t fslTaskStack_##task[(stackSize)/sizeof(
  task_stack_t)]; \
uint8_t fslTaskName_##task[] = name;                                    \
bool fslTaskFloatUse_##task = usesFloat
```

Parameters

task	The task function.
stackSize	Number of elements in the stack for this task.
name	String to assign to the task.
usesFloat	Boolean that indicates whether the task uses the floating point unit.

30.6.2.3 #define task_create(*task, priority, param, handler*)

Value:

```
(*(handler) = &(TCB_##task),                                    \
    __task_create(task,                                          \
                  fslTaskName_##task,                            \
                  sizeof(fslTaskStack_##task),                   \
                  fslTaskStack_##task,                           \
                  priority,                                      \
                  param,                                         \
                  fslTaskFloatUse_##task,                        \
                  handler))
```

Parameters

task	The task function.
priority	Initial priority of the task.
param	Pointer to be passed to the task when it is created.
handler	Returns the identifier to be used afterwards to destroy the task.

Return values

kSuccess	The task was successfully created.
kError	Creation of task failed.

30.6.2.4 #define sync_object_declare(*obj*) sync_object_t obj

To be used instead of a standard declaration.

Parameters

obj	The sync object to create.

30.6.2.5 #define lock_object_declare(*obj*) lock_object_t obj

To be used instead of a standard declaration.

Parameters

obj	The lock object to create.

30.6.2.6 #define MSG_QUEUE_DECLARE(*name, number, size*) msg_queue_t name

Parameters

name	Identifier for the memory region.
number	Number of elements in the queue.
size	Size of every elements in words.

30.6.3 Typedef Documentation

30.6.3.1 typedef OS_TCB* task_handler_t

30.6.3.2 typedef CPU_STK task_stack_t

30.6.3.3 typedef void(* task_t)(void *start)

30.6.3.4 typedef OS_SEM sync_object_t

30.6.3.5 typedef OS_MUTEX lock_object_t

30.6.3.6 typedef OS_FLAGS event_group_t

30.6.3.7 typedef OS_Q msg_queue_t

30.6.3.8 typedef void* msg_queue_item_t

30.7 FreeRTOS Abstraction Layer

The Kinetis SDK provides the FreeRTOS Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

Data Structures

- struct EVENT_FREE_RTOS
  Type for an event group object in FreeRTOS. More...

Macros

- #define kSyncWaitForever 0xFFFFFFFFU
  Constant to pass as timeout value in order to wait indefinitely.
- #define FSL_RTOS_CURRENT_TASK ((task_handler_t)NULL)
  Macro passed to the task_destroy function to destroy the current task.

Typedefs

- typedef xTaskHandle task_handler_t
  Type for a task handler, returned by the task_create function.
- typedef portSTACK_TYPE task_stack_t
  Type for a task stack.
- typedef pdTASK_CODE task_t
  Type for a task function.
- typedef xSemaphoreHandle lock_object_t
  Type for a lock object.
- typedef xSemaphoreHandle sync_object_t
  Type for a synchronization object.
- typedef uint32_t event_group_t
  Type for an event group object.
- typedef EVENT_FREE_RTOS event_object_t
  Type for an event group object.
- typedef xQueueHandle msg_queue_t
  Type for a message queue declaration and creation.
- typedef xQueueHandle msg_queue_handler_t
  Type for a message queue handler.
- typedef void * msg_queue_item_t
  Type for a message queue item.

Synchronization

- #define sync_object_declare(obj) sync_object_t obj=(sync_object_t)0
  Create the synchronization object.

Resource locking

- #define lock_object_declare(obj) lock_object_t obj=(lock_object_t)0
  Create the locking object.

Thread management

- #define FSL_RTOS_TASK_DEFINE(task, stackSize, name, usesFloat)
  Creates a task descriptor that is used to create the task with task_create.
- #define task_create(task, priority, param, handler)
  Creates and sets the task to active.

Message queues

- #define MSG_QUEUE_DECLARE(name, number, size) xQueueHandle name
  This macro statically reserves the memory required for the queue.

Critical Sections

- #define rtos_enter_critical taskENTER_CRITICAL
  Ensures the following code will not be preempted.
- #define rtos_exit_critical taskEXIT_CRITICAL
  Allows preemption.

30.7.1 Data Structure Documentation

30.7.1.1 struct EVENT_FREE_RTOS

Data Fields

- event_group_t flags
  Event flags.
- event_clear_type clearType
  Auto clear or manual clear.
- xSemaphoreHandle flagsSem
  Semaphore to protect flags.

30.7.2 Macro Definition Documentation

30.7.2.1 #define kSyncWaitForever 0xFFFFFFFFU

30.7.2.2 #define FSL_RTOS_CURRENT_TASK ((task_handler_t)NULL)

30.7.2.3 #define sync_object_declare(*obj*) sync_object_t obj=(sync_object_t)0

To be used instead of a standard declaration.

Parameters

obj	The sync object to create.

30.7.2.4 #define lock_object_declare(*obj*) lock_object_t obj=(lock_object_t)0

To be used instead of a standard declaration.

Parameters

obj	The lock object to create.

30.7.2.5 #define FSL_RTOS_TASK_DEFINE(*task, stackSize, name, usesFloat*)

Value:

```
static signed portCHAR fslTaskName_##task[]=name;          \
    static int fslTaskStackSize_##task = stackSize
```

Parameters

task	The task function.
stackSize	Number of elements in the stack for this task.
name	String to assign to the task.
usesFloat	Boolean that indicates whether the task uses the floating point unit.

30.7.2.6 #define task_create(*task, priority, param, handler*)

Value:

```
__task_create(task,                                  \
                (uint8_t *)fslTaskName_##task, \
                 fslTaskStackSize_##task,              \
                 NULL,                                 \
                 priority,                             \
                 param,                                \
                 false,                                \
                 handler)
```

Parameters

task	The task function.
priority	Initial priority of the task.
param	Pointer to be passed to the task when it is created.
handler	Returns the identifier to be used afterwards to destroy the task.

Return values

kSuccess	The task was successfully created.
kError	Creation of task failed.

30.7.2.7 #define MSG_QUEUE_DECLARE(*name, number, size*) xQueueHandle name

Parameters

name	Identifier for the memory region.
number	Number of elements in the queue.
size	Size of every elements in words.

30.7.3 Typedef Documentation

30.7.3.1 typedef xTaskHandle task_handler_t

30.7.3.2 typedef portSTACK_TYPE task_stack_t

30.7.3.3 typedef pdTASK_CODE task_t

30.7.3.4 typedef xSemaphoreHandle lock_object_t

30.7.3.5 typedef xSemaphoreHandle sync_object_t

30.7.3.6 typedef uint32_t event_group_t

30.7.3.7 typedef xQueueHandle msg_queue_t

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: KSDKAPIRM
Rev. 1.0.0-beta
03/2014