S32 SDK API Documentation

RTM 1.0.0

Generated by Doxygen 1.8.12

Mon Mar 27 2017 17:32:40

Contents

1 MISRA-C:2012 violations

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Module Documentation

3.1 ADC Driver

3.1.1 Detailed Description

Analog to Digital Converter Peripheral Driver.

Hardware background

The ADC of the S32K144 is a selectable resolution (8, 10, 12-bit), single-ended, SAR converter. It has 20 selectable input channels (16 external, 4 internal) and 16 control channels (each with a result register, a channel selection and interrupt enable).

Sample time is configurable through selection of A/D clock and a configurable sample time (in A/D clocks).

Also provided are the Hardware Average and Hardware Compare Features.

Hardware Average will sample a selectable number of measurements and average them before signaling a Conversion Complete.

Hardware Compare can be used to signal if an input channel goes outside (or inside) of a predefined range.

The **Calibration** features can be used to automatically calibrate or fine-tune the ADC before use.

Driver consideration

The ADC Driver provides access to all features, but not all need to be configured to use the ADC. The user application can use the default for most settings, changing only what is necessary. For example, if Compare or Average features are not used, the user does not need to configure them.

The Driver uses structures for configuration. Each structure contains members that are specific to its respective functionality. There is a **converter** structure, a hardware **compare** structure, a hardware **average** structure and a **calibration** structure. Each struct has a corresponding `InitStruct()` method that can be used to initialize the members to reset values, so the user can change only the values that are specific to the application.

All methods that access the hardware layer will return an error code to signal if the operation succeeded or failed. These values are defined by the adc_drv_status_t enumeration, and the possible values are:

- ADC_DRV_SUCCESS: operation succeeded
- ADC_DRV_FAIL: operation failed

Interrupt handling

The ADC Driver in S32 SDK does not use interrupts internally. These can be defined by the user application. There are two ways to add an ADC interrupt:

1. Using the weak symbols defined by start-up code. if the methods `ADC`**`x`**`_Handler(void)` (x denotes instance number) are not defined, the linker use a default ISR. An error will be generated if methods with the same name are defined multiple times. This method works regardless of the placement of the interrupt vector table (Flash or RAM).
2. Using the Interrupt Manager's `INT_SYS_InstallHandler()` method. This can be used to dynamically change the ISR at run-time. This method works only if the interrupt vector table is located in RAM (S32 SDK behavior). To get the ADC instance's interrupt number, use `ADC_DRV_GetInterruptNumber()`.

Clocking and pin configuration

The ADC Driver does not handle clock setup (from PCC) or any kind of pin configuration (done by PORT module). This is handled by the Clock Manager and PORT module, respectively. The driver assumes that correct clock configurations have been made, so it is the user's responsibility to set up clocking and pin configurations correctly.

Data Structures

- struct adc_converter_config_t

  Defines the converter configuration. More...
- struct adc_compare_config_t

  Defines the hardware compare configuration. More...
- struct adc_average_config_t

  Defines the hardware average configuration. More...
- struct adc_chan_config_t

  Defines the control channel configuration. More...
- struct adc_calibration_t

  Defines the user calibration configuration. More...

Enumerations

- enum adc_latch_clear_t { ADC_LATCH_CLEAR_WAIT, ADC_LATCH_CLEAR_FORCE }

  Defines the trigger latch clear method Implements : adc_latch_clear_t_Class.

Converter

Converter specific methods. These are used to configure and use the A/D Converter specific functionality, including:

- clock input and divider
- sample time in A/D clocks
- resolution
- trigger source
- voltage reference
- enable DMA
- enable continuous conversion on one channel

To start a conversion, a control channel (see Channel Configuration) and a trigger source must be configured. Once a conversion is started, the user application can wait for it to be finished by calling the ADC_DRV_WaitConvDone() function.

Only the first control channel can be triggered by software. To start a conversion in this case, an input channel must be written in the channel selection register using the ADC_DRV_ConfigChan() method. Writing a value to the control channel while a conversion is being performed on that channel will start a new conversion.

- void ADC_DRV_InitConverterStruct (adc_converter_config_t ∗const config)

  Initializes the converter configuration structure.
- void ADC_DRV_ConfigConverter (const uint32_t instance, const adc_converter_config_t ∗const config)

  Configures the converter with the given configuration structure.
- void ADC_DRV_GetConverterConfig (const uint32_t instance, adc_converter_config_t ∗const config)

  Gets the current converter configuration.
- void ADC_DRV_Reset (const uint32_t instance)

  Resets the converter (sets all configurations to reset values)
- void ADC_DRV_WaitConvDone (const uint32_t instance)

  Waits for a conversion/calibration to finish.

Hardware Compare

The Hardware Compare feature of the S32K144 ADC is a versatile mechanism that can be used to monitor that a value is within certain values. Measurements can be monitored to be within certain ranges:

- less than/ greater than a fixed value
- inside or outside of a certain range

Two compare values can be configured (the second value is used only for range function mode). The compare values must be written in 12-bit resolution mode regardless of the actual used resolution mode.

Once the hardware compare feature is enabled, a conversion is considered complete only when the measured value is within the allowable range set by the configuration.

- void ADC_DRV_InitHwCompareStruct (adc_compare_config_t ∗const config)

  Initializes the Hardware Compare configuration structure.
- void ADC_DRV_ConfigHwCompare (const uint32_t instance, const adc_compare_config_t ∗const config)

  Configures the Hardware Compare feature with the given configuration structure.
- void ADC_DRV_GetHwCompareConfig (const uint32_t instance, adc_compare_config_t ∗const config)

  Gets the current Hardware Compare configuration.

Hardware Average

The Hardware Average feature of the S32K144 allows for a set of measurements to be averaged together as a single conversion. The number of samples to be averaged is selectable (4, 8, 16 or 32 samples).

- void ADC_DRV_InitHwAverageStruct (adc_average_config_t ∗const config)

  Initializes the Hardware Average configuration structure.
- void ADC_DRV_ConfigHwAverage (const uint32_t instance, const adc_average_config_t ∗const config)

  Configures the Hardware Average feature with the given configuration structure.
- void ADC_DRV_GetHwAverageConfig (const uint32_t instance, adc_average_config_t ∗const config)

  Gets the current Hardware Average configuration.

Channel configuration

Control register specific functions. These functions control configurations for each control channel (input channel selection and interrupt enable).

When software triggering is enabled, calling the ADC_DRV_ConfigChan() method for control channel 0 starts a new conversion.

After a conversion is finished, the result can be retrieved using the ADC_DRV_GetChanResult() method.

- void ADC_DRV_InitChanStruct (adc_chan_config_t ∗const config)

  Initializes the control channel configuration structure.
- void ADC_DRV_ConfigChan (const uint32_t instance, const uint8_t chanIndex, const adc_chan_config_↩ t ∗const config)

  Configures the selected control channel with the given configuration structure.
- void ADC_DRV_GetChanConfig (const uint32_t instance, const uint8_t chanIndex, adc_chan_config_↩ t ∗const config)

  Gets the current control channel configuration for the selected channel index.
- void ADC_DRV_GetChanResult (const uint32_t instance, const uint8_t chanIndex, uint16_t ∗const result)

  Gets the last result for the selected control channel.

Automatic Calibration

These methods control the Calibration feature of the ADC.

The ADC_DRV_AutoCalibration() method can be called to execute a calibration sequence, or a calibration can be retrieved with the ADC_DRV_GetUserCalibration() and saved to non-volatile storage, to avoid calibration on every power-on. The calibration structure can be written with the ADC_DRV_ConfigUserCalibration() method.

- void ADC_DRV_AutoCalibration (const uint32_t instance)

  Executes an Auto-Calibration.
- void ADC_DRV_InitUserCalibrationStruct (adc_calibration_t ∗const config)

  Initializes the User Calibration configuration structure.
- void ADC_DRV_ConfigUserCalibration (const uint32_t instance, const adc_calibration_t ∗const config)

  Configures the User Calibration feature with the given configuration structure.
- void ADC_DRV_GetUserCalibration (const uint32_t instance, adc_calibration_t ∗const config)

  Gets the current User Calibration configuration.

Interrupts

This method returns the interrupt number for an ADC instance, which can be used to configure the interrupt, like in Interrupt Manager.

- IRQn_Type ADC_DRV_GetInterruptNumber (const uint32_t instance)

  Returns the interrupt number for the ADC instance.

Latched triggers processing

These functions provide basic operations for using the trigger latch mechanism.

- void ADC_DRV_ClearLatchedTriggers (const uint32_t instance, const adc_latch_clear_t clearMode)

  Clear latched triggers under processing.
- void ADC_DRV_ClearTriggerErrors (const uint32_t instance)

  Clear all latch trigger error.
- uint32_t ADC_DRV_GetTriggerErrorFlags (const uint32_t instance)

  This function returns the trigger error flags bits of the ADC instance.

3.1.2 Data Structure Documentation

3.1.2.1 struct adc_converter_config_t

Defines the converter configuration.

This structure is used to configure the ADC converter

Implements : adc_converter_config_t_Class

Data Fields

- adc_clk_divide_t clockDivide
- uint8_t sampleTime
- adc_resolution_t resolution
- adc_input_clock_t inputClock
- adc_trigger_t trigger
- bool dmaEnable
- adc_voltage_reference_t voltageRef
- bool continuousConvEnable

Field Documentation

3.1.2.1.1 clockDivide

```
adc_clk_divide_t clockDivide
```

Divider of the input clock for the ADC

3.1.2.1.2 continuousConvEnable

```
bool continuousConvEnable
```

Enable Continuous conversions

3.1.2.1.3 dmaEnable

```
bool dmaEnable
```

Enable DMA for the ADC

3.1.2.1.4 inputClock

```
adc_input_clock_t inputClock
```

Input clock source

3.1.2.1.5 resolution

```
adc_resolution_t resolution
```

ADC resolution (8,10,12 bit)

3.1.2.1.6 sampleTime

```
uint8_t sampleTime
```

Sample time in AD Clocks

3.1.2.1.7 trigger

```
adc_trigger_t trigger
```

ADC trigger type (software, hardware)

3.1.2.1.8 voltageRef

```
adc_voltage_reference_t voltageRef
```

Voltage reference used

3.1.2.2 struct adc_compare_config_t

Defines the hardware compare configuration.

This structure is used to configure the hardware compare feature for the ADC

Implements : adc_compare_config_t_Class

Data Fields

- bool compareEnable
- bool compareGreaterThanEnable
- bool compareRangeFuncEnable
- uint16_t compVal1
- uint16_t compVal2

Field Documentation

3.1.2.2.1 compareEnable

```
bool compareEnable
```

Enable the compare feature

3.1.2.2.2 compareGreaterThanEnable

```
bool compareGreaterThanEnable
```

Enable Greater-Than functionality

3.1.2.2.3 compareRangeFuncEnable

```
bool compareRangeFuncEnable
```

Enable Range functionality

3.1.2.2.4 compVal1

```
uint16_t compVal1
```

First Compare Value

3.1.2.2.5 compVal2

```
uint16_t compVal2
```

Second Compare Value

3.1.2.3 struct adc_average_config_t

Defines the hardware average configuration.

This structure is used to configure the hardware average feature for the ADC

Implements : adc_average_config_t_Class

Data Fields

- bool hwAvgEnable
- adc_average_t hwAverage

Field Documentation

3.1.2.3.1 hwAverage

```
adc_average_t hwAverage
```

Selection for number of samples used for averaging

3.1.2.3.2 hwAvgEnable

```
bool hwAvgEnable
```

Enable averaging functionality

3.1.2.4 struct adc_chan_config_t

Defines the control channel configuration.

This structure is used to configure a control channel of the ADC

Implements : adc_chan_config_t_Class

Data Fields

- bool interruptEnable
- adc_inputchannel_t channel

Field Documentation

3.1.2.4.1 channel

```
adc_inputchannel_t channel
```

Selection of input channel for measurement

3.1.2.4.2 interruptEnable

```
bool interruptEnable
```

Enable interrupts for this channel

3.1.2.5 struct adc_calibration_t

Defines the user calibration configuration.

This structure is used to configure the user calibration parameters of the ADC.

Implements : adc_calibration_t_Class

Data Fields

- uint16_t userGain
- uint16_t userOffset

Field Documentation

3.1.2.5.1 userGain

```
uint16_t userGain
```

User-configurable gain

3.1.2.5.2 userOffset

```
uint16_t userOffset
```

User-configurable Offset (2's complement, subtracted from result)

3.1.3 Enumeration Type Documentation

3.1.3.1 adc_latch_clear_t

```
enum adc_latch_clear_t
```

Defines the trigger latch clear method Implements : adc_latch_clear_t_Class.

Enumerator

ADC_LATCH_CLEAR_WAIT	Clear by waiting all latched triggers to be processed
ADC_LATCH_CLEAR_FORCE	Process current trigger and clear all latched

3.1.4 Function Documentation

3.1.4.1 ADC_DRV_AutoCalibration()

```
void ADC_DRV_AutoCalibration (
             const uint32_t instance )
```

Executes an Auto-Calibration.

This functions executes an Auto-Calibration sequence. It is recommended to run this sequence before using the ADC converter.

Parameters

in	*instance*	instance number

3.1.4.2 ADC_DRV_ClearLatchedTriggers()

```
void ADC_DRV_ClearLatchedTriggers (
             const uint32_t instance,
             const adc_latch_clear_t clearMode )
```

Clear latched triggers under processing.

This function clears all trigger latched flags of the ADC instance. This function must be called after the hardware trigger source for the ADC has been deactivated.

Parameters

in	*instance*	instance number of the ADC
in	*clearMode*	The clearing method for the latched triggers • ADC_LATCH_CLEAR_WAIT : Wait for all latched triggers to be processed. • ADC_LATCH_CLEAR_FORCE : Clear latched triggers and wait for trigger being process to finish.

3.1.4.3 ADC_DRV_ClearTriggerErrors()

```
void ADC_DRV_ClearTriggerErrors (
             const uint32_t instance )
```

Clear all latch trigger error.

This function clears all trigger error flags of the ADC instance.

Parameters

in	*instance*	instance number of the ADC

3.1.4.4 ADC_DRV_ConfigChan()

```
void ADC_DRV_ConfigChan (
             const uint32_t instance,
             const uint8_t chanIndex,
             const adc_chan_config_t *const config )
```

Configures the selected control channel with the given configuration structure.

This function sets a control channel configuration. In Software Trigger mode, only control channel (chanIndex) 0 can start conversions.

Parameters

in	*instance*	instance number
in	*chanIndex*	the control channel index
in	*config*	the configuration structure

3.1.4.5 ADC_DRV_ConfigConverter()

```
void ADC_DRV_ConfigConverter (
            const uint32_t instance,
            const adc_converter_config_t *const config )
```

Configures the converter with the given configuration structure.

This function configures the ADC converter with the options provided in the provided structure.

Parameters

in	*instance*	instance number
in	*config*	the configuration structure

3.1.4.6 ADC_DRV_ConfigHwAverage()

```
void ADC_DRV_ConfigHwAverage (
            const uint32_t instance,
            const adc_average_config_t *const config )
```

Configures the Hardware Average feature with the given configuration structure.

This function sets the configuration for the Hardware Average feature.

Parameters

in	*instance*	instance number
in	*config*	the configuration structure

3.1.4.7 ADC_DRV_ConfigHwCompare()

```
void ADC_DRV_ConfigHwCompare (
            const uint32_t instance,
            const adc_compare_config_t *const config )
```

Configures the Hardware Compare feature with the given configuration structure.

This functions sets the configuration for the Hardware Compare feature using the configuration structure.

Parameters

in	*instance*	instance number
in	*config*	the configuration structure

3.1.4.8 ADC_DRV_ConfigUserCalibration()

```
void ADC_DRV_ConfigUserCalibration (
             const uint32_t instance,
             const adc_calibration_t *const config )
```

Configures the User Calibration feature with the given configuration structure.

This function sets the configuration for the user calibration registers.

Parameters

in	*instance*	instance number
in	*config*	the configuration structure

3.1.4.9 ADC_DRV_GetChanConfig()

```
void ADC_DRV_GetChanConfig (
             const uint32_t instance,
             const uint8_t chanIndex,
             adc_chan_config_t *const config )
```

Gets the current control channel configuration for the selected channel index.

This function returns the configuration for a control channel

Parameters

in	*instance*	instance number
in	*chanIndex*	the control channel index
out	*config*	the configuration structure

3.1.4.10 ADC_DRV_GetChanResult()

```
void ADC_DRV_GetChanResult (
             const uint32_t instance,
             const uint8_t chanIndex,
             uint16_t *const result )
```

Gets the last result for the selected control channel.

This function returns the conversion result from a control channel.

Parameters

in	*instance*	instance number
in	*chanIndex*	the converter control channel index
out	*result*	the result raw value

3.1.4.11 ADC_DRV_GetConverterConfig()

```
void ADC_DRV_GetConverterConfig (
            const uint32_t instance,
            adc_converter_config_t *const config )
```

Gets the current converter configuration.

This functions returns the configuration for converter in the form of a configuration structure.

Parameters

in	*instance*	instance number
out	*config*	the configuration structure

3.1.4.12 ADC_DRV_GetHwAverageConfig()

```
void ADC_DRV_GetHwAverageConfig (
            const uint32_t instance,
            adc_average_config_t *const config )
```

Gets the current Hardware Average configuration.

This function returns the configuration for the Hardware Average feature.

Parameters

in	*instance*	instance number
out	*config*	the configuration structure

3.1.4.13 ADC_DRV_GetHwCompareConfig()

```
void ADC_DRV_GetHwCompareConfig (
            const uint32_t instance,
            adc_compare_config_t *const config )
```

Gets the current Hardware Compare configuration.

This function returns the configuration for the Hardware Compare feature.

Parameters

in	*instance*	instance number
out	*config*	the configuration structure

3.1.4.14 ADC_DRV_GetInterruptNumber()

```
IRQn_Type ADC_DRV_GetInterruptNumber (
            const uint32_t instance )
```

Returns the interrupt number for the ADC instance.

This function returns the interrupt number for the specified ADC instance.

Parameters

in	*instance*	instance number of the ADC

Returns

irq_number: the interrupt number (index) of the ADC instance, used to configure the interrupt

3.1.4.15 ADC_DRV_GetTriggerErrorFlags()

```
uint32_t ADC_DRV_GetTriggerErrorFlags (
             const uint32_t instance )
```

This function returns the trigger error flags bits of the ADC instance.

Parameters

in	*instance*	instance number of the ADC

Returns

trigErrorFlags The Trigger Error Flags bit-mask

3.1.4.16 ADC_DRV_GetUserCalibration()

```
void ADC_DRV_GetUserCalibration (
             const uint32_t instance,
             adc_calibration_t *const config )
```

Gets the current User Calibration configuration.

This function returns the current user calibration register values.

Parameters

in	*instance*	instance number
out	*config*	the configuration structure

3.1.4.17 ADC_DRV_InitChanStruct()

```
void ADC_DRV_InitChanStruct (
             adc_chan_config_t *const config )
```

Initializes the control channel configuration structure.

This function initializes the control channel configuration structure to default values (Reference Manual resets). This function should be called on a structure before using it to configure a channel (ADC_DRV_ConfigChan), otherwise all members must be written by the caller. This function insures that all members are written with safe values, so the user can modify only the desired members.

Parameters

out	*config*	the configuration structure

3.1.4.18 ADC_DRV_InitConverterStruct()

```
void ADC_DRV_InitConverterStruct (
            adc_converter_config_t *const config )
```

Initializes the converter configuration structure.

This function initializes the members of the adc_converter_config_t structure to default values (Reference Manual resets). This function should be called on a structure before using it to configure the converter with ADC_DRV↩_ConfigConverter(), otherwise all members must be written (initialized) by the user. This function insures that all members are written with safe values, so the user can modify only the desired members.

Parameters

out	*config*	the configuration structure

3.1.4.19 ADC_DRV_InitHwAverageStruct()

```
void ADC_DRV_InitHwAverageStruct (
            adc_average_config_t *const config )
```

Initializes the Hardware Average configuration structure.

This function initializes the Hardware Average configuration structure to default values (Reference Manual resets). This function should be called before configuring the Hardware Average feature (ADC_DRV_ConfigHwAverage), otherwise all members must be written by the caller. This function insures that all members are written with safe values, so the user can modify the desired members.

Parameters

out	*config*	the configuration structure

3.1.4.20 ADC_DRV_InitHwCompareStruct()

```
void ADC_DRV_InitHwCompareStruct (
            adc_compare_config_t *const config )
```

Initializes the Hardware Compare configuration structure.

This function initializes the Hardware Compare configuration structure to default values (Reference Manual resets). This function should be called before configuring the Hardware Compare feature (ADC_DRV_ConfigHwCompare), otherwise all members must be written by the caller. This function insures that all members are written with safe values, so the user can modify the desired members.

Parameters

out	*config*	the configuration structure

3.1.4.21 ADC_DRV_InitUserCalibrationStruct()

```
void ADC_DRV_InitUserCalibrationStruct (
            adc_calibration_t *const config )
```

Initializes the User Calibration configuration structure.

This function initializes the User Calibration configuration structure to default values (Reference Manual resets). This function should be called on a structure before using it to configure the User Calibration feature (ADC_DRV_↩ ConfigUserCalibration), otherwise all members must be written by the caller. This function insures that all members are written with safe values, so the user can modify only the desired members.

Parameters

out	*config*	the configuration structure

3.1.4.22 ADC_DRV_Reset()

```
void ADC_DRV_Reset (
            const uint32_t instance )
```

Resets the converter (sets all configurations to reset values)

This function resets all the internal ADC registers to reset values.

Parameters

in	*instance*	instance number

3.1.4.23 ADC_DRV_WaitConvDone()

```
void ADC_DRV_WaitConvDone (
            const uint32_t instance )
```

Waits for a conversion/calibration to finish.

This functions waits for a conversion to complete by continuously polling the Conversion Active Flag.

Parameters

in	*instance*	instance number

3.2 ADC HAL

3.2.1 Detailed Description

Analog to Digital Converter Hardware Abstraction Layer.

This HAL provides low-level access to all hardware features of the ADC.

Enumerations

- enum adc_clk_divide_t { ADC_CLK_DIVIDE_1 = 0x00U, ADC_CLK_DIVIDE_2 = 0x01U, ADC_CLK_DIVI↩ DE_4 = 0x02U, ADC_CLK_DIVIDE_8 = 0x03U }

  Clock Divider selection Implements : adc_clk_divide_t_Class.
- enum adc_resolution_t { ADC_RESOLUTION_8BIT = 0x00U, ADC_RESOLUTION_12BIT = 0x01U, ADC↩ _RESOLUTION_10BIT = 0x02U }

  Conversion resolution selection Implements : adc_resolution_t_Class.
- enum adc_input_clock_t { ADC_CLK_ALT_1 = 0x00U, ADC_CLK_ALT_2 = 0x01U, ADC_CLK_ALT_3 = 0x02U, ADC_CLK_ALT_4 = 0x03U }

  Input clock source selection Implements : adc_input_clock_t_Class.
- enum adc_trigger_t { ADC_TRIGGER_SOFTWARE = 0x00U, ADC_TRIGGER_HARDWARE = 0x01U }

  Trigger type selection Implements : adc_trigger_t_Class.
- enum adc_voltage_reference_t { ADC_VOLTAGEREF_VREF = 0x00U, ADC_VOLTAGEREF_VALT = 0x01U }

  Voltage reference selection Implements : adc_voltage_reference_t_Class.
- enum adc_average_t { ADC_AVERAGE_4 = 0x00U, ADC_AVERAGE_8 = 0x01U, ADC_AVERAGE_16 = 0x02U, ADC_AVERAGE_32 = 0x03U }

  Hardware average selection Implements : adc_average_t_Class.

Input Channel selection.

Enumerations and macros to select Input channel.

- enum adc_inputchannel_t {
  ADC_INPUTCHAN_AD0 = 0x00U, ADC_INPUTCHAN_AD1 = 0x01U, ADC_INPUTCHAN_AD2 = 0x02U, A↩ DC_INPUTCHAN_AD3 = 0x03U,
  ADC_INPUTCHAN_AD4 = 0x04U, ADC_INPUTCHAN_AD5 = 0x05U, ADC_INPUTCHAN_AD6 = 0x06U, A↩ DC_INPUTCHAN_AD7 = 0x07U,
  ADC_INPUTCHAN_AD8 = 0x08U, ADC_INPUTCHAN_AD9 = 0x09U, ADC_INPUTCHAN_AD10 = 0x0AU, ADC_INPUTCHAN_AD11 = 0x0BU,
  ADC_INPUTCHAN_AD12 = 0x0CU, ADC_INPUTCHAN_AD13 = 0x0DU, ADC_INPUTCHAN_AD14 = 0x0↩ EU, ADC_INPUTCHAN_AD15 = 0x0FU,
  ADC_INPUTCHAN_AD16 = 0x10U, ADC_INPUTCHAN_AD17 = 0x11U, ADC_INPUTCHAN_AD18 = 0x12U, ADC_INPUTCHAN_AD19 = 0x13U,
  ADC_INPUTCHAN_AD20 = 0x14U, ADC_INPUTCHAN_AD21 = 0x15U, ADC_INPUTCHAN_AD22 = 0x16U, ADC_INPUTCHAN_AD23 = 0x17U,
  ADC_INPUTCHAN_AD24 = 0x18U, ADC_INPUTCHAN_AD25 = 0x19U, ADC_INPUTCHAN_AD26 = 0x1AU, ADC_INPUTCHAN_AD27 = 0x1BU,
  ADC_INPUTCHAN_AD28 = 0x1CU, ADC_INPUTCHAN_AD29 = 0x1DU, ADC_INPUTCHAN_AD30 = 0x1↩ EU, ADC_INPUTCHAN_AD31 = 0x1FU }

  Input channel selection Implements : adc_inputchannel_t_Class.

Converter

General ADC functions.

- void ADC_HAL_Init (ADC_Type ∗const baseAddr)

  Initializes the ADC instance to reset values.
- static bool ADC_HAL_GetConvActiveFlag (const ADC_Type ∗const baseAddr)

  Gets the Conversion Active Flag.
- static adc_clk_divide_t ADC_HAL_GetClockDivide (const ADC_Type ∗const baseAddr)

  Gets the current ADC clock divider configuration.
- static void ADC_HAL_SetClockDivide (ADC_Type ∗const baseAddr, const adc_clk_divide_t clockDivide)

  Sets the ADC clock divider configuration.
- static uint8_t ADC_HAL_GetSampleTime (const ADC_Type ∗const baseAddr)

  Gets the Sample time in AD clock cycles.
- static void ADC_HAL_SetSampleTime (ADC_Type ∗const baseAddr, uint8_t sampletime)

  Sets the Sample time in AD clock cycles.
- static adc_resolution_t ADC_HAL_GetResolution (const ADC_Type ∗const baseAddr)

  Gets the Resolution Mode configuration.
- static void ADC_HAL_SetResolution (ADC_Type ∗const baseAddr, const adc_resolution_t resolution)

  Sets the Resolution Mode configuration.
- static adc_input_clock_t ADC_HAL_GetInputClock (const ADC_Type ∗const baseAddr)

  Gets the AD Clock Input configuration.
- static void ADC_HAL_SetInputClock (ADC_Type ∗const baseAddr, const adc_input_clock_t inputClock)

  Sets the AD Clock Input configuration.
- static adc_trigger_t ADC_HAL_GetTriggerMode (const ADC_Type ∗const baseAddr)

  Gets the ADC Trigger Mode.
- static void ADC_HAL_SetTriggerMode (ADC_Type ∗const baseAddr, const adc_trigger_t trigger)

  Sets the ADC Trigger Mode.
- static bool ADC_HAL_GetDMAEnableFlag (const ADC_Type ∗const baseAddr)

  Gets the DMA Enable Flag state.
- static void ADC_HAL_SetDMAEnableFlag (ADC_Type ∗const baseAddr, const bool state)

  Sets the DMA Enable Flag state.
- static adc_voltage_reference_t ADC_HAL_GetVoltageReference (const ADC_Type ∗const baseAddr)

  Gets the ADC Reference Voltage selection.
- static void ADC_HAL_SetVoltageReference (ADC_Type ∗const baseAddr, const adc_voltage_reference_↵
t voltageRef)

  Sets the ADC Reference Voltage selection.
- static bool ADC_HAL_GetContinuousConvFlag (const ADC_Type ∗const baseAddr)

  Gets the Continuous Conversion Flag state.
- static void ADC_HAL_SetContinuousConvFlag (ADC_Type ∗const baseAddr, const bool state)

  Sets the Continuous Conversion Flag state.

Hardware Compare.

Functions to configure the Hardware Compare feature.

- static bool ADC_HAL_GetHwCompareEnableFlag (const ADC_Type ∗const baseAddr)

  Gets the Hardware Compare Enable Flag state.
- static void ADC_HAL_SetHwCompareEnableFlag (ADC_Type ∗const baseAddr, const bool state)

Sets the Hardware Compare Enable Flag state.
- static bool ADC_HAL_GetHwCompareGtEnableFlag (const ADC_Type ∗const baseAddr)

  Gets the Hardware Compare Greater Than Enable Flag state.
- static void ADC_HAL_SetHwCompareGtEnableFlag (ADC_Type ∗const baseAddr, const bool state)

  Sets the Hardware Compare Greater Than Enable Flag state.
- static bool ADC_HAL_GetHwCompareRangeEnableFlag (const ADC_Type ∗const baseAddr)

  Gets the Hardware Compare Range Enable state.
- static void ADC_HAL_SetHwCompareRangeEnableFlag (ADC_Type ∗const baseAddr, const bool state)

  Sets the Hardware Compare Range Enable state.
- static uint16_t ADC_HAL_GetHwCompareComp1Value (const ADC_Type ∗const baseAddr)

  Gets the Compare Register 1 value.
- static void ADC_HAL_SetHwCompareComp1Value (ADC_Type ∗const baseAddr, const uint16_t value)

  Sets the Compare Register 1 value.
- static uint16_t ADC_HAL_GetHwCompareComp2Value (const ADC_Type ∗const baseAddr)

  Gets the Compare Register 2 value.
- static void ADC_HAL_SetHwCompareComp2Value (ADC_Type ∗const baseAddr, const uint16_t value)

  Sets the Compare Register 2 value.

Hardware Average.

Functions to configure the Hardware Averaging feature.

- static bool ADC_HAL_GetHwAverageEnableFlag (const ADC_Type ∗const baseAddr)

  Gets the Hardware Average Enable Flag state.
- static void ADC_HAL_SetHwAverageEnableFlag (ADC_Type ∗const baseAddr, const bool state)

  Sets the Hardware Average Enable Flag state.
- static adc_average_t ADC_HAL_GetHwAverageMode (const ADC_Type ∗const baseAddr)

  Gets the Hardware Average Mode.
- static void ADC_HAL_SetHwAverageMode (ADC_Type ∗const baseAddr, const adc_average_t average←Mode)

  Sets the Hardware Average Mode.

Automatic Calibration.

Functions configure and use the Automatic Calibration feature.

- static bool ADC_HAL_GetCalibrationActiveFlag (const ADC_Type ∗const baseAddr)

  Gets the Calibration Active Flag state.
- static void ADC_HAL_SetCalibrationActiveFlag (ADC_Type ∗const baseAddr, const bool state)

  Sets the Calibration Active Flag state.
- static uint16_t ADC_HAL_GetUserGainValue (const ADC_Type ∗const baseAddr)

  Gets the User Gain Register value.
- static void ADC_HAL_SetUserGainValue (ADC_Type ∗const baseAddr, const uint16_t value)

  Sets the User Gain Register value.
- static uint16_t ADC_HAL_GetUserOffsetValue (const ADC_Type ∗const baseAddr)

  Gets the User Offset Register value.
- static void ADC_HAL_SetUserOffsetValue (ADC_Type ∗const baseAddr, const uint16_t value)

  Sets the User Offset Register value.

Converter channels.

Functions to configure and access each ADC converter channel.

- static bool ADC_HAL_GetChanInterruptEnableFlag (const ADC_Type ∗const baseAddr, const uint8_t chanIndex)

  Gets the Channel Interrupt Enable state.
- static void ADC_HAL_SetChanInterruptEnableFlag (ADC_Type ∗const baseAddr, const uint8_t chanIndex, const bool state)

  Sets the Channel Interrupt Enable to a new state.
- static adc_inputchannel_t ADC_HAL_GetInputChannel (const ADC_Type ∗const baseAddr, const uint8_t chanIndex)

  Gets the configured input channel for the selected measurement channel.
- static void ADC_HAL_SetInputChannel (ADC_Type ∗const baseAddr, const uint8_t chanIndex, const adc_inputchannel_t inputChan)

  Sets the input channel configuration for the measurement channel.
- static bool ADC_HAL_GetConvCompleteFlag (const ADC_Type ∗const baseAddr, const uint8_t chanIndex)

  Gets the measurement channel Conversion Complete Flag state.
- static uint16_t ADC_HAL_GetChanResult (const ADC_Type ∗const baseAddr, const uint8_t chanIndex)

  Gets the conversion result for the selected measurement channel.

Trigger latches.

Functions using the trigger latch mechanism.

- static void ADC_HAL_ClearLatchTriggers (ADC_Type ∗const baseAddr)

  Clear the latched triggers.
- static uint32_t ADC_HAL_GetTriggerErrorFlags (const ADC_Type ∗const baseAddr)

  Get the trigger latch error flags.
- static void ADC_HAL_ClearTriggerErrorFlags (ADC_Type ∗const baseAddr)

  Clear the latch trigger error flags.
- static uint32_t ADC_HAL_GetTriggerLatchFlags (const ADC_Type ∗const baseAddr)

  Get the trigger latch flags bits.
- static uint32_t ADC_HAL_GetTriggerProcNumber (const ADC_Type ∗const baseAddr)

  Get the index of the trigger under process.

3.2.2 Enumeration Type Documentation

3.2.2.1 adc_average_t

```
enum adc_average_t
```

Hardware average selection Implements : adc_average_t_Class.

Enumerator

ADC_AVERAGE_4	Hardware average of 4 samples.
ADC_AVERAGE_8	Hardware average of 8 samples.
ADC_AVERAGE_16	Hardware average of 16 samples.
ADC_AVERAGE_32	Hardware average of 32 samples.

3.2.2.2 adc_clk_divide_t

```
enum adc_clk_divide_t
```

Clock Divider selection Implements : adc_clk_divide_t_Class.

Enumerator

ADC_CLK_DIVIDE↩_1	Input clock divided by 1.
ADC_CLK_DIVIDE↩_2	Input clock divided by 2.
ADC_CLK_DIVIDE↩_4	Input clock divided by 4.
ADC_CLK_DIVIDE↩_8	Input clock divided by 8.

3.2.2.3 adc_input_clock_t

```
enum adc_input_clock_t
```

Input clock source selection Implements : adc_input_clock_t_Class.

Enumerator

ADC_CLK_ALT↩_1	Input clock alternative 1.
ADC_CLK_ALT↩_2	Input clock alternative 2.
ADC_CLK_ALT↩_3	Input clock alternative 3.
ADC_CLK_ALT↩_4	Input clock alternative 4.

3.2.2.4 adc_inputchannel_t

```
enum adc_inputchannel_t
```

Input channel selection Implements : adc_inputchannel_t_Class.

Enumerator

ADC_INPUTCHAN_AD0	AD0
ADC_INPUTCHAN_AD1	AD1
ADC_INPUTCHAN_AD2	AD2
ADC_INPUTCHAN_AD3	AD3
ADC_INPUTCHAN_AD4	AD4
ADC_INPUTCHAN_AD5	AD5
ADC_INPUTCHAN_AD6	AD6
ADC_INPUTCHAN_AD7	AD7
ADC_INPUTCHAN_AD8	AD8
ADC_INPUTCHAN_AD9	AD9

Enumerator

ADC_INPUTCHAN_AD10	AD10
ADC_INPUTCHAN_AD11	AD11
ADC_INPUTCHAN_AD12	AD12
ADC_INPUTCHAN_AD13	AD13
ADC_INPUTCHAN_AD14	AD14
ADC_INPUTCHAN_AD15	AD15
ADC_INPUTCHAN_AD16	AD16
ADC_INPUTCHAN_AD17	AD17
ADC_INPUTCHAN_AD18	AD18
ADC_INPUTCHAN_AD19	AD19
ADC_INPUTCHAN_AD20	AD20
ADC_INPUTCHAN_AD21	AD21
ADC_INPUTCHAN_AD22	AD22
ADC_INPUTCHAN_AD23	AD23
ADC_INPUTCHAN_AD24	AD24
ADC_INPUTCHAN_AD25	AD25
ADC_INPUTCHAN_AD26	AD26
ADC_INPUTCHAN_AD27	AD27
ADC_INPUTCHAN_AD28	AD28
ADC_INPUTCHAN_AD29	AD29
ADC_INPUTCHAN_AD30	AD30
ADC_INPUTCHAN_AD31	AD31

3.2.2.5 adc_resolution_t

```
enum adc_resolution_t
```

Conversion resolution selection Implements : adc_resolution_t_Class.

Enumerator

ADC_RESOLUTION_8BIT	8-bit resolution mode
ADC_RESOLUTION_12BIT	12-bit resolution mode
ADC_RESOLUTION_10BIT	10-bit resolution mode

3.2.2.6 adc_trigger_t

```
enum adc_trigger_t
```

Trigger type selection Implements : adc_trigger_t_Class.

Enumerator

ADC_TRIGGER_SOFTWARE	Software trigger.
ADC_TRIGGER_HARDWARE	Hardware trigger.

3.2.2.7 adc_voltage_reference_t

```
enum adc_voltage_reference_t
```

Voltage reference selection Implements : adc_voltage_reference_t_Class.

Enumerator

ADC_VOLTAGEREF_VREF	VrefH and VrefL as Voltage reference.
ADC_VOLTAGEREF_VALT	ValtH and ValtL as Voltage reference.

3.2.3 Function Documentation

3.2.3.1 ADC_HAL_ClearLatchTriggers()

```
static void ADC_HAL_ClearLatchTriggers (
             ADC_Type *const baseAddr )  [inline], [static]
```

Clear the latched triggers.

This function clears the latched triggers, except for the one under process. Before calling this function, make sure the hardware trigger source of the ADC is disabled.

Parameters

in	*baseAddr*	adc base pointer

Implements : ADC_HAL_ClearLatchTriggers_Activity

3.2.3.2 ADC_HAL_ClearTriggerErrorFlags()

```
static void ADC_HAL_ClearTriggerErrorFlags (
             ADC_Type *const baseAddr )  [inline], [static]
```

Clear the latch trigger error flags.

This function clears the trigger latch error flags.

Parameters

in	*baseAddr*	adc base pointer

Implements : ADC_HAL_ClearTriggerErrorFlags_Activity

3.2.3.3 ADC_HAL_GetCalibrationActiveFlag()

```
static bool ADC_HAL_GetCalibrationActiveFlag (
             const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Calibration Active Flag state.

This function returns the state of the Calibration Active Flag. This flag is set if an Auto-Calibration sequence is taking place.

Parameters

in	*baseAddr*	adc base pointer

Returns

the Calibration Active Flag state

Implements : ADC_HAL_GetCalibrationActiveFlag_Activity

3.2.3.4 ADC_HAL_GetChanInterruptEnableFlag()

```
static bool ADC_HAL_GetChanInterruptEnableFlag (
             const ADC_Type *const baseAddr,
             const uint8_t chanIndex )  [inline], [static]
```

Gets the Channel Interrupt Enable state.

This function returns the state of the Channel Interrupt Enable Flag. If the flag is set, an interrupt is generated when the a conversion is completed for the channel.

Parameters

in	*baseAddr*	adc base pointer
in	*chanIndex*	the adc measurement channel index

Returns

the Channel Interrupt Enable Flag state

Implements : ADC_HAL_GetChanInterruptEnableFlag_Activity

3.2.3.5 ADC_HAL_GetChanResult()

```
static uint16_t ADC_HAL_GetChanResult (
             const ADC_Type *const baseAddr,
             const uint8_t chanIndex )  [inline], [static]
```

Gets the conversion result for the selected measurement channel.

This function returns the conversion result from a measurement channel. This automatically clears the Conversion Complete Flag (CoCo flag) for that channel.

Parameters

in	*baseAddr*	adc base pointer
in	*chanIndex*	the adc measurement channel index

Returns

the Measurement Channel Conversion Result

Implements : ADC_HAL_GetChanResult_Activity

3.2.3.6 ADC_HAL_GetClockDivide()

```
static adc_clk_divide_t ADC_HAL_GetClockDivide (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the current ADC clock divider configuration.

This function returns the configured clock divider bitfield value for the ADC instance.

Parameters

in	*baseAddr*	adc base pointer

Returns

the clock divider value. Possible values:

- ADC_CLK_DIVIDE_1 : Divider set to 1.
- ADC_CLK_DIVIDE_2 : Divider set to 2.
- ADC_CLK_DIVIDE_4 : Divider set to 4.
- ADC_CLK_DIVIDE_8 : Divider set to 8.

Implements : ADC_HAL_GetClockDivide_Activity

3.2.3.7 ADC_HAL_GetContinuousConvFlag()

```
static bool ADC_HAL_GetContinuousConvFlag (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Continuous Conversion Flag state.

This functions returns the state of the Continuous Conversion Flag. This feature can be used to continuously sample a single channel. When this is active, the channel cannot be changed (by software or hardware trigger) until this feature is turned off.

Parameters

in	*baseAddr*	adc base pointer

Returns

the Continuous Conversion Flag state

Implements : ADC_HAL_GetContinuousConvFlag_Activity

3.2.3.8 ADC_HAL_GetConvActiveFlag()

```
static bool ADC_HAL_GetConvActiveFlag (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Conversion Active Flag.

This function checks whether a conversion is currently taking place on the ADC module.

Parameters

in	*baseAddr*	adc base pointer

Returns

Conversion Active Flag state

Implements : ADC_HAL_GetConvActiveFlag_Activity

3.2.3.9 ADC_HAL_GetConvCompleteFlag()

```
static bool ADC_HAL_GetConvCompleteFlag (
            const ADC_Type *const baseAddr,
            const uint8_t chanIndex )  [inline], [static]
```

Gets the measurement channel Conversion Complete Flag state.

This function returns the state of the Conversion Complete Flag for a measurement channel. This flag is set when a conversion is complete or the the condition generated by the Hardware Compare feature is evaluated to true.

Parameters

in	*baseAddr*	adc base pointer
in	*chanIndex*	the adc measurement channel index

Returns

the Conversion Complete Flag state

Implements : ADC_HAL_GetConvCompleteFlag_Activity

3.2.3.10 ADC_HAL_GetDMAEnableFlag()

```
static bool ADC_HAL_GetDMAEnableFlag (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the DMA Enable Flag state.

This function returns the state of the DMA Enable flag. DMA can be used to transfer completed conversion values from the result registers to RAM without CPU intervention.

Parameters

in	*baseAddr*	adc base pointer

Returns

the DMA Enable Flag state

Implements : ADC_HAL_GetDMAEnableFlag_Activity

3.2.3.11 ADC_HAL_GetHwAverageEnableFlag()

```
static bool ADC_HAL_GetHwAverageEnableFlag (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Hardware Average Enable Flag state.

This function returns the state of the Hardware Average Enable Flag. Hardware averaging can be used to obtain an average value over multiple consecutive conversions on the same channel.

Parameters

in	*baseAddr*	adc base pointer

Returns

the Hardware Average Enable Flag state

Implements : ADC_HAL_GetHwAverageEnableFlag_Activity

3.2.3.12 ADC_HAL_GetHwAverageMode()

```
static adc_average_t ADC_HAL_GetHwAverageMode (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Hardware Average Mode.

This function returns the configured Hardware Average Mode. The mode selects the number of samples to average: 4, 8, 16 or 32.

Parameters

in	*baseAddr*	adc base pointer

Returns

the Hardware Average Mode selection. Possible values:

- ADC_AVERAGE_4 : Hardware average of 4 samples..
- ADC_AVERAGE_8 : Hardware average of 8 samples.
- ADC_AVERAGE_16 : Hardware average of 16 samples.

- ADC_AVERAGE_32 : Hardware average of 32 samples.

Implements : ADC_HAL_GetHwAverageMode_Activity

3.2.3.13 ADC_HAL_GetHwCompareComp1Value()

```
static uint16_t ADC_HAL_GetHwCompareComp1Value (
             const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Compare Register 1 value.

This function returns the value written in the Hardware Compare Register 1. This value defines the upper or lower limit for the Hardware Compare Range. This value is always 12-bit resolution value (for lower resolution modes, internal bit shifting will take place).

Parameters

in	*baseAddr*	adc base pointer

Returns

the Compare Register 1 value

Implements : ADC_HAL_GetHwCompareComp1Value_Activity

3.2.3.14 ADC_HAL_GetHwCompareComp2Value()

```
static uint16_t ADC_HAL_GetHwCompareComp2Value (
             const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Compare Register 2 value.

This function returns the value written in the Hardware Compare Register 2. This value defines the upper or lower limit for the Hardware Compare Range. This value is always 12-bit resolution (for lower resolution modes, internal bit shifting will take place).

Parameters

in	*baseAddr*	adc base pointer

Returns

the Compare Register 2 value

Implements : ADC_HAL_GetHwCompareComp2Value_Activity

3.2.3.15 ADC_HAL_GetHwCompareEnableFlag()

```
static bool ADC_HAL_GetHwCompareEnableFlag (
             const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Hardware Compare Enable Flag state.

This function returns the state of the Hardware Compare Enable Flag. Hardware Compare can be used to check if the ADC result is within or outside of a predefined range.

Parameters

in	*baseAddr*	adc base pointer

Returns

the Hardware Compare Enable Flag state

Implements : ADC_HAL_GetHwCompareEnableFlag_Activity

3.2.3.16 ADC_HAL_GetHwCompareGtEnableFlag()

```
static bool ADC_HAL_GetHwCompareGtEnableFlag (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Hardware Compare Greater Than Enable Flag state.

This function returns the Hardware Compare Greater Than Enable Flag. Using this feature, the ADC can be configured to check if the measured value is within or outside of a predefined range.

Parameters

in	*baseAddr*	adc base pointer

Returns

the Hardware Compare Greater Than Enable Flag state

Implements : ADC_HAL_GetHwCompareGtEnableFlag_Activity

3.2.3.17 ADC_HAL_GetHwCompareRangeEnableFlag()

```
static bool ADC_HAL_GetHwCompareRangeEnableFlag (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Hardware Compare Range Enable state.

This function returns the state of the Hardware Compare Range Enable Flag. This feature allows configuration of a range with two non-zero values or with a non-zero and zero value.

Parameters

in	*baseAddr*	adc base pointer

Returns

the Hardware Compare Range Enable Flag state

Implements : ADC_HAL_GetHwCompareRangeEnableFlag_Activity

3.2.3.18 ADC_HAL_GetInputChannel()

```
static adc_inputchannel_t ADC_HAL_GetInputChannel (
            const ADC_Type *const baseAddr,
            const uint8_t chanIndex ) [inline], [static]
```

Gets the configured input channel for the selected measurement channel.

This function returns the configured input channel for a measurement channel.

Parameters

in	*baseAddr*	adc base pointer
in	*chanIndex*	the adc measurement channel index

Returns

the Input Channel selected for the Measurement Channel. Possible values:

- ADC_INPUTCHAN_AD0 : AD0 selected as input.
- ADC_INPUTCHAN_AD1 : AD1 selected as input.
- ADC_INPUTCHAN_AD2 : AD2 selected as input.
- ADC_INPUTCHAN_AD3 : AD3 selected as input.
- ADC_INPUTCHAN_AD4 : AD4 selected as input.
- ADC_INPUTCHAN_AD5 : AD5 selected as input.
- ADC_INPUTCHAN_AD6 : AD6 selected as input.
- ADC_INPUTCHAN_AD7 : AD7 selected as input.
- ADC_INPUTCHAN_AD8 : AD8 selected as input.
- ADC_INPUTCHAN_AD9 : AD9 selected as input.
- ADC_INPUTCHAN_AD10 : AD10 selected as input.
- ADC_INPUTCHAN_AD11 : AD11 selected as input.
- ADC_INPUTCHAN_AD12 : AD12 selected as input.
- ADC_INPUTCHAN_AD13 : AD13 selected as input.
- ADC_INPUTCHAN_AD14 : AD14 selected as input.
- ADC_INPUTCHAN_AD15 : AD15 selected as input.
- ADC_INPUTCHAN_TEMP : Temp Sensor selected as input.
- ADC_INPUTCHAN_BANDGAP : Band Gap selected as input.
- ADC_INPUTCHAN_VREFSH : VREFSH selected as input.
- ADC_INPUTCHAN_VREFSL : VREFSL selected as input.
- ADC_INPUTCHAN_DISABLED : Channel Disabled.

Implements : ADC_HAL_GetInputChannel_Activity

3.2.3.19 ADC_HAL_GetInputClock()

```
static adc_input_clock_t ADC_HAL_GetInputClock (
            const ADC_Type *const baseAddr ) [inline], [static]
```

Gets the AD Clock Input configuration.

This function returns the configured clock input source for the ADC.

Parameters

in	*baseAddr*	adc base pointer

Returns

the input clock source. Possible values:

- ADC_CLK_ALT_1 : ADC Input clock source alternative 1.
- ADC_CLK_ALT_2 : ADC Input clock source alternative 2.
- ADC_CLK_ALT_3 : ADC Input clock source alternative 3.
- ADC_CLK_ALT_4 : ADC Input clock source alternative 4.

Implements : ADC_HAL_GetInputClock_Activity

3.2.3.20 ADC_HAL_GetResolution()

```
static adc_resolution_t ADC_HAL_GetResolution (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Resolution Mode configuration.

This function returns the configured resolution mode for the ADC.

Parameters

in	*baseAddr*	adc base pointer

Returns

the ADC resolution mode. Possible values:

- ADC_RESOLUTION_8BIT : 8-bit resolution mode.
- ADC_RESOLUTION_10BIT : 10-bit resolution mode.
- ADC_RESOLUTION_12BIT : 12-bit resolution mode.

Implements : ADC_HAL_GetResolution_Activity

3.2.3.21 ADC_HAL_GetSampleTime()

```
static uint8_t ADC_HAL_GetSampleTime (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the Sample time in AD clock cycles.

This function gets the sample time (in AD clocks) configured for the ADC. Selection of 2 to 256 ADCK is possible. The value returned by this function is the sample time minus 1. A sample time of 1 is not supported.

Parameters

in	*baseAddr*	adc base pointer

Returns

the Sample Time in AD Clocks

Implements : ADC_HAL_GetSampleTime_Activity

3.2.3.22 ADC_HAL_GetTriggerErrorFlags()

```
static uint32_t ADC_HAL_GetTriggerErrorFlags (
             const ADC_Type *const baseAddr )  [inline], [static]
```

Get the trigger latch error flags.

This function returns the latch trigger error flags

Parameters

in	*baseAddr*	adc base pointer

Returns

The trigger latch error flags

Implements : ADC_HAL_GetTriggerErrorFlags_Activity

3.2.3.23 ADC_HAL_GetTriggerLatchFlags()

```
static uint32_t ADC_HAL_GetTriggerLatchFlags (
             const ADC_Type *const baseAddr )  [inline], [static]
```

Get the trigger latch flags bits.

/ This function returns the trigger latch flags.

Parameters

in	*baseAddr*	adc base pointer

Returns

trigger latch flags

Implements : ADC_HAL_GetTriggerLatchFlags_Activity

3.2.3.24 ADC_HAL_GetTriggerMode()

```
static adc_trigger_t ADC_HAL_GetTriggerMode (
             const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the ADC Trigger Mode.

This function returns the configured triggering mode for the ADC. In Software Triggering Mode, the user can start conversions by setting an input channel in the ADC measurement channel A (index 0). When in Hardware trigger mode, a conversion is started by another peripheral (like PDB or TRGMUX).

Parameters

in	*baseAddr*	adc base pointer

Returns

the current trigger mode. Possible values:
- ADC_TRIGGER_SOFTWARE : Software triggering.
- ADC_TRIGGER_HARDWARE : Hardware triggering.

Implements : ADC_HAL_GetTriggerMode_Activity

3.2.3.25 ADC_HAL_GetTriggerProcNumber()

```
static uint32_t ADC_HAL_GetTriggerProcNumber (
             const ADC_Type *const baseAddr )  [inline], [static]
```

Get the index of the trigger under process.

This function returns the index of the trigger currently being processed.

Parameters

in	*baseAddr*	adc base pointer

Returns

the trigger index

Implements : ADC_HAL_GetTriggerProcNumber_Activity

3.2.3.26 ADC_HAL_GetUserGainValue()

```
static uint16_t ADC_HAL_GetUserGainValue (
             const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the User Gain Register value.

This function returns the value in the User Gain Register. The value in this register is the amplification applied to the measured data before being written in the result register.

Parameters

in	*baseAddr*	adc base pointer

Returns

the User Gain Register value

Implements : ADC_HAL_GetUserGainValue_Activity

3.2.3.27 ADC_HAL_GetUserOffsetValue()

```
static uint16_t ADC_HAL_GetUserOffsetValue (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the User Offset Register value.

This function returns the value in the User Offset Register. The value in this register is subtracted from the measured data before being written in the result register. This value is 16-bit signed value. To preserve resolution, lower-order bits will be ignored in low resolution-modes.

Parameters

in	*baseAddr*	adc base pointer

Returns

the User Offset Register value

Implements : ADC_HAL_GetUserOffsetValue_Activity

3.2.3.28 ADC_HAL_GetVoltageReference()

```
static adc_voltage_reference_t ADC_HAL_GetVoltageReference (
            const ADC_Type *const baseAddr )  [inline], [static]
```

Gets the ADC Reference Voltage selection.

This function returns the configured reference voltage selection for the ADC. Reference voltage can be selected between the pairs (VrefH, VrefL) and (ValtH, ValtL).

Parameters

in	*baseAddr*	adc base pointer

Returns

the voltage reference input pair. Possible values:

- ADC_VOLTAGEREF_VREF : VrefL and VrefH used as voltage reference.
- ADC_VOLTAGEREF_VALT : ValtL and ValtH used as voltage reference.

Implements : ADC_HAL_GetVoltageReference_Activity

3.2.3.29 ADC_HAL_Init()

```
void ADC_HAL_Init (
            ADC_Type *const baseAddr )
```

Initializes the ADC instance to reset values.

This function initializes the ADC instance to a known state (the register are written with their reset values from the Reference Manual).

Parameters

in	*baseAddr*	adc base pointer

3.2.3.30 ADC_HAL_SetCalibrationActiveFlag()

```
static void ADC_HAL_SetCalibrationActiveFlag (
            ADC_Type *const baseAddr,
            const bool state )  [inline], [static]
```

Sets the Calibration Active Flag state.

This functions starts or aborts an Auto-Calibration sequence. If this is set, it will remain set until the sequence is finished.

Parameters

in	*baseAddr*	adc base pointer
in	*state*	the new Calibration Active Flag state

Implements : ADC_HAL_SetCalibrationActiveFlag_Activity

3.2.3.31 ADC_HAL_SetChanInterruptEnableFlag()

```
static void ADC_HAL_SetChanInterruptEnableFlag (
            ADC_Type *const baseAddr,
            const uint8_t chanIndex,
            const bool state )  [inline], [static]
```

Sets the Channel Interrupt Enable to a new state.

This function configures the state of the Interrupt Enable Flag for a measurement channel. If the flag is set, an interrupt is generated when the a conversion is completed for the channel.

Parameters

in	*baseAddr*	adc base pointer
in	*chanIndex*	the adc measurement channel index
in	*state*	the new Channel Interrupt Enable Flag state

Implements : ADC_HAL_SetChanInterruptEnableFlag_Activity

3.2.3.32 ADC_HAL_SetClockDivide()

```
static void ADC_HAL_SetClockDivide (
            ADC_Type *const baseAddr,
            const adc_clk_divide_t clockDivide )  [inline], [static]
```

Sets the ADC clock divider configuration.

This functions configures the ADC instance clock divider.

Parameters

in	*baseAddr*	adc base pointer
in	*clockDivide*	clk divider • ADC_CLK_DIVIDE_1 : Divider set to 1. • ADC_CLK_DIVIDE_2 : Divider set to 2. • ADC_CLK_DIVIDE_4 : Divider set to 4. • ADC_CLK_DIVIDE_8 : Divider set to 8.

Implements : ADC_HAL_SetClockDivide_Activity

3.2.3.33 ADC_HAL_SetContinuousConvFlag()

```
static void ADC_HAL_SetContinuousConvFlag (
            ADC_Type *const baseAddr,
            const bool state )  [inline], [static]
```

Sets the Continuous Conversion Flag state.

This function configures the Continuous Conversion. This feature can be used to continuously sample a single channel. When this is active, the channel cannot be changed (by software or hardware trigger) until this feature is turned off.

Parameters

in	*baseAddr*	adc base pointer
in	*state*	the new Continuous Conversion Flag state

Implements : ADC_HAL_SetContinuousConvFlag_Activity

3.2.3.34 ADC_HAL_SetDMAEnableFlag()

```
static void ADC_HAL_SetDMAEnableFlag (
            ADC_Type *const baseAddr,
            const bool state )  [inline], [static]
```

Sets the DMA Enable Flag state.

This function configures the DMA Enable Flag. DMA can be used to transfer completed conversion values from the result registers to RAM without CPU intervention.

Parameters

in	*baseAddr*	adc base pointer
in	*state*	the new DMA Enable Flag state

Implements : ADC_HAL_SetDMAEnableFlag_Activity

3.2.3.35 ADC_HAL_SetHwAverageEnableFlag()

```
static void ADC_HAL_SetHwAverageEnableFlag (
             ADC_Type *const baseAddr,
             const bool state )  [inline], [static]
```

Sets the Hardware Average Enable Flag state.

This function configures the Hardware Average Enable Flag. Hardware averaging can be used to obtain an average value over multiple consecutive conversions on the same channel.

Parameters

in	*baseAddr*	adc base pointer
in	*state*	the new Hardware Average Enable Flag state

Implements : ADC_HAL_SetHwAverageEnableFlag_Activity

3.2.3.36 ADC_HAL_SetHwAverageMode()

```
static void ADC_HAL_SetHwAverageMode (
             ADC_Type *const baseAddr,
             const adc_average_t averageMode )  [inline], [static]
```

Sets the Hardware Average Mode.

This function configures the Hardware Average Mode. The mode selects the number of samples to average: 4, 8, 16 or 32.

Parameters

in	*baseAddr*	adc base pointer
in	*averageMode*	the new Hardware Average Mode. • ADC_AVERAGE_4 : Hardware average of 4 samples.. • ADC_AVERAGE_8 : Hardware average of 8 samples. • ADC_AVERAGE_16 : Hardware average of 16 samples. • ADC_AVERAGE_32 : Hardware average of 32 samples.

Implements : ADC_HAL_SetHwAverageMode_Activity

3.2.3.37 ADC_HAL_SetHwCompareComp1Value()

```
static void ADC_HAL_SetHwCompareComp1Value (
             ADC_Type *const baseAddr,
             const uint16_t value )  [inline], [static]
```

Sets the Compare Register 1 value.

This function writes a 12-bit value in the Hardware Compare Register 1. This value defines the upper or lower limit for the Hardware Compare Range. This value is always 12-bit resolution (for lower resolution modes, internal bit shifting will take place).

Parameters

in	*baseAddr*	adc base pointer
in	*value*	the new Compare Register 1 value

Implements : ADC_HAL_SetHwCompareComp1Value_Activity

3.2.3.38 ADC_HAL_SetHwCompareComp2Value()

```
static void ADC_HAL_SetHwCompareComp2Value (
            ADC_Type *const baseAddr,
            const uint16_t value ) [inline], [static]
```

Sets the Compare Register 2 value.

This function writes a 12-bit value in the Hardware Compare Register 2. This value defines the upper or lower limit for the Hardware Compare Range. This value is always 12-bit resolution value (for lower resolution modes, internal bit shifting will take place).

Parameters

in	*baseAddr*	adc base pointer
in	*value*	the new Compare Register 2 value

Implements : ADC_HAL_SetHwCompareComp2Value_Activity

3.2.3.39 ADC_HAL_SetHwCompareEnableFlag()

```
static void ADC_HAL_SetHwCompareEnableFlag (
            ADC_Type *const baseAddr,
            const bool state ) [inline], [static]
```

Sets the Hardware Compare Enable Flag state.

This functions configures the Hardware Compare Enable Flag. Hardware Compare can be used to check if the ADC result is within or outside of a predefined range.

Parameters

in	*baseAddr*	adc base pointer
in	*state*	the new Hardware Compare Enable Flag state

Implements : ADC_HAL_SetHwCompareEnableFlag_Activity

3.2.3.40 ADC_HAL_SetHwCompareGtEnableFlag()

```
static void ADC_HAL_SetHwCompareGtEnableFlag (
            ADC_Type *const baseAddr,
            const bool state ) [inline], [static]
```

Sets the Hardware Compare Greater Than Enable Flag state.

This function configures the Hardware Compare Greater Than Enable Flag. Using this feature, the ADC can be configured to check if the measured value is within or outside of a predefined range.

Parameters

in	*baseAddr*	adc base pointer
in	*state*	the new Hardware Compare Greater Than Enable Flag state

Implements : ADC_HAL_SetHwCompareGtEnableFlag_Activity

3.2.3.41 ADC_HAL_SetHwCompareRangeEnableFlag()

```
static void ADC_HAL_SetHwCompareRangeEnableFlag (
            ADC_Type *const baseAddr,
            const bool state )  [inline], [static]
```

Sets the Hardware Compare Range Enable state.

This function configures the Hardware Compare Range Enable Flag. This feature allows configuration of a range with two non-zero values or with a non-zero and zero value.

Parameters

in	*baseAddr*	adc base pointer
in	*state*	the new Hardware Compare Range Enable Flag state

Implements : ADC_HAL_SetHwCompareRangeEnableFlag_Activity

3.2.3.42 ADC_HAL_SetInputChannel()

```
static void ADC_HAL_SetInputChannel (
            ADC_Type *const baseAddr,
            const uint8_t chanIndex,
            const adc_inputchannel_t inputChan )  [inline], [static]
```

Sets the input channel configuration for the measurement channel.

This function configures the input channel for a measurement channel. In software trigger mode, configuring channel A (index 0) will start a new conversion immediately.

Parameters

in	*baseAddr*	adc base pointer
in	*chanIndex*	the adc measurement channel index

Parameters

in	*inputChan*	the Input Channel selected for the Measurement Channel • ADC_INPUTCHAN_AD0 : AD0 selected as input. • ADC_INPUTCHAN_AD1 : AD1 selected as input. • ADC_INPUTCHAN_AD2 : AD2 selected as input. • ADC_INPUTCHAN_AD3 : AD3 selected as input. • ADC_INPUTCHAN_AD4 : AD4 selected as input. • ADC_INPUTCHAN_AD5 : AD5 selected as input. • ADC_INPUTCHAN_AD6 : AD6 selected as input. • ADC_INPUTCHAN_AD7 : AD7 selected as input. • ADC_INPUTCHAN_AD8 : AD8 selected as input. • ADC_INPUTCHAN_AD9 : AD9 selected as input. • ADC_INPUTCHAN_AD10 : AD10 selected as input. • ADC_INPUTCHAN_AD11 : AD11 selected as input. • ADC_INPUTCHAN_AD12 : AD12 selected as input. • ADC_INPUTCHAN_AD13 : AD13 selected as input. • ADC_INPUTCHAN_AD14 : AD14 selected as input. • ADC_INPUTCHAN_AD15 : AD15 selected as input. • ADC_INPUTCHAN_TEMP : Temp Sensor selected as input. • ADC_INPUTCHAN_BANDGAP : Band Gap selected as input. • ADC_INPUTCHAN_VREFSH : VREFSH selected as input. • ADC_INPUTCHAN_VREFSL : VREFSL selected as input. • ADC_INPUTCHAN_DISABLED : Channel Disabled.

Implements : ADC_HAL_SetInputChannel_Activity

3.2.3.43 ADC_HAL_SetInputClock()

```
static void ADC_HAL_SetInputClock (
            ADC_Type *const baseAddr,
            const adc_input_clock_t inputClock )  [inline], [static]
```

Sets the AD Clock Input configuration.

This function configures the clock input source for the ADC.

Parameters

in	*baseAddr*	adc base pointer

Parameters

in	*inputClock*	the new input clock source • ADC_CLK_ALT_1 : ADC Input clock source alternative 1. • ADC_CLK_ALT_2 : ADC Input clock source alternative 2. • ADC_CLK_ALT_3 : ADC Input clock source alternative 3. • ADC_CLK_ALT_4 : ADC Input clock source alternative 4.

Implements : ADC_HAL_SetInputClock_Activity

3.2.3.44 ADC_HAL_SetResolution()

```
static void ADC_HAL_SetResolution (
            ADC_Type *const baseAddr,
            const adc_resolution_t resolution )  [inline], [static]
```

Sets the Resolution Mode configuration.

This function configures the ADC resolution mode.

Parameters

in	*baseAddr*	adc base pointer
in	*resolution*	the adc resolution mode • ADC_RESOLUTION_8BIT : 8-bit resolution mode. • ADC_RESOLUTION_10BIT : 10-bit resolution mode. • ADC_RESOLUTION_12BIT : 12-bit resolution mode.

Implements : ADC_HAL_SetResolution_Activity

3.2.3.45 ADC_HAL_SetSampleTime()

```
static void ADC_HAL_SetSampleTime (
            ADC_Type *const baseAddr,
            uint8_t sampletime )  [inline], [static]
```

Sets the Sample time in AD clock cycles.

This function configures the sample time for the ADC (in ADCK clocks). The actual sample time will be the value provided plus 1. Selection of 2 to 256 ADCK is possible. A real sample time of 1 is not supported (a parameter value of 0 will be automatically be changed to 1).

Parameters

in	*baseAddr*	adc base pointer
in	*sampletime*	Sample time in AD Clocks

Implements : ADC_HAL_SetSampleTime_Activity

3.2.3.46 ADC_HAL_SetTriggerMode()

```
static void ADC_HAL_SetTriggerMode (
            ADC_Type *const baseAddr,
            const adc_trigger_t trigger )  [inline], [static]
```

Sets the ADC Trigger Mode.

This function configures the ADC triggering mode. In Software Triggering Mode, the user can start conversions by setting an input channel in the ADC measurement channel A (index 0). When in Hardware trigger mode, a conversion is started by another peripheral (like PDB or TRGMUX).

Parameters

in	*baseAddr*	adc base pointer
in	*trigger*	the desired trigger mode • ADC_TRIGGER_SOFTWARE : Software triggering. • ADC_TRIGGER_HARDWARE : Hardware triggering.

Implements : ADC_HAL_SetTriggerMode_Activity

3.2.3.47 ADC_HAL_SetUserGainValue()

```
static void ADC_HAL_SetUserGainValue (
            ADC_Type *const baseAddr,
            const uint16_t value )  [inline], [static]
```

Sets the User Gain Register value.

This function configures the User Gain Register. The value in this register is the amplification applied to the measured data before being written in the result register.

Parameters

in	*baseAddr*	adc base pointer
in	*value*	the new User Gain Register value

Implements : ADC_HAL_SetUserGainValue_Activity

3.2.3.48 ADC_HAL_SetUserOffsetValue()

```
static void ADC_HAL_SetUserOffsetValue (
            ADC_Type *const baseAddr,
            const uint16_t value )  [inline], [static]
```

Sets the User Offset Register value.

This function configures the User Offset Register. The value in this register is subtracted from the measured data before being written in the result register. This value is 16-bit signed value. To preserve resolution, lower-order bits will be ignored in low resolution-modes.

Parameters

in	*baseAddr*	adc base pointer
in	*value*	the new User Offset Register value

Implements : ADC_HAL_SetUserOffsetValue_Activity

3.2.3.49 ADC_HAL_SetVoltageReference()

```
static void ADC_HAL_SetVoltageReference (
             ADC_Type *const baseAddr,
             const adc_voltage_reference_t voltageRef )  [inline], [static]
```

Sets the ADC Reference Voltage selection.

This function configures the ADC Reference Voltage. Reference voltage can be selected between the pairs (VrefH, VrefL) and (ValtH, ValtL).

Parameters

in	*baseAddr*	adc base pointer
in	*voltageRef*	the new voltage reference input • ADC_VOLTAGEREF_VREF : VrefL and VrefH used as voltage reference. • ADC_VOLTAGEREF_VALT : ValtL and ValtH used as voltage reference.

Implements : ADC_HAL_SetVoltageReference_Activity

3.3 Analog to Digital Converter (ADC)

3.3.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Drivers for the Analog to Digital Converter (ADC) module of S32 SDK devices.

The ADC is a configurable 12-bit (selectable to between 8-bit, 10-bit and 12-bit resolution) single-ended SAR converter.

Features of the ADC include:

- 16 control channels, with configurable triggers
- a maximum of 16 external input sources and 4 internal input sources
- hardware compare and average functions
- auto-calibration feature

Modules

- ADC Driver

  Analog to Digital Converter Peripheral Driver.
- ADC HAL

  Analog to Digital Converter Hardware Abstraction Layer.

3.4 CRC Driver

3.4.1 Detailed Description

Cyclic Redundancy Check Peripheral Driver.

This section describes the programming interface of the CRC driver.

Data Structures

- struct crc_user_config_t

  CRC configuration structure. Implements : crc_user_config_t_Class. More...

Variables

- CRC_Type ∗const g_crcBase [CRC_INSTANCE_COUNT]

  Table of base addresses for CRC instances.

CRC DRIVER API

- status_t CRC_DRV_Init (uint32_t instance, const crc_user_config_t ∗userConfigPtr)

  Initializes the CRC module.
- status_t CRC_DRV_Deinit (uint32_t instance)

  Sets the default configuration.
- void CRC_DRV_WriteData (uint32_t instance, const uint8_t ∗data, uint32_t dataSize)

  Appends a block of bytes to the current CRC calculation.
- uint32_t CRC_DRV_GetCrcResult (uint32_t instance)

  Returns the current result of the CRC calculation.
- status_t CRC_DRV_Configure (uint32_t instance, const crc_user_config_t ∗userConfigPtr)

  Configures the CRC module from a user configuration structure.

3.4.2 Data Structure Documentation

3.4.2.1 struct crc_user_config_t

CRC configuration structure. Implements : crc_user_config_t_Class.

Data Fields

- crc_bit_width_t crcWidth
- uint32_t seed
- uint32_t polynomial
- crc_transpose_t writeTranspose
- crc_transpose_t readTranspose
- bool complementChecksum

Field Documentation

3.4.2.1.1 complementChecksum

```
bool complementChecksum
```

True if the result shall be complement of the actual checksum.

3.4.2.1.2 crcWidth

```
crc_bit_width_t crcWidth
```

Selects 16-bit or 32-bit CRC protocol.

3.4.2.1.3 polynomial

```
uint32_t polynomial
```

CRC Polynomial, MSBit first.
Example polynomial: 0x1021U = 1_0000_0010_0001 = x^12+x^5+1

3.4.2.1.4 readTranspose

```
crc_transpose_t readTranspose
```

Type of transpose when reading CRC result.

3.4.2.1.5 seed

```
uint32_t seed
```

Starting checksum value.

3.4.2.1.6 writeTranspose

```
crc_transpose_t writeTranspose
```

Type of transpose when writing CRC input data.

3.4.3 Function Documentation

3.4.3.1 CRC_DRV_Configure()

```
status_t CRC_DRV_Configure (
             uint32_t instance,
             const crc_user_config_t * userConfigPtr )
```

Configures the CRC module from a user configuration structure.

This function configures the CRC module from a user configuration structure

Parameters

in	*instance*	The CRC instance number
in	*userConfigPtr*	Pointer to structure of initialization

Returns

Execution status (success)

3.4.3.2 CRC_DRV_Deinit()

```
status_t CRC_DRV_Deinit (
            uint32_t instance )
```

Sets the default configuration.

This function sets the default configuration

Parameters

in	*instance*	The CRC instance number

Returns

Execution status (success)

3.4.3.3 CRC_DRV_GetCrcResult()

```
uint32_t CRC_DRV_GetCrcResult (
            uint32_t instance )
```

Returns the current result of the CRC calculation.

This function returns the current result of the CRC calculation

Parameters

in	*instance*	The CRC instance number

Returns

Result of CRC calculation

3.4.3.4 CRC_DRV_Init()

```
status_t CRC_DRV_Init (
            uint32_t instance,
            const crc_user_config_t * userConfigPtr )
```

Initializes the CRC module.

This function initializes CRC driver based on user configuration input. The user must make sure that the clock is enabled

Parameters

in	*instance*	The CRC instance number
in	*userConfigPtr*	Pointer to structure of initialization

Returns

Execution status (success)

3.4.3.5 CRC_DRV_WriteData()

```
void CRC_DRV_WriteData (
            uint32_t instance,
            const uint8_t * data,
            uint32_t dataSize )
```

Appends a block of bytes to the current CRC calculation.

This function appends a block of bytes to the current CRC calculation

Parameters

in	*instance*	The CRC instance number
in	*data*	Data for current CRC calculation
in	*dataSize*	Length of data to be calculated

3.4.4 Variable Documentation

3.4.4.1 g_crcBase

```
CRC_Type* const g_crcBase[CRC_INSTANCE_COUNT]
```

Table of base addresses for CRC instances.

3.5 CRC HAL

3.5.1 Detailed Description

Cyclic Redundancy Check Hardware Abstraction Layer.

This section describes the programming interface of the CRC HAL.

Enumerations

- enum crc_transpose_t { CRC_TRANSPOSE_NONE = 0x00U, CRC_TRANSPOSE_BITS = 0x01U, CRC_↩ TRANSPOSE_BITS_AND_BYTES = 0x02U, CRC_TRANSPOSE_BYTES = 0x03U }

  CRC type of transpose of read write data Implements : crc_transpose_t_Class.
- enum crc_bit_width_t { CRC_BITS_16 = 0U, CRC_BITS_32 = 1U }

  CRC bit width Implements : crc_bit_width_t_Class.

CRC HAL API

- void CRC_HAL_Init (CRC_Type ∗const base)

  Initializes the CRC module.
- uint32_t CRC_HAL_GetCrc32 (CRC_Type ∗const base, uint32_t data, bool newSeed, uint32_t seed)

  Appends 32-bit data to the current CRC calculation and returns new result.
- uint32_t CRC_HAL_GetCrc16 (CRC_Type ∗const base, uint16_t data, bool newSeed, uint32_t seed)

  Appends 16-bit data to the current CRC calculation and returns new result.
- uint32_t CRC_HAL_GetCrc8 (CRC_Type ∗const base, uint8_t data, bool newSeed, uint32_t seed)

  Appends 8-bit data to the current CRC calculation and returns new result.
- uint32_t CRC_HAL_GetCrcResult (const CRC_Type ∗const base)

  Returns the current result of the CRC calculation.
- static uint32_t CRC_HAL_GetDataReg (const CRC_Type ∗const base)

  Gets the current CRC result.
- static void CRC_HAL_SetDataReg (CRC_Type ∗const base, uint32_t value)

  Sets the 32 bits of CRC data register.
- static uint16_t CRC_HAL_GetDataHReg (const CRC_Type ∗const base)

  Gets the upper 16 bits of the current CRC result.
- static void CRC_HAL_SetDataHReg (CRC_Type ∗const base, uint16_t value)

  Sets the upper 16 bits of CRC data register.
- static uint16_t CRC_HAL_GetDataLReg (const CRC_Type ∗const base)

  Gets the lower 16 bits of the current CRC result.
- static void CRC_HAL_SetDataLReg (CRC_Type ∗const base, uint16_t value)

  Sets the lower 16 bits of CRC data register.
- static void CRC_HAL_SetDataHUReg (CRC_Type ∗const base, uint8_t value)

  Sets the High Upper Byte - HU.
- static void CRC_HAL_SetDataHLReg (CRC_Type ∗const base, uint8_t value)

  Sets the High Lower Byte - HL.
- static void CRC_HAL_SetDataLUReg (CRC_Type ∗const base, uint8_t value)

  Sets the Low Upper Byte - LU.
- static void CRC_HAL_SetDataLLReg (CRC_Type ∗const base, uint8_t value)

  Sets the Low Lower Byte - LL.
- static uint32_t CRC_HAL_GetPolyReg (const CRC_Type ∗const base)

  Gets the polynomial register value.

- static void CRC_HAL_SetPolyReg (CRC_Type ∗const base, uint32_t value)
  Sets the polynomial register.
- static uint16_t CRC_HAL_GetPolyHReg (const CRC_Type ∗const base)
  Gets the upper 16 bits of polynomial register.
- static void CRC_HAL_SetPolyHReg (CRC_Type ∗const base, uint16_t value)
  Sets the upper 16 bits of polynomial register.
- static uint16_t CRC_HAL_GetPolyLReg (const CRC_Type ∗const base)
  Gets the lower 16 bits of polynomial register.
- static void CRC_HAL_SetPolyLReg (CRC_Type ∗const base, uint16_t value)
  Sets the lower 16 bits of polynomial register.
- static bool CRC_HAL_GetSeedOrDataMode (const CRC_Type ∗const base)
  Gets the CRC_DATA register mode.
- static void CRC_HAL_SetSeedOrDataMode (CRC_Type ∗const base, bool enable)
  Sets the CRC_DATA register mode.
- static bool CRC_HAL_GetFXorMode (const CRC_Type ∗const base)
  Gets complement read of CRC data register.
- static void CRC_HAL_SetFXorMode (CRC_Type ∗const base, bool enable)
  Sets complement read of CRC data register.
- static crc_bit_width_t CRC_HAL_GetProtocolWidth (const CRC_Type ∗const base)
  Gets the CRC protocol width.
- static void CRC_HAL_SetProtocolWidth (CRC_Type ∗const base, crc_bit_width_t width)
  Sets the CRC protocol width.
- static crc_transpose_t CRC_HAL_GetWriteTranspose (const CRC_Type ∗const base)
  Gets the CRC transpose type for writes.
- static void CRC_HAL_SetWriteTranspose (CRC_Type ∗const base, crc_transpose_t transp)
  Sets the CRC transpose type for writes.
- static crc_transpose_t CRC_HAL_GetReadTranspose (const CRC_Type ∗const base)
  Gets the CRC transpose type for reads.
- static void CRC_HAL_SetReadTranspose (CRC_Type ∗const base, crc_transpose_t transp)
  Sets the CRC transpose type for reads.

3.5.2 Enumeration Type Documentation

3.5.2.1 crc_bit_width_t

```
enum crc_bit_width_t
```

CRC bit width Implements : crc_bit_width_t_Class.

Enumerator

CRC_BITS_16	Generate 16-bit CRC code
CRC_BITS_32	Generate 32-bit CRC code

3.5.2.2 crc_transpose_t

```
enum crc_transpose_t
```

CRC type of transpose of read write data Implements : crc_transpose_t_Class.

Enumerator

CRC_TRANSPOSE_NONE	No transpose
CRC_TRANSPOSE_BITS	Transpose bits in bytes
CRC_TRANSPOSE_BITS_AND_BYTES	Transpose bytes and bits in bytes
CRC_TRANSPOSE_BYTES	Transpose bytes

3.5.3 Function Documentation

3.5.3.1 CRC_HAL_GetCrc16()

```
uint32_t CRC_HAL_GetCrc16 (
            CRC_Type *const base,
            uint16_t data,
            bool newSeed,
            uint32_t seed )
```

Appends 16-bit data to the current CRC calculation and returns new result.

This function appends 16-bit data to the current CRC calculation and returns new result. If the newSeed is true, seed set and result are calculated from the seed new value (new CRC calculation)

Parameters

in	*base*	The CRC peripheral base address
in	*data*	Input data for CRC calculation
in	*newSeed*	Sets new CRC calculation • true: New seed set and used for new calculation. • false: Seed argument ignored, continues old calculation.
in	*seed*	New seed if newSeed is true, else ignored

Returns

New CRC result

3.5.3.2 CRC_HAL_GetCrc32()

```
uint32_t CRC_HAL_GetCrc32 (
            CRC_Type *const base,
            uint32_t data,
            bool newSeed,
            uint32_t seed )
```

Appends 32-bit data to the current CRC calculation and returns new result.

This function appends 32-bit data to the current CRC calculation and returns new result. If the newSeed is true, seed set and result are calculated from the seed new value (new CRC calculation)

Parameters

in	*base*	The CRC peripheral base address
in	*data*	Input data for CRC calculation
in	*newSeed*	Sets new CRC calculation • true: New seed set and used for new calculation. • false: Seed argument ignored, continues old calculation.
in	*seed*	New seed if newSeed is true, else ignored

Returns

New CRC result

3.5.3.3 CRC_HAL_GetCrc8()

```
uint32_t CRC_HAL_GetCrc8 (
            CRC_Type *const base,
            uint8_t data,
            bool newSeed,
            uint32_t seed )
```

Appends 8-bit data to the current CRC calculation and returns new result.

This function appends 8-bit data to the current CRC calculation and returns new result. If the newSeed is true, seed set and result are calculated from the seed new value (new CRC calculation)

Parameters

in	*base*	The CRC peripheral base address
in	*data*	Input data for CRC calculation
in	*newSeed*	Sets new CRC calculation • true: New seed set and used for new calculation. • false: Seed argument ignored, continues old calculation.
in	*seed*	New seed if newSeed is true, else ignored

Returns

New CRC result

3.5.3.4 CRC_HAL_GetCrcResult()

```
uint32_t CRC_HAL_GetCrcResult (
            const CRC_Type *const base )
```

Returns the current result of the CRC calculation.

This function returns the current result of the CRC calculation

Parameters

in	*base*	The CRC peripheral base address

Returns

Result of CRC calculation

3.5.3.5 CRC_HAL_GetDataHReg()

```
static uint16_t CRC_HAL_GetDataHReg (
            const CRC_Type *const base )  [inline], [static]
```

Gets the upper 16 bits of the current CRC result.

This function gets the upper 16 bits of the current CRC result from the data register

Parameters

in	*base*	The CRC peripheral base address

Returns

Returns the upper 16 bits of the current CRC result Implements : CRC_HAL_GetDataHReg_Activity

3.5.3.6 CRC_HAL_GetDataLReg()

```
static uint16_t CRC_HAL_GetDataLReg (
            const CRC_Type *const base )  [inline], [static]
```

Gets the lower 16 bits of the current CRC result.

This function gets the lower 16 bits of the current CRC result from the data register

Parameters

in	*base*	The CRC peripheral base address

Returns

Returns the lower 16 bits of the current CRC result Implements : CRC_HAL_GetDataLReg_Activity

3.5.3.7 CRC_HAL_GetDataReg()

```
static uint32_t CRC_HAL_GetDataReg (
            const CRC_Type *const base )  [inline], [static]
```

Gets the current CRC result.

This function gets the current CRC result from the data register

Parameters

in	*base*	The CRC peripheral base address

Returns

Returns the current CRC result Implements : CRC_HAL_GetDataReg_Activity

3.5.3.8 CRC_HAL_GetFXorMode()

```
static bool CRC_HAL_GetFXorMode (
            const CRC_Type *const base )  [inline], [static]
```

Gets complement read of CRC data register.

This function gets complement read of CRC data register. Some CRC protocols require the final checksum to be XORed with 0xFFFFFFFF or 0xFFFF. Complement mode enables "on the fly" complementing of read data

Parameters

in	*base*	The CRC peripheral base address

Returns

Complement read -true: Invert or complement the read value of the CRC Data register. -false: No XOR on reading. Implements : CRC_HAL_GetFXorMode_Activity

3.5.3.9 CRC_HAL_GetPolyHReg()

```
static uint16_t CRC_HAL_GetPolyHReg (
            const CRC_Type *const base )  [inline], [static]
```

Gets the upper 16 bits of polynomial register.

This function gets the upper 16 bits of polynomial register. Note that this upper part of the register is not used in 16-bit CRC mode

Parameters

in	*base*	The CRC peripheral base address

Returns

Returns the upper 16 bits of polynomial register Implements : CRC_HAL_GetPolyHReg_Activity

3.5.3.10 CRC_HAL_GetPolyLReg()

```
static uint16_t CRC_HAL_GetPolyLReg (
            const CRC_Type *const base )  [inline], [static]
```

Gets the lower 16 bits of polynomial register.

This function gets the lower 16 bits of polynomial register

Parameters

in	*base*	The CRC peripheral base address

Returns

Returns the lower 16 bits of polynomial register Implements : CRC_HAL_GetPolyLReg_Activity

3.5.3.11 CRC_HAL_GetPolyReg()

```
static uint32_t CRC_HAL_GetPolyReg (
            const CRC_Type *const base )  [inline], [static]
```

Gets the polynomial register value.

This function gets the polynomial register value

Parameters

in	*base*	The CRC peripheral base address

Returns

Returns the polynomial register value Implements : CRC_HAL_GetPolyReg_Activity

3.5.3.12 CRC_HAL_GetProtocolWidth()

```
static crc_bit_width_t CRC_HAL_GetProtocolWidth (
            const CRC_Type *const base )  [inline], [static]
```

Gets the CRC protocol width.

This function gets the CRC protocol width

Parameters

in	*base*	The CRC peripheral base address

Returns

CRC protocol width
- CRC_BITS_16: 16-bit CRC protocol.
- CRC_BITS_32: 32-bit CRC protocol. Implements : CRC_HAL_GetProtocolWidth_Activity

3.5.3.13 CRC_HAL_GetReadTranspose()

```
static crc_transpose_t CRC_HAL_GetReadTranspose (
            const CRC_Type *const base )  [inline], [static]
```

Gets the CRC transpose type for reads.

This function gets the CRC transpose type for reads

Parameters

in	*base*	The CRC peripheral base address

Returns

CRC output transpose type Implements : CRC_HAL_GetReadTranspose_Activity

3.5.3.14 CRC_HAL_GetSeedOrDataMode()

```
static bool CRC_HAL_GetSeedOrDataMode (
            const CRC_Type *const base )  [inline], [static]
```

Gets the CRC_DATA register mode.

This function gets the CRC_DATA register mode

Parameters

in	*base*	The CRC peripheral base address

Returns

CRC_DATA register mode -true: CRC_DATA register is used for seed values. -false: CRC_DATA register is used for data values. Implements : CRC_HAL_GetSeedOrDataMode_Activity

3.5.3.15 CRC_HAL_GetWriteTranspose()

```
static crc_transpose_t CRC_HAL_GetWriteTranspose (
            const CRC_Type *const base )  [inline], [static]
```

Gets the CRC transpose type for writes.

This function gets the CRC transpose type for writes

Parameters

in	*base*	The CRC peripheral base address

Returns

CRC input transpose type for writes Implements : CRC_HAL_GetWriteTranspose_Activity

3.5.3.16 CRC_HAL_Init()

```
void CRC_HAL_Init (
            CRC_Type *const base )
```

Initializes the CRC module.

This function initializes the module to default configuration (Initial checksum: 0U, Default polynomial: 0x1021U, Type of read transpose: CRC_TRANSPOSE_NONE, Type of write transpose: CRC_TRANSPOSE_NONE, No complement of checksum read, 32-bit CRC)

Parameters

in	*base*	The CRC peripheral base address

3.5.3.17 CRC_HAL_SetDataHLReg()

```
static void CRC_HAL_SetDataHLReg (
            CRC_Type *const base,
            uint8_t value ) [inline], [static]
```

Sets the High Lower Byte - HL.

This function sets the High Lower Byte - HL of CRC data register

Parameters

in	*base*	The CRC peripheral base address
in	*value*	New data for CRC computation Implements : CRC_HAL_SetDataHLReg_Activity

3.5.3.18 CRC_HAL_SetDataHReg()

```
static void CRC_HAL_SetDataHReg (
            CRC_Type *const base,
            uint16_t value ) [inline], [static]
```

Sets the upper 16 bits of CRC data register.

This function sets the upper 16 bits of CRC data register

Parameters

in	*base*	The CRC peripheral base address
in	*value*	New data for CRC computation Implements : CRC_HAL_SetDataHReg_Activity

3.5.3.19 CRC_HAL_SetDataHUReg()

```
static void CRC_HAL_SetDataHUReg (
            CRC_Type *const base,
            uint8_t value ) [inline], [static]
```

Sets the High Upper Byte - HU.

This function sets the High Upper Byte - HU of CRC data register

Parameters

in	*base*	The CRC peripheral base address
in	*value*	New data for CRC computation Implements : CRC_HAL_SetDataHUReg_Activity

3.5.3.20 CRC_HAL_SetDataLLReg()

```
static void CRC_HAL_SetDataLLReg (
             CRC_Type *const base,
             uint8_t value )  [inline], [static]
```

Sets the Low Lower Byte - LL.

This function sets the Low Lower Byte - LL of CRC data register

Parameters

in	*base*	The CRC peripheral base address
in	*value*	New data for CRC computation Implements : CRC_HAL_SetDataLLReg_Activity

3.5.3.21 CRC_HAL_SetDataLReg()

```
static void CRC_HAL_SetDataLReg (
             CRC_Type *const base,
             uint16_t value )  [inline], [static]
```

Sets the lower 16 bits of CRC data register.

This function sets the lower 16 bits of CRC data register

Parameters

in	*base*	The CRC peripheral base address
in	*value*	New data for CRC computation Implements : CRC_HAL_SetDataLReg_Activity

3.5.3.22 CRC_HAL_SetDataLUReg()

```
static void CRC_HAL_SetDataLUReg (
             CRC_Type *const base,
             uint8_t value )  [inline], [static]
```

Sets the Low Upper Byte - LU.

This function sets the Low Upper Byte - LU of CRC data register

Parameters

in	*base*	The CRC peripheral base address
in	*value*	New data for CRC computation Implements : CRC_HAL_SetDataLUReg_Activity

3.5.3.23 CRC_HAL_SetDataReg()

```
static void CRC_HAL_SetDataReg (
             CRC_Type *const base,
             uint32_t value )  [inline], [static]
```

Sets the 32 bits of CRC data register.

This function sets the 32 bits of CRC data register

Parameters

in	*base*	The CRC peripheral base address
in	*value*	New data for CRC computation Implements : CRC_HAL_SetDataReg_Activity

3.5.3.24 CRC_HAL_SetFXorMode()

```
static void CRC_HAL_SetFXorMode (
            CRC_Type *const base,
            bool enable )  [inline], [static]
```

Sets complement read of CRC data register.

This function sets complement read of CRC data register. Some CRC protocols require the final checksum to be XORed with 0xFFFFFFFF or 0xFFFF. Complement mode enables "on the fly" complementing of read data

Parameters

in	*base*	The CRC peripheral base address
in	*enable*	Enable or disable complementing of read data Implements : CRC_HAL_SetFXorMode_Activity

3.5.3.25 CRC_HAL_SetPolyHReg()

```
static void CRC_HAL_SetPolyHReg (
            CRC_Type *const base,
            uint16_t value )  [inline], [static]
```

Sets the upper 16 bits of polynomial register.

This function sets the upper 16 bits of polynomial register. Note that this upper part of the register is ignored in 16-bit CRC mode

Parameters

in	*base*	The CRC peripheral base address
in	*value*	Polynomial value Implements : CRC_HAL_SetPolyHReg_Activity

3.5.3.26 CRC_HAL_SetPolyLReg()

```
static void CRC_HAL_SetPolyLReg (
            CRC_Type *const base,
            uint16_t value )  [inline], [static]
```

Sets the lower 16 bits of polynomial register.

This function sets the lower 16 bits of polynomial register

Parameters

in	*base*	The CRC peripheral base address
in	*value*	Polynomial value Implements : CRC_HAL_SetPolyLReg_Activity

3.5.3.27 CRC_HAL_SetPolyReg()

```
static void CRC_HAL_SetPolyReg (
            CRC_Type *const base,
            uint32_t value )  [inline], [static]
```

Sets the polynomial register.

This function sets the polynomial register

Parameters

in	*base*	The CRC peripheral base address
in	*value*	Polynomial value Implements : CRC_HAL_SetPolyReg_Activity

3.5.3.28 CRC_HAL_SetProtocolWidth()

```
static void CRC_HAL_SetProtocolWidth (
            CRC_Type *const base,
            crc_bit_width_t width )  [inline], [static]
```

Sets the CRC protocol width.

This function sets the CRC protocol width

Parameters

in	*base*	The CRC peripheral base address
in	*width*	The CRC protocol width • CRC_BITS_16: 16-bit CRC protocol. • CRC_BITS_32: 32-bit CRC protocol. Implements : CRC_HAL_SetProtocolWidth_Activity

3.5.3.29 CRC_HAL_SetReadTranspose()

```
static void CRC_HAL_SetReadTranspose (
            CRC_Type *const base,
            crc_transpose_t transp )  [inline], [static]
```

Sets the CRC transpose type for reads.

This function sets the CRC transpose type for reads

Parameters

in	*base*	The CRC peripheral base address
in	*transp*	The CRC output transpose type Implements : CRC_HAL_SetReadTranspose_Activity

3.5.3.30 CRC_HAL_SetSeedOrDataMode()

```
static void CRC_HAL_SetSeedOrDataMode (
            CRC_Type *const base,
            bool enable )  [inline], [static]
```

Sets the CRC_DATA register mode.

This function sets the CRC_DATA register mode

Parameters

in	*base*	The CRC peripheral base address
in	*enable*	Enable CRC data register to use for seed value -true: use CRC data register for seed values. -false: use CRC data register for data values. Implements : CRC_HAL_SetSeedOrDataMode_Activity

3.5.3.31 CRC_HAL_SetWriteTranspose()

```
static void CRC_HAL_SetWriteTranspose (
            CRC_Type *const base,
            crc_transpose_t transp )  [inline], [static]
```

Sets the CRC transpose type for writes.

This function sets the CRC transpose type for writes

Parameters

in	*base*	The CRC peripheral base address
in	*transp*	The CRC input transpose type Implements : CRC_HAL_SetWriteTranspose_Activity

3.6 CSEc Driver

3.6.1 Detailed Description

Cryptographic Services Engine Peripheral Driver.

How to use the CSEc driver in your application

To access the command feature set, the part must be configured for EEE operation, using the PGMPART command. This can be implemented by using the Flash driver. By enabling security features and configuring a number of user keys, the total size of the 4 KByte EEERAM will be reduced by the space required to store the user keys. The user key space will then effectively be unaddressable space in the EEERAM.

At the bottom of this page is an example of making this configuration using the Flash driver. For more details related to the FLASH_DRV_DEFlashPartition function, please refer to the Flash driver documentation. Please note that this configuration is required only once and should not be lanched from Flash memory.

In order to use the CSEc driver in your application, the **CSEC_DRV_Init** function should be called prior to using the rest of the API. The parameter of this function is used for holding the internal state of the driver throughout the lifetime of the application.

Key/seed/random number generation

This is the high level flow in which to initialize and generate random numbers.

1. Run **CSEC_DRV_InitRNG** to initialize a random seed from the internal TRNG
    - **CSEC_DRV_InitRNG** must be run after every POR, and before the first execution of CSEC_DRV_↩ GenerateRND
    - Note that if the next step (run **CSEC_DRV_GenerateRND**) is run without initializing the seed, **CSEC↩ _RNG_SEED** will be returned.
2. Run **CSEC_DRV_GenerateRND** to generate a random numer The PRNG uses the PRNG_STATE/KEY and Seed per SHE spec and the AIS20 standard.
3. For additional random numbers the user may continue executing **CSEC_DRV_GenerateRND** unless a POR event occurred.

Memory update protocol

In order to update a key, the user must have knowledge of a valid authentication secret, i.e. another key (AuthID). If the key AuthID is empty, the key update will only work if AuthID = ID (the key that will be updated will represent the AuthID from now on), otherwise **CSEC_KEY_EMPTY** is returned.

The M1-M3 values need to be computed according to the SHE Specification in order to update a key slot. The **CSEC_DRV_LoadKey** function will require those values. After successfully updating the key slot, two verification values will be returned: M4 and M5. The user can compute the two values and compare them with the ones returned by the **CSEC_DRV_LoadKey** function in order to ensure the slot was updated as desired. Please refer to the CSEc driver example for a reference implementation of the memory update protocol.

Examples:

Using the Flash driver to partition Flash for CSEc operation

```
flash_ssd_config_t flashSSDConfig;

FLASH_DRV_Init(&flash1_InitConfig0, &flashSSDConfig);

/* Configure the part for EEE operation, with 20 keys for CSEc */
FLASH_DRV_DEFlashPartition(&flashSSDConfig, 0x2, 0x4, 0x3, false);
```

Encryption using AES EBC mode

```
uint8_t plainText[16] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
    0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};
uint8_t plainKey[16] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
    0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

csec_error_code_t stat;
uint8_t cipherText[16];

csec_state_t csecState;

CSEC_DRV_Init(&csecState);

stat = CSEC_DRV_LoadPlainKey(plainKey);
if (stat != CSEC_NO_ERROR)
{
    /* Loading the key failed, encryption will not have the expected result */
    return false;
}

stat = CSEC_DRV_EncryptECB(CSEC_RAM_KEY, plainText, 16, cipherText);
if (stat != CSEC_NO_ERROR)
{
    /* Encryption was successful */
    return true;
}
```

Generating and verifying CMAC for a message

```
uint8_t plainKey[16] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab,
    0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
uint8_t msg[16] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
    0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
uint8_t cmac[16];
bool verifStatus;
csec_error_code_t stat;

csec_state_t csecState;

CSEC_DRV_Init(&csecState);

stat = CSEC_DRV_LoadPlainKey(plainKey);
if (stat != CSEC_NO_ERROR)
    return false;

stat = CSEC_DRV_GenerateMAC(CSEC_RAM_KEY, msg, 128, cmac);
if (stat != CSEC_NO_ERROR)
    return false;

stat = CSEC_DRV_VerifyMAC(CSEC_RAM_KEY, msg, 128, cmac, 128, &verifStatus);
if (stat != CSEC_NO_ERROR)
    return false;

if (!verifStatus)
{
    /* The given CMAC did not matched with the one computed internally */
    return false;
}
```

Generating random bits

```
csec_error_code_t stat;
csec_status_t status;
uint8_t rnd[16];

csec_state_t csecState;

CSEC_DRV_Init(&csecState);
```

```
stat = CSEC_DRV_InitRNG();
if (stat != CSEC_NO_ERROR)
    return false;

/* Check RNG is initialized */
status = CSEC_DRV_GetStatus();
if (!(status & CSEC_STATUS_RND_INIT))
    return false;

stat = CSEC_DRV_GenerateRND(rnd);
if (stat != CSEC_NO_ERROR)
    return false;
```

Data Structures

- struct csec_state_t

    Internal driver state information. More...

Typedefs

- typedef void(∗ csec_callback_t) (csec_cmd_t completedCmd, void ∗callbackParam)

    CSEc asynchronous command complete callback function type.

Enumerations

- enum csec_boot_flavor_t { CSEC_BOOT_STRICT, CSEC_BOOT_SERIAL, CSEC_BOOT_PARALLEL, C←SEC_BOOT_NOT_DEFINED }

    Specifies the boot type for the BOOT_DEFINE command.

Functions

- void CSEC_DRV_Init (csec_state_t ∗state)

    Initializes the internal state of the driver and enables the FTFC interrupt.
- void CSEC_DRV_Deinit (void)

    Clears the internal state of the driver and disables the FTFC interrupt.
- status_t CSEC_DRV_EncryptECB (csec_key_id_t keyId, const uint8_t ∗plainText, uint32_t length, uint8_t ∗cipherText)

    Performs the AES-128 encryption in ECB mode.
- status_t CSEC_DRV_DecryptECB (csec_key_id_t keyId, const uint8_t ∗cipherText, uint32_t length, uint8_t ∗plainText)

    Performs the AES-128 decryption in ECB mode.
- status_t CSEC_DRV_EncryptCBC (csec_key_id_t keyId, const uint8_t ∗plainText, uint32_t length, const uint8_t ∗iv, uint8_t ∗cipherText)

    Performs the AES-128 encryption in CBC mode.
- status_t CSEC_DRV_DecryptCBC (csec_key_id_t keyId, const uint8_t ∗cipherText, uint16_t length, const uint8_t ∗iv, uint8_t ∗plainText)

    Performs the AES-128 decryption in CBC mode.
- status_t CSEC_DRV_GenerateMAC (csec_key_id_t keyId, const uint8_t ∗msg, uint32_t msgLen, uint8_←t ∗cmac)

    Calculates the MAC of a given message using CMAC with AES-128.
- status_t CSEC_DRV_GenerateMACAddrMode (csec_key_id_t keyId, const uint8_t ∗msg, uint32_t msgLen, uint8_t ∗cmac)

    Calculates the MAC of a given message (located in Flash) using CMAC with AES-128.

- status_t CSEC_DRV_VerifyMAC (csec_key_id_t keyId, const uint8_t ∗msg, uint32_t msgLen, const uint8_t ∗mac, uint16_t macLen, bool ∗verifStatus)

  Verifies the MAC of a given message using CMAC with AES-128.
- status_t CSEC_DRV_VerifyMACAddrMode (csec_key_id_t keyId, const uint8_t ∗msg, uint32_t msgLen, const uint8_t ∗mac, uint16_t macLen, bool ∗verifStatus)

  Verifies the MAC of a given message (located in Flash) using CMAC with AES-128.
- status_t CSEC_DRV_LoadKey (csec_key_id_t keyId, const uint8_t ∗m1, const uint8_t ∗m2, const uint8_t ∗m3, uint8_t ∗m4, uint8_t ∗m5)

  Updates an internal key per the SHE specification.
- status_t CSEC_DRV_LoadPlainKey (const uint8_t ∗plainKey)

  Updates the RAM key memory slot with a 128-bit plaintext.
- status_t CSEC_DRV_ExportRAMKey (uint8_t ∗m1, uint8_t ∗m2, uint8_t ∗m3, uint8_t ∗m4, uint8_t ∗m5)

  Exports the RAM_KEY into a format protected by SECRET_KEY.
- status_t CSEC_DRV_InitRNG (void)

  Initializes the seed and derives a key for the PRNG.
- status_t CSEC_DRV_ExtendSeed (const uint8_t ∗entropy)

  Extends the seed of the PRNG.
- status_t CSEC_DRV_GenerateRND (uint8_t ∗rnd)

  Generates a vector of 128 random bits.
- status_t CSEC_DRV_BootFailure (void)

  Signals a failure detected during later stages of the boot process.
- status_t CSEC_DRV_BootOK (void)

  Marks a successful boot verification during later stages of the boot process.
- status_t CSEC_DRV_BootDefine (uint32_t bootSize, csec_boot_flavor_t bootFlavor)

  Implements an extension of the SHE standard to define both the user boot size and boot method.
- static csec_status_t CSEC_DRV_GetStatus (void)

  Returns the content of the status register.
- status_t CSEC_DRV_GetID (const uint8_t ∗challenge, uint8_t ∗uid, uint8_t ∗sreg, uint8_t ∗mac)

  Returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.
- status_t CSEC_DRV_DbgChal (uint8_t ∗challenge)

  Obtains a random number which the user shall use along with the MASTER_ECU_KEY and UID to return an authorization request.
- status_t CSEC_DRV_DbgAuth (const uint8_t ∗authorization)

  Erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed by CSEc.
- status_t CSEC_DRV_MPCompress (const uint8_t ∗msg, uint16_t msgLen, uint8_t ∗mpCompress)

  Compresses the given messages by accessing the Miyaguchi-Prenell compression feature with in the CSEc feature set.
- status_t CSEC_DRV_EncryptECBAsync (csec_key_id_t keyId, const uint8_t ∗plainText, uint32_t length, uint8_t ∗cipherText)

  Asynchronously performs the AES-128 encryption in ECB mode.
- status_t CSEC_DRV_DecryptECBAsync (csec_key_id_t keyId, const uint8_t ∗cipherText, uint32_t length, uint8_t ∗plainText)

  Asynchronously performs the AES-128 decryption in ECB mode.
- status_t CSEC_DRV_EncryptCBCAsync (csec_key_id_t keyId, const uint8_t ∗cipherText, uint16_t length, const uint8_t ∗iv, uint8_t ∗plainText)

  Asynchronously performs the AES-128 encryption in CBC mode.
- status_t CSEC_DRV_DecryptCBCAsync (csec_key_id_t keyId, const uint8_t ∗cipherText, uint32_t length, const uint8_t ∗iv, uint8_t ∗plainText)

  Asynchronously performs the AES-128 decryption in CBC mode.
- status_t CSEC_DRV_GenerateMACAsync (csec_key_id_t keyId, const uint8_t ∗msg, uint32_t msgLen, uint8_t ∗cmac)

  Asynchronously calculates the MAC of a given message using CMAC with AES-128.

- status_t CSEC_DRV_VerifyMACAsync (csec_key_id_t keyId, const uint8_t ∗msg, uint32_t msgLen, const uint8_t ∗mac, uint16_t macLen, bool ∗verifStatus)

  Asynchronously verifies the MAC of a given message using CMAC with AES-128.
- status_t CSEC_DRV_GetAsyncCmdStatus (void)

  Checks the status of the execution of an asynchronous command.
- void CSEC_DRV_InstallCallback (csec_callback_t callbackFunc, void ∗callbackParam)

  Installs a callback function which will be invoked when an asynchronous command finishes its execution.

3.6.2 Data Structure Documentation

3.6.2.1 struct csec_state_t

Internal driver state information.

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.

Implements : csec_state_t_Class

Data Fields

- bool cmdInProgress
- csec_cmd_t cmd
- const uint8_t ∗ inputBuff
- uint8_t ∗ outputBuff
- uint32_t index
- uint32_t fullSize
- uint32_t partSize
- csec_key_id_t keyId
- status_t errCode
- const uint8_t ∗ iv
- csec_call_sequence_t seq
- uint32_t msgLen
- bool ∗ verifStatus
- bool macWritten
- const uint8_t ∗ mac
- uint32_t macLen
- csec_callback_t callback
- void ∗ callbackParam

Field Documentation

3.6.2.1.1 callback

```
csec_callback_t callback
```

The callback invoked when an asynchronous command is completed

3.6.2.1.2 callbackParam

```
void* callbackParam
```

User parameter for the command completion callback

3.6.2.1.3 cmd

```
csec_cmd_t cmd
```

Specifies the type of the command in execution

3.6.2.1.4 cmdInProgress

```
bool cmdInProgress
```

Specifies if a command is in progress

3.6.2.1.5 errCode

```
status_t errCode
```

Specifies the error code of the last executed command

3.6.2.1.6 fullSize

```
uint32_t fullSize
```

Specifies the size of the input of the command in execution

3.6.2.1.7 index

```
uint32_t index
```

Specifies the index in the input buffer of the command in execution

3.6.2.1.8 inputBuff

```
const uint8_t* inputBuff
```

Specifies the input of the command in execution

3.6.2.1.9 iv

```
const uint8_t* iv
```

Specifies the IV of the command in execution (for encryption/decryption using CBC mode)

3.6.2.1.10 keyId

```
csec_key_id_t keyId
```

Specifies the key used for the command in execution

3.6.2.1.11 mac

```
const uint8_t* mac
```

Specifies the MAC to be verified for a MAC verification command

3.6.2.1.12 macLen

```
uint32_t macLen
```

Specifies the number of bits of the MAC to be verified for a MAC verification command

3.6.2.1.13 macWritten

```
bool macWritten
```

Specifies if the MAC to be verified was written in CSE_PRAM for a MAC verification command

3.6.2.1.14 msgLen

```
uint32_t msgLen
```

Specifies the message size (in bits) for the command in execution (for MAC generation/verification)

3.6.2.1.15 outputBuff

```
uint8_t* outputBuff
```

Specifies the output of the command in execution

3.6.2.1.16 partSize

```
uint32_t partSize
```

Specifies the size of the chunck of the input currently processed

3.6.2.1.17 seq

```
csec_call_sequence_t seq
```

Specifies if the information is the first or a following function call.

3.6.2.1.18 verifStatus

```
bool* verifStatus
```

Specifies the result of the last executed MAC verification command

3.6.3 Typedef Documentation

3.6.3.1 csec_callback_t

```
typedef void(* csec_callback_t) (csec_cmd_t completedCmd, void *callbackParam)
```

CSEc asynchronous command complete callback function type.

Implements : csec_callback_t_Class

3.6.4 Enumeration Type Documentation

3.6.4.1 csec_boot_flavor_t

```
enum csec_boot_flavor_t
```

Specifies the boot type for the BOOT_DEFINE command.

Implements : csec_boot_flavor_t_Class

Enumerator

CSEC_BOOT_STRICT	
CSEC_BOOT_SERIAL	
CSEC_BOOT_PARALLEL	
CSEC_BOOT_NOT_DEFINED	

3.6.5 Function Documentation

3.6.5.1 CSEC_DRV_BootDefine()

```
status_t CSEC_DRV_BootDefine (
             uint32_t bootSize,
             csec_boot_flavor_t bootFlavor )
```

Implements an extension of the SHE standard to define both the user boot size and boot method.

The function implements an extension of the SHE standard to define both the user boot size and boot method.

Parameters

in	*bootSize*	Number of blocks of 128-bit data to check on boot. Maximum size is 512kBytes.
in	*bootFlavor*	The boot method.

Returns

Error Code after command execution.

3.6.5.2 CSEC_DRV_BootFailure()

```
status_t CSEC_DRV_BootFailure (
             void )
```

Signals a failure detected during later stages of the boot process.

The function is called during later stages of the boot process to detect a failure.

Returns

Error Code after command execution.

3.6.5.3 CSEC_DRV_BootOK()

```
status_t CSEC_DRV_BootOK (
             void )
```

Marks a successful boot verification during later stages of the boot process.

The function is called during later stages of the boot process to mark successful boot verification.

Returns

Error Code after command execution.

3.6.5.4 CSEC_DRV_DbgAuth()

```
status_t CSEC_DRV_DbgAuth (
            const uint8_t * authorization )
```

Erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed by CSEc.

This function erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed by CSEc.

Parameters

in	*authorization*	Pointer to the 128-bit buffer containing the authorization value.

Returns

Error Code after command execution.

3.6.5.5 CSEC_DRV_DbgChal()

```
status_t CSEC_DRV_DbgChal (
            uint8_t * challenge )
```

Obtains a random number which the user shall use along with the MASTER_ECU_KEY and UID to return an authorization request.

This function obtains a random number which the user shall use along with the MASTER_ECU_KEY and UID to return an authorization request.

Parameters

out	*challenge*	Pointer to the 128-bit buffer where the challenge data will be stored.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.6 CSEC_DRV_DecryptCBC()

```
status_t CSEC_DRV_DecryptCBC (
            csec_key_id_t keyId,
            const uint8_t * cipherText,
            uint16_t length,
            const uint8_t * iv,
            uint8_t * plainText )
```

Performs the AES-128 decryption in CBC mode.

This function performs the AES-128 decryption in CBC mode of the input cipher text buffer.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.

Parameters

in	*cipherText*	Pointer to the cipher text buffer.
in	*length*	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
in	*iv*	Pointer to the initialization vector buffer.
out	*plainText*	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.7 CSEC_DRV_DecryptCBCAsync()

```
status_t CSEC_DRV_DecryptCBCAsync (
            csec_key_id_t keyId,
            const uint8_t * cipherText,
            uint32_t length,
            const uint8_t * iv,
            uint8_t * plainText )
```

Asynchronously performs the AES-128 decryption in CBC mode.

This function performs the AES-128 decryption in CBC mode of the input cipher text buffer, in an asynchronous manner.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*cipherText*	Pointer to the cipher text buffer.
in	*length*	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
in	*iv*	Pointer to the initialization vector buffer.
out	*plainText*	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

Returns

STATUS_SUCCESS if the command was successfully launched, STATUS_BUSY if another command was already launched. CSEC_DRV_GetAsyncCmdStatus can be used in order to check the execution status.

3.6.5.8 CSEC_DRV_DecryptECB()

```
status_t CSEC_DRV_DecryptECB (
            csec_key_id_t keyId,
            const uint8_t * cipherText,
            uint32_t length,
            uint8_t * plainText )
```

Performs the AES-128 decryption in ECB mode.

This function performs the AES-128 decryption in ECB mode of the input cipher text buffer.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation
in	*cipherText*	Pointer to the cipher text buffer.
in	*length*	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
out	*plainText*	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.9 CSEC_DRV_DecryptECBAsync()

```
status_t CSEC_DRV_DecryptECBAsync (
             csec_key_id_t keyId,
             const uint8_t * cipherText,
             uint32_t length,
             uint8_t * plainText )
```

Asynchronously performs the AES-128 decryption in ECB mode.

This function performs the AES-128 decryption in ECB mode of the input cipher text buffer, in an asynchronous manner.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation
in	*cipherText*	Pointer to the cipher text buffer.
in	*length*	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
out	*plainText*	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

Returns

STATUS_SUCCESS if the command was successfully launched, STATUS_BUSY if another command was already launched. CSEC_DRV_GetAsyncCmdStatus can be used in order to check the execution status.

3.6.5.10 CSEC_DRV_Deinit()

```
void CSEC_DRV_Deinit (
             void )
```

Clears the internal state of the driver and disables the FTFC interrupt.

3.6.5.11 CSEC_DRV_EncryptCBC()

```
status_t CSEC_DRV_EncryptCBC (
             csec_key_id_t keyId,
             const uint8_t * plainText,
             uint32_t length,
             const uint8_t * iv,
             uint8_t * cipherText )
```

Performs the AES-128 encryption in CBC mode.

This function performs the AES-128 encryption in CBC mode of the input plaintext buffer.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*plainText*	Pointer to the plain text buffer.
in	*length*	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
in	*iv*	Pointer to the initialization vector buffer.
out	*cipherText*	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.12 CSEC_DRV_EncryptCBCAsync()

```
status_t CSEC_DRV_EncryptCBCAsync (
             csec_key_id_t keyId,
             const uint8_t * cipherText,
             uint16_t length,
             const uint8_t * iv,
             uint8_t * plainText )
```

Asynchronously performs the AES-128 encryption in CBC mode.

This function performs the AES-128 encryption in CBC mode of the input plaintext buffer, in an asynchronous manner.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*plainText*	Pointer to the plain text buffer.
in	*length*	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
in	*iv*	Pointer to the initialization vector buffer.
out	*cipherText*	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.

Returns

STATUS_SUCCESS if the command was successfully launched, STATUS_BUSY if another command was already launched. CSEC_DRV_GetAsyncCmdStatus can be used in order to check the execution status.

3.6.5.13 CSEC_DRV_EncryptECB()

```
status_t CSEC_DRV_EncryptECB (
             csec_key_id_t keyId,
             const uint8_t * plainText,
             uint32_t length,
             uint8_t * cipherText )
```

Performs the AES-128 encryption in ECB mode.

This function performs the AES-128 encryption in ECB mode of the input plain text buffer

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*plainText*	Pointer to the plain text buffer.
in	*length*	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
out	*cipherText*	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.14 CSEC_DRV_EncryptECBAsync()

```
status_t CSEC_DRV_EncryptECBAsync (
             csec_key_id_t keyId,
             const uint8_t * plainText,
             uint32_t length,
             uint8_t * cipherText )
```

Asynchronously performs the AES-128 encryption in ECB mode.

This function performs the AES-128 encryption in ECB mode of the input plain text buffer, in an asynchronous manner.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*plainText*	Pointer to the plain text buffer.
in	*length*	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
out	*cipherText*	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.

Returns

STATUS_SUCCESS if the command was successfully launched, STATUS_BUSY if another command was already launched. CSEC_DRV_GetAsyncCmdStatus can be used in order to check the execution status.

3.6.5.15 CSEC_DRV_ExportRAMKey()

```
status_t CSEC_DRV_ExportRAMKey (
             uint8_t * m1,
             uint8_t * m2,
             uint8_t * m3,
             uint8_t * m4,
             uint8_t * m5 )
```

Exports the RAM_KEY into a format protected by SECRET_KEY.

This function exports the RAM_KEY into a format protected by SECRET_KEY.

Parameters

out	*m1*	Pointer to a buffer where the M1 parameter will be exported.
out	*m2*	Pointer to a buffer where the M2 parameter will be exported.
out	*m3*	Pointer to a buffer where the M3 parameter will be exported.
out	*m4*	Pointer to a buffer where the M4 parameter will be exported.
out	*m5*	Pointer to a buffer where the M5 parameter will be exported.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.16 CSEC_DRV_ExtendSeed()

```
status_t CSEC_DRV_ExtendSeed (
            const uint8_t * entropy )
```

Extends the seed of the PRNG.

Extends the seed of the PRNG by compressing the former seed value and the supplied entropy into a new seed. This new seed is then to be used to generate a random number by invoking the CMD_RND command. The random number generator must be initialized by CMD_INIT_RNG before the seed may be extended.

Parameters

in	*entropy*	Pointer to a 128-bit buffer containing the entropy.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.17 CSEC_DRV_GenerateMAC()

```
status_t CSEC_DRV_GenerateMAC (
            csec_key_id_t keyId,
            const uint8_t * msg,
            uint32_t msgLen,
            uint8_t * cmac )
```

Calculates the MAC of a given message using CMAC with AES-128.

This function calculates the MAC of a given message using CMAC with AES-128.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*msg*	Pointer to the message buffer.
in	*msgLen*	Number of bits of message on which CMAC will be computed.
out	*cmac*	Pointer to the buffer containing the result of the CMAC computation.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.18 CSEC_DRV_GenerateMACAddrMode()

```
status_t CSEC_DRV_GenerateMACAddrMode (
             csec_key_id_t keyId,
             const uint8_t * msg,
             uint32_t msgLen,
             uint8_t * cmac )
```

Calculates the MAC of a given message (located in Flash) using CMAC with AES-128.

This function calculates the MAC of a given message using CMAC with AES-128. It is different from the CSEC_↩ DRV_GenerateMAC function in the sense that it does not involve an extra copy of the data on which the CMAC is computed and the message pointer should be a pointer to Flash memory.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*msg*	Pointer to the message buffer (pointing to Flash memory).
in	*msgLen*	Number of bits of message on which CMAC will be computed.
out	*cmac*	Pointer to the buffer containing the result of the CMAC computation.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.19 CSEC_DRV_GenerateMACAsync()

```
status_t CSEC_DRV_GenerateMACAsync (
             csec_key_id_t keyId,
             const uint8_t * msg,
             uint32_t msgLen,
             uint8_t * cmac )
```

Asynchronously calculates the MAC of a given message using CMAC with AES-128.

This function calculates the MAC of a given message using CMAC with AES-128, in an asynchronous manner.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*msg*	Pointer to the message buffer.
in	*msgLen*	Number of bits of message on which CMAC will be computed.
out	*cmac*	Pointer to the buffer containing the result of the CMAC computation.

Returns

STATUS_SUCCESS if the command was successfully launched, STATUS_BUSY if another command was already launched. CSEC_DRV_GetAsyncCmdStatus can be used in order to check the execution status.

3.6.5.20 CSEC_DRV_GenerateRND()

```
status_t CSEC_DRV_GenerateRND (
            uint8_t * rnd )
```

Generates a vector of 128 random bits.

The function returns a vector of 128 random bits. The random number generator has to be initialized by calling CSEC_DRV_InitRNG before random numbers can be supplied.

Parameters

out	*rnd*	Pointer to a 128-bit buffer where the generated random number has to be stored.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.21 CSEC_DRV_GetAsyncCmdStatus()

```
status_t CSEC_DRV_GetAsyncCmdStatus (
            void )
```

Checks the status of the execution of an asynchronous command.

This function checks the status of the execution of an asynchronous command. If the command is still in progress, returns STATUS_BUSY.

Returns

Error Code after command execution.

3.6.5.22 CSEC_DRV_GetID()

```
status_t CSEC_DRV_GetID (
            const uint8_t * challenge,
            uint8_t * uid,
            uint8_t * sreg,
            uint8_t * mac )
```

Returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.

This function returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.

Parameters

in	*challenge*	Pointer to the 128-bit buffer containing Challenge data.
out	*uid*	Pointer to 120 bit buffer where the UID will be stored.
out	*sreg*	Value of the status register.
out	*mac*	Pointer to the 128 bit buffer where the MAC generated over challenge and UID and status will be stored.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.23 CSEC_DRV_GetStatus()

```
static csec_status_t CSEC_DRV_GetStatus (
             void ) [inline], [static]
```

Returns the content of the status register.

The function shall return the content of the status register.

Returns

Value of the status register.

Implements : CSEC_DRV_GetStatus_Activity

3.6.5.24 CSEC_DRV_Init()

```
void CSEC_DRV_Init (
             csec_state_t * state )
```

Initializes the internal state of the driver and enables the FTFC interrupt.

Parameters

in	*state*	Pointer to the state structure which will be used for holding the internal state of the driver.

3.6.5.25 CSEC_DRV_InitRNG()

```
status_t CSEC_DRV_InitRNG (
             void )
```

Initializes the seed and derives a key for the PRNG.

The function initializes the seed and derives a key for the PRNG. The function must be called before CMD_RND after every power cycle/reset.

Returns

Error Code after command execution.

3.6.5.26 CSEC_DRV_InstallCallback()

```
void CSEC_DRV_InstallCallback (
             csec_callback_t callbackFunc,
             void * callbackParam )
```

Installs a callback function which will be invoked when an asynchronous command finishes its execution.

Parameters

in	*callbackFunc*	The function to be invoked.
in	*callbackParam*	The parameter to be passed to the callback function.

3.6.5.27 CSEC_DRV_LoadKey()

```
status_t CSEC_DRV_LoadKey (
            csec_key_id_t keyId,
            const uint8_t * m1,
            const uint8_t * m2,
            const uint8_t * m3,
            uint8_t * m4,
            uint8_t * m5 )
```

Updates an internal key per the SHE specification.

This function updates an internal key per the SHE specification.

Parameters

in	*key*↩ *Id*	KeyID of the key to be updated.
in	*m1*	Pointer to the 128-bit M1 message containing the UID, Key ID and Authentication Key ID.
in	*m2*	Pointer to the 256-bit M2 message contains the new security flags, counter and the key value all encrypted using a derived key generated from the Authentication Key.
in	*m3*	Pointer to the 128-bit M3 message is a MAC generated over messages M1 and M2.
out	*m4*	Pointer to a 256 bits buffer where the computed M4 parameter is stored.
out	*m5*	Pointer to a 128 bits buffer where the computed M5 parameters is stored.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.28 CSEC_DRV_LoadPlainKey()

```
status_t CSEC_DRV_LoadPlainKey (
            const uint8_t * plainKey )
```

Updates the RAM key memory slot with a 128-bit plaintext.

The function updates the RAM key memory slot with a 128-bit plaintext. The key is loaded without encryption and verification of the key, i.e. the key is handed over in plaintext. A plain key can only be loaded into the RAM_KEY slot.

Parameters

in	*plainKey*	Pointer to the 128-bit buffer containing the key that needs to be copied in RAM_KEY slot.

Returns

Error Code after command execution.

3.6.5.29 CSEC_DRV_MPCompress()

```
status_t CSEC_DRV_MPCompress (
            const uint8_t * msg,
            uint16_t msgLen,
            uint8_t * mpCompress )
```

Compresses the given messages by accessing the Miyaguchi-Prenell compression feature with in the CSEc feature set.

This function accesses a Miyaguchi-Prenell compression feature within the CSEc feature set to compress the given messages.

Parameters

in	*msg*	Pointer to the messages to be compressed. Messages must be pre-processed per SHE specification if they do not already meet the full 128-bit block size requirement.
in	*msgLen*	The number of 128 bit messages to be compressed.
out	*mpCompress*	Pointer to the 128 bit buffer storing the compressed data.

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.30 CSEC_DRV_VerifyMAC()

```
status_t CSEC_DRV_VerifyMAC (
            csec_key_id_t keyId,
            const uint8_t * msg,
            uint32_t msgLen,
            const uint8_t * mac,
            uint16_t macLen,
            bool * verifStatus )
```

Verifies the MAC of a given message using CMAC with AES-128.

This function verifies the MAC of a given message using CMAC with AES-128.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*msg*	Pointer to the message buffer.
in	*msgLen*	Number of bits of message on which CMAC will be computed.
in	*mac*	Pointer to the buffer containing the CMAC to be verified.
in	*macLen*	Number of bits of the CMAC to be compared. A macLength value of zero indicates that all 128-bits are compared.
out	*verifStatus*	Status of MAC verification command (true: verification operation passed, false: verification operation failed).

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.31 CSEC_DRV_VerifyMACAddrMode()

```
status_t CSEC_DRV_VerifyMACAddrMode (
             csec_key_id_t keyId,
             const uint8_t * msg,
             uint32_t msgLen,
             const uint8_t * mac,
             uint16_t macLen,
             bool * verifStatus )
```

Verifies the MAC of a given message (located in Flash) using CMAC with AES-128.

This function verifies the MAC of a given message using CMAC with AES-128. It is different from the CSEC_DRV↩_VerifyMAC function in the sense that it does not involve an extra copy of the data on which the CMAC is computed and the message pointer should be a pointer to Flash memory.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*msg*	Pointer to the message buffer (pointing to Flash memory).
in	*msgLen*	Number of bits of message on which CMAC will be computed.
in	*mac*	Pointer to the buffer containing the CMAC to be verified.
in	*macLen*	Number of bits of the CMAC to be compared. A macLength value of zero indicates that all 128-bits are compared.
out	*verifStatus*	Status of MAC verification command (true: verification operation passed, false: verification operation failed).

Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS_SUCCESS.

3.6.5.32 CSEC_DRV_VerifyMACAsync()

```
status_t CSEC_DRV_VerifyMACAsync (
             csec_key_id_t keyId,
             const uint8_t * msg,
             uint32_t msgLen,
             const uint8_t * mac,
             uint16_t macLen,
             bool * verifStatus )
```

Asynchronously verifies the MAC of a given message using CMAC with AES-128.

This function verifies the MAC of a given message using CMAC with AES-128, in an asynchronous manner.

Parameters

in	*keyId*	KeyID used to perform the cryptographic operation.
in	*msg*	Pointer to the message buffer.

Parameters

in	*msgLen*	Number of bits of message on which CMAC will be computed.
in	*mac*	Pointer to the buffer containing the CMAC to be verified.
in	*macLen*	Number of bits of the CMAC to be compared. A macLength value of zero indicates that all 128-bits are compared.
out	*verifStatus*	Status of MAC verification command (true: verification operation passed, false: verification operation failed).

Returns

STATUS_SUCCESS if the command was successfully launched, STATUS_BUSY if another command was already launched. CSEC_DRV_GetAsyncCmdStatus can be used in order to check the execution status.

3.7 CSEc HAL

3.7.1 Detailed Description

Macros

- #define CSEC_STATUS_BUSY (0x1U)

  The bit is set whenever SHE is processing a command.
- #define CSEC_STATUS_SECURE_BOOT (0x2U)

  The bit is set if the secure booting is activated.
- #define CSEC_STATUS_BOOT_INIT (0x4U)

  The bit is set if the secure booting has been personalized during the boot sequence.
- #define CSEC_STATUS_BOOT_FINISHED (0x8U)

  The bit is set when the secure booting has been finished by calling either CMD_BOOT_FAILURE or CMD_BOOT_OK or if CMD_SECURE_BOOT failed in verifying BOOT_MAC.
- #define CSEC_STATUS_BOOT_OK (0x10U)

  The bit is set if the secure booting (CMD_SECURE_BOOT) succeeded. If CMD_BOOT_FAILURE is called the bit is erased.
- #define CSEC_STATUS_RND_INIT (0x20U)

  The bit is set if the random number generator has been initialized.
- #define CSEC_STATUS_EXT_DEBUGGER (0x40U)

  The bit is set if an external debugger is connected to the chip.
- #define CSEC_STATUS_INT_DEBUGGER (0x80U)

  The bit is set if the internal debugging mechanisms of SHE are activated.
- #define CSEC_NO_ERROR 0x1U

  Represents the result of the execution of a command. Provides one bit for each error code as per SHE specification.
- #define CSEC_SEQUENCE_ERROR 0x2U
- #define CSEC_KEY_NOT_AVAILABLE 0x4U
- #define CSEC_KEY_INVALID 0x8U
- #define CSEC_KEY_EMPTY 0x10U
- #define CSEC_NO_SECURE_BOOT 0x20U
- #define CSEC_KEY_WRITE_PROTECTED 0x40U
- #define CSEC_KEY_UPDATE_ERROR 0x80U
- #define CSEC_RNG_SEED 0x100U
- #define CSEC_NO_DEBUGGING 0x200U
- #define CSEC_MEMORY_FAILURE 0x400U
- #define CSEC_GENERAL_ERROR 0x800U

Typedefs

- typedef uint8_t csec_status_t

  Represents the status of the CSEc module. Provides one bit for each status code as per SHE specification. CSE↩ C_STATUS_∗ masks can be used for verifying the status.

Enumerations

- enum csec_cmd_t {
CSEC_CMD_ENC_ECB = 0x1U, CSEC_CMD_ENC_CBC, CSEC_CMD_DEC_ECB, CSEC_CMD_DEC_↩
CBC,
CSEC_CMD_GENERATE_MAC, CSEC_CMD_VERIFY_MAC, CSEC_CMD_LOAD_KEY, CSEC_CMD_L↩
OAD_PLAIN_KEY,
CSEC_CMD_EXPORT_RAM_KEY, CSEC_CMD_INIT_RNG, CSEC_CMD_EXTEND_SEED, CSEC_CM↩
D_RND,
CSEC_CMD_RESERVED_1, CSEC_CMD_BOOT_FAILURE, CSEC_CMD_BOOT_OK, CSEC_CMD_GE↩
T_ID,
CSEC_CMD_BOOT_DEFINE, CSEC_CMD_DBG_CHAL, CSEC_CMD_DBG_AUTH, CSEC_CMD_TRN↩
G_RND,
CSEC_CMD_RESERVED_2, CSEC_CMD_MP_COMPRESS }

  CSEc commands which follow the same values as the SHE command definition.
- enum csec_key_id_t {
CSEC_SECRET_KEY = 0x0U, CSEC_MASTER_ECU, CSEC_BOOT_MAC_KEY, CSEC_BOOT_MAC,
CSEC_KEY_1, CSEC_KEY_2, CSEC_KEY_3, CSEC_KEY_4,
CSEC_KEY_5, CSEC_KEY_6, CSEC_KEY_7, CSEC_KEY_8,
CSEC_KEY_9, CSEC_KEY_10, CSEC_RAM_KEY = 0xFU, CSEC_KEY_11 = 0x14U,
CSEC_KEY_12, CSEC_KEY_13, CSEC_KEY_14, CSEC_KEY_15,
CSEC_KEY_16, CSEC_KEY_17, CSEC_KEY_18, CSEC_KEY_19,
CSEC_KEY_20, CSEC_KEY_21 }

  Specify the KeyID to be used to implement the requested cryptographic operation.
- enum csec_func_format_t { CSEC_FUNC_FORMAT_COPY, CSEC_FUNC_FORMAT_ADDR }

  Specifies how the data is transferred to/from the CSE. There are two use cases. One is to copy all data and the command function call method and the other is a pointer and function call method.
- enum csec_call_sequence_t { CSEC_CALL_SEQ_FIRST, CSEC_CALL_SEQ_SUBSEQUENT }

  Specifies if the information is the first or a following function call.

Functions

- static void CSEC_HAL_WriteCommandHeader (csec_cmd_t funcId, csec_func_format_t funcFormat, csec↩
_call_sequence_t callSeq, csec_key_id_t keyId)

  Writes the command header to CSE_PRAM.
- void CSEC_HAL_WriteCommandBytes (uint8_t offset, const uint8_t ∗bytes, uint8_t numBytes)

  Writes command bytes to CSE_PRAM.
- void CSEC_HAL_WriteCommandHalfWord (uint8_t offset, uint16_t halfWord)

  Writes a command half word to CSE_PRAM.
- void CSEC_HAL_WriteCommandByte (uint8_t offset, uint8_t byte)

  Writes a command byte to CSE_PRAM.
- void CSEC_HAL_WriteCommandWords (uint8_t offset, const uint32_t ∗words, uint8_t numWords)

  Writes command words to CSE_PRAM.
- void CSEC_HAL_ReadCommandBytes (uint8_t offset, uint8_t ∗bytes, uint8_t numBytes)

  Reads command bytes from CSE_PRAM.
- uint16_t CSEC_HAL_ReadCommandHalfWord (uint8_t offset)

  Reads a command half word from CSE_PRAM.
- uint8_t CSEC_HAL_ReadCommandByte (uint8_t offset)

  Reads a command byte from CSE_PRAM.
- void CSEC_HAL_ReadCommandWords (uint8_t offset, uint32_t ∗words, uint8_t numWords)

  Reads command words from CSE_PRAM.
- static void CSEC_HAL_WaitCommandCompletion (void)

  Waits for the completion of a CSEc command.

- static status_t CSEC_HAL_ReadErrorBits (void)

  Reads the error bits from PRAM.
- static csec_status_t CSEC_HAL_ReadStatus (void)

  Reads the status of the CSEc module.
- START_FUNCTION_DECLARATION_RAMSECTION void CSEC_HAL_WriteCmdAndWait (csec_cmd_↵ t funcId, csec_func_format_t funcFormat, csec_call_sequence_t callSeq, csec_key_id_t keyId) END_FU↵ NCTION_DECLARATION_RAMSECTION static inline void CSEC_HAL_SetInterrupt(bool enable)

  Writes the command header to CSE_PRAM and waits for completion.

3.7.2 Macro Definition Documentation

3.7.2.1 CSEC_GENERAL_ERROR

```
#define CSEC_GENERAL_ERROR 0x800U
```

3.7.2.2 CSEC_KEY_EMPTY

```
#define CSEC_KEY_EMPTY 0x10U
```

3.7.2.3 CSEC_KEY_INVALID

```
#define CSEC_KEY_INVALID 0x8U
```

3.7.2.4 CSEC_KEY_NOT_AVAILABLE

```
#define CSEC_KEY_NOT_AVAILABLE 0x4U
```

3.7.2.5 CSEC_KEY_UPDATE_ERROR

```
#define CSEC_KEY_UPDATE_ERROR 0x80U
```

3.7.2.6 CSEC_KEY_WRITE_PROTECTED

```
#define CSEC_KEY_WRITE_PROTECTED 0x40U
```

3.7.2.7 CSEC_MEMORY_FAILURE

```
#define CSEC_MEMORY_FAILURE 0x400U
```

3.7.2.8 CSEC_NO_DEBUGGING

```
#define CSEC_NO_DEBUGGING 0x200U
```

3.7.2.9 CSEC_NO_ERROR

```
#define CSEC_NO_ERROR 0x1U
```

Represents the result of the execution of a command. Provides one bit for each error code as per SHE specification.

3.7.2.10 CSEC_NO_SECURE_BOOT

```
#define CSEC_NO_SECURE_BOOT 0x20U
```

3.7.2.11 CSEC_RNG_SEED

```
#define CSEC_RNG_SEED 0x100U
```

3.7.2.12 CSEC_SEQUENCE_ERROR

```
#define CSEC_SEQUENCE_ERROR 0x2U
```

3.7.2.13 CSEC_STATUS_BOOT_FINISHED

```
#define CSEC_STATUS_BOOT_FINISHED (0x8U)
```

The bit is set when the secure booting has been finished by calling either CMD_BOOT_FAILURE or CMD_BOO↩T_OK or if CMD_SECURE_BOOT failed in verifying BOOT_MAC.

3.7.2.14 CSEC_STATUS_BOOT_INIT

```
#define CSEC_STATUS_BOOT_INIT (0x4U)
```

The bit is set if the secure booting has been personalized during the boot sequence.

3.7.2.15 CSEC_STATUS_BOOT_OK

```
#define CSEC_STATUS_BOOT_OK (0x10U)
```

The bit is set if the secure booting (CMD_SECURE_BOOT) succeeded. If CMD_BOOT_FAILURE is called the bit is erased.

3.7.2.16 CSEC_STATUS_BUSY

```
#define CSEC_STATUS_BUSY (0x1U)
```

The bit is set whenever SHE is processing a command.

3.7.2.17 CSEC_STATUS_EXT_DEBUGGER

```
#define CSEC_STATUS_EXT_DEBUGGER (0x40U)
```

The bit is set if an external debugger is connected to the chip.

3.7.2.18 CSEC_STATUS_INT_DEBUGGER

```
#define CSEC_STATUS_INT_DEBUGGER (0x80U)
```

The bit is set if the internal debugging mechanisms of SHE are activated.

3.7.2.19 CSEC_STATUS_RND_INIT

```
#define CSEC_STATUS_RND_INIT (0x20U)
```

The bit is set if the random number generator has been initialized.

3.7.2.20 CSEC_STATUS_SECURE_BOOT

```
#define CSEC_STATUS_SECURE_BOOT (0x2U)
```

The bit is set if the secure booting is activated.

3.7.3 Typedef Documentation

3.7.3.1 csec_status_t

```
typedef uint8_t csec_status_t
```

Represents the status of the CSEc module. Provides one bit for each status code as per SHE specification. CSE↩
C_STATUS_∗ masks can be used for verifying the status.

Implements : csec_status_t_Class

3.7.4 Enumeration Type Documentation

3.7.4.1 csec_call_sequence_t

```
enum csec_call_sequence_t
```

Specifies if the information is the first or a following function call.

Implements : csec_call_sequence_t_Class

Enumerator

CSEC_CALL_SEQ_FIRST	
CSEC_CALL_SEQ_SUBSEQUENT	

3.7.4.2 csec_cmd_t

```
enum csec_cmd_t
```

CSEc commands which follow the same values as the SHE command definition.

Implements : csec_cmd_t_Class

Enumerator

CSEC_CMD_ENC_ECB	
CSEC_CMD_ENC_CBC	

Enumerator

CSEC_CMD_DEC_ECB	
CSEC_CMD_DEC_CBC	
CSEC_CMD_GENERATE_MAC	
CSEC_CMD_VERIFY_MAC	
CSEC_CMD_LOAD_KEY	
CSEC_CMD_LOAD_PLAIN_KEY	
CSEC_CMD_EXPORT_RAM_KEY	
CSEC_CMD_INIT_RNG	
CSEC_CMD_EXTEND_SEED	
CSEC_CMD_RND	
CSEC_CMD_RESERVED_1	
CSEC_CMD_BOOT_FAILURE	
CSEC_CMD_BOOT_OK	
CSEC_CMD_GET_ID	
CSEC_CMD_BOOT_DEFINE	
CSEC_CMD_DBG_CHAL	
CSEC_CMD_DBG_AUTH	
CSEC_CMD_TRNG_RND	
CSEC_CMD_RESERVED_2	
CSEC_CMD_MP_COMPRESS	

3.7.4.3 csec_func_format_t

```
enum csec_func_format_t
```

Specifies how the data is transferred to/from the CSE. There are two use cases. One is to copy all data and the command function call method and the other is a pointer and function call method.

Implements : csec_func_format_t_Class

Enumerator

CSEC_FUNC_FORMAT_COPY	
CSEC_FUNC_FORMAT_ADDR	

3.7.4.4 csec_key_id_t

```
enum csec_key_id_t
```

Specify the KeyID to be used to implement the requested cryptographic operation.

Implements : csec_key_id_t_Class

Enumerator

CSEC_SECRET_KEY	
CSEC_MASTER_ECU	
CSEC_BOOT_MAC_KEY	
CSEC_BOOT_MAC	

Enumerator

CSEC_KEY_1	
CSEC_KEY_2	
CSEC_KEY_3	
CSEC_KEY_4	
CSEC_KEY_5	
CSEC_KEY_6	
CSEC_KEY_7	
CSEC_KEY_8	
CSEC_KEY_9	
CSEC_KEY_10	
CSEC_RAM_KEY	
CSEC_KEY_11	
CSEC_KEY_12	
CSEC_KEY_13	
CSEC_KEY_14	
CSEC_KEY_15	
CSEC_KEY_16	
CSEC_KEY_17	
CSEC_KEY_18	
CSEC_KEY_19	
CSEC_KEY_20	
CSEC_KEY_21	

3.7.5 Function Documentation

3.7.5.1 CSEC_HAL_ReadCommandByte()

```
uint8_t CSEC_HAL_ReadCommandByte (
             uint8_t offset )
```

Reads a command byte from CSE_PRAM.

This function reads a command byte from CSE_PRAM.

Parameters

in	*offset*	The offset (in bytes) from which the byte shall be read.

Returns

The byte read.

3.7.5.2 CSEC_HAL_ReadCommandBytes()

```
void CSEC_HAL_ReadCommandBytes (
             uint8_t offset,
             uint8_t * bytes,
             uint8_t numBytes )
```

Reads command bytes from CSE_PRAM.

This function reads command bytes from CSE_PRAM, from a 32-bit aligned offset.

Parameters

in	*offset*	The offset (in bytes) from which the bytes shall be read.
out	*bytes*	The buffer containing the bytes read.
in	*numBytes*	The number of bytes to be read.

3.7.5.3 CSEC_HAL_ReadCommandHalfWord()

```
uint16_t CSEC_HAL_ReadCommandHalfWord (
            uint8_t offset )
```

Reads a command half word from CSE_PRAM.

This function reads a command half word from CSE_PRAM, from a 16-bit aligned offset.

Parameters

in	*offset*	The offset (in bytes) from which the half word shall be read.

Returns

The half word read.

3.7.5.4 CSEC_HAL_ReadCommandWords()

```
void CSEC_HAL_ReadCommandWords (
            uint8_t offset,
            uint32_t * words,
            uint8_t numWords )
```

Reads command words from CSE_PRAM.

This function reads command words from CSE_PRAM, from a 32-bit aligned offset.

Parameters

in	*offset*	The offset (in bytes) from which the words shall be read.
out	*words*	The buffer containing the words read.
in	*numWords*	The number of words to be read.

3.7.5.5 CSEC_HAL_ReadErrorBits()

```
static status_t CSEC_HAL_ReadErrorBits (
            void ) [inline], [static]
```

Reads the error bits from PRAM.

This function reads the error bits reported after running a CSEc command.

Returns

Error Code after command execution.

Implements : CSEC_HAL_ReadErrorBits_Activity

3.7.5.6 CSEC_HAL_ReadStatus()

```
static csec_status_t CSEC_HAL_ReadStatus (
             void  )  [inline], [static]
```

Reads the status of the CSEc module.

This function reads the contents od the status register.

Returns

Status represented as a mask of CSEC_STATUS_∗.

Implements : CSEC_HAL_ReadStatus_Activity

3.7.5.7 CSEC_HAL_WaitCommandCompletion()

```
static void CSEC_HAL_WaitCommandCompletion (
             void  )  [inline], [static]
```

Waits for the completion of a CSEc command.

This function waits for the completion of a CSEc command.

Implements : CSEC_HAL_WaitCommandCompletion_Activity

3.7.5.8 CSEC_HAL_WriteCmdAndWait()

```
START_FUNCTION_DECLARATION_RAMSECTION void CSEC_HAL_WriteCmdAndWait (
             csec_cmd_t funcId,
             csec_func_format_t funcFormat,
             csec_call_sequence_t callSeq,
             csec_key_id_t keyId )
```

Writes the command header to CSE_PRAM and waits for completion.

This function writes the header of a command and waits for completion. The function is always located in RAM, and is used for CSEc commands using pointer methods, in order to allow the MGATE to read from FLASH without causing a read collision.

Parameters

in	*funcId*	The ID of the operation to be started.
in	*funcFormat*	Specifies how the data is transferred to/from the CSE.
in	*callSeq*	Specifies if the information is the first or a following function call.
in	*keyId*	Specify the KeyID to be used to implement the requested cryptographic operation.

Enables/Disables the command completion interrupt.

This function enables/disables the command completion interrupt.

Parameters

in	*enable*	Enable/Disable the command completion interrupt.

Implements : CSEC_HAL_SetInterrupt_Activity

3.7.5.9 CSEC_HAL_WriteCommandByte()

```
void CSEC_HAL_WriteCommandByte (
            uint8_t offset,
            uint8_t byte )
```

Writes a command byte to CSE_PRAM.

This function writes a command byte to CSE_PRAM.

Parameters

in	*offset*	The offset (in bytes) at which the byte shall be written.
in	*byte*	The byte to be written.

3.7.5.10 CSEC_HAL_WriteCommandBytes()

```
void CSEC_HAL_WriteCommandBytes (
            uint8_t offset,
            const uint8_t * bytes,
            uint8_t numBytes )
```

Writes command bytes to CSE_PRAM.

This function writes command bytes to CSE_PRAM, at a 32-bit aligned offset.

Parameters

in	*offset*	The offset (in bytes) at which the bytes shall be written.
in	*bytes*	The buffer containing the bytes to be written.
in	*numBytes*	The number of bytes to be written.

3.7.5.11 CSEC_HAL_WriteCommandHalfWord()

```
void CSEC_HAL_WriteCommandHalfWord (
            uint8_t offset,
            uint16_t halfWord )
```

Writes a command half word to CSE_PRAM.

This function writes a command half word to CSE_PRAM, at a 16-bit aligned offset.

Parameters

in	*offset*	The offset (in bytes) at which the half word shall be written.
in	*halfWord*	The half word to be written.

3.7.5.12 CSEC_HAL_WriteCommandHeader()

```
static void CSEC_HAL_WriteCommandHeader (
            csec_cmd_t funcId,
            csec_func_format_t funcFormat,
            csec_call_sequence_t callSeq,
            csec_key_id_t keyId )  [inline], [static]
```

Writes the command header to CSE_PRAM.

This function writes the command header to CSE_PRAM, triggering the CSEc operation.

Parameters

in	*funcId*	The ID of the operation to be started.
in	*funcFormat*	Specifies how the data is transferred to/from the CSE.
in	*callSeq*	Specifies if the information is the first or a following function call.
in	*keyId*	Specify the KeyID to be used to implement the requested cryptographic operation.

Implements : CSEC_HAL_WriteCommandHeader_Activity

3.7.5.13 CSEC_HAL_WriteCommandWords()

```
void CSEC_HAL_WriteCommandWords (
            uint8_t offset,
            const uint32_t * words,
            uint8_t numWords )
```

Writes command words to CSE_PRAM.

This function writes command words to CSE_PRAM, at a 32-bit aligned offset.

Parameters

in	*offset*	The offset (in bytes) at which the words shall be written.
in	*words*	The buffer containing the words to be written.
in	*numWords*	The number of words to be written.

3.8 Clock Manager

3.8.1 Detailed Description

This module covers the clock management API and clock related functionality.

This section describes the programming interface of the clock_manager driver. Clock_manager achieves its functionality by relying on several HAL components corresponding to the hardware modules involved in clock distribution and management.

Notes

Current implementation assumes that the clock configurations are valid and are applied in a valid sequence. Mainly this means that the configuration doesn't reinitialize the clock used as the system clock.

Code Example

This is an example for switching between two configurations:

```
CLOCK_SYS_Init(g_clockManConfigsArr,
               CLOCK_MANAGER_CONFIG_CNT,
               g_clockManCallbacksArr,
               CLOCK_MANAGER_CALLBACK_CNT);

CLOCK_SYS_UpdateConfiguration(0,
      CLOCK_MANAGER_POLICY_FORCIBLE);
CLOCK_SYS_UpdateConfiguration(1,
      CLOCK_MANAGER_POLICY_FORCIBLE);
```

Modules

- Clock Manager Driver

  This module covers the device-specific clock_manager functionality implemented for S32K144 SOC.
- Peripheral Clock Control (PCC)

  This module covers Peripheral Clock Control module functionality.
- System Clock Generator (SCG)

  This module covers System Clock Generator module functionality.
- System Integration Module (SIM)

  This module cover System Integration Module functionality.

Data Structures

- struct clock_manager_user_config_t

  Clock configuration structure. Implements clock_manager_user_config_t_Class. More...
- struct clock_notify_struct_t

  Clock notification structure passed to clock callback function. Implements clock_notify_struct_t_Class. More...
- struct clock_manager_callback_user_config_t

  Structure for callback function and its parameter. Implements clock_manager_callback_user_config_t_Class. More...
- struct clock_manager_state_t

  Clock manager state structure. Implements clock_manager_state_t_Class. More...

Typedefs

- typedef status_t(∗ clock_manager_callback_t) (clock_notify_struct_t ∗notify, void ∗callbackData)

  Type of clock callback functions.

Enumerations

- enum clock_manager_notify_t { CLOCK_MANAGER_NOTIFY_RECOVER = 0x00U, CLOCK_MANAGER↩_NOTIFY_BEFORE = 0x01U, CLOCK_MANAGER_NOTIFY_AFTER = 0x02U }

  The clock notification type. Implements clock_manager_notify_t_Class.
- enum clock_manager_callback_type_t { CLOCK_MANAGER_CALLBACK_BEFORE = 0x01U, CLOCK_M↩ANAGER_CALLBACK_AFTER = 0x02U, CLOCK_MANAGER_CALLBACK_BEFORE_AFTER = 0x03U }

  The callback type, indicates what kinds of notification this callback handles. Implements clock_manager_callback↩_type_t_Class.
- enum clock_manager_policy_t { CLOCK_MANAGER_POLICY_AGREEMENT, CLOCK_MANAGER_POL↩ICY_FORCIBLE }

  Clock transition policy. Implements clock_manager_policy_t_Class.

Dynamic clock setting

- status_t CLOCK_SYS_Init (clock_manager_user_config_t const ∗∗clockConfigsPtr, uint8_t configsNumber, clock_manager_callback_user_config_t ∗∗callbacksPtr, uint8_t callbacksNumber)

  Install pre-defined clock configurations.
- status_t CLOCK_SYS_UpdateConfiguration (uint8_t targetConfigIndex, clock_manager_policy_t policy)

  Set system clock configuration according to pre-defined structure.
- status_t CLOCK_SYS_SetConfiguration (clock_manager_user_config_t const ∗config)

  Set system clock configuration.
- uint8_t CLOCK_SYS_GetCurrentConfiguration (void)

  Get current system clock configuration.
- clock_manager_callback_user_config_t ∗ CLOCK_SYS_GetErrorCallback (void)

  Get the callback which returns error in last clock switch.
- status_t CLOCK_SYS_GetFreq (clock_names_t clockName, uint32_t ∗frequency)

  Gets the clock frequency for a specific clock name.

3.8.2 Data Structure Documentation

3.8.2.1 struct clock_manager_user_config_t

Clock configuration structure. Implements clock_manager_user_config_t_Class.

Data Fields

- scg_config_t scgConfig
- sim_clock_config_t simConfig
- pcc_config_t pccConfig
- pmc_config_t pmcConfig

Field Documentation

3.8.2.1.1 pccConfig

```
pcc_config_t pccConfig
```

PCC Clock configuration.

3.8.2.1.2 pmcConfig

```
pmc_config_t pmcConfig
```

PMC Clock configuration.

3.8.2.1.3 scgConfig

```
scg_config_t scgConfig
```

SCG Clock configuration.

3.8.2.1.4 simConfig

```
sim_clock_config_t simConfig
```

SIM Clock configuration.

3.8.2.2 struct clock_notify_struct_t

Clock notification structure passed to clock callback function. Implements clock_notify_struct_t_Class.

Data Fields

- uint8_t targetClockConfigIndex
- clock_manager_policy_t policy
- clock_manager_notify_t notifyType

Field Documentation

3.8.2.2.1 notifyType

```
clock_manager_notify_t notifyType
```

Clock notification type.

3.8.2.2.2 policy

```
clock_manager_policy_t policy
```

Clock transition policy.

3.8.2.2.3 targetClockConfigIndex

```
uint8_t targetClockConfigIndex
```

Target clock configuration index.

3.8.2.3 struct clock_manager_callback_user_config_t

Structure for callback function and its parameter. Implements clock_manager_callback_user_config_t_Class.

Data Fields

- clock_manager_callback_t callback
- clock_manager_callback_type_t callbackType
- void ∗ callbackData

Field Documentation

3.8.2.3.1 callback

```
clock_manager_callback_t callback
```

Entry of callback function.

3.8.2.3.2 callbackData

```
void∗ callbackData
```

Parameter of callback function.

3.8.2.3.3 callbackType

```
clock_manager_callback_type_t callbackType
```

Callback type.

3.8.2.4 struct clock_manager_state_t

Clock manager state structure. Implements clock_manager_state_t_Class.

Data Fields

- clock_manager_user_config_t const ∗∗ configTable
- uint8_t clockConfigNum
- uint8_t curConfigIndex
- clock_manager_callback_user_config_t ∗∗ callbackConfig
- uint8_t callbackNum
- uint8_t errorCallbackIndex

Field Documentation

3.8.2.4.1 callbackConfig

`clock_manager_callback_user_config_t** callbackConfig`

Pointer to callback table.

3.8.2.4.2 callbackNum

`uint8_t callbackNum`

Number of clock callbacks.

3.8.2.4.3 clockConfigNum

`uint8_t clockConfigNum`

Number of clock configurations.

3.8.2.4.4 configTable

`clock_manager_user_config_t const** configTable`

Pointer to clock configure table.

3.8.2.4.5 curConfigIndex

`uint8_t curConfigIndex`

Index of current configuration.

3.8.2.4.6 errorCallbackIndex

`uint8_t errorCallbackIndex`

Index of callback returns error.

3.8.3 Typedef Documentation

3.8.3.1 clock_manager_callback_t

```
typedef status_t(* clock_manager_callback_t) (clock_notify_struct_t *notify, void *callback←
Data)
```

Type of clock callback functions.

3.8.4 Enumeration Type Documentation

3.8.4.1 clock_manager_callback_type_t

`enum clock_manager_callback_type_t`

The callback type, indicates what kinds of notification this callback handles. Implements clock_manager_callback←
_type_t_Class.

Enumerator

CLOCK_MANAGER_CALLBACK_BEFORE	Callback handles BEFORE notification.
CLOCK_MANAGER_CALLBACK_AFTER	Callback handles AFTER notification.
CLOCK_MANAGER_CALLBACK_BEFORE_AFTER	Callback handles BEFORE and AFTER notification

3.8.4.2 clock_manager_notify_t

```
enum clock_manager_notify_t
```

The clock notification type. Implements clock_manager_notify_t_Class.

Enumerator

CLOCK_MANAGER_NOTIFY_RECOVER	Notify IP to recover to previous work state.
CLOCK_MANAGER_NOTIFY_BEFORE	Notify IP that system will change clock setting.
CLOCK_MANAGER_NOTIFY_AFTER	Notify IP that have changed to new clock setting.

3.8.4.3 clock_manager_policy_t

```
enum clock_manager_policy_t
```

Clock transition policy. Implements clock_manager_policy_t_Class.

Enumerator

CLOCK_MANAGER_POLICY_AGREEMENT	Clock transfers gracefully.
CLOCK_MANAGER_POLICY_FORCIBLE	Clock transfers forcefully.

3.8.5 Function Documentation

3.8.5.1 CLOCK_SYS_GetCurrentConfiguration()

```
uint8_t CLOCK_SYS_GetCurrentConfiguration (
            void  )
```

Get current system clock configuration.

Returns

Current clock configuration index.

3.8.5.2 CLOCK_SYS_GetErrorCallback()

```
clock_manager_callback_user_config_t* CLOCK_SYS_GetErrorCallback (
            void  )
```

Get the callback which returns error in last clock switch.

When graceful policy is used, if some IP is not ready to change clock setting, the callback will return error and system stay in current configuration. Applications can use this function to check which IP callback returns error.

Returns

Pointer to the callback which returns error.

3.8.5.3 CLOCK_SYS_GetFreq()

```
status_t CLOCK_SYS_GetFreq (
            clock_names_t clockName,
            uint32_t * frequency )
```

Gets the clock frequency for a specific clock name.

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock_names_t. The SCG 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 STATUS_UNSUPPORTED. If frequency is required for a peripheral and the module is not clocked, then STATUS_MCU_GATED_OFF status is returned. Frequency is returned if a valid address is provided. If frequency is required for a peripheral that doesn't support functional clock, the zero value is provided.

Parameters

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

Returns

status Error code defined in status_t

3.8.5.4 CLOCK_SYS_Init()

```
status_t CLOCK_SYS_Init (
            clock_manager_user_config_t const ** clockConfigsPtr,
            uint8_t configsNumber,
            clock_manager_callback_user_config_t ** callbacksPtr,
            uint8_t callbacksNumber )
```

Install pre-defined clock configurations.

This function installs the pre-defined clock configuration table to clock manager.

Parameters

in	*clockConfigsPtr*	Pointer to the clock configuration table.
in	*configsNumber*	Number of clock configurations in table.
in	*callbacksPtr*	Pointer to the callback configuration table.
in	*callbacksNumber*	Number of callback configurations in table.

Returns

Error code.

3.8.5.5 CLOCK_SYS_SetConfiguration()

```
status_t CLOCK_SYS_SetConfiguration (
            clock_manager_user_config_t const * config )
```

Set system clock configuration.

This function sets the system to target configuration, it only sets the clock modules registers for clock mode change, but not send notifications to drivers. This function is different by different SoCs.

Parameters

in	*config*	Target configuration.

Returns

Error code.

Note

If external clock is used in the target mode, please make sure it is enabled, for example, if the external oscillator is used, please setup EREFS/HGO correctly and make sure OSCINIT is set.

3.8.5.6 CLOCK_SYS_UpdateConfiguration()

```
status_t CLOCK_SYS_UpdateConfiguration (
            uint8_t targetConfigIndex,
            clock_manager_policy_t policy )
```

Set system clock configuration according to pre-defined structure.

This function sets system to target clock configuration; before transition, clock manager will send notifications to all drivers registered to the callback table. When graceful policy is used, if some drivers are not ready to change, clock transition will not occur, all drivers still work in previous configuration and error is returned. When forceful policy is used, all drivers should stop work and system changes to new clock configuration.

Parameters

in	*targetConfigIndex*	Index of the clock configuration.
in	*policy*	Transaction policy, graceful or forceful.

Returns

Error code.

Note

If external clock is used in the target mode, please make sure it is enabled, for example, if the external oscillator is used, please setup EREFS/HGO correctly and make sure OSCINIT is set.

3.9 Clock Manager Driver

3.9.1 Detailed Description

This module covers the device-specific clock_manager functionality implemented for S32K144 SOC.

The support for S32K144 consist in the following items:

1. Clock names enumeration clock_names_t is an enumeration which contains all clock names which are relevant for S32K144.
2. Submodule configuration structures
    - scg_config_t
    - pcc_config_t
    - sim_clock_config_t
3. Submodule configuration functions The following functions were implemented for S32K144:
    - CLOCK_SYS_SetScgConfiguration
    - CLOCK_SYS_SetPccConfiguration
    - CLOCK_SYS_SetSimConfiguration

Functions

- status_t CLOCK_SYS_SetScgConfiguration (const scg_config_t ∗scgConfig)

  Configures SCG module.
- void CLOCK_SYS_SetPccConfiguration (const pcc_config_t ∗peripheralClockConfig)

  Configures PCC module.
- void CLOCK_SYS_SetSimConfiguration (const sim_clock_config_t ∗simClockConfig)

  Configures SIM module.
- void CLOCK_SYS_SetPmcConfiguration (const pmc_config_t ∗pmcConfig)

  Configures PMC module.

3.9.2 Function Documentation

3.9.2.1 CLOCK_SYS_SetPccConfiguration()

```
void CLOCK_SYS_SetPccConfiguration (
            const pcc_config_t * peripheralClockConfig )
```

Configures PCC module.

This function configures the PCC module according to the configuration.

Parameters

in	*peripheralClockConfig*	Pointer to the configuration structure.

3.9.2.2 CLOCK_SYS_SetPmcConfiguration()

```
void CLOCK_SYS_SetPmcConfiguration (
            const pmc_config_t * pmcConfig )
```

Configures PMC module.

This function configures the PMC module according to the configuration.

Parameters

in	*pmcConfig*	Pointer to the configuration structure.

3.9.2.3 CLOCK_SYS_SetScgConfiguration()

```
status_t CLOCK_SYS_SetScgConfiguration (
            const scg_config_t * scgConfig )
```

Configures SCG module.

This function configures the SCG module according to the configuration.

Parameters

in	*scgConfig*	Pointer to the configuration structure.

Returns

Status of module initialization

3.9.2.4 CLOCK_SYS_SetSimConfiguration()

```
void CLOCK_SYS_SetSimConfiguration (
            const sim_clock_config_t * simClockConfig )
```

Configures SIM module.

This function configures the SIM module according to the configuration.

Parameters

in	*simClockConfig*	Pointer to the configuration structure.

3.10 Comparator (CMP)

3.10.1 Detailed Description

Hardware background

The comparator (CMP) module is an analog comparator integrated in MCU.

Features of the CMP module include:

- 8 bit DAC with 2 voltage reference source
- 8 analog inputs from external pins
- Round robin check. In summary, this allow the CMP to operate independently in STOP and VLPS mode, whilst being triggered periodically to sample up to 8 inputs. Only if an input changes state is a full wakeup generated.
- Operational over the entire supply range
- Inputs may range from rail to rail
- Programmable hysteresis control
- Selectable interrupt on rising-edge, falling-edge, or both rising or falling edges of the comparator output
- Selectable inversion on comparator output
- Capability to produce a wide range of outputs such as: sampled, windowed, which is ideal for certain PWM zero-crossing-detection applications and digitally filtered
- A comparison event can be selected to trigger a DMA transfer
- The window and filter functions are not available in STOP modes.

How to use the CMP driver in your application

The user can configure the CMP in many ways: -CMP_DRV_Init - configures all CMP features -CMP_DRV_←ConfigDAC - configures only DAC features -CMP_DRV_ConfigTriggerMode - configures only trigger mode features -CMP_DRV_ConfigComparator - configures only analog comparator features -CMP_DRV_ConfigMUX - configures only MUX features

Also the current configuration can be read using: -CMP_DRV_GetConfigAll - gets all CMP configuration -CM←P_DRV_GetDACConfig - gets only DAC configuration -CMP_DRV_GetMUXConfig - gets only MUX configuration -CMP_DRV_GetInitTriggerMode - gets only trigger mode configuration -CMP_DRV_GetComparatorConfig - gets only analog comparator features

When the MCU exits from STOP mode CMP_DRV_GetInputFlags can be used to get the channel which triggered the wakeup. Please use this function only in this use case. CMP_DRV_ClearInputFlags will be used to clear this input change flags.

CMP_DRV_GetOutputFlags can be used to get output flag state and CMP_DRV_GetOutputFlags to clear them.

The main structure used to configure your application is **cmp_module_t**. This structure includes configuration structures for trigger mode, MUX, DAC and comparator: **cmp_comparator_t**, **cmp_anmux_t**, **cmp_dac_t** and **cmp_trigger_mode_t**

Example:

The next example will compare 2 external signals (CMP input 0 an CMP input 1). The output can be measured on port E, pin 4.

```
const cmp_module_t cmp_general_config =
 {
     {
         .dmaTriggerState         = false,
         .outputInterruptTrigger = CMP_NO_EVENT,
         .mode                    = CMP_CONTINUOUS,
         .filterSamplePeriod      = 0,
         .filterSampleCount       = 0,
         .powerMode               = CMP_LOW_SPEED,
         .inverterState           = CMP_NORMAL,
         .outputSelect            = CMP_COUT,
         .pinState                = CMP_AVAILABLE,
         .offsetLevel             = CMP_LEVEL_OFFSET_0,
         .hysteresisLevel         = CMP_LEVEL_HYS_0
     },

     {
         .positivePortMux         = CMP_MUX,
         .negativePortMux         = CMP_MUX,
         .positiveInputMux        = 0,
         .negativeInputMux        = 1
     },

     {
         .voltageReferenceSource = CMP_VIN1,
         .voltage                 = 120,
         .state                   = false,

     },

     {
         .roundRobinState             = false,
         .roundRobinInterruptState    = false,
         .fixedPort                   = CMP_PLUS_FIXED,
         .fixedChannel                = 0,
         .samples                     = 0,
         .initializationDelay         = 0,
         /* Channel 0 is enabled for round robin check */
         /* Channel 1 is enabled for round robin check */
         /* Channel 2 is enabled for round robin check */
         /* Channel 3 is enabled for round robin check */
         /* Channel 4 is enabled for round robin check */
         /* Channel 5 is enabled for round robin check */
         /* Channel 6 is enabled for round robin check */
         /* Channel 7 is enabled for round robin check */
         .roundRobinChannelsState     = 255,
         /* Initial comparison result for channel 0 is 1 */
         /* Initial comparison result for channel 1 is 1 */
         /* Initial comparison result for channel 2 is 1 */
         /* Initial comparison result for channel 3 is 1 */
         /* Initial comparison result for channel 4 is 1 */
         /* Initial comparison result for channel 5 is 1 */
         /* Initial comparison result for channel 6 is 1 */
         /* Initial comparison result for channel 7 is 1 */
         .programedState              = 255
     }
 };

 #define COMPARATOR_PORT          PORTA
 #define COMPARATOR_INPUT1_PIN    0UL
 #define COMPARATOR_INPUT2_PIN    1UL
 #define COMPARATOR_OUTPUT        4UL
 #define COMPARATOR_INSTANCE      0UL

 int main(void)
 {
     /* Write your local variable definition here */
     PCC_Type *pccBase = PCC_BASE_PTRS;
     /* Enable clock source for CMP0 */
     PCC_HAL_SetClockMode(pccBase, PCC_CMP0_CLOCK, true);

     /* Enable clock source for PORTA */
     PCC_HAL_SetClockMode(pccBase, PCC_PORTA_CLOCK, true);

     /* Set pins used by CMP */
     /* The negative port is connected to PTA0 and positive port is connected to PTA1. The
      comparator output can be visualized on PTA4 */
     PORT_HAL_SetMuxModeSel(COMPARATOR_PORT, COMPARATOR_INPUT1_PIN,
      PORT_PIN_DISABLED);
```

```
    PORT_HAL_SetMuxModeSel(COMPARATOR_PORT, COMPARATOR_INPUT2_PIN,
     PORT_PIN_DISABLED);
    /* Please DISCONNECT JTAG. If not the comparator output will be connected to JTAG_TMS.*/
    PORT_HAL_SetMuxModeSel(COMPARATOR_PORT, COMPARATOR_OUTPUT,
     PORT_MUX_ALT4 );
    /* Init CMP module */
    CMP_DRV_Init(COMPARATOR_INSTANCE, &cmp_general_config);
    for (;;)
        {}
    return(0);
}
```

Modules

- Comparator Driver

  Comparator Peripheral Driver.
- Comparator HAL

  Comparator Hardware Abstraction Layer.

3.11 Comparator Driver

3.11.1 Detailed Description

Comparator Peripheral Driver.

Definitions

Data Structures

- struct cmp_comparator_t

  Defines the block configuration. More...
- struct cmp_anmux_t

  Defines the analog mux. More...
- struct cmp_dac_t

  Defines the DAC block. More...
- struct cmp_trigger_mode_t

  Defines the trigger mode. More...
- struct cmp_module_t

  Defines the comparator module configuration. More...

cMP DRV.

- status_t CMP_DRV_Reset (const uint32_t instance)

  Reset all registers.
- status_t CMP_DRV_GetInitConfigAll (cmp_module_t ∗config)

  Get reset configuration for all registers.
- status_t CMP_DRV_Init (const uint32_t instance, const cmp_module_t ∗const config)

  Configure all comparator features with the given configuration structure.
- status_t CMP_DRV_GetConfigAll (const uint32_t instance, cmp_module_t ∗const config)

  Gets the current comparator configuration.
- status_t CMP_DRV_GetInitConfigDAC (cmp_dac_t ∗config)

  Get reset configuration for registers related with DAC.
- status_t CMP_DRV_ConfigDAC (const uint32_t instance, const cmp_dac_t ∗config)

  Configure only the DAC component.
- status_t CMP_DRV_GetDACConfig (const uint32_t instance, cmp_dac_t ∗const config)

  Return current configuration for DAC.
- status_t CMP_DRV_GetInitConfigMUX (cmp_anmux_t ∗config)

  Get reset configuration for registers related with MUX.
- status_t CMP_DRV_ConfigMUX (const uint32_t instance, const cmp_anmux_t ∗config)

  Configure only the MUX component.
- status_t CMP_DRV_GetMUXConfig (const uint32_t instance, cmp_anmux_t ∗const config)

  Return configuration only for the MUX component.
- status_t CMP_DRV_GetInitTriggerMode (cmp_trigger_mode_t ∗config)

  Get reset configuration for registers related with Trigger Mode.
- status_t CMP_DRV_ConfigTriggerMode (const uint32_t instance, const cmp_trigger_mode_t ∗config)

  Configure trigger mode.
- status_t CMP_DRV_GetTriggerModeConfig (const uint32_t instance, cmp_trigger_mode_t ∗const config)

  Get current trigger mode configuration.

- status_t CMP_DRV_GetOutputFlags (const uint32_t instance, cmp_output_trigger_t ∗flags)

  Get comparator output flags.
- status_t CMP_DRV_ClearOutputFlags (const uint32_t instance)

  Clear comparator output flags.
- status_t CMP_DRV_GetInputFlags (const uint32_t instance, cmp_ch_list_t ∗flags)

  Gets input channels change flags.
- status_t CMP_DRV_ClearInputFlags (const uint32_t instance)

  Clear comparator input channels flags.
- status_t CMP_DRV_GetInitConfigComparator (cmp_comparator_t ∗config)

  Get reset configuration for registers related with comparator features.
- status_t CMP_DRV_ConfigComparator (const uint32_t instance, const cmp_comparator_t ∗config)

  Configure only comparator features.
- status_t CMP_DRV_GetComparatorConfig (const uint32_t instance, cmp_comparator_t ∗config)

  Return configuration for comparator from CMP module.

3.11.2 Data Structure Documentation

3.11.2.1 struct cmp_comparator_t

Defines the block configuration.

This structure is used to configure only comparator block module(filtering, sampling, power_mode etc.) Implements : cmp_comparator_t_Class

Data Fields

- bool dmaTriggerState
- cmp_output_trigger_t outputInterruptTrigger
- cmp_mode_t mode
- uint8_t filterSamplePeriod
- uint8_t filterSampleCount
- cmp_power_mode_t powerMode
- cmp_inverter_t inverterState
- cmp_output_enable_t pinState
- cmp_output_select_t outputSelect
- cmp_offset_t offsetLevel
- cmp_hysteresis_t hysteresisLevel

Field Documentation

3.11.2.1.1 dmaTriggerState

```
bool dmaTriggerState
```

True if DMA transfer trigger from comparator is enable.

3.11.2.1.2 filterSampleCount

```
uint8_t filterSampleCount
```

Number of sample count for filtering.

3.11.2.1.3 filterSamplePeriod

```
uint8_t filterSamplePeriod
```

Filter sample period.

3.11.2.1.4 hysteresisLevel

```
cmp_hysteresis_t hysteresisLevel
```

CMP_LEVEL_HYS_0 if hard block output has level 0 hysteresis. CMP_LEVEL_HYS_1 if hard block output has level 1 hysteresis. CMP_LEVEL_HYS_2 if hard block output has level 2 hysteresis. CMP_LEVEL_HYS_3 if hard block output has level 3 hysteresis.

3.11.2.1.5 inverterState

```
cmp_inverter_t inverterState
```

CMP_NORMAL if does not invert the comparator output. CMP_INVERT if inverts the comparator output.

3.11.2.1.6 mode

```
cmp_mode_t mode
```

Configuration structure which define: the comparator functional mode, sample period and sample count.

3.11.2.1.7 offsetLevel

```
cmp_offset_t offsetLevel
```

CMP_LEVEL_OFFSET_0 if hard block output has level 0 offset. CMP_LEVEL_OFFSET_1 if hard block output has level 1 offset.

3.11.2.1.8 outputInterruptTrigger

```
cmp_output_trigger_t outputInterruptTrigger
```

CMP_NO_INTERRUPT comparator output would not trigger any interrupt. CMP_FALLING_EDGE comparator output would trigger an interrupt on falling edge. CMP_RISING_EDGE comparator output would trigger an interrupt on rising edge. CMP_BOTH_EDGES comparator output would trigger an interrupt on rising and falling edges.

3.11.2.1.9 outputSelect

```
cmp_output_select_t outputSelect
```

CMP_COUT if output signal is equal to COUT(filtered). CMP_COUTA if output signal is equal to COUTA(unfiltered).

3.11.2.1.10 pinState

```
cmp_output_enable_t pinState
```

CMP_UNAVAILABLE if comparator output is not available to package pin. CMP_AVAILABLE if comparator output is available to package pin.

3.11.2.1.11 powerMode

```
cmp_power_mode_t powerMode
```

CMP_LOW_SPEED if low speed mode is selected. CMP_HIGH_SPEED if high speed mode is selected

3.11.2.2 struct cmp_anmux_t

Defines the analog mux.

This structure is used to configure the analog multiplexor to select compared signals Implements : cmp_anmux_↩
t_Class

Data Fields

- cmp_port_mux_t positivePortMux
- cmp_port_mux_t negativePortMux
- cmp_ch_number_t positiveInputMux
- cmp_ch_number_t negativeInputMux

Field Documentation

3.11.2.2.1 negativeInputMux

```
cmp_ch_number_t negativeInputMux
```

Select which channel is selected for the minus mux.

3.11.2.2.2 negativePortMux

```
cmp_port_mux_t negativePortMux
```

Select negative port signal. CMP_DAC if source is digital to analog converter. CMP_MUX if source is 8 ch MUX

3.11.2.2.3 positiveInputMux

```
cmp_ch_number_t positiveInputMux
```

Select which channel is selected for the plus mux.

3.11.2.2.4 positivePortMux

```
cmp_port_mux_t positivePortMux
```

Select positive port signal. CMP_DAC if source is digital to analog converter. CMP_MUX if source is 8 ch MUX

3.11.2.3 struct cmp_dac_t

Defines the DAC block.

This structure is used to configure the DAC block integrated in comparator module Implements : cmp_dac_t_Class

Data Fields

- cmp_voltage_reference_t voltageReferenceSource
- uint8_t voltage
- bool state

Field Documentation

3.11.2.3.1 state

```
bool state
```

True if DAC is enabled.

3.11.2.3.2 voltage

```
uint8_t voltage
```

The digital value which is converted to analog signal.

3.11.2.3.3 voltageReferenceSource

```
cmp_voltage_reference_t voltageReferenceSource
```

CMP_VIN1 if selected voltage reference is VIN1. CMP_VIN2 if selected voltage reference is VIN2.

3.11.2.4 struct cmp_trigger_mode_t

Defines the trigger mode.

This structure is used to configure the trigger mode operation when MCU enters STOP modes Implements : cmp←_trigger_mode_t_Class

Data Fields

- bool roundRobinState
- bool roundRobinInterruptState
- cmp_fixed_port_t fixedPort
- cmp_ch_number_t fixedChannel
- uint8_t samples
- uint8_t initializationDelay
- cmp_ch_list_t roundRobinChannelsState
- cmp_ch_list_t programedState

Field Documentation

3.11.2.4.1 fixedChannel

```
cmp_ch_number_t fixedChannel
```

Select which channel would be assigned to the fixed port.

3.11.2.4.2 fixedPort

```
cmp_fixed_port_t fixedPort
```

CMP_PLUS_FIXED if plus port is fixed. CMP_MINUS_FIXED if minus port is fixed.

3.11.2.4.3 initializationDelay

```
uint8_t initializationDelay
```

Select dac and comparator initialization delay(clock cycles).

3.11.2.4.4 programedState

```
cmp_ch_list_t programedState
```

Pre-programmed state for comparison result.

3.11.2.4.5 roundRobinChannelsState

```
cmp_ch_list_t roundRobinChannelsState
```

One bite for each channel state. |———|———|——|———|———| |CH7_state|CH6_state|.....|CH1_state|CH0↩
_state| |--------—|--------—|---—|--------—|-------—|

3.11.2.4.6 roundRobinInterruptState

```
bool roundRobinInterruptState
```

True if Round-Robin interrupt is enabled.

3.11.2.4.7 roundRobinState

```
bool roundRobinState
```

True if Round-Robin is enabled.

3.11.2.4.8 samples

```
uint8_t samples
```

Select number of round-robin clock cycles for a given channel.

3.11.2.5 struct cmp_module_t

Defines the comparator module configuration.

This structure is used to configure all components of comparator module Implements : cmp_module_t_Class

Data Fields

- cmp_comparator_t comparator
- cmp_anmux_t mux
- cmp_dac_t dac
- cmp_trigger_mode_t triggerMode

Field Documentation

3.11.2.5.1 comparator

```
cmp_comparator_t comparator
```

3.11.2.5.2 dac

```
cmp_dac_t dac
```

3.11.2.5.3 mux

```
cmp_anmux_t mux
```

3.11.2.5.4 triggerMode

```
cmp_trigger_mode_t triggerMode
```

3.11.3 Function Documentation

3.11.3.1 CMP_DRV_ClearInputFlags()

```
status_t CMP_DRV_ClearInputFlags (
             const uint32_t instance )
```

Clear comparator input channels flags.

This function clear comparator input channels flags.

Parameters

instance	- instance number

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.2 CMP_DRV_ClearOutputFlags()

```
status_t CMP_DRV_ClearOutputFlags (
             const uint32_t instance )
```

Clear comparator output flags.

This function clear comparator output flags(rising and falling edge).

Parameters

instance	- instance number

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.3 CMP_DRV_ConfigComparator()

```
status_t CMP_DRV_ConfigComparator (
            const uint32_t instance,
            const cmp_comparator_t * config )
```

Configure only comparator features.

This function configure only features related with comparator: DMA request, power mode, output select, interrupts enable, invert, offset, hysteresis.

Parameters

instance	- instance number
config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.4 CMP_DRV_ConfigDAC()

```
status_t CMP_DRV_ConfigDAC (
            const uint32_t instance,
            const cmp_dac_t * config )
```

Configure only the DAC component.

This function configures the DAC with the options provided in the config structure.

Parameters

instance	- instance number
config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.5 CMP_DRV_ConfigMUX()

```
status_t CMP_DRV_ConfigMUX (
            const uint32_t instance,
            const cmp_anmux_t * config )
```

Configure only the MUX component.

This function configures the MUX with the options provided in the config structure.

Parameters

instance	- instance number
config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.6 CMP_DRV_ConfigTriggerMode()

```
status_t CMP_DRV_ConfigTriggerMode (
            const uint32_t instance,
            const cmp_trigger_mode_t * config )
```

Configure trigger mode.

This function configures the trigger mode with the options provided in the config structure.

Parameters

instance	- instance number
config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.7 CMP_DRV_GetComparatorConfig()

```
status_t CMP_DRV_GetComparatorConfig (
            const uint32_t instance,
            cmp_comparator_t * config )
```

Return configuration for comparator from CMP module.

This function return configuration for features related with comparator: DMA request, power mode, output select, interrupts enable, invert, offset, hysteresis.

Parameters

instance	- instance number
config	- the configuration structure returned

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.8 CMP_DRV_GetConfigAll()

```
status_t CMP_DRV_GetConfigAll (
            const uint32_t instance,
            cmp_module_t *const config )
```

Gets the current comparator configuration.

This function returns the current configuration for comparator as a configuration structure.

Parameters

instance	- instance number
config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.9 CMP_DRV_GetDACConfig()

```
status_t CMP_DRV_GetDACConfig (
            const uint32_t instance,
            cmp_dac_t *const config )
```

Return current configuration for DAC.

This function returns current configuration only for DAC.

Parameters

instance	- instance number
config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.10 CMP_DRV_GetInitConfigAll()

```
status_t CMP_DRV_GetInitConfigAll (
            cmp_module_t * config )
```

Get reset configuration for all registers.

This function returns a configuration structure with reset values for all registers from comparator module.

Parameters

config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.11 CMP_DRV_GetInitConfigComparator()

```
status_t CMP_DRV_GetInitConfigComparator (
            cmp_comparator_t * config )
```

Get reset configuration for registers related with comparator features.

This function return a configuration structure with reset values for features associated with comparator (DMA request, power mode, output select, interrupts enable, invert, offset, hysteresis).

Parameters

config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.12 CMP_DRV_GetInitConfigDAC()

```
status_t CMP_DRV_GetInitConfigDAC (
            cmp_dac_t * config )
```

Get reset configuration for registers related with DAC.

This function returns a configuration structure with reset values for features associated with DAC.

Parameters

config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.13 CMP_DRV_GetInitConfigMUX()

```
status_t CMP_DRV_GetInitConfigMUX (
             cmp_anmux_t * config )
```

Get reset configuration for registers related with MUX.

This function returns a configuration structure with reset values for features associated with MUX.

Parameters

config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.14 CMP_DRV_GetInitTriggerMode()

```
status_t CMP_DRV_GetInitTriggerMode (
             cmp_trigger_mode_t * config )
```

Get reset configuration for registers related with Trigger Mode.

This function returns a configuration structure with reset values for features associated with Trigger Mode.

Parameters

config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.15 CMP_DRV_GetInputFlags()

```
status_t CMP_DRV_GetInputFlags (
             const uint32_t instance,
             cmp_ch_list_t * flags )
```

Gets input channels change flags.

This function return in <flags> all input channels flags as uint8_t(1 bite for each channel flag).

Parameters

instance	- instance number
flags	- pointer to input flags

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.16 CMP_DRV_GetMUXConfig()

```
status_t CMP_DRV_GetMUXConfig (
             const uint32_t instance,
             cmp_anmux_t *const config )
```

Return configuration only for the MUX component.

This function returns current configuration to determine which signals go to comparator ports.

Parameters

instance	- instance number
config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.17 CMP_DRV_GetOutputFlags()

```
status_t CMP_DRV_GetOutputFlags (
             const uint32_t instance,
             cmp_output_trigger_t * flags )
```

Get comparator output flags.

This function returns in <flags> comparator output flags(rising and falling edge).

Parameters

instance	- instance number
-	flags - pointer to output flags NO_EVENT RISING_EDGE FALLING_EDGE BOTH_EDGE

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.18 CMP_DRV_GetTriggerModeConfig()

```
status_t CMP_DRV_GetTriggerModeConfig (
            const uint32_t instance,
            cmp_trigger_mode_t *const config )
```

Get current trigger mode configuration.

This function returns the current trigger mode configuration for trigger mode.

Parameters

instance	- instance number
config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.19 CMP_DRV_Init()

```
status_t CMP_DRV_Init (
            const uint32_t instance,
            const cmp_module_t *const config )
```

Configure all comparator features with the given configuration structure.

This function configures the comparator module with the options provided in the config structure.

Parameters

instance	- instance number
config	- the configuration structure

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.11.3.20 CMP_DRV_Reset()

```
status_t CMP_DRV_Reset (
            const uint32_t instance )
```

Reset all registers.

This function set all CMP registers to reset values.

Parameters

instance	- instance number

Returns

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.12 Comparator HAL

3.12.1 Detailed Description

Comparator Hardware Abstraction Layer.

Macros

- #define CMP_INPUT_FLAGS_MASK 0xFF0000
- #define CMP_INPUT_FLAGS_SHIFT 16U
- #define CMP_ROUND_ROBIN_CHANNELS_MASK 0xFF0000
- #define CMP_ROUND_ROBIN_CHANNELS_SHIFT 16U

Typedefs

- typedef uint8_t cmp_ch_list_t

  Comparator channels list (1bit/channel) |--------|--------|----|--------|--------| |CH7_state|CH6_state|.....|CH1_↩ state|CH0_state| |--------|--------|----|--------|--------| Implements : cmp_ch_list_t_Class.
- typedef uint8_t cmp_ch_number_t

  Number of channel Implements : cmp_ch_number_t_Class.

Enumerations

- enum cmp_power_mode_t { CMP_LOW_SPEED = 0U, CMP_HIGH_SPEED = 1U }

  Power Modes selection Implements : cmp_power_mode_t_Class.
- enum cmp_voltage_reference_t { CMP_VIN1 = 0U, CMP_VIN2 = 1U }

  Voltage Reference selection Implements : cmp_voltage_reference_t_Class.
- enum cmp_port_mux_t { CMP_DAC = 0U, CMP_MUX = 1U }

  Port Mux Source selection Implements : cmp_port_mux_t_Class.
- enum cmp_inverter_t { CMP_NORMAL = 0U, CMP_INVERT = 1U }

  Comparator output invert selection Implements : cmp_inverter_t_Class.
- enum cmp_output_select_t { CMP_COUT = 0U, CMP_COUTA = 1U }

  Comparator output select selection Implements : cmp_output_select_t_Class.
- enum cmp_output_enable_t { CMP_UNAVAILABLE = 0U, CMP_AVAILABLE = 1U }

  Comparator output pin enable selection Implements : cmp_output_enable_t_Class.
- enum cmp_offset_t { CMP_LEVEL_OFFSET_0 = 0U, CMP_LEVEL_OFFSET_1 = 1U }

  Comparator hard block offset control Implements : cmp_offset_t_Class.
- enum cmp_hysteresis_t { CMP_LEVEL_HYS_0 = 0U, CMP_LEVEL_HYS_1 = 1U, CMP_LEVEL_HYS_2 = 2U, CMP_LEVEL_HYS_3 = 3U }

  Comparator hysteresis control Implements : cmp_hysteresis_t_Class.
- enum cmp_fixed_port_t { CMP_PLUS_FIXED = 0U, CMP_MINUS_FIXED = 1U }

  Comparator Round-Robin fixed port Implements : cmp_fixed_port_t_Class.
- enum cmp_output_trigger_t { CMP_NO_EVENT = 0U, CMP_FALLING_EDGE = 1U, CMP_RISING_EDGE = 2U, CMP_BOTH_EDGES = 3U }

  Comparator output interrupt configuration Implements : cmp_output_trigger_t_Class.
- enum cmp_mode_t {
  CMP_DISABLED = 0U, CMP_CONTINUOUS = 1U, CMP_SAMPLED_NONFILTRED_INT_CLK = 2U, CM↩ P_SAMPLED_NONFILTRED_EXT_CLK = 3U,
  CMP_SAMPLED_FILTRED_INT_CLK = 4U, CMP_SAMPLED_FILTRED_EXT_CLK = 5U, CMP_WINDO↩ WED = 6U, CMP_WINDOWED_RESAMPLED = 7U,
  CMP_WINDOWED_FILTRED = 8U }

  Comparator functional modes Implements : cmp_mode_t_Class.

CMP_HAL.

- void CMP_HAL_Init (CMP_Type ∗baseAddr)

  Initializes the comparator registers with reset values.
- cmp_mode_t CMP_HAL_GetFunctionalMode (const CMP_Type ∗baseAddr)

  Gets the comparator functional mode. If you want to get filter count and filter period please use CMP_HAL_Get↩ FilterSamplePeriod and CMP_HAL_GetSamplingState.
- void CMP_HAL_SetFunctionalMode (CMP_Type ∗baseAddr, cmp_mode_t mode, uint8_t filter_sample_↩ count, uint8_t filter_sample_period)

  Sets the comparator functional mode (mode, filter count, filter period)
- static bool CMP_HAL_GetDMATriggerState (const CMP_Type ∗baseAddr)

  Verify if the DMA transfer trigger is enabled.
- static void CMP_HAL_SetDMATriggerState (CMP_Type ∗baseAddr, bool to_set)

  Configure the DMA transfer trigger.
- static cmp_output_trigger_t CMP_HAL_GetOutputInterruptTrigger (const CMP_Type ∗baseAddr)

  Return the comparator output interrupts source configuration(none, rising edge, falling edge or both edges)
- static void CMP_HAL_SetOutputInterruptTrigger (CMP_Type ∗baseAddr, cmp_output_trigger_t to_set)

  Set the comparator output interrupts source configuration(none, rising edge, falling edge or both edges)
- static cmp_output_trigger_t CMP_HAL_GetOutputEvent (const CMP_Type ∗baseAddr)

  Return type of event occurred at the comparator output.
- static void CMP_HAL_ClearOutputEvent (CMP_Type ∗baseAddr)

  Clear all output flags.
- static bool CMP_HAL_GetOutputRisingFlag (const CMP_Type ∗baseAddr)

  Verify if a rising edge occurred on COUT.
- static void CMP_HAL_ClearOutputRisingFlag (CMP_Type ∗baseAddr)

  Clear rising edge flag.
- static bool CMP_HAL_GetOutputFallingFlag (const CMP_Type ∗baseAddr)

  Verify if a falling-edge occurred on COUT.
- static void CMP_HAL_ClearOutputFallingFlag (CMP_Type ∗baseAddr)

  Clear falling edge flag.
- static bool CMP_HAL_GetComparatorOutput (const CMP_Type ∗baseAddr)

  Return the analog comparator output value.
- static uint8_t CMP_HAL_GetFilterSamplePeriod (const CMP_Type ∗baseAddr)

  Return the sample period for filter(clock cycles)
- static void CMP_HAL_SetFilterSamplePeriod (CMP_Type ∗baseAddr, uint8_t to_set)

  Set the filter sample period(clock cycles)
- static bool CMP_HAL_GetSamplingState (const CMP_Type ∗baseAddr)

  Verify if the sampling mode is selected.
- static void CMP_HAL_SetSamplingState (CMP_Type ∗baseAddr, bool to_set)

  Set the sampling mode state.
- static bool CMP_HAL_GetWindowingModeState (const CMP_Type ∗baseAddr)
- static void CMP_HAL_SetWindowingModeState (CMP_Type ∗baseAddr, bool to_set)

  Set the windowing mode state.
- static cmp_power_mode_t CMP_HAL_GetPowerMode (const CMP_Type ∗baseAddr)

  Return the current power mode.
- static void CMP_HAL_SetPowerMode (CMP_Type ∗baseAddr, cmp_power_mode_t to_set)

  Set the power mode.
- static cmp_inverter_t CMP_HAL_GetInverterState (const CMP_Type ∗baseAddr)

  Return the current comparator output inverter.
- static void CMP_HAL_SetInverterState (CMP_Type ∗baseAddr, cmp_inverter_t to_set)

  Configure the comparator output inverter mode.

- static cmp_output_select_t CMP_HAL_GetComparatorOutputSource (const CMP_Type ∗baseAddr)

  Return the current comparator output selected.
- static void CMP_HAL_SetComparatorOutputSource (CMP_Type ∗baseAddr, cmp_output_select_t to_set)

  Select the comparator output signal source.
- static cmp_output_enable_t CMP_HAL_GetOutputPinState (const CMP_Type ∗baseAddr)

  Verify if the comparator output state(available/not available in a packaged pin)
- static void CMP_HAL_SetOutputPinState (CMP_Type ∗baseAddr, cmp_output_enable_t to_set)

  Set the comparator output pin state(available/not available in a packaged pin)
- static bool CMP_HAL_GetAnalogComparatorState (const CMP_Type ∗baseAddr)

  Verify if the analog comparator module is enabled.
- static void CMP_HAL_SetAnalogComparatorState (CMP_Type ∗baseAddr, bool to_set)

  Set the analog comparator module state.
- static uint8_t CMP_HAL_GetFilterSampleCount (const CMP_Type ∗baseAddr)

  Return the number of consecutive samples that must agree prior to the comparator output filter accepting a new output state.
- static void CMP_HAL_SetFilterSampleCount (CMP_Type ∗baseAddr, uint8_t to_set)

  Set the number of consecutive samples that must agree prior to the comparator output filter accepting a new output state.
- static cmp_offset_t CMP_HAL_GetOffset (const CMP_Type ∗baseAddr)

  Return the current offset level.
- static void CMP_HAL_SetOffset (CMP_Type ∗baseAddr, cmp_offset_t to_set)

  Set the offset level.
- static cmp_hysteresis_t CMP_HAL_GetHysteresis (const CMP_Type ∗baseAddr)

  Return the current hysteresis level.
- static void CMP_HAL_SetHysteresis (CMP_Type ∗baseAddr, cmp_hysteresis_t to_set)

  Set the hysteresis level.
- static void CMP_HAL_SetDACOutputState (CMP_Type ∗baseAddr, bool to_set)

  Set if the DAC output is enabled to go outside of this block.
- static bool CMP_HAL_GetDACOutputState (const CMP_Type ∗baseAddr)

  Get if the DAC output is enabled to go outside of this block.
- static cmp_port_mux_t CMP_HAL_GetPositivePortInput (const CMP_Type ∗baseAddr)

  Return the current source for positive port of the comparator.
- static void CMP_HAL_SetPositivePortInput (CMP_Type ∗baseAddr, cmp_port_mux_t to_set)

  Set the source for positive port of the comparator.
- static cmp_port_mux_t CMP_HAL_GetNegativePortInput (const CMP_Type ∗baseAddr)

  Return the current source for negative port of the comparator.
- static void CMP_HAL_SetNegativePortInput (CMP_Type ∗baseAddr, cmp_port_mux_t to_set)

  Set the source for negative port of the comparator.
- static cmp_ch_list_t CMP_HAL_GetRoundRobinChannels (const CMP_Type ∗baseAddr)

  Return which channels are used for round-robin checker.
- static void CMP_HAL_SetRoundRobinChannels (CMP_Type ∗baseAddr, cmp_ch_list_t to_set)

  Set which channels are use for round-robin checker.
- static bool CMP_HAL_GetDACState (const CMP_Type ∗baseAddr)

  Verify if the DAC is enabled.
- static void CMP_HAL_SetDACState (CMP_Type ∗baseAddr, bool to_set)

  Set the DAC state (enabled/disabled)
- static cmp_voltage_reference_t CMP_HAL_GetVoltageReference (const CMP_Type ∗baseAddr)

  Return the current voltage reference.
- static void CMP_HAL_SetVoltageReference (CMP_Type ∗baseAddr, cmp_voltage_reference_t to_set)

  Set the voltage reference.
- static cmp_ch_number_t CMP_HAL_GetPlusMUXControl (const CMP_Type ∗baseAddr)

Determine which input is selected for the plus mux.

- static void CMP_HAL_SetPlusMuxControl (CMP_Type ∗baseAddr, cmp_ch_number_t to_set)

  Select input for the plus mux.
- static cmp_ch_number_t CMP_HAL_GetMinusMUXControl (const CMP_Type ∗baseAddr)

  Determine which input is selected for the minus mux.
- static void CMP_HAL_SetMinusMUXControl (CMP_Type ∗baseAddr, cmp_ch_number_t to_set)

  Select input for the minus mux.
- static uint8_t CMP_HAL_GetVoltage (const CMP_Type ∗baseAddr)

  Return the current output voltage level(0-255)
- static void CMP_HAL_SetVoltage (CMP_Type ∗baseAddr, uint8_t to_set)

  Set the output voltage level.
- static bool CMP_HAL_GetRoundRobinState (const CMP_Type ∗baseAddr)

  Verify if the round robin operation is enabled.
- static void CMP_HAL_SetRoundRobinState (CMP_Type ∗baseAddr, bool to_set)

  Set the round robin operation state.
- static bool CMP_HAL_GetRoundRobinInterruptState (const CMP_Type ∗baseAddr)

  Verify if the round robin interrupt is enabled.
- static void CMP_HAL_SetRoundRobinInterruptState (CMP_Type ∗baseAddr, bool to_set)

  Set the round robin interrupt state.
- static cmp_fixed_port_t CMP_HAL_GetFixedPort (const CMP_Type ∗baseAddr)

  Return the port fixed for round-robin operation.
- static void CMP_HAL_SetFixedPort (CMP_Type ∗baseAddr, cmp_fixed_port_t to_set)

  Set the fixed port for round-robin operation.
- static cmp_ch_number_t CMP_HAL_GetFixedChannel (const CMP_Type ∗baseAddr)

  Return which channel is selected for fixed mux port(as fixed reference)
- static void CMP_HAL_SetFixedChannel (CMP_Type ∗baseAddr, cmp_ch_number_t to_set)

  Set which channel is used as the fixed reference input for the fixed mux port.
- static cmp_ch_list_t CMP_HAL_GetInputChangedFlags (const CMP_Type ∗baseAddr)

  Return all input changed flags.
- static void CMP_HAL_ClearInputChangedFlags (CMP_Type ∗baseAddr)

  Clear all input changed flags.
- static uint8_t CMP_HAL_GetRoundRobinSamplesNumber (const CMP_Type ∗baseAddr)

  Return how many round-robin clock cycles takes sampling.
- static void CMP_HAL_SetRoundRobinSamplesNumber (CMP_Type ∗baseAddr, uint8_t to_set)

  Set how many round-robin clock cycles takes sampling.
- static uint8_t CMP_HAL_GetInitDelay (const CMP_Type ∗baseAddr)

  Return the comparator and DAC initialization delay.
- static void CMP_HAL_SetInitDelay (CMP_Type ∗baseAddr, uint8_t to_set)

  Set the comparator and DAC initialization delay.
- static cmp_ch_list_t CMP_HAL_GetLastComparisonResult (const CMP_Type ∗baseAddr)

  Return last input comparison results for all channels.
- static void CMP_HAL_SetPresetState (CMP_Type ∗baseAddr, cmp_ch_list_t to_set)

  Defines the pre-set state of input channels.

3.12.2 Macro Definition Documentation

3.12.2.1 CMP_INPUT_FLAGS_MASK

```
#define CMP_INPUT_FLAGS_MASK 0xFF0000
```

3.12.2.2 CMP_INPUT_FLAGS_SHIFT

```
#define CMP_INPUT_FLAGS_SHIFT 16U
```

3.12.2.3 CMP_ROUND_ROBIN_CHANNELS_MASK

```
#define CMP_ROUND_ROBIN_CHANNELS_MASK 0xFF0000
```

3.12.2.4 CMP_ROUND_ROBIN_CHANNELS_SHIFT

```
#define CMP_ROUND_ROBIN_CHANNELS_SHIFT 16U
```

3.12.3 Typedef Documentation

3.12.3.1 cmp_ch_list_t

```
typedef uint8_t cmp_ch_list_t
```

Comparator channels list (1bit/channel) |--------—|--------—|-—–|--------—|--------—| |CH7_state|CH6_state|.....|CH1_↩ state|CH0_state| |--------—|--------—|-—–|--------—|--------—| Implements : cmp_ch_list_t_Class.

3.12.3.2 cmp_ch_number_t

```
typedef uint8_t cmp_ch_number_t
```

Number of channel Implements : cmp_ch_number_t_Class.

3.12.4 Enumeration Type Documentation

3.12.4.1 cmp_fixed_port_t

```
enum cmp_fixed_port_t
```

Comparator Round-Robin fixed port Implements : cmp_fixed_port_t_Class.

Enumerator

CMP_PLUS_FIXED	The Plus port is fixed. Only the inputs to the Minus port are swept in each round.
CMP_MINUS_FIXED	The Minus port is fixed. Only the inputs to the Plus port are swept in each round.

3.12.4.2 cmp_hysteresis_t

```
enum cmp_hysteresis_t
```

Comparator hysteresis control Implements : cmp_hysteresis_t_Class.

Enumerator

CMP_LEVEL_HYS↩_0	
CMP_LEVEL_HYS↩_1	
CMP_LEVEL_HYS↩_2	
CMP_LEVEL_HYS↩_3	

3.12.4.3 cmp_inverter_t

```
enum cmp_inverter_t
```

Comparator output invert selection Implements : cmp_inverter_t_Class.

Enumerator

CMP_NORMAL	Output signal isn't inverted.
CMP_INVERT	Output signal is inverted.

3.12.4.4 cmp_mode_t

```
enum cmp_mode_t
```

Comparator functional modes Implements : cmp_mode_t_Class.

Enumerator

CMP_DISABLED	
CMP_CONTINUOUS	
CMP_SAMPLED_NONFILTRED_INT_CLK	
CMP_SAMPLED_NONFILTRED_EXT_CLK	
CMP_SAMPLED_FILTRED_INT_CLK	
CMP_SAMPLED_FILTRED_EXT_CLK	
CMP_WINDOWED	
CMP_WINDOWED_RESAMPLED	
CMP_WINDOWED_FILTRED	

3.12.4.5 cmp_offset_t

```
enum cmp_offset_t
```

Comparator hard block offset control Implements : cmp_offset_t_Class.

Enumerator

CMP_LEVEL_OFFSET↩_0	
CMP_LEVEL_OFFSET↩_1	

3.12.4.6 cmp_output_enable_t

```
enum cmp_output_enable_t
```

Comparator output pin enable selection Implements : cmp_output_enable_t_Class.

Enumerator

CMP_UNAVAILABLE	Comparator output isn't available to a specific pin
CMP_AVAILABLE	Comparator output is available to a specific pin

3.12.4.7 cmp_output_select_t

```
enum cmp_output_select_t
```

Comparator output select selection Implements : cmp_output_select_t_Class.

Enumerator

CMP_COUT	Select COUT as comparator output signal.
CMP_COUTA	Select COUTA as comparator output signal.

3.12.4.8 cmp_output_trigger_t

```
enum cmp_output_trigger_t
```

Comparator output interrupt configuration Implements : cmp_output_trigger_t_Class.

Enumerator

CMP_NO_EVENT	Comparator output interrupts are disabled OR no event occurred.
CMP_FALLING_EDGE	Comparator output interrupts will be generated only on falling edge OR only falling edge event occurred.
CMP_RISING_EDGE	Comparator output interrupts will be generated only on rising edge OR only rising edge event occurred.
CMP_BOTH_EDGES	Comparator output interrupts will be generated on both edges OR both edges event occurred.

3.12.4.9 cmp_port_mux_t

```
enum cmp_port_mux_t
```

Port Mux Source selection Implements : cmp_port_mux_t_Class.

Enumerator

CMP_DAC	Select DAC as source for the comparator port.
CMP_MUX	Select MUX8 as source for the comparator port.

3.12.4.10 cmp_power_mode_t

```
enum cmp_power_mode_t
```

Power Modes selection Implements : cmp_power_mode_t_Class.

Enumerator

CMP_LOW_SPEED	Module in low speed mode.
CMP_HIGH_SPEED	Module in high speed mode.

3.12.4.11 cmp_voltage_reference_t

```
enum cmp_voltage_reference_t
```

Voltage Reference selection Implements : cmp_voltage_reference_t_Class.

Enumerator

CMP_VIN1	Use Vin1 as supply reference source for DAC.
CMP_VIN2	Use Vin2 as supply reference source for DAC.

3.12.5 Function Documentation

3.12.5.1 CMP_HAL_ClearInputChangedFlags()

```
static void CMP_HAL_ClearInputChangedFlags (
             CMP_Type * baseAddr )  [inline], [static]
```

Clear all input changed flags.

Parameters

baseAddr	- cmp base pointer

Returns

- void Implements : CMP_HAL_ClearInputChangedFlags_Activity

3.12.5.2 CMP_HAL_ClearOutputEvent()

```
static void CMP_HAL_ClearOutputEvent (
             CMP_Type * baseAddr )  [inline], [static]
```

Clear all output flags.

Parameters

baseAddr	- cmp base pointer

Returns

- void Implements : CMP_HAL_ClearOutputEvent_Activity

3.12.5.3 CMP_HAL_ClearOutputFallingFlag()

```
static void CMP_HAL_ClearOutputFallingFlag (
            CMP_Type * baseAddr )  [inline], [static]
```

Clear falling edge flag.

Parameters

baseAddr	- cmp base pointer

Returns

- void Implements : CMP_HAL_ClearOutputFallingFlag_Activity

3.12.5.4 CMP_HAL_ClearOutputRisingFlag()

```
static void CMP_HAL_ClearOutputRisingFlag (
            CMP_Type * baseAddr )  [inline], [static]
```

Clear rising edge flag.

Parameters

baseAddr	- cmp base pointer

Returns

- void Implements : CMP_HAL_ClearOutputRisingFlag_Activity

3.12.5.5 CMP_HAL_GetAnalogComparatorState()

```
static bool CMP_HAL_GetAnalogComparatorState (
            const CMP_Type * baseAddr )  [inline], [static]
```

Verify if the analog comparator module is enabled.

Parameters

baseAddr	- cmp base pointer

Returns

- module state true - module is enabled false - module is disabled Implements : CMP_HAL_GetAnalog↩ ComparatorState_Activity

3.12.5.6 CMP_HAL_GetComparatorOutput()

```
static bool CMP_HAL_GetComparatorOutput (
             const CMP_Type * baseAddr )  [inline], [static]
```

Return the analog comparator output value.

Parameters

baseAddr	- cmp base pointer

Returns

- analog comparator output value Implements : CMP_HAL_GetComparatorOutput_Activity

3.12.5.7 CMP_HAL_GetComparatorOutputSource()

```
static cmp_output_select_t CMP_HAL_GetComparatorOutputSource (
             const CMP_Type * baseAddr )  [inline], [static]
```

Return the current comparator output selected.

Parameters

baseAddr	- cmp base pointer

Returns

- comparator output signal source CMP_COUT CMP_COUTA Implements : CMP_HAL_GetComparator↩ OutputSource_Activity

3.12.5.8 CMP_HAL_GetDACOutputState()

```
static bool CMP_HAL_GetDACOutputState (
             const CMP_Type * baseAddr )  [inline], [static]
```

Get if the DAC output is enabled to go outside of this block.

Parameters

baseAddr	- cmp base pointer
to_set	- DAC output state true - DAC output go outside of DAC block(to packaged pin) false - DAC output doesn't go outside of DAC block(to packaged pin)

Returns

- void Implements : CMP_HAL_GetDACOutputState_Activity

3.12.5.9 CMP_HAL_GetDACState()

```
static bool CMP_HAL_GetDACState (
            const CMP_Type * baseAddr )  [inline], [static]
```

Verify if the DAC is enabled.

Parameters

baseAddr	- cmp base pointer

Returns

- dac state true - DAC is enabled false - DAC is disabled Implements : CMP_HAL_GetDACState_Activity

3.12.5.10 CMP_HAL_GetDMATriggerState()

```
static bool CMP_HAL_GetDMATriggerState (
            const CMP_Type * baseAddr )  [inline], [static]
```

Verify if the DMA transfer trigger is enabled.

Parameters

baseAddr	- cmp base pointer

Returns

- DMA transfer trigger state true - DMA trigger is enabled false - DAM trigger is disabled Implements : CMP↩
_HAL_GetDMATriggerState_Activity

3.12.5.11 CMP_HAL_GetFilterSampleCount()

```
static uint8_t CMP_HAL_GetFilterSampleCount (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return the number of consecutive samples that must agree prior to the comparator output filter accepting a new output state.

Parameters

baseAddr	- cmp base pointer

Returns

- filter sample count Implements : CMP_HAL_GetFilterSampleCount_Activity

3.12.5.12 CMP_HAL_GetFilterSamplePeriod()

```
static uint8_t CMP_HAL_GetFilterSamplePeriod (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return the sample period for filter(clock cycles)

Parameters

baseAddr	- cmp base pointer

Returns

- sampling period(in bus cycles) Implements : CMP_HAL_GetFilterSamplePeriod_Activity

3.12.5.13 CMP_HAL_GetFixedChannel()

```
static cmp_ch_number_t CMP_HAL_GetFixedChannel (
             const CMP_Type * baseAddr )  [inline], [static]
```

Return which channel is selected for fixed mux port(as fixed reference)

Parameters

baseAddr	- cmp base pointer

Returns

- fixed channel CMP_PLUS_FIXED CMP_MINUS_FIXED Implements : CMP_HAL_GetFixedChannel_↩ Activity

3.12.5.14 CMP_HAL_GetFixedPort()

```
static cmp_fixed_port_t CMP_HAL_GetFixedPort (
             const CMP_Type * baseAddr )  [inline], [static]
```

Return the port fixed for round-robin operation.

Parameters

baseAddr	- cmp base pointer

Returns

- fixed port Implements : CMP_HAL_GetFixedPort_Activity

3.12.5.15 CMP_HAL_GetFunctionalMode()

```
cmp_mode_t CMP_HAL_GetFunctionalMode (
             const CMP_Type * baseAddr )
```

Gets the comparator functional mode. If you want to get filter count and filter period please use CMP_HAL_Get↩ FilterSamplePeriod and CMP_HAL_GetSamplingState.

Parameters

baseAddr	- cmp base pointer

Returns

- functional mode CMP_DISABLED CMP_CONTINUOUS CMP_SAMPLED_NONFILTRED_INT_CLK CM←P_SAMPLED_NONFILTRED_EXT_CLK CMP_SAMPLED_FILTRED_INT_CLK CMP_SAMPLED_FILTRE←D_EXT_CLK CMP_WINDOWED CMP_WINDOWED_RESAMPLED CMP_WINDOWED_FILTRED

3.12.5.16 CMP_HAL_GetHysteresis()

```
static cmp_hysteresis_t CMP_HAL_GetHysteresis (
            const CMP_Type * baseAddr ) [inline], [static]
```

Return the current hysteresis level.

Parameters

baseAddr	- cmp base pointer

Returns

- current hysteresis level CMP_LEVEL_HYS_0 CMP_LEVEL_HYS_1 CMP_LEVEL_HYS_2 CMP_LEVEL_←HYS_3 Implements : CMP_HAL_GetHysteresis_Activity

3.12.5.17 CMP_HAL_GetInitDelay()

```
static uint8_t CMP_HAL_GetInitDelay (
            const CMP_Type * baseAddr ) [inline], [static]
```

Return the comparator and DAC initialization delay.

Parameters

baseAddr	- cmp base pointer

Returns

- delay(round-robin clock period) Implements : CMP_HAL_GetInitDelay_Activity

3.12.5.18 CMP_HAL_GetInputChangedFlags()

```
static cmp_ch_list_t CMP_HAL_GetInputChangedFlags (
            const CMP_Type * baseAddr ) [inline], [static]
```

Return all input changed flags.

Parameters

baseAddr	- cmp base pointer

Returns

- flags status |-------|-------|--—|-------|-------| |CH7_flag|CH6_flag|.....|CH1_flag|CH0_flag| |-------|-------|--—|-------|-------| Implements : CMP_HAL_GetInputChangedFlags_Activity

3.12.5.19 CMP_HAL_GetInverterState()

```
static cmp_inverter_t CMP_HAL_GetInverterState (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return the current comparator output inverter.

Parameters

baseAddr	cmp base pointer

Returns

- inverter state CMP_NORMAL CMP_INVERT Implements : CMP_HAL_GetInverterState_Activity

3.12.5.20 CMP_HAL_GetLastComparisonResult()

```
static cmp_ch_list_t CMP_HAL_GetLastComparisonResult (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return last input comparison results for all channels.

Parameters

baseAddr	- cmp base pointer

Returns

- comparison results Implements : CMP_HAL_GetLastComparisonResult_Activity

3.12.5.21 CMP_HAL_GetMinusMUXControl()

```
static cmp_ch_number_t CMP_HAL_GetMinusMUXControl (
            const CMP_Type * baseAddr )  [inline], [static]
```

Determine which input is selected for the minus mux.

Parameters

baseAddr	- cmp base pointer

Returns

- channel for the minus mux Implements : CMP_HAL_GetMinusMUXControl_Activity

3.12.5.22 CMP_HAL_GetNegativePortInput()

```
static cmp_port_mux_t CMP_HAL_GetNegativePortInput (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return the current source for negative port of the comparator.

Parameters

baseAddr	- cmp base pointer

Returns

- signal source CMP_DAC CMP_MUX Implements : CMP_HAL_GetNegativePortInput_Activity

3.12.5.23 CMP_HAL_GetOffset()

```
static cmp_offset_t CMP_HAL_GetOffset (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return the current offset level.

Parameters

baseAddr	- cmp base pointer

Returns

- offset level CMP_LEVEL_OFFSET_0 CMP_LEVEL_OFFSET_1 Implements : CMP_HAL_GetOffset_↩
Activity

3.12.5.24 CMP_HAL_GetOutputEvent()

```
static cmp_output_trigger_t CMP_HAL_GetOutputEvent (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return type of event occurred at the comparator output.

Parameters

baseAddr	- cmp base pointer

Returns

- comparator output flags CMP_NO_EVENT CMP_FALLING_EDGE CMP_RISING_EDGE CMP_BOTH_E↩
DGES Implements : CMP_HAL_GetOutputEvent_Activity

3.12.5.25 CMP_HAL_GetOutputFallingFlag()

```
static bool CMP_HAL_GetOutputFallingFlag (
            const CMP_Type * baseAddr )  [inline], [static]
```

Verify if a falling-edge occurred on COUT.

Parameters

baseAddr	cmp base pointer

Returns

- Falling edge flag state true - falling-edge event occurred on COUT false - falling-edge event doesn't occurred on COUT Implements : CMP_HAL_GetOutputFallingFlag_Activity

3.12.5.26 CMP_HAL_GetOutputInterruptTrigger()

```
static cmp_output_trigger_t CMP_HAL_GetOutputInterruptTrigger (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return the comparator output interrupts source configuration(none, rising edge, falling edge or both edges)

Parameters

baseAddr	- cmp base pointer

Returns

- comparator output interrupts configuration CMP_NO_EVENT CMP_FALLING_EDGE CMP_RISING_EDGE CMP_BOTH_EDGES Implements : CMP_HAL_GetOutputInterruptTrigger_Activity

3.12.5.27 CMP_HAL_GetOutputPinState()

```
static cmp_output_enable_t CMP_HAL_GetOutputPinState (
            const CMP_Type * baseAddr )  [inline], [static]
```

Verify if the comparator output state(available/not available in a packaged pin)

Parameters

baseAddr	- cmp base pointer

Returns

- comparator output state CMP_UNAVAILABLE CMP_AVAILABLE Implements : CMP_HAL_GetOutputPin↩State_Activity

3.12.5.28 CMP_HAL_GetOutputRisingFlag()

```
static bool CMP_HAL_GetOutputRisingFlag (
            const CMP_Type * baseAddr )  [inline], [static]
```

Verify if a rising edge occurred on COUT.

Parameters

baseAddr	- cmp base pointer

Returns

- rising-edge flag state true - rising-edge event occurred on COUT false - rising-edge event doesn't occurred on COUT Implements : CMP_HAL_GetOutputRisingFlag_Activity

3.12.5.29 CMP_HAL_GetPlusMUXControl()

```
static cmp_ch_number_t CMP_HAL_GetPlusMUXControl (
            const CMP_Type * baseAddr )  [inline], [static]
```

Determine which input is selected for the plus mux.

Parameters

baseAddr	- cmp base pointer

Returns

- channel for the plus mux Implements : CMP_HAL_GetPlusMUXControl_Activity

3.12.5.30 CMP_HAL_GetPositivePortInput()

```
static cmp_port_mux_t CMP_HAL_GetPositivePortInput (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return the current source for positive port of the comparator.

Parameters

baseAddr	- cmp base pointer

Returns

- signal source CMP_DAC CMP_MUX Implements : CMP_HAL_GetPositivePortInput_Activity

3.12.5.31 CMP_HAL_GetPowerMode()

```
static cmp_power_mode_t CMP_HAL_GetPowerMode (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return the current power mode.

Parameters

baseAddr	- cmp base pointer

Returns

- current power mode CMP_LOW_SPEED CMP_HIGH_SPEED Implements : CMP_HAL_GetPowerMode↩ _Activity

3.12.5.32 CMP_HAL_GetRoundRobinChannels()

```
static cmp_ch_list_t CMP_HAL_GetRoundRobinChannels (
            const CMP_Type * baseAddr ) [inline], [static]
```

Return which channels are used for round-robin checker.

Parameters

baseAddr	- cmp base pointer

Returns

- channels states, one bite for each channel state |---------|---------|----|---------|---------| |CH7_state|CH6_↩ state|.....|CH1_state|CH0_state| |---------|---------|----|---------|---------| Implements : CMP_HAL_GetRound↩ RobinChannels_Activity

3.12.5.33 CMP_HAL_GetRoundRobinInterruptState()

```
static bool CMP_HAL_GetRoundRobinInterruptState (
            const CMP_Type * baseAddr ) [inline], [static]
```

Verify if the round robin interrupt is enabled.

Parameters

baseAddr	- cmp base pointer

Returns

- round-robin interrupt state true - round robin interrupt is enabled false - round robin interrupt is disabled Implements : CMP_HAL_GetRoundRobinInterruptState_Activity

3.12.5.34 CMP_HAL_GetRoundRobinSamplesNumber()

```
static uint8_t CMP_HAL_GetRoundRobinSamplesNumber (
            const CMP_Type * baseAddr ) [inline], [static]
```

Return how many round-robin clock cycles takes sampling.

Parameters

baseAddr	- cmp base pointer

Returns

- number of sample clocks Implements : CMP_HAL_GetRoundRobinSamplesNumber_Activity

3.12.5.35 CMP_HAL_GetRoundRobinState()

```
static bool CMP_HAL_GetRoundRobinState (
            const CMP_Type * baseAddr )  [inline], [static]
```

Verify if the round robin operation is enabled.

Parameters

baseAddr	-cmp base pointer

Returns

- round-robin operation state true - round robin operation is enabled false - round robin operation is disabled Implements : CMP_HAL_GetRoundRobinState_Activity

3.12.5.36 CMP_HAL_GetSamplingState()

```
static bool CMP_HAL_GetSamplingState (
            const CMP_Type * baseAddr )  [inline], [static]
```

Verify if the sampling mode is selected.

Parameters

baseAddr	- cmp base pointer

Returns

- sampling mode state true - sampling mode is used false - sampling mode isn't used Implements : CMP_↩
HAL_GetSamplingState_Activity

3.12.5.37 CMP_HAL_GetVoltage()

```
static uint8_t CMP_HAL_GetVoltage (
            const CMP_Type * baseAddr )  [inline], [static]
```

Return the current output voltage level(0-255)

Parameters

baseAddr	- cmp base pointer

Returns

- voltage level Implements : CMP_HAL_GetVoltage_Activity

3.12.5.38 CMP_HAL_GetVoltageReference()

```
static cmp_voltage_reference_t CMP_HAL_GetVoltageReference (
             const CMP_Type * baseAddr )  [inline], [static]
```

Return the current voltage reference.

Parameters

baseAddr	- cmp base pointer

Returns

- voltage referece CMP_VIN1 CMP_VIN2 Implements : CMP_HAL_GetVoltageReference_Activity

3.12.5.39 CMP_HAL_GetWindowingModeState()

```
static bool CMP_HAL_GetWindowingModeState (
             const CMP_Type * baseAddr )  [inline], [static]
```

Verify if the windowing mode is selected

Parameters

baseAddr	- cmp base pointer

Returns

- windowing mode state true - windowing mode is used false - windowing mode isn't used Implements : CM↩
P_HAL_GetWindowingModeState_Activity

3.12.5.40 CMP_HAL_Init()

```
void CMP_HAL_Init (
             CMP_Type * baseAddr )
```

Initializes the comparator registers with reset values.

Parameters

baseAddr	- cmp base pointer

Returns

- void

3.12.5.41 CMP_HAL_SetAnalogComparatorState()

```
static void CMP_HAL_SetAnalogComparatorState (
            CMP_Type * baseAddr,
            bool to_set ) [inline], [static]
```

Set the analog comparator module state.

Parameters

baseAddr	- cmp base pointer
to_set	- analog comparator module state true - module is enabled false - module is disabled Implements : CMP_HAL_SetAnalogComparatorState_Activity

3.12.5.42 CMP_HAL_SetComparatorOutputSource()

```
static void CMP_HAL_SetComparatorOutputSource (
            CMP_Type * baseAddr,
            cmp_output_select_t to_set ) [inline], [static]
```

Select the comparator output signal source.

Parameters

baseAddr	- cmp base pointer
to_set	- comparator output signal source CMP_COUT CMP_COUTA

Returns

void Implements : CMP_HAL_SetComparatorOutputSource_Activity

3.12.5.43 CMP_HAL_SetDACOutputState()

```
static void CMP_HAL_SetDACOutputState (
            CMP_Type * baseAddr,
            bool to_set ) [inline], [static]
```

Set if the DAC output is enabled to go outside of this block.

Parameters

baseAddr	- cmp base pointer
to_set	- DAC output state true - DAC output go outside of DAC block(to packaged pin) false - DAC output doesn't go outside of DAC block(to packaged pin)

Returns

- void Implements : CMP_HAL_SetDACOutputState_Activity

3.12.5.44 CMP_HAL_SetDACState()

```
static void CMP_HAL_SetDACState (
            CMP_Type * baseAddr,
            bool to_set ) [inline], [static]
```

Set the DAC state (enabled/disabled)

Parameters

baseAddr	- cmp base pointer
to_set	- DAC state true - DAC is enabled false - DAC is disabled

Returns

- void Implements : CMP_HAL_SetDACState_Activity

3.12.5.45 CMP_HAL_SetDMATriggerState()

```
static void CMP_HAL_SetDMATriggerState (
            CMP_Type * baseAddr,
            bool to_set ) [inline], [static]
```

Configure the DMA transfer trigger.

Parameters

baseAddr	- cmp base pointer
to_set	- DMA transfer trigger state true - DMA trigger is enabled false - DAM trigger is disabled

Returns

- void Implements : CMP_HAL_SetDMATriggerState_Activity

3.12.5.46 CMP_HAL_SetFilterSampleCount()

```
static void CMP_HAL_SetFilterSampleCount (
            CMP_Type * baseAddr,
            uint8_t to_set ) [inline], [static]
```

Set the number of consecutive samples that must agree prior to the comparator output filter accepting a new output state.

Parameters

baseAddr	- cmp base pointer
to_set	- filter sample count(min value 0, max value 7)

Returns

- void Implements : CMP_HAL_SetFilterSampleCount_Activity

3.12.5.47 CMP_HAL_SetFilterSamplePeriod()

```
static void CMP_HAL_SetFilterSamplePeriod (
             CMP_Type * baseAddr,
             uint8_t to_set )  [inline], [static]
```

Set the filter sample period(clock cycles)

Parameters

baseAddr	-cmp base pointer
to_set	- number of bus cycles

Returns

- void Implements : CMP_HAL_SetFilterSamplePeriod_Activity

3.12.5.48 CMP_HAL_SetFixedChannel()

```
static void CMP_HAL_SetFixedChannel (
             CMP_Type * baseAddr,
             cmp_ch_number_t to_set )  [inline], [static]
```

Set which channel is used as the fixed reference input for the fixed mux port.

Parameters

baseAddr	- cmp base pointer
to_set	- fixed channel

Returns

- void Implements : CMP_HAL_SetFixedChannel_Activity

3.12.5.49 CMP_HAL_SetFixedPort()

```
static void CMP_HAL_SetFixedPort (
             CMP_Type * baseAddr,
             cmp_fixed_port_t to_set )  [inline], [static]
```

Set the fixed port for round-robin operation.

Parameters

baseAddr	- cmp base pointer
to_set	- fixed port CMP_PLUS_FIXED CMP_MINUS_FIXED

Returns

- void Implements : CMP_HAL_SetFixedPort_Activity

3.12.5.50 CMP_HAL_SetFunctionalMode()

```
void CMP_HAL_SetFunctionalMode (
            CMP_Type * baseAddr,
            cmp_mode_t mode,
            uint8_t filter_sample_count,
            uint8_t filter_sample_period )
```

Sets the comparator functional mode (mode, filter count, filter period)

Parameters

baseAddr	- cmp base pointer
mode	- functional mode CMP_DISABLED CMP_CONTINUOUS CMP_SAMPLED_NONFILTRED_INT_CLK CMP_SAMPLED_NONFILTRED_EXT_CLK CMP_SAMPLED_FILTRED_INT_CLK CMP_SAMPLED_FILTRED_EXT_CLK CMP_WINDOWED CMP_WINDOWED_RESAMPLED CMP_WINDOWED_FILTRED
filter_sample_count	- number of consecutive samples that must agree prior to the comparator ouput filter accepting a new output state
filter_sample_period	- sampling period

Returns

-void

3.12.5.51 CMP_HAL_SetHysteresis()

```
static void CMP_HAL_SetHysteresis (
            CMP_Type * baseAddr,
            cmp_hysteresis_t to_set )  [inline], [static]
```

Set the hysteresis level.

Parameters

baseAddr	- cmp base pointer
to_set	- hysteresis level CMP_LEVEL_HYS_0 CMP_LEVEL_HYS_1 CMP_LEVEL_HYS_2 CMP_LEVEL_HYS_3

Returns

- void Implements : CMP_HAL_SetHysteresis_Activity

3.12.5.52 CMP_HAL_SetInitDelay()

```
static void CMP_HAL_SetInitDelay (
            CMP_Type * baseAddr,
            uint8_t to_set )  [inline], [static]
```

Set the comparator and DAC initialization delay.

Parameters

baseAddr	- cmp base pointer
to_set	- delay (min value 0, max value 63)

Returns

- void Implements : CMP_HAL_SetInitDelay_Activity

3.12.5.53 CMP_HAL_SetInverterState()

```
static void CMP_HAL_SetInverterState (
             CMP_Type * baseAddr,
             cmp_inverter_t to_set )  [inline], [static]
```

Configure the comparator output inverter mode.

Parameters

baseAddr	- cmp base pointer
to_set	- comparator output inverter mode CMP_NORMAL CMP_INVERT

Returns

- void Implements : CMP_HAL_SetInverterState_Activity

3.12.5.54 CMP_HAL_SetMinusMUXControl()

```
static void CMP_HAL_SetMinusMUXControl (
             CMP_Type * baseAddr,
             cmp_ch_number_t to_set )  [inline], [static]
```

Select input for the minus mux.

Parameters

baseAddr	- cmp base pointer
to_set	- channel for the minus mux

Returns

- void Implements : CMP_HAL_SetMinusMUXControl_Activity

3.12.5.55 CMP_HAL_SetNegativePortInput()

```
static void CMP_HAL_SetNegativePortInput (
             CMP_Type * baseAddr,
             cmp_port_mux_t to_set )  [inline], [static]
```

Set the source for negative port of the comparator.

Parameters

baseAddr	- cmp base pointer
to_set	- signal source CMP_DAC CMP_MUX

Returns

- void Implements : CMP_HAL_SetNegativePortInput_Activity

3.12.5.56 CMP_HAL_SetOffset()

```
static void CMP_HAL_SetOffset (
            CMP_Type * baseAddr,
            cmp_offset_t to_set )  [inline], [static]
```

Set the offset level.

Parameters

baseAddr	- cmp base pointer
to_set	- offset level CMP_LEVEL_OFFSET_0 CMP_LEVEL_OFFSET_1

Returns

- void Implements : CMP_HAL_SetOffset_Activity

3.12.5.57 CMP_HAL_SetOutputInterruptTrigger()

```
static void CMP_HAL_SetOutputInterruptTrigger (
            CMP_Type * baseAddr,
            cmp_output_trigger_t to_set )  [inline], [static]
```

Set the comparator output interrupts source configuration(none, rising edge, falling edge or both edges)

Parameters

baseAddr	- cmp base pointer
to_set	- comparator output interrupts configuration CMP_NO_EVENT CMP_FALLING_EDGE CMP_RISING_EDGE CMP_BOTH_EDGES

Returns

- void Implements : CMP_HAL_SetOutputInterruptTrigger_Activity

3.12.5.58 CMP_HAL_SetOutputPinState()

```
static void CMP_HAL_SetOutputPinState (
            CMP_Type * baseAddr,
            cmp_output_enable_t to_set )  [inline], [static]
```

Set the comparator output pin state(available/not available in a packaged pin)

Parameters

baseAddr	- cmp base pointer
to_set	- comparator output state CMP_UNAVAILABLE CMP_AVAILABLE

Returns

- void Implements : CMP_HAL_SetOutputPinState_Activity

3.12.5.59 CMP_HAL_SetPlusMuxControl()

```
static void CMP_HAL_SetPlusMuxControl (
            CMP_Type * baseAddr,
            cmp_ch_number_t to_set ) [inline], [static]
```

Select input for the plus mux.

Parameters

baseAddr	cmp base pointer
to_set	- channel for the plus mux

Returns

- void Implements : CMP_HAL_SetPlusMuxControl_Activity

3.12.5.60 CMP_HAL_SetPositivePortInput()

```
static void CMP_HAL_SetPositivePortInput (
            CMP_Type * baseAddr,
            cmp_port_mux_t to_set ) [inline], [static]
```

Set the source for positive port of the comparator.

Parameters

baseAddr	cmp base pointer
to_set	- signal source CMP_DAC CMP_MUX

Returns

- void Implements : CMP_HAL_SetPositivePortInput_Activity

3.12.5.61 CMP_HAL_SetPowerMode()

```
static void CMP_HAL_SetPowerMode (
            CMP_Type * baseAddr,
            cmp_power_mode_t to_set ) [inline], [static]
```

Set the power mode.

Parameters

baseAddr	- cmp base pointer
to_set	- power mode CMP_LOW_SPEED CMP_HIGH_SPEED Implements : CMP_HAL_SetPowerMode_Activity

3.12.5.62 CMP_HAL_SetPresetState()

```
static void CMP_HAL_SetPresetState (
            CMP_Type * baseAddr,
            cmp_ch_list_t to_set )  [inline], [static]
```

Defines the pre-set state of input channels.

Parameters

baseAddr	cmp base pointer
to_set	- state

Returns

void Implements : CMP_HAL_SetPresetState_Activity

3.12.5.63 CMP_HAL_SetRoundRobinChannels()

```
static void CMP_HAL_SetRoundRobinChannels (
            CMP_Type * baseAddr,
            cmp_ch_list_t to_set )  [inline], [static]
```

Set which channels are use for round-robin checker.

Parameters

baseAddr	- cmp base pointer
to_set	- channels states, one bite for each channel state |---------|---------|----|---------|---------| |CH7_state|CH6_state|.....|CH1_state|CH0_state| |---------|---------|----|---------|---------|

Returns

- void Implements : CMP_HAL_SetRoundRobinChannels_Activity

3.12.5.64 CMP_HAL_SetRoundRobinInterruptState()

```
static void CMP_HAL_SetRoundRobinInterruptState (
            CMP_Type * baseAddr,
            bool to_set )  [inline], [static]
```

Set the round robin interrupt state.

Parameters

baseAddr	- cmp base pointer
to_set	- round robin interrupt state true - round robin interrupt is enabled false - round robin interrupt is disabled

Returns

- void Implements : CMP_HAL_SetRoundRobinInterruptState_Activity

3.12.5.65 CMP_HAL_SetRoundRobinSamplesNumber()

```
static void CMP_HAL_SetRoundRobinSamplesNumber (
            CMP_Type * baseAddr,
            uint8_t to_set )  [inline], [static]
```

Set how many round-robin clock cycles takes sampling.

Parameters

baseAddr	- cmp base pointer
to_set	- number of sample clocks(min value 0, max value 3)

Returns

- void Implements : CMP_HAL_SetRoundRobinSamplesNumber_Activity

3.12.5.66 CMP_HAL_SetRoundRobinState()

```
static void CMP_HAL_SetRoundRobinState (
            CMP_Type * baseAddr,
            bool to_set )  [inline], [static]
```

Set the round robin operation state.

Parameters

baseAddr	cmp base pointer
to_set	- round robin operation state true - round robin operation is enabled false - round robin operation is disabled

Returns

- void Implements : CMP_HAL_SetRoundRobinState_Activity

3.12.5.67 CMP_HAL_SetSamplingState()

```
static void CMP_HAL_SetSamplingState (
            CMP_Type * baseAddr,
            bool to_set )  [inline], [static]
```

Set the sampling mode state.

Parameters

baseAddr	- cmp the base pointer
to_set	- sampling mode state true - sampling mode is used false - sampling mode isn't used

Returns

- void Implements : CMP_HAL_SetSamplingState_Activity

3.12.5.68 CMP_HAL_SetVoltage()

```
static void CMP_HAL_SetVoltage (
             CMP_Type * baseAddr,
             uint8_t to_set )  [inline], [static]
```

Set the output voltage level.

Parameters

baseAddr	- cmp base pointer
to_set	- voltage level

Returns

- void Implements : CMP_HAL_SetVoltage_Activity

3.12.5.69 CMP_HAL_SetVoltageReference()

```
static void CMP_HAL_SetVoltageReference (
             CMP_Type * baseAddr,
             cmp_voltage_reference_t to_set )  [inline], [static]
```

Set the voltage reference.

Parameters

baseAddr	- cmp base pointer
to_set	- voltage reference CMP_VIN1 CMP_VIN2

Returns

- void Implements : CMP_HAL_SetVoltageReference_Activity

3.12.5.70 CMP_HAL_SetWindowingModeState()

```
static void CMP_HAL_SetWindowingModeState (
             CMP_Type * baseAddr,
             bool to_set )  [inline], [static]
```

Set the windowing mode state.

Parameters

baseAddr	- cmp base pointer
to_set	- windowing mode state; true - windowing mode is used false - windowing mode isn't used

Returns

void Implements : CMP_HAL_SetWindowingModeState_Activity

3.13 Controller Area Network with Flexible Data Rate (FlexCAN)

3.13.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Driver for the FlexCAN module of S32 SDK devices.

Hardware background

The FlexCAN module is a communication controller implementing the CAN protocol according to the ISO 11898-1 standard and CAN 2.0 B protocol specifications. The FlexCAN module is a full implementation of the CAN protocol specification, the CAN with Flexible Data rate (CAN FD) protocol and the CAN 2.0 version B protocol, which supports both standard and extended message frames and long payloads up to 64 bytes transferred at faster rates up to 8 Mbps. The message buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The FlexCAN module includes these distinctive features:

- Full implementation of the CAN with Flexible Data Rate (CAN FD) protocol specification and CAN protocol specification, Version 2.0 B
  - Standard data frames
  - Extended data frames
  - Zero to sixty four bytes data length
  - Programmable bit rate (see the chip-specific FlexCAN information for the specific maximum rate configuration)
  - Content-related addressing
- Compliant with the ISO 11898-1 standard
- Flexible mailboxes configurable to store 0 to 8, 16, 32 or 64 bytes data length
- Each mailbox configurable as receive or transmit, all supporting standard and extended messages
- Individual Rx Mask registers per mailbox
- Full-featured Rx FIFO with storage capacity for up to six frames and automatic internal pointer handling with DMA support
- Transmission abort capability
- Flexible message buffers (MBs), totaling 32 message buffers of 8 bytes data length each, configurable as Rx or Tx
- Programmable clock source to the CAN Protocol Interface, either peripheral clock or oscillator clock
- RAM not used by reception or transmission structures can be used as general purpose RAM space
- Listen-Only mode capability
- Programmable Loop-Back mode supporting self-test operation
- Maskable interrupts
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes or matching with received frames (Pretended Networking)
- Transceiver Delay Compensation feature when transmitting CAN FD messages at faster data rates
- Remote request frames may be handled automatically or by software
- CAN bit time settings and configuration bits can only be written in Freeze mode

- SYNCH bit available in Error in Status 1 register to inform that the module is synchronous with CAN bus
- CRC status for transmitted message
- Rx FIFO Global Mask register
- Selectable priority between mailboxes and Rx FIFO during matching process
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 128 extended, 256 standard, or 512 partial (8 bit) IDs, with up to 32 individual masking capability
- 100% backward compatibility with previous FlexCAN version
- Supports Pretended Networking functionality in low power: Stop mode

Modules

- FlexCAN Driver
- FlexCAN HAL

3.14 Cryptographic Services Engine (CSEc)

3.14.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Drivers for the Cryptographic Services Engine (CSEc) module of S32 SDK devices.

The FTFC module has added features to comply with the SHE specification. By using an embedded processor, firmware and hardware assisted AES-128 sub-block, the FTFC macro enables encryption, decryption and message generation and authentication algorithms for secure messaging applications. Additionally a TRNG and Miyaguchi-Prenell compression sub-blocks enables true random number generation (entropy generator for PRNG in AES sub-block).

Hardware background

Features of the CSEc module include:

- Secure cryptographic key storage (ranging from 3 to 21 user keys)
- AES-128 encryption and decryption
- AES-128 CMAC (Cipher-based Message Authentication Code) calculation and authentication
- ECB (Electronic Cypher Book) Mode - encryption and decryption
- CBC (Cipher Block Chaining) Mode - encryption and decryption
- True and Pseudo random number generation
- Miyaguchi-Prenell compression function
- Secure Boot Mode (user configurable)
  - Sequential Boot Mode
  - Parallel Boot Mode
  - Strict Sequential Boot Mode (unchangeable once set)

Modules

- CSEc Driver

  Cryptographic Services Engine Peripheral Driver.
- CSEc HAL

3.15 Cyclic Redundancy Check (CRC)

3.15.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Drivers for the Cyclic Redundancy Check (CRC) module of S32K devices.

The cyclic redundancy check (CRC) module generates 16/32-bit CRC code for error detection.
The CRC module provides a programmable polynomial, seed, and other parameters required to implement a 16-bit or 32-bit CRC standard.
The 16/32-bit code is calculated for 32 bits of data at a time.

3.15.2 CRC Driver Initialization

To initialize the CRC module, call CRC_DRV_Init() function and pass the user configuration data structure to it.

This is example code to configure the CRC driver:

```
#define INST_CRC1 (0U)

/* CRC-16-CCITT standard configuration as follows */
/* Configuration structure crc1_InitConfig0 */
const crc_user_config_t crc1_InitConfig0 = {
    .crcWidth = CRC_BITS_16,
    .seed = 0xFFFFU,
    .polynomial = 0x1021U,
    .writeTranspose = CRC_TRANSPOSE_NONE,
    .readTranspose = CRC_TRANSPOSE_NONE,
    .complementChecksum = false
};

/* KERMIT standard configuration as follows */
/* Configuration structure crc1_InitConfig1 */
const crc_user_config_t crc1_InitConfig1 = {
    .crcWidth = CRC_BITS_16,
    .seed = 0U,
    .polynomial = 0x1021U,
    .writeTranspose = CRC_TRANSPOSE_BITS,
    .readTranspose = CRC_TRANSPOSE_BITS,
    .complementChecksum = false
};

/* Initializes the CRC */
CRC_DRV_Init(INST_CRC1, &crc1_InitConfig0);
```

3.15.3 CRC Driver Operation

Function CRC_DRV_Configure() shall be used to write user configuration to CRC hardware module before starting operation by calling CRC_DRV_WriteData(). Finally, using CRC_DRV_GetCrcResult() function to get the result of CRC calculation.

This is example code to Configure and get CRC block:

```
#define INST_CRC1 (0U)

uint8_t buffer[] = { 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30 };
uint32_t result;

/* Set the CRC configuration */
CRC_DRV_Configure(INST_CRC1, &crc1_InitConfig0);
/* Write data to the current CRC calculation */
CRC_DRV_WriteData(INST_CRC1, buffer, 10U);
/* Get result of CRC calculation (0x3218U) */
result = CRC_DRV_GetCrcResult(INST_CRC1);

/* Set the other CRC configuration */
CRC_DRV_Configure(INST_CRC1, &crc1_InitConfig1);
/* Write data to the current CRC calculation */
CRC_DRV_WriteData(INST_CRC1, buffer, 10U);
/* Get result of CRC calculation (0x6B28U) */
result = CRC_DRV_GetCrcResult(INST_CRC1);

/* De-init */
CRC_DRV_Deinit(INST_CRC1);
```

Modules

- CRC Driver

    Cyclic Redundancy Check Peripheral Driver.
- CRC HAL

    Cyclic Redundancy Check Hardware Abstraction Layer.

3.16 DMAMUX HAL

3.16.1 Detailed Description

This module covers the functionality of the Direct Memory Access Multiplexer (DMAMUX) peripheral hardware abstraction layer.

DMAMUX HAL provides the API for reading and writing register bit-fields belonging to the DMAMUX module. It also provides an initialization function for bringing the module to the reset state.

The DMAMUX HAL functions are used by the eDMA driver. The main functionality provided by DMAMUX HAL API is mapping the DMA requests from various modules on the chip to eDMA channels.

For higher-level functionality, use the eDMA driver.

DMAMUX HAL function

- void DMAMUX_HAL_Init (DMAMUX_Type ∗base)

  Initializes the DMAMUX module to the reset state.
- static void DMAMUX_HAL_SetChannelCmd (DMAMUX_Type ∗base, uint32_t channel, bool enable)

  Enables/Disables the DMAMUX channel.
- static void DMAMUX_HAL_SetPeriodTriggerCmd (DMAMUX_Type ∗base, uint32_t channel, bool enable)

  Enables/Disables the period trigger.
- static void DMAMUX_HAL_SetChannelSource (DMAMUX_Type ∗base, uint32_t channel, uint8_t source)

  Configures the DMA request for the DMAMUX channel.

3.16.2 Function Documentation

3.16.2.1 DMAMUX_HAL_Init()

```
void DMAMUX_HAL_Init (
             DMAMUX_Type ∗ base )
```

Initializes the DMAMUX module to the reset state.

Initializes the DMAMUX module to the reset state.

Parameters

base	Register base address for DMAMUX module.

3.16.2.2 DMAMUX_HAL_SetChannelCmd()

```
static void DMAMUX_HAL_SetChannelCmd (
             DMAMUX_Type ∗ base,
             uint32_t channel,
             bool enable )  [inline], [static]
```

Enables/Disables the DMAMUX channel.

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

Parameters

base	Register base address for DMAMUX module.
channel	DMAMUX channel number.
enable	Enables (true) or Disables (false) DMAMUX channel. Implements : DMAMUX_HAL_SetChannelCmd_Activity

3.16.2.3 DMAMUX_HAL_SetChannelSource()

```
static void DMAMUX_HAL_SetChannelSource (
             DMAMUX_Type * base,
             uint32_t channel,
             uint8_t source )  [inline], [static]
```

Configures the DMA request for the DMAMUX channel.

Selects which DMA source is routed to a DMA channel. The DMA sources are defined in the file <MCU>_↩ Features.h

Parameters

base	Register base address for DMAMUX module.
channel	DMAMUX channel number.
source	DMA request source. Implements : DMAMUX_HAL_SetChannelSource_Activity

3.16.2.4 DMAMUX_HAL_SetPeriodTriggerCmd()

```
static void DMAMUX_HAL_SetPeriodTriggerCmd (
             DMAMUX_Type * base,
             uint32_t channel,
             bool enable )  [inline], [static]
```

Enables/Disables the period trigger.

Parameters

base	Register base address for DMAMUX module.
channel	DMAMUX channel number.
enable	Enables (true) or Disables (false) period trigger. Implements : DMAMUX_HAL_SetPeriodTriggerCmd_Activity

3.17 Direct Memory Access (DMA)

3.17.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Driver for the Enhanced Direct Memory Access (eDMA) module, as well as HAL for the Direct Memory Access Multiplexer (DMAMUX) module of S32 SDK devices.
The direct memory access engine features are used for performing complex data transfers with minimal intervention from the host processor. These sections describe the S32 SDK software modules API that can be used for initializing, configuring and triggering DMA transfers.

Modules

- DMAMUX HAL

  This module covers the functionality of the Direct Memory Access Multiplexer (DMAMUX) peripheral hardware abstraction layer.
- EDMA Driver

  This module covers the functionality of the Enhanced Direct Memory Access (eDMA) peripheral driver.
- EDMA HAL

  This module covers the functionality of the Enhanced Direct Memory Access (eDMA) peripheral hardware abstraction layer.

3.18 EDMA Driver

3.18.1 Detailed Description

This module covers the functionality of the Enhanced Direct Memory Access (eDMA) peripheral driver.

The eDMA driver implements direct memory access support in the S32144K processor; the main usage of this module is to offload the bus read/write accesses from the core to the eDMA engine.

Features

- Memory-to-memory, peripheral-to-memory, memory-to-peripheral transfers
- Simple single-block transfers with minimum configuration
- Loop transfers for complex use-cases (e.g. double buffering)
- Scatter/gather
- Dynamic channel allocation

Functionality

Initialization

In order to use the eDMA driver, the module must be first initialized, using EDMA_DRV_Init() function. Once initialized, it cannot be initialized again until it is de-initialized, using EDMA_DRV_Deinit(). The initialization function does the following operations:

- resets eDMA and DMAMUX modules
- clears the eDMA driver state structure
- sets the arbitration mode and halt settings
- enables error and channel interrupts

Upon module initialization, the application must initialize the channel(s) to be used, using EDMA_DRV_Channel↩ Init() function. This operation means enabling a eDMA channel number (or dynamically allocating one), selecting a source trigger (DMA request multiplexed via DMAMUX) and setting the channel priority. Additionally, a user callback can be installed for each channel, which will be called when the corresponding interrupt is triggered.

Transfer Configuration

After initialization, the transfer control descriptor for the selected channel must be configured before use. Depending on the application use-case, on of the three transfer configuration methods should be called.

Single-block transfer

For the simplest use-case where a contiguous chunk of data must be transferred, the most suitable function is EDMA_DRV_ConfigSingleBlockTransfer(). This takes the source/destination addresses as parameters, as well as transfer type/size and data buffer size, and configures the channel TCD to read/write the data in a single request. The looping and scatter/gather features are not used in this scenario. The driver computes the appropriate offsets for source/destination addresses and set the other TCD fields.

Loop transfer

The eDMA IP on S32K144 supports complex addressing modes. One of the methods to configure complex transfers in multiple requests is using the minor/major loop support. The EDMA_DRV_ConfigLoopTransfer() function sets up the transfer control descriptor for subsequent requests to trigger multiple transfers. The addresses are adjusted after each minor/major loop, according to user setup. This method takes a transfer configuration structure as parameter, with settings for all the fields that control addressing mode (source/destination offsets, minor loop offset, channel linking, minor/major loop count, address last adjustments). It is the responsibility of the application to correctly initialize the configuration structure passed to this function, according to the addressed use-case.

Scatter/gather

The eDMA driver also supports scatter/gather feature, which allows various transfer scenarios. When scatter/gather is enabled, a new TCD structure is automatically loaded in the current channel's TCD registers when a transfer is complete, allowing the application to define multiple different subsequent transfers. The EDMA_DRV_Config←ScatterGatherTransfer() function sets up a list of TCD structures based on the parameters received and configures the eDMA channel for the first transfer; upon completion, the second TCD from the list will be loaded and the channel will be ready to start the new transfer when a new request is received.
The application must allocate memory for the TCD list passed to this function (with an extra 32-bytes buffer, as the TCD structures need to be 32 bytes aligned); nevertheless, the driver will take care of initializing the array of descriptors, based on the other parameters passed. The function also received two lists of scatter/gather configuration structures (for source and destination, respectively), which define the address, length and type for each transfer. Besides these, the other parameters received are the transfer size, the number of bytes to be transferred on each request and the number of TCD structures to be used. This method will initialize all the descriptors according to user input and link them together; the linkage is done by writing the address of the next descriptor in the appropriate field of each one, similar to a linked-list data structure. The first descriptor is also copied to the TCD registers of the selected channel; if no errors are returned, after calling this function the channel is configured for the transfer defined by the first descriptor.

Channel Control

The eDMA driver provides functions that allow the user to start, stop, allocate and release an eDMA channel.
The EDMA_DRV_StartChannel() enables the DMA requests for a channel; this function should be called when the channel is already initialized, as the first request received after the function call will trigger the transfer based on the current values of the channel's TCD registers.
The EDMA_DRV_StopChannel() function disables requests for the selected channel; this function should be called whenever the application needs to ignore DMA requests for a channel. It is automatically called when the channel is released.
The EDMA_DRV_RequestChannel() function selects a channel to be used by application and updates the driver state structure accordingly. Two types of channel allocation are available:

- static: the user passes the channel number as parameter; if the channel is already allocated, the function returns an error;
- dynamic: the driver allocates the first available channel and returns its number (or an error if no channel is availabe).

The EDMA_DRV_ReleaseChannel() function frees the hw and sw resources allocated for that channel; it clears the channel state structure, updates the driver state and disables requests for that channel.

Important Notes

- Before using the eDMA driver the clock for eDMA and DMAMUX modules must be configured
- The driver enables the interrupts for the eDMA module, but any interrupt priority must be done by the application
- The driver assumes there is only one eDMA instance implemented in the SoC

Data Structures

- struct edma_user_config_t

  The user configuration structure for the eDMA driver. More...
- struct edma_chn_state_t

  Data structure for the eDMA channel state. Implements : edma_chn_state_t_Class. More...
- struct edma_channel_config_t

  The user configuration structure for the an eDMA driver channel. More...
- struct edma_scatter_gather_list_t

  Data structure for configuring a discrete memory transfer. Implements : edma_scatter_gather_list_t_Class. More...
- struct edma_state_t

  Runtime state structure for the eDMA driver. More...
- struct edma_loop_transfer_config_t

  eDMA loop transfer configuration. More...
- struct edma_transfer_config_t

  eDMA transfer size configuration. More...
- struct edma_software_tcd_t

  eDMA TCD Implements : edma_software_tcd_t_Class More...

Macros

- #define STCD_SIZE(number) (((number) ∗ 32U) - 1U)

  Macro for the memory size needed for the software TCD.
- #define STCD_ADDR(address) (((uint32_t)address + 31UL) & ∼0x1FUL)
- #define EDMA_ERR_LSB_MASK 1U

  Macro for accessing the least significant bit of the ERR register.

Typedefs

- typedef void(∗ edma_callback_t) (void ∗parameter, edma_chn_status_t status)

  Definition for the eDMA channel callback function.

Enumerations

- enum edma_chn_status_t { EDMA_CHN_NORMAL = 0U, EDMA_CHN_IDLE, EDMA_CHN_ERROR }

  Channel status for eDMA channel.
- enum edma_transfer_type_t { EDMA_TRANSFER_PERIPH2MEM, EDMA_TRANSFER_MEM2PERIPH, E←
DMA_TRANSFER_MEM2MEM }

  A type for the DMA transfer. Implements : edma_transfer_type_t_Class.

Variables

- DMA_Type ∗const g_edmaBase [DMA_INSTANCE_COUNT]

  Array for the eDMA module register base address.
- DMAMUX_Type ∗const g_dmamuxBase [DMAMUX_INSTANCE_COUNT]

  Array for DMAMUX module register base address.
- const IRQn_Type g_edmaIrqId [FEATURE_CHANNEL_INTERRUPT_LINES]

  Array for eDMA channel interrupt vector number.
- const clock_names_t g_edmaClockNames [DMA_INSTANCE_COUNT]

  Array for eDMA clock sources.
- const clock_names_t g_dmamuxClockNames [DMAMUX_INSTANCE_COUNT]
- const IRQn_Type g_edmaErrIrqId [FEATURE_ERROR_INTERRUPT_LINES]

  Array for eDMA module's error interrupt vector number.

eDMA peripheral driver module level functions

- status_t EDMA_DRV_Init (edma_state_t ∗edmaState, const edma_user_config_t ∗userConfig, edma_chn↩ _state_t ∗const chnStateArray[ ], const edma_channel_config_t ∗const chnConfigArray[ ], uint8_t chnCount)

  Initializes the eDMA module.
- status_t EDMA_DRV_Deinit (void)

  De-initializes the eDMA module.

eDMA peripheral driver channel management functions

- status_t EDMA_DRV_ChannelInit (edma_chn_state_t ∗edmaChannelState, const edma_channel_config_t ∗edmaChannelConfig)

  Initializes an eDMA channel.
- status_t EDMA_DRV_ReleaseChannel (uint8_t channel)

  Releases an eDMA channel.

eDMA peripheral driver transfer setup functions

- void EDMA_DRV_PushConfigToReg (uint8_t channel, const edma_transfer_config_t ∗tcd)

  Copies the channel configuration to the TCD registers.
- void EDMA_DRV_PushConfigToSTCD (const edma_transfer_config_t ∗config, edma_software_tcd_t ∗stcd)

  Copies the channel configuration to the software TCD structure.
- status_t EDMA_DRV_ConfigSingleBlockTransfer (uint8_t channel, edma_transfer_type_t type, uint32_t src↩ Addr, uint32_t destAddr, edma_transfer_size_t transferSize, uint32_t dataBufferSize)

  Configures a simple single block data transfer with DMA.
- status_t EDMA_DRV_ConfigLoopTransfer (uint8_t channel, const edma_transfer_config_t ∗transferConfig)

  Configures the DMA transfer in loop mode.
- status_t EDMA_DRV_ConfigScatterGatherTransfer (uint8_t channel, edma_software_tcd_t ∗stcd, edma_↩ transfer_size_t transferSize, uint32_t bytesOnEachRequest, const edma_scatter_gather_list_t ∗srcList, const edma_scatter_gather_list_t ∗destList, uint8_t tcdCount)

  Configures the DMA transfer in a scatter-gather mode.

eDMA Peripheral driver channel operation functions

- status_t EDMA_DRV_StartChannel (uint8_t channel)

  Starts an eDMA channel.
- status_t EDMA_DRV_StopChannel (uint8_t channel)

  Stops the eDMA channel.

eDMA Peripheral callback and interrupt functions

- status_t EDMA_DRV_InstallCallback (uint8_t channel, edma_callback_t callback, void ∗parameter)

  Registers the callback function and the parameter for eDMA channel.

eDMA Peripheral driver miscellaneous functions

- edma_chn_status_t EDMA_DRV_GetChannelStatus (uint8_t channel)

  Gets the eDMA channel status.

3.18.2 Data Structure Documentation

3.18.2.1 struct edma_user_config_t

The user configuration structure for the eDMA driver.

Use an instance of this structure with the EDMA_DRV_Init() function. This allows the user to configure settings of the EDMA peripheral with a single function call. Implements : edma_user_config_t_Class

Data Fields

- edma_arbitration_algorithm_t chnArbitration
- bool notHaltOnError

Field Documentation

3.18.2.1.1 chnArbitration

```
edma_arbitration_algorithm_t chnArbitration
```

eDMA channel arbitration.

3.18.2.1.2 notHaltOnError

```
bool notHaltOnError
```

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

3.18.2.2 struct edma_chn_state_t

Data structure for the eDMA channel state. Implements : edma_chn_state_t_Class.

Data Fields

- uint8_t channel
- edma_callback_t callback
- void ∗ parameter
- volatile edma_chn_status_t status

Field Documentation

3.18.2.2.1 callback

```
edma_callback_t callback
```

Callback function pointer for the eDMA channel. It will be called at the eDMA channel complete and eDMA channel error.

3.18.2.2.2 channel

```
uint8_t channel
```

Virtual channel indicator.

3.18.2.2.3 parameter

```
void* parameter
```

Parameter for the callback function pointer.

3.18.2.2.4 status

```
volatile edma_chn_status_t status
```

eDMA channel status.

3.18.2.3 struct edma_channel_config_t

The user configuration structure for the an eDMA driver channel.

Use an instance of this structure with the EDMA_DRV_ChannelInit() function. This allows the user to configure settings of the EDMA channel with a single function call. Implements : edma_channel_config_t_Class

Data Fields

- edma_channel_priority_t priority
- uint8_t channel
- dma_request_source_t source
- edma_callback_t callback
- void * callbackParam

Field Documentation

3.18.2.3.1 callback

```
edma_callback_t callback
```

Callback that will be registered for this channel

3.18.2.3.2 callbackParam

```
void* callbackParam
```

Parameter passed to the channel callback

3.18.2.3.3 channel

```
uint8_t channel
```

eDMA channel number - use EDMA_ANY_CHANNEL for dynamic allocation

3.18.2.3.4 priority

```
edma_channel_priority_t priority
```

eDMA channel priority - only used when channel arbitration mode is 'Fixed priority'.

3.18.2.3.5 source

```
dma_request_source_t source
```

Selects the source of the DMA request for this channel

3.18.2.4 struct edma_scatter_gather_list_t

Data structure for configuring a discrete memory transfer. Implements : edma_scatter_gather_list_t_Class.

Data Fields

- uint32_t address
- uint32_t length
- edma_transfer_type_t type

Field Documentation

3.18.2.4.1 address

```
uint32_t address
```

Address of buffer.

3.18.2.4.2 length

```
uint32_t length
```

Length of buffer.

3.18.2.4.3 type

```
edma_transfer_type_t type
```

Type of the DMA transfer

3.18.2.5 struct edma_state_t

Runtime state structure for the eDMA driver.

This structure holds data that is used by the eDMA peripheral driver to manage multi eDMA channels. The user passes the memory for this run-time state structure and the eDMA driver populates the members. Implements : edma_state_t_Class

Data Fields

- edma_chn_state_t ∗volatile chn [FEATURE_EDMA_MODULE_CHANNELS]

Field Documentation

3.18.2.5.1 chn

```
edma_chn_state_t* volatile chn[FEATURE_EDMA_MODULE_CHANNELS]
```

Pointer array storing channel state.

3.18.2.6 struct edma_loop_transfer_config_t

eDMA loop transfer configuration.

This structure configures the basic minor/major loop attributes. Implements : edma_loop_transfer_config_t_Class

Data Fields

- uint32_t majorLoopIterationCount
- bool srcOffsetEnable
- bool dstOffsetEnable
- int32_t minorLoopOffset
- bool minorLoopChnLinkEnable
- uint8_t minorLoopChnLinkNumber
- bool majorLoopChnLinkEnable
- uint8_t majorLoopChnLinkNumber

Field Documentation

3.18.2.6.1 dstOffsetEnable

```
bool dstOffsetEnable
```

Selects whether the minor loop offset is applied to the destination address upon minor loop completion.

3.18.2.6.2 majorLoopChnLinkEnable

```
bool majorLoopChnLinkEnable
```

Enables channel-to-channel linking on major loop complete.

3.18.2.6.3 majorLoopChnLinkNumber

```
uint8_t majorLoopChnLinkNumber
```

The number of the next channel to be started by DMA engine when major loop completes.

3.18.2.6.4 majorLoopIterationCount

```
uint32_t majorLoopIterationCount
```

Number of major loop iterations.

3.18.2.6.5 minorLoopChnLinkEnable

```
bool minorLoopChnLinkEnable
```

Enables channel-to-channel linking on minor loop complete.

3.18.2.6.6 minorLoopChnLinkNumber

```
uint8_t minorLoopChnLinkNumber
```

The number of the next channel to be started by DMA engine when minor loop completes.

3.18.2.6.7 minorLoopOffset

```
int32_t minorLoopOffset
```

Sign-extended offset applied to the source or destination address to form the next-state value after the minor loop completes.

3.18.2.6.8 srcOffsetEnable

```
bool srcOffsetEnable
```

Selects whether the minor loop offset is applied to the source address upon minor loop completion.

3.18.2.7 struct edma_transfer_config_t

eDMA transfer size configuration.

This structure configures the basic source/destination transfer attribute. Implements : edma_transfer_config_t_↩
Class

Data Fields

- uint32_t srcAddr
- uint32_t destAddr
- edma_transfer_size_t srcTransferSize
- edma_transfer_size_t destTransferSize
- int16_t srcOffset
- int16_t destOffset
- int32_t srcLastAddrAdjust
- int32_t destLastAddrAdjust
- edma_modulo_t srcModulo
- edma_modulo_t destModulo
- uint32_t minorByteTransferCount
- bool scatterGatherEnable
- uint32_t scatterGatherNextDescAddr
- bool interruptEnable
- edma_loop_transfer_config_t ∗ loopTransferConfig

Field Documentation

3.18.2.7.1 destAddr

```
uint32_t destAddr
```

Memory address pointing to the destination data.

3.18.2.7.2 destLastAddrAdjust

```
int32_t destLastAddrAdjust
```

Last destination address adjustment. Note here it is only valid when scatter/gather feature is not enabled.

3.18.2.7.3 destModulo

```
edma_modulo_t destModulo
```

Destination address modulo.

3.18.2.7.4 destOffset

```
int16_t destOffset
```

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

3.18.2.7.5 destTransferSize

```
edma_transfer_size_t destTransferSize
```

Destination data transfer size.

3.18.2.7.6 interruptEnable

```
bool interruptEnable
```

Enable the interrupt request when the major loop count completes

3.18.2.7.7 loopTransferConfig

```
edma_loop_transfer_config_t* loopTransferConfig
```

Pointer to loop transfer configuration structure (defines minor/major loop attributes) Note: this field is only used when minor loop mapping is enabled from DMA configuration.

3.18.2.7.8 minorByteTransferCount

```
uint32_t minorByteTransferCount
```

Number of bytes to be transferred in each service request of the channel.

3.18.2.7.9 scatterGatherEnable

```
bool scatterGatherEnable
```

Enable scatter gather feature.

3.18.2.7.10 scatterGatherNextDescAddr

```
uint32_t scatterGatherNextDescAddr
```

The address of the next descriptor to be used, when scatter/gather feature is enabled. Note: this value is not used when scatter/gather feature is disabled.

3.18.2.7.11 srcAddr

```
uint32_t srcAddr
```

Memory address pointing to the source data.

3.18.2.7.12 srcLastAddrAdjust

```
int32_t srcLastAddrAdjust
```

Last source address adjustment.

3.18.2.7.13 srcModulo

```
edma_modulo_t srcModulo
```

Source address modulo.

3.18.2.7.14 srcOffset

```
int16_t srcOffset
```

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

3.18.2.7.15 srcTransferSize

```
edma_transfer_size_t srcTransferSize
```

Source data transfer size.

3.18.2.8 struct edma_software_tcd_t

eDMA TCD Implements : edma_software_tcd_t_Class

Data Fields

- uint32_t SADDR
- int16_t SOFF
- uint16_t ATTR
- uint32_t NBYTES
- int32_t SLAST
- uint32_t DADDR
- int16_t DOFF
- uint16_t CITER
- int32_t DLAST_SGA
- uint16_t CSR
- uint16_t BITER

Field Documentation

3.18.2.8.1 ATTR

```
uint16_t ATTR
```

3.18.2.8.2 BITER

```
uint16_t BITER
```

3.18.2.8.3 CITER

```
uint16_t CITER
```

3.18.2.8.4 CSR

```
uint16_t CSR
```

3.18.2.8.5 DADDR

```
uint32_t DADDR
```

3.18.2.8.6 DLAST_SGA

```
int32_t DLAST_SGA
```

3.18.2.8.7 DOFF

```
int16_t DOFF
```

3.18.2.8.8 NBYTES

```
uint32_t NBYTES
```

3.18.2.8.9 SADDR

```
uint32_t SADDR
```

3.18.2.8.10 SLAST

```
int32_t SLAST
```

3.18.2.8.11 SOFF

```
int16_t SOFF
```

3.18.3 Macro Definition Documentation

3.18.3.1 EDMA_ERR_LSB_MASK

```
#define EDMA_ERR_LSB_MASK 1U
```

Macro for accessing the least significant bit of the ERR register.

The erroneous channels are retrieved from ERR register by subsequently right shifting all the ERR bits + "AND"-ing the result with this mask.

3.18.3.2 STCD_ADDR

```
#define STCD_ADDR(
              address ) (((uint32_t)address + 31UL) & ~0x1FUL)
```

3.18.3.3 STCD_SIZE

```
#define STCD_SIZE(
              number ) (((number) * 32U) - 1U)
```

Macro for the memory size needed for the software TCD.

Software TCD is aligned to 32 bytes. We don't need a software TCD structure for the first descriptor, since the configuration is pushed directly to registers. To make sure the software TCD can meet the eDMA module requirement regarding alignment, allocate memory for the remaining descriptors with extra 31 bytes.

3.18.4 Typedef Documentation

3.18.4.1 edma_callback_t

```
typedef void(* edma_callback_t) (void *parameter, edma_chn_status_t status)
```

Definition for the eDMA channel callback function.

Prototype for the callback function registered in the eDMA driver. Implements : edma_callback_t_Class

3.18.5 Enumeration Type Documentation

3.18.5.1 edma_chn_status_t

```
enum edma_chn_status_t
```

Channel status for eDMA channel.

A structure describing the eDMA channel status. The user can get the status by callback parameter or by calling EDMA_DRV_getStatus() function. Implements : edma_chn_status_t_Class

Enumerator

EDMA_CHN_NORMAL	eDMA channel is occupied.
EDMA_CHN_IDLE	eDMA channel is idle.
EDMA_CHN_ERROR	An error occurs in the eDMA channel.

3.18.5.2 edma_transfer_type_t

```
enum edma_transfer_type_t
```

A type for the DMA transfer. Implements : edma_transfer_type_t_Class.

Enumerator

EDMA_TRANSFER_PERIPH2MEM	Transfer from peripheral to memory
EDMA_TRANSFER_MEM2PERIPH	Transfer from memory to peripheral
EDMA_TRANSFER_MEM2MEM	Transfer from memory to memory

3.18.6 Function Documentation

3.18.6.1 EDMA_DRV_ChannelInit()

```
status_t EDMA_DRV_ChannelInit (
            edma_chn_state_t * edmaChannelState,
            const edma_channel_config_t * edmaChannelConfig )
```

Initializes an eDMA channel.

This function initializes the run-time state structure for a eDMA channel, based on user configuration. It will request the channel, set up the channel priority and install the callback.

Parameters

edmaChannelState	Pointer to the eDMA channel state structure. The user passes the memory for this run-time state structure and the eDMA peripheral driver populates the members. This run-time state structure keeps track of the eDMA channel status. The memory must be kept valid before calling the EDMA_DRV_ReleaseChannel.
edmaChannelConfig	User configuration structure for eDMA channel. The user populates the members of this structure and passes the pointer of this structure into the function.

Returns

STATUS_ERROR or STATUS_SUCCESS.

3.18.6.2 EDMA_DRV_ConfigLoopTransfer()

```
status_t EDMA_DRV_ConfigLoopTransfer (
            uint8_t channel,
            const edma_transfer_config_t * transferConfig )
```

Configures the DMA transfer in loop mode.

This function configures the DMA transfer in a loop chain. The user passes a block of memory into this function that configures the loop transfer properties (minor/major loop count, address offsets, channel linking). The DMA driver copies the configuration to TCD registers, only when the loop properties are set up correctly and minor loop mapping is enabled for the eDMA module.

Parameters

chn	Pointer to the channel state structure.
transferConfig	Pointer to the transfer configuration strucutre; this structure defines fields for setting up the basic transfer and also a pointer to a memory strucure that defines the loop chain properties (minor/major).

Returns

STATUS_ERROR or STATUS_SUCCESS

3.18.6.3 EDMA_DRV_ConfigScatterGatherTransfer()

```
status_t EDMA_DRV_ConfigScatterGatherTransfer (
             uint8_t channel,
             edma_software_tcd_t * stcd,
             edma_transfer_size_t transferSize,
             uint32_t bytesOnEachRequest,
             const edma_scatter_gather_list_t * srcList,
             const edma_scatter_gather_list_t * destList,
             uint8_t tcdCount )
```

Configures the DMA transfer in a scatter-gather mode.

This function configures the descriptors into a single-ended chain. The user passes blocks of memory into this function. The interrupt is triggered only when the last memory block is completed. The memory block information is passed with the edma_scatter_gather_list_t data structure, which can tell the memory address and length. The DMA driver configures the descriptor for each memory block, transfers the descriptor from the first one to the last one, and stops.

Parameters

chn	Pointer to the channel state structure.
stcd	Array of empty software TCD structures. The user must prepare this memory block. We don't need a software TCD structure for the first descriptor, since the configuration is pushed directly to registers.The "stcd" buffer must align with 32 bytes; if not, an error occurs in the eDMA driver. Thus, the required memory size for "stcd" is equal to tcdCount $*$ size_of(edma_software_tcd_t) - 1; the driver will take care of the memory alignment if the provided memory buffer is big enough. For proper allocation of the "stcd" buffer it is recommended to use STCD_SIZE macro.
transferSize	The number of bytes to be transferred on every DMA write/read.
bytesOnEachRequest	Bytes to be transferred in each DMA request.
srcList	Data structure storing the address, length and type of transfer (M->M, M->P, P->M) for the bytes to be transferred for source memory blocks. If the source memory is peripheral, the length is not used.
destList	Data structure storing the address, length and type of transfer (M->M, M->P, P->M) for the bytes to be transferred for destination memory blocks. In the memory-to-memory transfer mode, the user must ensure that the length of the destination scatter gather list is equal to the source scatter gather list. If the
	destination memory is a peripheral register, the length is not used.
tcdCount	The number of TCD memory blocks contained in the scatter gather list.

Returns

STATUS_ERROR or STATUS_SUCCESS

3.18.6.4 EDMA_DRV_ConfigSingleBlockTransfer()

```
status_t EDMA_DRV_ConfigSingleBlockTransfer (
            uint8_t channel,
            edma_transfer_type_t type,
            uint32_t srcAddr,
            uint32_t destAddr,
            edma_transfer_size_t transferSize,
            uint32_t dataBufferSize )
```

Configures a simple single block data transfer with DMA.

This function configures the descriptor for a single block transfer. The function considers contiguous memory blocks, thus it configures the TCD source/destination offset fields to cover the data buffer without gaps, according to "transferSize" parameter (the offset is equal to the number of bytes transferred in a source read/destination write).

NOTE: For memory-to-peripheral or peripheral-to-memory transfers, make sure the transfer size is equal to the data buffer size of the peripheral used, otherwise only truncated chunks of data may be transferred (e.g. for a communication IP with an 8-bit data register the transfer size should be 1B, whereas for a 32-bit data register, the transfer size should be 4B). The rationale of this constraint is that, on the peripheral side, the address offset is set to zero, allowing to read/write data from/to the peripheral in a single source read/destination write operation.

Parameters

chn	Pointer to the channel state structure.
type	Transfer type (M->M, P->M, M->P).
srcAddr	A source register address or a source memory address.
destAddr	A destination register address or a destination memory address.
transferSize	The number of bytes to be transferred on every DMA write/read. Source/Dest share the same write/read size.
dataBufferSize	The total number of bytes to be transferred.

Returns

STATUS_ERROR or STATUS_SUCCESS

3.18.6.5 EDMA_DRV_Deinit()

```
status_t EDMA_DRV_Deinit (
            void )
```

De-initializes the eDMA module.

This function resets the eDMA module to reset state and disables the interrupt to the core.

Returns

STATUS_ERROR or STATUS_SUCCESS.

3.18.6.6 EDMA_DRV_GetChannelStatus()

```
edma_chn_status_t EDMA_DRV_GetChannelStatus (
            uint8_t channel )
```

Gets the eDMA channel status.

Parameters

chn	Channel number.

Returns

Channel status.

3.18.6.7 EDMA_DRV_Init()

```
status_t EDMA_DRV_Init (
            edma_state_t * edmaState,
            const edma_user_config_t * userConfig,
            edma_chn_state_t *const chnStateArray[ ],
            const edma_channel_config_t *const chnConfigArray[ ],
            uint8_t chnCount )
```

Initializes the eDMA module.

This function initializes the run-time state structure to provide the eDMA channel allocation release, protect, and track the state for channels. This function also resets the eDMA modules, initializes the module to user-defined settings and default settings.

Parameters

edmaState	The pointer to the eDMA peripheral driver state structure. The user passes the memory for this run-time state structure and the eDMA peripheral driver populates the members. This run-time state structure keeps track of the eDMA channels status. The memory must be kept valid before calling the EDMA_DRV_DeInit.
userConfig	User configuration structure for eDMA peripheral drivers. The user populates the members of this structure and passes the pointer of this structure into the function.
chnStateArray	Array of pointers to run-time state structures for eDMA channels; will populate the state structures inside the eDMA driver state structure.
chnConfigArray	Array of pointers to channel initialization structures.
chnCount	The number of eDMA channels to be initialized.

Returns

STATUS_ERROR or STATUS_SUCCESS.

3.18.6.8 EDMA_DRV_InstallCallback()

```
status_t EDMA_DRV_InstallCallback (
            uint8_t channel,
            edma_callback_t callback,
            void * parameter )
```

Registers the callback function and the parameter for eDMA channel.

This function registers the callback function and the parameter into the eDMA channel state structure. The callback function is called when the channel is complete or a channel error occurs. The eDMA driver passes the channel status to this callback function to indicate whether it is caused by the channel complete event or the channel error event.

To un-register the callback function, set the callback function to "NULL" and call this function.

Parameters

chn	The pointer to the channel state structure.
callback	The pointer to the callback function.
parameter	The pointer to the callback function's parameter.

Returns

STATUS_ERROR or STATUS_SUCCESS.

3.18.6.9 EDMA_DRV_PushConfigToReg()

```
void EDMA_DRV_PushConfigToReg (
            uint8_t channel,
            const edma_transfer_config_t * tcd )
```

Copies the channel configuration to the TCD registers.

Parameters

chn	Pointer to the channel state structure.
config	Pointer to the channel configuration structure.

3.18.6.10 EDMA_DRV_PushConfigToSTCD()

```
void EDMA_DRV_PushConfigToSTCD (
            const edma_transfer_config_t * config,
            edma_software_tcd_t * stcd )
```

Copies the channel configuration to the software TCD structure.

This function copies the properties from the channel configuration to the software TCD structure; the address of the software TCD can be used to enable scatter/gather operation (pointer to the next TCD).

Parameters

chn	Pointer to the channel state structure.
config	Pointer to the channel configuration structure.
stcd	Pointer to the software TCD structure.

3.18.6.11 EDMA_DRV_ReleaseChannel()

```
status_t EDMA_DRV_ReleaseChannel (
            uint8_t channel )
```

Releases an eDMA channel.

This function stops the eDMA channel and disables the interrupt of this channel. The channel state structure can be released after this function is called.

Parameters

chn	The pointer to the channel state structure.

Returns

STATUS_ERROR or STATUS_SUCCESS.

3.18.6.12 EDMA_DRV_StartChannel()

```
status_t EDMA_DRV_StartChannel (
            uint8_t channel )
```

Starts an eDMA channel.

This function enables the eDMA channel DMA request.

Parameters

chn	Pointer to the channel state structure.

Returns

STATUS_ERROR or STATUS_SUCCESS.

3.18.6.13 EDMA_DRV_StopChannel()

```
status_t EDMA_DRV_StopChannel (
            uint8_t channel )
```

Stops the eDMA channel.

This function disables the eDMA channel DMA request.

Parameters

chn	Pointer to the channel state structure.

Returns

STATUS_ERROR or STATUS_SUCCESS.

3.18.7 Variable Documentation

3.18.7.1 g_dmamuxBase

```
DMAMUX_Type* const g_dmamuxBase[DMAMUX_INSTANCE_COUNT]
```

Array for DMAMUX module register base address.

3.18.7.2 g_dmamuxClockNames

```
const clock_names_t g_dmamuxClockNames[DMAMUX_INSTANCE_COUNT]
```

3.18.7.3 g_edmaBase

```
DMA_Type* const g_edmaBase[DMA_INSTANCE_COUNT]
```

Array for the eDMA module register base address.

3.18.7.4 g_edmaClockNames

```
const clock_names_t g_edmaClockNames[DMA_INSTANCE_COUNT]
```

Array for eDMA clock sources.

Array for eDMA & DMAMUX clock sources.

3.18.7.5 g_edmaErrIrqId

```
const IRQn_Type g_edmaErrIrqId[FEATURE_ERROR_INTERRUPT_LINES]
```

Array for eDMA module's error interrupt vector number.

3.18.7.6 g_edmaIrqId

```
const IRQn_Type g_edmaIrqId[FEATURE_CHANNEL_INTERRUPT_LINES]
```

Array for eDMA channel interrupt vector number.

3.19 EDMA HAL

3.19.1 Detailed Description

This module covers the functionality of the Enhanced Direct Memory Access (eDMA) peripheral hardware abstraction layer.

eDMA HAL provides the API for reading and writing register bit-fields belonging to the eDMA module. It also provides an initialization function for bringing the module to the reset state.

The eDMA HAL functions are used by the eDMA driver, as well as other drivers for peripherals that have DMA features (communication/analogue).

For higher-level functionality, use the eDMA driver.

Enumerations

- enum edma_arbitration_algorithm_t { EDMA_ARBITRATION_FIXED_PRIORITY = 0U, EDMA_ARBITRAT↩ ION_ROUND_ROBIN }

  eDMA channel arbitration algorithm used for selection among channels. Implements : edma_arbitration_algorithm↩ _t_Class
- enum edma_channel_priority_t {
  EDMA_CHN_PRIORITY_0 = 0U, EDMA_CHN_PRIORITY_1 = 1U, EDMA_CHN_PRIORITY_2 = 2U, EDM↩ A_CHN_PRIORITY_3 = 3U,
  EDMA_CHN_PRIORITY_4 = 4U, EDMA_CHN_PRIORITY_5 = 5U, EDMA_CHN_PRIORITY_6 = 6U, EDM↩ A_CHN_PRIORITY_7 = 7U,
  EDMA_CHN_PRIORITY_8 = 8U, EDMA_CHN_PRIORITY_9 = 9U, EDMA_CHN_PRIORITY_10 = 10U, E↩ DMA_CHN_PRIORITY_11 = 11U,
  EDMA_CHN_PRIORITY_12 = 12U, EDMA_CHN_PRIORITY_13 = 13U, EDMA_CHN_PRIORITY_14 = 14U,
  EDMA_CHN_PRIORITY_15 = 15U,
  EDMA_CHN_DEFAULT_PRIORITY = 255U }

  eDMA channel priority setting Implements : edma_channel_priority_t_Class
- enum edma_modulo_t {
  EDMA_MODULO_OFF = 0U, EDMA_MODULO_2B, EDMA_MODULO_4B, EDMA_MODULO_8B,
  EDMA_MODULO_16B, EDMA_MODULO_32B, EDMA_MODULO_64B, EDMA_MODULO_128B,
  EDMA_MODULO_256B, EDMA_MODULO_512B, EDMA_MODULO_1KB, EDMA_MODULO_2KB,
  EDMA_MODULO_4KB, EDMA_MODULO_8KB, EDMA_MODULO_16KB, EDMA_MODULO_32KB,
  EDMA_MODULO_64KB, EDMA_MODULO_128KB, EDMA_MODULO_256KB, EDMA_MODULO_512KB,
  EDMA_MODULO_1MB, EDMA_MODULO_2MB, EDMA_MODULO_4MB, EDMA_MODULO_8MB,
  EDMA_MODULO_16MB, EDMA_MODULO_32MB, EDMA_MODULO_64MB, EDMA_MODULO_128MB,
  EDMA_MODULO_256MB, EDMA_MODULO_512MB, EDMA_MODULO_1GB, EDMA_MODULO_2GB }

  eDMA modulo configuration Implements : edma_modulo_t_Class
- enum edma_transfer_size_t {
  EDMA_TRANSFER_SIZE_1B = 0x0U, EDMA_TRANSFER_SIZE_2B = 0x1U, EDMA_TRANSFER_SIZE_↩ 4B = 0x2U, EDMA_TRANSFER_SIZE_16B = 0x4U,
  EDMA_TRANSFER_SIZE_32B = 0x5U }

  eDMA transfer configuration Implements : edma_transfer_size_t_Class
- enum edma_bandwidth_config_t { EDMA_BANDWIDTH_STALL_NONE = 0U, EDMA_BANDWIDTH_STA↩ LL_4_CYCLES = 2U, EDMA_BANDWIDTH_STALL_8_CYCLES = 3U }

  Bandwidth control configuration Implements : edma_bandwidth_config_t_Class.

eDMA HAL driver module level operation

- void EDMA_HAL_Init (DMA_Type ∗base)
  Initializes eDMA module to known state.
- void EDMA_HAL_CancelTransfer (DMA_Type ∗base)
  Cancels the remaining data transfer.
- void EDMA_HAL_CancelTransferWithError (DMA_Type ∗base)
  Cancels the remaining data transfer and treats it as an error condition.
- static void EDMA_HAL_SetHaltCmd (DMA_Type ∗base, bool halt)
  Halts/Un-halts the DMA Operations.
- static void EDMA_HAL_SetHaltOnErrorCmd (DMA_Type ∗base, bool haltOnError)
  Halts or does not halt the eDMA module when an error occurs.
- static void EDMA_HAL_SetDebugCmd (DMA_Type ∗base, bool enable)
  Enables/Disables the eDMA DEBUG mode.

eDMA HAL error checking

- static bool EDMA_HAL_GetValidErrorNotCleared (const DMA_Type ∗base)
  Checks for valid errors.
- static bool EDMA_HAL_GetTransferCancelledError (const DMA_Type ∗base)
  Checks for cancelled transfers.
- static bool EDMA_HAL_GetChannelPriorityError (const DMA_Type ∗base)
  Checks for channel priority errors.
- static bool EDMA_HAL_GetSourceAddressError (const DMA_Type ∗base)
  Checks for source address errors.
- static bool EDMA_HAL_GetSourceOffsetError (const DMA_Type ∗base)
  Checks for source offset errors.
- static bool EDMA_HAL_GetDestinationAddressError (const DMA_Type ∗base)
  Checks for destination address errors.
- static bool EDMA_HAL_GetDestinationOffsetError (const DMA_Type ∗base)
  Checks for destination offset errors.
- static bool EDMA_HAL_GetMinorMajorLoopConfigError (const DMA_Type ∗base)
  Checks for minor/major loop configuration errors.
- static bool EDMA_HAL_GetScatterGatherError (const DMA_Type ∗base)
  Checks for scatter/gather configuration errors.
- static bool EDMA_HAL_GetSourceBusError (const DMA_Type ∗base)
  Checks for source bus errors.
- static bool EDMA_HAL_GetDestinationBusError (const DMA_Type ∗base)
  Checks for destination bus errors.
- static uint8_t EDMA_HAL_GetChannelWithError (const DMA_Type ∗base)
  Error channel number.

eDMA HAL driver channel priority and arbitration configuration

- static void EDMA_HAL_SetChannelPreemptMode (DMA_Type ∗base, uint32_t channel, bool preemptive, bool preemptible)
  Sets the preemption feature for the eDMA channel.
- static void EDMA_HAL_SetChannelPriority (DMA_Type ∗base, uint32_t channel, edma_channel_priority_t priority)
  Sets the eDMA channel priority.
- static void EDMA_HAL_SetChannelArbitrationMode (DMA_Type ∗base, edma_arbitration_algorithm_↩
t channelArbitration)
  Sets the channel arbitration algorithm.
- static edma_arbitration_algorithm_t EDMA_HAL_GetChannelArbitrationMode (const DMA_Type ∗base)
  Gets the channel arbitration algorithm.

eDMA HAL driver configuration and operation

- static void EDMA_HAL_SetMinorLoopMappingCmd (DMA_Type ∗base, bool enable)
  Enables/Disables the minor loop mapping.
- static void EDMA_HAL_SetContinuousLinkCmd (DMA_Type ∗base, bool continuous)
  Enables or disables the continuous transfer mode.
- void EDMA_HAL_SetErrorIntCmd (DMA_Type ∗base, uint8_t channel, bool enable)
  Enables/Disables the error interrupt for channels.
- static uint32_t EDMA_HAL_GetErrorIntStatusFlag (const DMA_Type ∗base)
  Gets the eDMA error interrupt status.
- static void EDMA_HAL_ClearErrorIntStatusFlag (DMA_Type ∗base, uint8_t channel)
  Clears the error interrupt status for the eDMA channel or channels.
- void EDMA_HAL_SetDmaRequestCmd (DMA_Type ∗base, uint8_t channel, bool enable)
  Enables/Disables the DMA request for the channel or all channels.
- static bool EDMA_HAL_GetDmaRequestStatusFlag (const DMA_Type ∗base, uint32_t channel)
  Gets the eDMA channel DMA request status.
- static void EDMA_HAL_ClearDoneStatusFlag (DMA_Type ∗base, uint8_t channel)
  Clears the done status for a channel or all channels.
- static void EDMA_HAL_TriggerChannelStart (DMA_Type ∗base, uint8_t channel)
  Triggers the eDMA channel.
- static bool EDMA_HAL_GetIntStatusFlag (const DMA_Type ∗base, uint32_t channel)
  Gets the eDMA channel interrupt request status.
- static void EDMA_HAL_ClearIntStatusFlag (DMA_Type ∗base, uint8_t channel)
  Clears the interrupt status for the eDMA channel or all channels.
- static void EDMA_HAL_SetAsyncRequestInStopModeCmd (DMA_Type ∗base, uint32_t channel, bool enable)
  Enables/Disables an asynchronous request in stop mode.

eDMA HAL driver TCD configuration functions

- void EDMA_HAL_TCDClearReg (DMA_Type ∗base, uint32_t channel)
  Clears all registers to 0 for the hardware TCD.
- static void EDMA_HAL_TCDSetSrcAddr (DMA_Type ∗base, uint32_t channel, uint32_t address)
  Configures the source address for the hardware TCD.
- static void EDMA_HAL_TCDSetSrcOffset (DMA_Type ∗base, uint32_t channel, int16_t offset)
  Configures the source address signed offset for the hardware TCD.
- void EDMA_HAL_TCDSetAttribute (DMA_Type ∗base, uint32_t channel, edma_modulo_t srcModulo, edma_modulo_t destModulo, edma_transfer_size_t srcTransferSize, edma_transfer_size_t destTransferSize)
  Configures the transfer attribute for the eDMA channel.
- void EDMA_HAL_TCDSetNbytes (DMA_Type ∗base, uint32_t channel, uint32_t nbytes)
  Configures the nbytes for the eDMA channel.
- uint32_t EDMA_HAL_TCDGetNbytes (const DMA_Type ∗base, uint32_t channel)
  Gets the nbytes configuration data for the TCD.
- static void EDMA_HAL_TCDSetSrcMinorLoopOffsetCmd (DMA_Type ∗base, uint32_t channel, bool enable)
  Enables/disables the source minor loop offset feature for the TCD.
- static void EDMA_HAL_TCDSetDestMinorLoopOffsetCmd (DMA_Type ∗base, uint32_t channel, bool enable)
  Enables/disables the destination minor loop offset feature for the TCD.
- void EDMA_HAL_TCDSetMinorLoopOffset (DMA_Type ∗base, uint32_t channel, int32_t offset)
  Configures the minor loop offset for the TCD.
- static void EDMA_HAL_TCDSetSrcLastAdjust (DMA_Type ∗base, uint32_t channel, int32_t size)
  Configures the last source address adjustment for the TCD.

- static void EDMA_HAL_TCDSetDestAddr (DMA_Type ∗base, uint32_t channel, uint32_t address)

  Configures the destination address for the TCD.
- static void EDMA_HAL_TCDSetDestOffset (DMA_Type ∗base, uint32_t channel, int16_t offset)

  Configures the destination address signed offset for the TCD.
- static void EDMA_HAL_TCDSetDestLastAdjust (DMA_Type ∗base, uint32_t channel, int32_t adjust)

  Configures the last source address adjustment.
- void EDMA_HAL_TCDSetScatterGatherLink (DMA_Type ∗base, uint32_t channel, uint32_t nextTCDAddr)

  Configures the memory address for the next transfer TCD for the TCD.
- static void EDMA_HAL_TCDSetBandwidth (DMA_Type ∗base, uint32_t channel, edma_bandwidth_config_t bandwidth)

  Configures the bandwidth for the TCD.
- static void EDMA_HAL_TCDSetChannelMajorLink (DMA_Type ∗base, uint32_t channel, uint32_t major←LinkChannel, bool enable)

  Configures the major channel link the TCD.
- static void EDMA_HAL_TCDSetScatterGatherCmd (DMA_Type ∗base, uint32_t channel, bool enable)

  Enables/Disables the scatter/gather feature for the TCD.
- static void EDMA_HAL_TCDSetDisableDmaRequestAfterTCDDoneCmd (DMA_Type ∗base, uint32_t channel, bool disable)

  Disables/Enables the DMA request after the major loop completes for the TCD.
- static void EDMA_HAL_TCDSetHalfCompleteIntCmd (DMA_Type ∗base, uint32_t channel, bool enable)

  Enables/Disables the half complete interrupt for the TCD.
- static void EDMA_HAL_TCDSetIntCmd (DMA_Type ∗base, uint32_t channel, bool enable)

  Enables/Disables the interrupt after the major loop completes for the TCD.
- static void EDMA_HAL_TCDTriggerChannelStart (DMA_Type ∗base, uint32_t channel)

  Triggers the start bits for the TCD.
- static bool EDMA_HAL_TCDGetChannelActiveStatus (const DMA_Type ∗base, uint32_t channel)

  Checks whether the channel is running for the TCD.
- void EDMA_HAL_TCDSetChannelMinorLink (DMA_Type ∗base, uint32_t channel, uint32_t linkChannel, bool enable)

  Sets the channel minor link for the TCD.
- void EDMA_HAL_TCDSetMajorCount (DMA_Type ∗base, uint32_t channel, uint32_t count)

  Sets the major iteration count according to minor loop channel link setting.
- uint32_t EDMA_HAL_TCDGetBeginMajorCount (const DMA_Type ∗base, uint32_t channel)

  Returns the begin major iteration count.
- uint32_t EDMA_HAL_TCDGetCurrentMajorCount (const DMA_Type ∗base, uint32_t channel)

  Returns the current major iteration count.
- uint32_t EDMA_HAL_TCDGetFinishedBytes (const DMA_Type ∗base, uint32_t channel)

  Gets the number of bytes already transferred for the TCD.
- uint32_t EDMA_HAL_TCDGetUnfinishedBytes (const DMA_Type ∗base, uint32_t channel)

  Gets the number of bytes haven't transferred for the TCD.
- static bool EDMA_HAL_TCDGetDoneStatusFlag (const DMA_Type ∗base, uint32_t channel)

  Gets the channel done status.

3.19.2 Enumeration Type Documentation

3.19.2.1 edma_arbitration_algorithm_t

```
enum edma_arbitration_algorithm_t
```

eDMA channel arbitration algorithm used for selection among channels. Implements : edma_arbitration_algorithm←_t_Class

Enumerator

EDMA_ARBITRATION_FIXED_PRIORITY	Fixed Priority
EDMA_ARBITRATION_ROUND_ROBIN	Round-Robin arbitration

3.19.2.2 edma_bandwidth_config_t

```
enum edma_bandwidth_config_t
```

Bandwidth control configuration Implements : edma_bandwidth_config_t_Class.

Enumerator

EDMA_BANDWIDTH_STALL_NONE	No eDMA engine stalls.
EDMA_BANDWIDTH_STALL_4_CYCLES	eDMA engine stalls for 4 cycles after each read/write.
EDMA_BANDWIDTH_STALL_8_CYCLES	eDMA engine stalls for 8 cycles after each read/write.

3.19.2.3 edma_channel_priority_t

```
enum edma_channel_priority_t
```

eDMA channel priority setting Implements : edma_channel_priority_t_Class

Enumerator

EDMA_CHN_PRIORITY_0	
EDMA_CHN_PRIORITY_1	
EDMA_CHN_PRIORITY_2	
EDMA_CHN_PRIORITY_3	
EDMA_CHN_PRIORITY_4	
EDMA_CHN_PRIORITY_5	
EDMA_CHN_PRIORITY_6	
EDMA_CHN_PRIORITY_7	
EDMA_CHN_PRIORITY_8	
EDMA_CHN_PRIORITY_9	
EDMA_CHN_PRIORITY_10	
EDMA_CHN_PRIORITY_11	
EDMA_CHN_PRIORITY_12	
EDMA_CHN_PRIORITY_13	
EDMA_CHN_PRIORITY_14	
EDMA_CHN_PRIORITY_15	
EDMA_CHN_DEFAULT_PRIORITY	

3.19.2.4 edma_modulo_t

```
enum edma_modulo_t
```

eDMA modulo configuration Implements : edma_modulo_t_Class

Enumerator

EDMA_MODULO_OFF	
EDMA_MODULO_2B	
EDMA_MODULO_4B	
EDMA_MODULO_8B	
EDMA_MODULO_16B	
EDMA_MODULO_32B	
EDMA_MODULO_64B	
EDMA_MODULO_128B	
EDMA_MODULO_256B	
EDMA_MODULO_512B	
EDMA_MODULO_1KB	
EDMA_MODULO_2KB	
EDMA_MODULO_4KB	
EDMA_MODULO_8KB	
EDMA_MODULO_16KB	
EDMA_MODULO_32KB	
EDMA_MODULO_64KB	
EDMA_MODULO_128KB	
EDMA_MODULO_256KB	
EDMA_MODULO_512KB	
EDMA_MODULO_1MB	
EDMA_MODULO_2MB	
EDMA_MODULO_4MB	
EDMA_MODULO_8MB	
EDMA_MODULO_16MB	
EDMA_MODULO_32MB	
EDMA_MODULO_64MB	
EDMA_MODULO_128MB	
EDMA_MODULO_256MB	
EDMA_MODULO_512MB	
EDMA_MODULO_1GB	
EDMA_MODULO_2GB	

3.19.2.5 edma_transfer_size_t

```
enum edma_transfer_size_t
```

eDMA transfer configuration Implements : edma_transfer_size_t_Class

Enumerator

EDMA_TRANSFER_SIZE_1B	
EDMA_TRANSFER_SIZE_2B	
EDMA_TRANSFER_SIZE_4B	
EDMA_TRANSFER_SIZE_16B	
EDMA_TRANSFER_SIZE_32B	

3.19.3 Function Documentation

3.19.3.1 EDMA_HAL_CancelTransfer()

```
void EDMA_HAL_CancelTransfer (
            DMA_Type * base )
```

Cancels the remaining data transfer.

This function 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

base	Register base address for eDMA module.

3.19.3.2 EDMA_HAL_CancelTransferWithError()

```
void EDMA_HAL_CancelTransferWithError (
            DMA_Type * base )
```

Cancels the remaining data transfer and treats it as an error condition.

This function 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 honoured. This cancel retires the channel normally as if the minor loop had completed. Additional thing is to treat this operation as an error condition.

Parameters

base	Register base address for eDMA module.

3.19.3.3 EDMA_HAL_ClearDoneStatusFlag()

```
static void EDMA_HAL_ClearDoneStatusFlag (
            DMA_Type * base,
            uint8_t channel ) [inline], [static]
```

Clears the done status for a channel or all channels.

Parameters

base	Register base address for eDMA module.
channel	Channel indicator. If kEDMAAllChannel is selected, all channels' done status will be cleared. Implements : EDMA_HAL_ClearDoneStatusFlag_Activity

3.19.3.4 EDMA_HAL_ClearErrorIntStatusFlag()

```
static void EDMA_HAL_ClearErrorIntStatusFlag (
            DMA_Type * base,
            uint8_t channel ) [inline], [static]
```

Clears the error interrupt status for the eDMA channel or channels.

Parameters

base	Register base address for eDMA module.
channel	Channel indicator. If kEDMAAllChannel is selected, all channels' error interrupt status will be cleared. Implements : EDMA_HAL_ClearErrorIntStatusFlag_Activity

3.19.3.5 EDMA_HAL_ClearIntStatusFlag()

```
static void EDMA_HAL_ClearIntStatusFlag (
            DMA_Type * base,
            uint8_t channel )  [inline], [static]
```

Clears the interrupt status for the eDMA channel or all channels.

Parameters

base	Register base address for eDMA module.
channel	Channel indicator. If kEDMAAllChannel is selected, all channels' interrupt status will be cleared. Implements : EDMA_HAL_ClearIntStatusFlag_Activity

3.19.3.6 EDMA_HAL_GetChannelArbitrationMode()

```
static edma_arbitration_algorithm_t EDMA_HAL_GetChannelArbitrationMode (
            const DMA_Type * base )  [inline], [static]
```

Gets the channel arbitration algorithm.

Parameters

base	Register base address for eDMA module.

Returns

edma_arbitration_algorithm_t variable indicating the selected channel arbitration: Round-Robin way or fixed priority way Implements : EDMA_HAL_GetChannelArbitrationMode_Activity

3.19.3.7 EDMA_HAL_GetChannelPriorityError()

```
static bool EDMA_HAL_GetChannelPriorityError (
            const DMA_Type * base )  [inline], [static]
```

Checks for channel priority errors.

Returns whether the channel priorities configuration was erroneous (priorities not unique).

Parameters

base	Register base address for eDMA module.

Returns

true, if the channels priorities were misconfigured, false otherwise. Implements : EDMA_HAL_GetChannel←
PriorityError_Activity

3.19.3.8 EDMA_HAL_GetChannelWithError()

```
static uint8_t EDMA_HAL_GetChannelWithError (
            const DMA_Type * base )  [inline], [static]
```

Error channel number.

Returns the channel number of the last recorded error.

Parameters

base	Register base address for eDMA module.

Returns

the channel number of the last recorded error. Implements : EDMA_HAL_GetChannelWithError_Activity

3.19.3.9 EDMA_HAL_GetDestinationAddressError()

```
static bool EDMA_HAL_GetDestinationAddressError (
            const DMA_Type * base )  [inline], [static]
```

Checks for destination address errors.

Returns whether the destination address configuration was erroneous.

Parameters

base	Register base address for eDMA module.

Returns

true, if the destination address was misconfigured, false otherwise. Implements : EDMA_HAL_Get←
DestinationAddressError_Activity

3.19.3.10 EDMA_HAL_GetDestinationBusError()

```
static bool EDMA_HAL_GetDestinationBusError (
            const DMA_Type * base )  [inline], [static]
```

Checks for destination bus errors.

Returns whether there was a bus error on a destination write.

Parameters

base	Register base address for eDMA module.

Returns

true, if there was a destination bus error, false otherwise. Implements : EDMA_HAL_GetDestinationBus↩ Error_Activity

3.19.3.11 EDMA_HAL_GetDestinationOffsetError()

```
static bool EDMA_HAL_GetDestinationOffsetError (
             const DMA_Type * base )  [inline], [static]
```

Checks for destination offset errors.

Returns whether the destination offset configuration was erroneous.

Parameters

base	Register base address for eDMA module.

Returns

true, if the destination offset was misconfigured, false otherwise. Implements : EDMA_HAL_GetDestination↩ OffsetError_Activity

3.19.3.12 EDMA_HAL_GetDmaRequestStatusFlag()

```
static bool EDMA_HAL_GetDmaRequestStatusFlag (
             const DMA_Type * base,
             uint32_t channel )  [inline], [static]
```

Gets the eDMA channel DMA request status.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.

Returns

Hardware request is triggered in this eDMA channel (true) or not be triggered in this channel (false). Implements : EDMA_HAL_GetDmaRequestStatusFlag_Activity

3.19.3.13 EDMA_HAL_GetErrorIntStatusFlag()

```
static uint32_t EDMA_HAL_GetErrorIntStatusFlag (
             const DMA_Type * base )  [inline], [static]
```

Gets the eDMA error interrupt status.

Parameters

base	Register base address for eDMA module.

Returns

32 bit variable indicating error channels. If error happens on eDMA channel n, the bit n of this variable is '1'. If not, the bit n of this variable is '0'. Implements : EDMA_HAL_GetErrorIntStatusFlag_Activity

3.19.3.14 EDMA_HAL_GetIntStatusFlag()

```
static bool EDMA_HAL_GetIntStatusFlag (
            const DMA_Type * base,
            uint32_t channel )  [inline], [static]
```

Gets the eDMA channel interrupt request status.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.

Returns

Interrupt request happens in this eDMA channel (true) or not happen in this channel (false). Implements : EDMA_HAL_GetIntStatusFlag_Activity

3.19.3.15 EDMA_HAL_GetMinorMajorLoopConfigError()

```
static bool EDMA_HAL_GetMinorMajorLoopConfigError (
            const DMA_Type * base )  [inline], [static]
```

Checks for minor/major loop configuration errors.

Returns whether the NBYTES or CITER fields configuration was erroneous.

Parameters

base	Register base address for eDMA module.

Returns

true, if the NBYTES/CITER was misconfigured, false otherwise. Implements : EDMA_HAL_GetMinorMajor←LoopConfigError_Activity

3.19.3.16 EDMA_HAL_GetScatterGatherError()

```
static bool EDMA_HAL_GetScatterGatherError (
            const DMA_Type * base )  [inline], [static]
```

Checks for scatter/gather configuration errors.

Returns whether the scatter/gather configuration was erroneous.

Parameters

base	Register base address for eDMA module.

Returns

true, if the scatter/gather feature was misconfigured, false otherwise. Implements : EDMA_HAL_GetScatter↩ GatherError_Activity

3.19.3.17 EDMA_HAL_GetSourceAddressError()

```
static bool EDMA_HAL_GetSourceAddressError (
            const DMA_Type * base )  [inline], [static]
```

Checks for source address errors.

Returns whether the source address configuration was erroneous.

Parameters

base	Register base address for eDMA module.

Returns

true, if the source address was misconfigured, false otherwise. Implements : EDMA_HAL_GetSource↩ AddressError_Activity

3.19.3.18 EDMA_HAL_GetSourceBusError()

```
static bool EDMA_HAL_GetSourceBusError (
            const DMA_Type * base )  [inline], [static]
```

Checks for source bus errors.

Returns whether there was a bus error on a source read.

Parameters

base	Register base address for eDMA module.

Returns

true, if there was a source bus error, false otherwise. Implements : EDMA_HAL_GetSourceBusError_Activity

3.19.3.19 EDMA_HAL_GetSourceOffsetError()

```
static bool EDMA_HAL_GetSourceOffsetError (
            const DMA_Type * base )  [inline], [static]
```

Checks for source offset errors.

Returns whether the source offset configuration was erroneous.

Parameters

base	Register base address for eDMA module.

Returns

true, if the source offset was misconfigured, false otherwise. Implements : EDMA_HAL_GetSourceOffset↩ Error_Activity

3.19.3.20 EDMA_HAL_GetTransferCancelledError()

```
static bool EDMA_HAL_GetTransferCancelledError (
            const DMA_Type * base ) [inline], [static]
```

Checks for cancelled transfers.

Returns whether the last entry was a cancelled transfer, by the error cancel transfer input

Parameters

base	Register base address for eDMA module.

Returns

true, if the last transfer was cancelled, false otherwise. Implements : EDMA_HAL_GetTransferCancelled↩ Error_Activity

3.19.3.21 EDMA_HAL_GetValidErrorNotCleared()

```
static bool EDMA_HAL_GetValidErrorNotCleared (
            const DMA_Type * base ) [inline], [static]
```

Checks for valid errors.

Returns whether a valid error exists, that has not been cleared.

Parameters

base	Register base address for eDMA module.

Returns

true, if a valid uncleared error exists, false otherwise. Implements : EDMA_HAL_GetValidErrorNotCleared_↩ Activity

3.19.3.22 EDMA_HAL_Init()

```
void EDMA_HAL_Init (
            DMA_Type * base )
```

Initializes eDMA module to known state.

Parameters

base	Register base address for eDMA module.

3.19.3.23 EDMA_HAL_SetAsyncRequestInStopModeCmd()

```
static void EDMA_HAL_SetAsyncRequestInStopModeCmd (
            DMA_Type * base,
            uint32_t channel,
            bool enable )  [inline], [static]
```

Enables/Disables an asynchronous request in stop mode.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
enable	Enable (true) or Disable (false) async DMA request. Implements : EDMA_HAL_SetAsyncRequestInStopModeCmd_Activity

3.19.3.24 EDMA_HAL_SetChannelArbitrationMode()

```
static void EDMA_HAL_SetChannelArbitrationMode (
            DMA_Type * base,
            edma_arbitration_algorithm_t channelArbitration )  [inline], [static]
```

Sets the channel arbitration algorithm.

Parameters

base	Register base address for eDMA module.
channelArbitration	Round-Robin way or fixed priority way. Implements : EDMA_HAL_SetChannelArbitrationMode_Activity

3.19.3.25 EDMA_HAL_SetChannelPreemptMode()

```
static void EDMA_HAL_SetChannelPreemptMode (
            DMA_Type * base,
            uint32_t channel,
            bool preemptive,
            bool preemptible )  [inline], [static]
```

Sets the preemption feature for the eDMA channel.

This function sets the preemption features.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.

Parameters

preemptive	eDMA channel can suspend a lower priority channel (true). eDMA channel cannot suspend a lower priority channel (false).
preemptible	eDMA channel can be temporarily suspended by the service request of a higher priority channel (true). eDMA channel can't be suspended by a higher priority channel (false). Implements : EDMA_HAL_SetChannelPreemptMode_Activity

3.19.3.26 EDMA_HAL_SetChannelPriority()

```
static void EDMA_HAL_SetChannelPriority (
             DMA_Type * base,
             uint32_t channel,
             edma_channel_priority_t priority )  [inline], [static]
```

Sets the eDMA channel priority.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
priority	Priority of the DMA channel. Different channels should have different priority setting inside a group. Implements : EDMA_HAL_SetChannelPriority_Activity

3.19.3.27 EDMA_HAL_SetContinuousLinkCmd()

```
static void EDMA_HAL_SetContinuousLinkCmd (
             DMA_Type * base,
             bool continuous )  [inline], [static]
```

Enables or disables the continuous transfer mode.

This function enables or disables the continuous transfer. 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

base	Register base address for eDMA module.
continuous	Enables (true) or Disable (false) continuous transfer mode. Implements : EDMA_HAL_SetContinuousLinkCmd_Activity

3.19.3.28 EDMA_HAL_SetDebugCmd()

```
static void EDMA_HAL_SetDebugCmd (
             DMA_Type * base,
             bool enable )  [inline], [static]
```

Enables/Disables the eDMA DEBUG mode.

This function enables/disables the eDMA Debug mode. 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

base	Register base address for eDMA module.
enable	Enables (true) or Disable (false) eDMA module debug mode. Implements : EDMA_HAL_SetDebugCmd_Activity

3.19.3.29 EDMA_HAL_SetDmaRequestCmd()

```
void EDMA_HAL_SetDmaRequestCmd (
            DMA_Type * base,
            uint8_t channel,
            bool enable )
```

Enables/Disables the DMA request for the channel or all channels.

Parameters

base	Register base address for eDMA module.
enable	Enable(true) or Disable (false) DMA request.
channel	Channel indicator. If kEDMAAllChannel is selected, all channels DMA request are enabled/disabled.

3.19.3.30 EDMA_HAL_SetErrorIntCmd()

```
void EDMA_HAL_SetErrorIntCmd (
            DMA_Type * base,
            uint8_t channel,
            bool enable )
```

Enables/Disables the error interrupt for channels.

Parameters

base	Register base address for eDMA module.
channel	Channel indicator. If kEDMAAllChannel is selected, all channels' error interrupt will be enabled/disabled.
enable	Enable(true) or Disable (false) error interrupt.

3.19.3.31 EDMA_HAL_SetHaltCmd()

```
static void EDMA_HAL_SetHaltCmd (
            DMA_Type * base,
            bool halt ) [inline], [static]
```

Halts/Un-halts the DMA Operations.

This function stalls/un-stalls the start of any new channels. Executing channels are allowed to be completed.

Parameters

base	Register base address for eDMA module.
halt	Halts (true) or un-halts (false) eDMA transfer. Implements : EDMA_HAL_SetHaltCmd_Activity

3.19.3.32 EDMA_HAL_SetHaltOnErrorCmd()

```
static void EDMA_HAL_SetHaltOnErrorCmd (
            DMA_Type * base,
            bool haltOnError ) [inline], [static]
```

Halts or does not halt the eDMA module when an error occurs.

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

Parameters

base	Register base address for eDMA module.
haltOnError	Halts (true) or not halt (false) eDMA module when an error occurs. Implements : EDMA_HAL_SetHaltOnErrorCmd_Activity

3.19.3.33 EDMA_HAL_SetMinorLoopMappingCmd()

```
static void EDMA_HAL_SetMinorLoopMappingCmd (
            DMA_Type * base,
            bool enable ) [inline], [static]
```

Enables/Disables the minor loop mapping.

This function enables/disables the minor loop mapping feature. If enabled, the NBYTES is redefined to include the 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

base	Register base address for eDMA module.
enable	Enables (true) or Disable (false) minor loop mapping. Implements : EDMA_HAL_SetMinorLoopMappingCmd_Activity

3.19.3.34 EDMA_HAL_TCDClearReg()

```
void EDMA_HAL_TCDClearReg (
            DMA_Type * base,
            uint32_t channel )
```

Clears all registers to 0 for the hardware TCD.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.

3.19.3.35 EDMA_HAL_TCDGetBeginMajorCount()

```
uint32_t EDMA_HAL_TCDGetBeginMajorCount (
            const DMA_Type * base,
            uint32_t channel )
```

Returns the begin major iteration count.

Gets the begin major iteration count according to minor loop channel link settings.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.

Returns

begin iteration count

3.19.3.36 EDMA_HAL_TCDGetChannelActiveStatus()

```
static bool EDMA_HAL_TCDGetChannelActiveStatus (
            const DMA_Type * base,
            uint32_t channel )  [inline], [static]
```

Checks whether the channel is running for the TCD.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.

Returns

True stands for running. False stands for not. Implements : EDMA_HAL_TCDGetChannelActiveStatus_↩ Activity

3.19.3.37 EDMA_HAL_TCDGetCurrentMajorCount()

```
uint32_t EDMA_HAL_TCDGetCurrentMajorCount (
            const DMA_Type * base,
            uint32_t channel )
```

Returns the current major iteration count.

Gets the current major iteration count according to minor loop channel link settings.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.

Returns

current iteration count

3.19.3.38 EDMA_HAL_TCDGetDoneStatusFlag()

```
static bool EDMA_HAL_TCDGetDoneStatusFlag (
            const DMA_Type * base,
            uint32_t channel ) [inline], [static]
```

Gets the channel done status.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.

Returns

If channel done. Implements : EDMA_HAL_TCDGetDoneStatusFlag_Activity

3.19.3.39 EDMA_HAL_TCDGetFinishedBytes()

```
uint32_t EDMA_HAL_TCDGetFinishedBytes (
            const DMA_Type * base,
            uint32_t channel )
```

Gets the number of bytes already transferred for the TCD.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.

Returns

data bytes already transferred

3.19.3.40 EDMA_HAL_TCDGetNbytes()

```
uint32_t EDMA_HAL_TCDGetNbytes (
            const DMA_Type * base,
            uint32_t channel )
```

Gets the nbytes configuration data for the TCD.

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

base	Register base address for eDMA module.
channel	eDMA channel number.

Returns

nbytes configuration according to minor loop setting.

3.19.3.41 EDMA_HAL_TCDGetUnfinishedBytes()

```
uint32_t EDMA_HAL_TCDGetUnfinishedBytes (
            const DMA_Type * base,
            uint32_t channel )
```

Gets the number of bytes haven't transferred for the TCD.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.

Returns

data bytes already transferred

3.19.3.42 EDMA_HAL_TCDSetAttribute()

```
void EDMA_HAL_TCDSetAttribute (
            DMA_Type * base,
            uint32_t channel,
            edma_modulo_t srcModulo,
            edma_modulo_t destModulo,
            edma_transfer_size_t srcTransferSize,
            edma_transfer_size_t destTransferSize )
```

Configures the transfer attribute for the eDMA channel.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
srcModulo	enumeration type for an allowed source modulo. 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.
destModulo	Enum type for an allowed destination modulo.
srcTransferSize	Enum type for source transfer size.
destTransferSize	Enum type for destination transfer size.

3.19.3.43 EDMA_HAL_TCDSetBandwidth()

```
static void EDMA_HAL_TCDSetBandwidth (
             DMA_Type * base,
             uint32_t channel,
             edma_bandwidth_config_t bandwidth )  [inline], [static]
```

Configures the bandwidth for the TCD.

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

base	Register base address for eDMA module.
channel	eDMA channel number.
bandwidth	enum type for bandwidth control Implements : EDMA_HAL_TCDSetBandwidth_Activity

3.19.3.44 EDMA_HAL_TCDSetChannelMajorLink()

```
static void EDMA_HAL_TCDSetChannelMajorLink (
             DMA_Type * base,
             uint32_t channel,
             uint32_t majorLinkChannel,
             bool enable )  [inline], [static]
```

Configures the major channel link the TCD.

If the major link 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

base	Register base address for eDMA module.
channel	eDMA channel number.
majorLinkChannel	channel number for major link
enable	Enables (true) or Disables (false) channel major link. Implements : EDMA_HAL_TCDSetChannelMajorLink_Activity

3.19.3.45 EDMA_HAL_TCDSetChannelMinorLink()

```
void EDMA_HAL_TCDSetChannelMinorLink (
             DMA_Type * base,
             uint32_t channel,
             uint32_t linkChannel,
             bool enable )
```

Sets the channel minor link for the TCD.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
linkChannel	Channel to be linked on minor loop complete.
enable	Enable (true)/Disable (false) channel minor link.

3.19.3.46 EDMA_HAL_TCDSetDestAddr()

```
static void EDMA_HAL_TCDSetDestAddr (
            DMA_Type * base,
            uint32_t channel,
            uint32_t address )  [inline], [static]
```

Configures the destination address for the TCD.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
address	The pointer to the destination address. Implements : EDMA_HAL_TCDSetDestAddr_Activity

3.19.3.47 EDMA_HAL_TCDSetDestLastAdjust()

```
static void EDMA_HAL_TCDSetDestLastAdjust (
            DMA_Type * base,
            uint32_t channel,
            int32_t adjust )  [inline], [static]
```

Configures the last source address adjustment.

This function adds 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

base	Register base address for eDMA module.
channel	eDMA channel number.
adjust	adjustment value Implements : EDMA_HAL_TCDSetDestLastAdjust_Activity

3.19.3.48 EDMA_HAL_TCDSetDestMinorLoopOffsetCmd()

```
static void EDMA_HAL_TCDSetDestMinorLoopOffsetCmd (
            DMA_Type * base,
            uint32_t channel,
            bool enable )  [inline], [static]
```

Enables/disables the destination minor loop offset feature for the TCD.

Configures whether the minor loop offset is applied to the destination address upon minor loop completion. NOTE: EMLM bit needs to be enabled prior to calling this function, otherwise it has no effect.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
enable	Enables (true) or disables (false) destination minor loop offset. Implements : EDMA_HAL_TCDSetDestMinorLoopOffsetCmd_Activity

3.19.3.49 EDMA_HAL_TCDSetDestOffset()

```
static void EDMA_HAL_TCDSetDestOffset (
            DMA_Type * base,
            uint32_t channel,
            int16_t offset )  [inline], [static]
```

Configures the destination address signed offset for the TCD.

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

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
offset	signed-offset Implements : EDMA_HAL_TCDSetDestOffset_Activity

3.19.3.50 EDMA_HAL_TCDSetDisableDmaRequestAfterTCDDoneCmd()

```
static void EDMA_HAL_TCDSetDisableDmaRequestAfterTCDDoneCmd (
            DMA_Type * base,
            uint32_t channel,
            bool disable )  [inline], [static]
```

Disables/Enables the DMA request after the major loop completes for the TCD.

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

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
disable	Disable (true)/Enable (false) DMA request after TCD complete. Implements : EDMA_HAL_TCDSetDisableDmaRequestAfterTCDDoneCmd_Activity

3.19.3.51 EDMA_HAL_TCDSetHalfCompleteIntCmd()

```
static void EDMA_HAL_TCDSetHalfCompleteIntCmd (
            DMA_Type * base,
            uint32_t channel,
            bool enable )  [inline], [static]
```

Enables/Disables the half complete interrupt for the TCD.

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

base	Register base address for eDMA module.
channel	eDMA channel number.
enable	Enable (true) /Disable (false) half complete interrupt. Implements : EDMA_HAL_TCDSetHalfCompleteIntCmd_Activity

3.19.3.52 EDMA_HAL_TCDSetIntCmd()

```
static void EDMA_HAL_TCDSetIntCmd (
            DMA_Type * base,
            uint32_t channel,
            bool enable )  [inline], [static]
```

Enables/Disables the interrupt after the major loop completes for the TCD.

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

base	Register base address for eDMA module.
channel	eDMA channel number.
enable	Enable (true) /Disable (false) interrupt after TCD done. Implements : EDMA_HAL_TCDSetIntCmd_Activity

3.19.3.53 EDMA_HAL_TCDSetMajorCount()

```
void EDMA_HAL_TCDSetMajorCount (
            DMA_Type * base,
            uint32_t channel,
            uint32_t count )
```

Sets the major iteration count according to minor loop channel link setting.

Note here that user need to first set the minor loop channel link and then call this function. The execute flow inside this function is dependent on the minor loop channel link setting.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
count	major loop count

3.19.3.54 EDMA_HAL_TCDSetMinorLoopOffset()

```
void EDMA_HAL_TCDSetMinorLoopOffset (
            DMA_Type * base,
            uint32_t channel,
            int32_t offset )
```

Configures the minor loop offset for the TCD.

Configures the offset value. If neither source nor destination offset is enabled, offset is not configured. NOTE: EMLM bit needs to be enabled prior to calling this function, otherwise it has no effect.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
offset	Minor loop offset

3.19.3.55 EDMA_HAL_TCDSetNbytes()

```
void EDMA_HAL_TCDSetNbytes (
            DMA_Type * base,
            uint32_t channel,
            uint32_t nbytes )
```

Configures the nbytes for the eDMA channel.

Note here that user need firstly configure the minor loop mapping feature and then call this function.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
nbytes	Number of bytes to be transferred in each service request of the channel

3.19.3.56 EDMA_HAL_TCDSetScatterGatherCmd()

```
static void EDMA_HAL_TCDSetScatterGatherCmd (
            DMA_Type * base,
            uint32_t channel,
            bool enable ) [inline], [static]
```

Enables/Disables the scatter/gather feature for the TCD.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
enable	Enables (true) /Disables (false) scatter/gather feature. Implements : EDMA_HAL_TCDSetScatterGatherCmd_Activity

3.19.3.57 EDMA_HAL_TCDSetScatterGatherLink()

```
void EDMA_HAL_TCDSetScatterGatherLink (
             DMA_Type * base,
             uint32_t channel,
             uint32_t nextTCDAddr )
```

Configures the memory address for the next transfer TCD for the TCD.

This function enables the scatter/gather feature for the TCD and configures the next TCD's address. This address points to the beginning of a 0-modulo-32 byte region containing the next transfer TCD 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.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
nextTCDAddr	The address of the next TCD to be linked to this TCD.

3.19.3.58 EDMA_HAL_TCDSetSrcAddr()

```
static void EDMA_HAL_TCDSetSrcAddr (
             DMA_Type * base,
             uint32_t channel,
             uint32_t address ) [inline], [static]
```

Configures the source address for the hardware TCD.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
address	The pointer to the source memory address. Implements : EDMA_HAL_TCDSetSrcAddr_Activity

3.19.3.59 EDMA_HAL_TCDSetSrcLastAdjust()

```
static void EDMA_HAL_TCDSetSrcLastAdjust (
             DMA_Type * base,
             uint32_t channel,
             int32_t size ) [inline], [static]
```

Configures the last source address adjustment for the TCD.

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

base	Register base address for eDMA module.
channel	eDMA channel number.
size	adjustment value Implements : EDMA_HAL_TCDSetSrcLastAdjust_Activity

3.19.3.60 EDMA_HAL_TCDSetSrcMinorLoopOffsetCmd()

```
static void EDMA_HAL_TCDSetSrcMinorLoopOffsetCmd (
            DMA_Type * base,
            uint32_t channel,
            bool enable )  [inline], [static]
```

Enables/disables the source minor loop offset feature for the TCD.

Configures whether the minor loop offset is applied to the source address upon minor loop completion. NOTE: EMLM bit needs to be enabled prior to calling this function, otherwise it has no effect.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
enable	Enables (true) or disables (false) source minor loop offset. Implements : EDMA_HAL_TCDSetSrcMinorLoopOffsetCmd_Activity

3.19.3.61 EDMA_HAL_TCDSetSrcOffset()

```
static void EDMA_HAL_TCDSetSrcOffset (
            DMA_Type * base,
            uint32_t channel,
            int16_t offset )  [inline], [static]
```

Configures the source address signed offset for the hardware TCD.

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

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number.
offset	signed-offset for source address. Implements : EDMA_HAL_TCDSetSrcOffset_Activity

3.19.3.62 EDMA_HAL_TCDTriggerChannelStart()

```
static void EDMA_HAL_TCDTriggerChannelStart (
            DMA_Type * base,
            uint32_t channel )  [inline], [static]
```

Triggers the start bits for the TCD.

The eDMA hardware automatically clears this flag after the channel begins execution.

Parameters

base	Register base address for eDMA module.
channel	eDMA channel number. Implements : EDMA_HAL_TCDTriggerChannelStart_Activity

3.19.3.63 EDMA_HAL_TriggerChannelStart()

```
static void EDMA_HAL_TriggerChannelStart (
            DMA_Type * base,
            uint8_t channel )  [inline], [static]
```

Triggers the eDMA channel.

Parameters

base	Register base address for eDMA module.
channel	Channel indicator. If kEDMAAllChannel is selected, all channels are tirggere. Implements : EDMA_HAL_TriggerChannelStart_Activity

3.20 EIM Driver

3.20.1 Detailed Description

Error Injection Module Peripheral Driver.
EIM PD provides a set of high-level APIs/services to configure the Error Injection Module (EIM) module.

Basic Operations of EIM

1. To initialize EIM, call EIM_DRV_Init() with an user channel configuration array. In the following code, EIM is initialized with default settings (after reset) for check-bit mask and data mask and both channels is enabled.

```
#define INST_EIM1 (0U)

/* Configuration structure array */
eim_user_channel_config_t userChannelConfigArr[] =
{
    /* Configuration channel 0 */
    {
        .channel = 0x0U,
        .checkBitMask = 0x00U,
        .dataMask = 0x00U,
        .enable = true
    },
    /* Configuration channel 1 */
    {
        .channel = 0x1U,
        .checkBitMask = 0x00U,
        .dataMask = 0x00U,
        .enable = true
    }
};

/* Initialize the EIM instance 0 with configured channel number of 2 and userChannelConfigArr */
EIM_DRV_Init(INST_EIM1, 2U, userChannelConfigArr);
```

2. Check-bit mask defines a bit-mapped mask that specifies whether the corresponding bit of the check-bit bus from the target RAM should be inverted or remain unmodified. Data mask defines a bit-mapped mask that specifies whether the corresponding bit of the read data bus from the target RAM should be inverted or remain unmodified. To reconfigure check-bit mask or data mask:

   (a) The first make sure that EIM module is disabled by call EIM_HAL_Disable().

   (b) After that, call EIM_DRV_ConfigChannel() with a new check-bit mask or data mask wants to change.

   (c) And finally, call EIM_HAL_Enable() to re-enable EIM module. The following sample code reconfigures EIM to invert bit 0, 2, 5, 7 of check-bit bus for channel 0 and invert bit 7, 15, 23, 31 of bytes 0-3 of the read data bus for channel 1.

```
/* Disable EIM module */
EIM_HAL_Disable(g_eimBase[INST_EIM1]);

/* Change user configuration of EIM channel 0 */
userChannelConfigArr[0].checkBitMask = 0b10100101U;
/* Set new user configuration for channel 0*/
EIM_DRV_ConfigChannel(INST_EIM1, userChannelConfigArr[0]);

/* Change user configuration of EIM channel 1 */
userChannelConfigArr[1].dataMask = 0x80808080U;
/* Set new user configuration for channel 1*/
EIM_DRV_ConfigChannel(INST_EIM1, userChannelConfigArr[1]);

/* Enable EIM module */
EIM_HAL_Enable(g_eimBase[INST_EIM1]);
```

3. To get the current register configuration (data mask, check-bit mask and enable status) of a channel in EIM, just call EIM_DRV_GetChannelConfig(). Make sure that the operation is not execute in target RAM where EIM inject the error

```
eim_user_channel_config_t channelConfig;

/* Get configuration of EIM channel 1*/
EIM_DRV_GetChannelConfig(INST_EIM1, 1U, &channelConfig);
```

4. To de-initialize EIM, just call the EIM_DRV_Deinit() function. This function sets all registers to reset values and disables EIM.

```
/* De-initializes the EIM module */
EIM_DRV_Deinit(INST_EIM1);
```

Data Structures

- struct eim_user_channel_config_t

    EIM channel configuration structure. More...

Variables

- EIM_Type ∗const g_eimBase [EIM_INSTANCE_COUNT]

    Table of base addresses for EIM instances.

EIM Driver API

- void EIM_DRV_Init (uint32_t instance, uint8_t channelCnt, const eim_user_channel_config_t ∗channel←ConfigArr)

    Initializes the EIM module.
- void EIM_DRV_Deinit (uint32_t instance)

    De-initializes the EIM module.
- void EIM_DRV_ConfigChannel (uint32_t instance, const eim_user_channel_config_t ∗userChannelConfig)

    Initializes the EIM channel.
- void EIM_DRV_GetChannelConfig (uint32_t instance, uint8_t channel, eim_user_channel_config_←t ∗channelConfig)

    Gets the EIM channel configuration.

3.20.2 Data Structure Documentation

3.20.2.1 struct eim_user_channel_config_t

EIM channel configuration structure.

This structure holds the configuration settings for the EIM channel Implements : eim_user_channel_config_t_Class

Data Fields

- uint8_t channel
- uint8_t checkBitMask
- uint32_t dataMask
- bool enable

Field Documentation

3.20.2.1.1 channel

```
uint8_t channel
```

EIM channel number

3.20.2.1.2 checkBitMask

```
uint8_t checkBitMask
```

Specifies whether the corresponding bit of the checkbit bus from the target RAM should be inverted or remain unmodified

3.20.2.1.3 dataMask

```
uint32_t dataMask
```

Specifies whether the corresponding bit of the read data bus from the target RAM should be inverted or remain unmodified

3.20.2.1.4 enable

```
bool enable
```

true : EIM channel operation is enabled false : EIM channel operation is disabled

3.20.3 Function Documentation

3.20.3.1 EIM_DRV_ConfigChannel()

```
void EIM_DRV_ConfigChannel (
             uint32_t instance,
             const eim_user_channel_config_t * userChannelConfig )
```

Initializes the EIM channel.

This function configures check bit mask, data mask and operation status(enable/disable) for EIM channel. The EIM channel configuration structure shall be passed as arguments.

This is an example demonstrating how to define a EIM channel configuration structure:

```
eim_user_channel_config_t eimTestInit = {
     .channel = 0x1U,
     .checkBitMask = 0x25U,
     .dataMask = 0x11101100U,
     .enable = true
};
```

Parameters

in	*instance*	EIM module instance number
in	*userChannelConfig*	Pointer to EIM channel configuration structure

3.20.3.2 EIM_DRV_Deinit()

```
void EIM_DRV_Deinit (
             uint32_t instance )
```

De-initializes the EIM module.

This function sets all registers to reset value and disables EIM module. In order to use the EIM module again, EIM_DRV_Init must be called.

Parameters

in	*instance*	EIM module instance number

3.20.3.3 EIM_DRV_GetChannelConfig()

```
void EIM_DRV_GetChannelConfig (
             uint32_t instance,
             uint8_t channel,
             eim_user_channel_config_t * channelConfig )
```

Gets the EIM channel configuration.

This function gets check bit mask, data mask and operation status of EIM channel.

Parameters

in	*instance*	EIM module instance number
in	*channel*	EIM channel number
out	*channelConfig*	Pointer to EIM channel configuration structure

3.20.3.4 EIM_DRV_Init()

```
void EIM_DRV_Init (
             uint32_t instance,
             uint8_t channelCnt,
             const eim_user_channel_config_t * channelConfigArr )
```

Initializes the EIM module.

This function configures for EIM channels. The EIM channel configuration structure array and number of configured channels shall be passed as arguments. This function should be called before calling any other EIM driver function.

This is an example demonstrating how to define a EIM channel configuration structure array:

```
eim_user_channel_config_t channelConfigArr[] =
{
     {
     .channel = 0x0U,
     .checkBitMask = 0x12U,
     .dataMask = 0x01234567U,
     .enable = true
     },
     {
     .channel = 0x1U,
     .checkBitMask = 0x22U,
     .dataMask = 0x01234444U,
     .enable = false
     }
};
```

Parameters

`in`	*instance*	EIM module instance number.
`in`	*channelCnt*	Number of configured channels
`in`	*channelConfigArr*	EIM channel configuration structure array

3.20.4 Variable Documentation

3.20.4.1 g_eimBase

```
EIM_Type* const g_eimBase[EIM_INSTANCE_COUNT]
```

Table of base addresses for EIM instances.

3.21 EIM HAL

3.21.1 Detailed Description

Error Injection Module Hardware Abstraction Level. EIM HAL provides low level APIs for reading and writing register bit-fields belonging to the EIM module.

Macros

- #define POS_MSB_EIM_EICHEN (31U)

  The position of the most significant bit in Error Injection Channel Enable register.

EIM HAL API

- void EIM_HAL_Init (EIM_Type ∗const base)

  Resets for the registers of EIM descriptor.
- static void EIM_HAL_Enable (EIM_Type ∗const base)

  Enables EIM module.
- static void EIM_HAL_Disable (EIM_Type ∗const base)

  Disables the EIM module.
- static void EIM_HAL_EnableChannelCmd (EIM_Type ∗const base, uint8_t channel, bool enable)

  Enables or disables EIM channel operation.
- static bool EIM_HAL_IsChannelEnabled (const EIM_Type ∗const base, uint8_t channel)

  Checks whether EIM channel is enabled.
- static void EIM_HAL_SetCheckBitMask (EIM_Type ∗const base, uint8_t channel, uint8_t checkBitMask)

  Sets check bit mask for EIM channel.
- static uint8_t EIM_HAL_GetCheckBitMask (const EIM_Type ∗const base, uint8_t channel)

  Gets check bit mask of EIM channel.
- static void EIM_HAL_SetDataMask (EIM_Type ∗const base, uint8_t channel, uint32_t dataMask)

  Sets data mask for EIM channel.
- static uint32_t EIM_HAL_GetDataMask (const EIM_Type ∗const base, uint8_t channel)

  Gets data mask of EIM channel.

3.21.2 Macro Definition Documentation

3.21.2.1 POS_MSB_EIM_EICHEN

```
#define POS_MSB_EIM_EICHEN (31U)
```

The position of the most significant bit in Error Injection Channel Enable register.

3.21.3 Function Documentation

3.21.3.1 EIM_HAL_Disable()

```
static void EIM_HAL_Disable (
            EIM_Type ∗const base )  [inline], [static]
```

Disables the EIM module.

This function disables the error injection function of the EIM.

Parameters

in	*base*	EIM peripheral base address Implements : EIM_HAL_Disable_Activity

3.21.3.2 EIM_HAL_Enable()

```
static void EIM_HAL_Enable (
            EIM_Type *const base )  [inline], [static]
```

Enables EIM module.

This function enables the error injection function of the EIM.

Parameters

in	*base*	EIM peripheral base address Implements : EIM_HAL_Enable_Activity

3.21.3.3 EIM_HAL_EnableChannelCmd()

```
static void EIM_HAL_EnableChannelCmd (
            EIM_Type *const base,
            uint8_t channel,
            bool enable )  [inline], [static]
```

Enables or disables EIM channel operation.

This function enables the EIM channel given as argument.

Parameters

in	*base*	EIM peripheral base address
in	*channel*	EIM channel number
in	*enable*	EIM channel operation • true : enables EIM channel • false: disables EIM channel Implements : EIM_HAL_EnableChannelCmd_Activity

3.21.3.4 EIM_HAL_GetCheckBitMask()

```
static uint8_t EIM_HAL_GetCheckBitMask (
            const EIM_Type *const base,
            uint8_t channel )  [inline], [static]
```

Gets check bit mask of EIM channel.

This function gets check bit mask of EIM channel given as argument.

Parameters

in	*base*	EIM peripheral base address
in	*channel*	EIM channel number

Returns

Checkbit mask Implements : EIM_HAL_GetCheckBitMask_Activity

3.21.3.5 EIM_HAL_GetDataMask()

```
static uint32_t EIM_HAL_GetDataMask (
            const EIM_Type *const base,
            uint8_t channel )  [inline], [static]
```

Gets data mask of EIM channel.

This function gets data mask of EIM channel given as argument.

Parameters

in	*base*	EIM peripheral base address
in	*channel*	EIM channel number

Returns

Data mask Implements : EIM_HAL_GetDataMask_Activity

3.21.3.6 EIM_HAL_Init()

```
void EIM_HAL_Init (
            EIM_Type *const base )
```

Resets for the registers of EIM descriptor.

This function disables all channels and clears checkbit and data masks of all the channels.

Parameters

in	*base*	EIM peripheral base address

3.21.3.7 EIM_HAL_IsChannelEnabled()

```
static bool EIM_HAL_IsChannelEnabled (
            const EIM_Type *const base,
            uint8_t channel )  [inline], [static]
```

Checks whether EIM channel is enabled.

This function check whether the EIM channel given as argument is enabled.

Parameters

in	*base*	EIM peripheral base address
in	*channel*	EIM channel number

Returns

EIM channel operation status -true: EIM channel is enabled -false: EIM channel is disabled Implements : EIM_HAL_IsChannelEnabled_Activity

3.21.3.8 EIM_HAL_SetCheckBitMask()

```
static void EIM_HAL_SetCheckBitMask (
             EIM_Type *const base,
             uint8_t channel,
             uint8_t checkBitMask )  [inline], [static]
```

Sets check bit mask for EIM channel.

This function sets the check bit mask of the EIM channel given as argument.

Parameters

in	*base*	EIM peripheral base address
in	*channel*	EIM channel number
in	*checkBitMask*	Checkbit mask Implements : EIM_HAL_SetCheckBitMask_Activity

3.21.3.9 EIM_HAL_SetDataMask()

```
static void EIM_HAL_SetDataMask (
             EIM_Type *const base,
             uint8_t channel,
             uint32_t dataMask )  [inline], [static]
```

Sets data mask for EIM channel.

This function sets data mask of the EIM channel given as argument.

Parameters

in	*base*	EIM peripheral base address
in	*channel*	EIM channel number
in	*dataMask*	Data mask Implements : EIM_HAL_SetDataMask_Activity

3.22 ERM Driver

3.22.1 Detailed Description

Error Reporting Module Peripheral Driver.

This section describes the programming interface of the ERM driver.

Data Structures

- struct erm_interrupt_config_t

  ERM interrupt notification configuration structure Implements : erm_interrupt_config_t_Class. More...
- struct erm_user_config_t

  ERM user configuration structure Implements : erm_user_config_t_Class. More...

Variables

- ERM_Type ∗const g_ermBase [ERM_INSTANCE_COUNT]

  Table of base addresses for ERM instances.

ERM DRIVER API

- void ERM_DRV_Init (uint32_t instance, uint8_t channelCnt, const erm_user_config_t ∗userConfigArr)

  Initializes the ERM module.
- void ERM_DRV_Deinit (uint32_t instance)

  Sets the default configuration.
- void ERM_DRV_SetInterruptConfig (uint32_t instance, uint8_t channel, erm_interrupt_config_t interruptCfg)

  Sets interrupt notification.
- void ERM_DRV_GetInterruptConfig (uint32_t instance, uint8_t channel, erm_interrupt_config_t ∗const interruptPtr)

  Gets interrupt notification.
- void ERM_DRV_ClearEvent (uint32_t instance, uint8_t channel, erm_ecc_event_t eccEvent)

  Clears error event and the corresponding interrupt notification.
- erm_ecc_event_t ERM_DRV_GetErrorDetail (uint32_t instance, uint8_t channel, uint32_t ∗addressPtr)

  Gets the address of the last ECC event in Memory n and ECC event.

3.22.2 Data Structure Documentation

3.22.2.1 struct erm_interrupt_config_t

ERM interrupt notification configuration structure Implements : erm_interrupt_config_t_Class.

Data Fields

- bool enableSingleCorrection
- bool enableNonCorrectable

Field Documentation

3.22.2.1.1 enableNonCorrectable

```
bool enableNonCorrectable
```

Enable Non-Correctable Interrupt Notification

3.22.2.1.2 enableSingleCorrection

```
bool enableSingleCorrection
```

Enable Single Correction Interrupt Notification

3.22.2.2 struct erm_user_config_t

ERM user configuration structure Implements : erm_user_config_t_Class.

Data Fields

- uint8_t channel
- const erm_interrupt_config_t ∗ interruptCfg

Field Documentation

3.22.2.2.1 channel

```
uint8_t channel
```

The channel assignments

3.22.2.2.2 interruptCfg

```
const erm_interrupt_config_t* interruptCfg
```

Interrupt configuration

3.22.3 Function Documentation

3.22.3.1 ERM_DRV_ClearEvent()

```
void ERM_DRV_ClearEvent (
             uint32_t instance,
             uint8_t channel,
             erm_ecc_event_t eccEvent )
```

Clears error event and the corresponding interrupt notification.

This function clears the record of an event. If the corresponding interrupt is enabled, the interrupt notification will be cleared

Parameters

in	*instance*	The ERM instance number
in	*channel*	The configured memory channel
in	*eccEvent*	The types of ECC events

3.22.3.2 ERM_DRV_Deinit()

```
void ERM_DRV_Deinit (
            uint32_t instance )
```

Sets the default configuration.

This function sets the default configuration

Parameters

in	*instance*	The ERM instance number

3.22.3.3 ERM_DRV_GetErrorDetail()

```
erm_ecc_event_t ERM_DRV_GetErrorDetail (
            uint32_t instance,
            uint8_t channel,
            uint32_t * addressPtr )
```

Gets the address of the last ECC event in Memory n and ECC event.

This function gets the address of the last ECC event in Memory n and the types of the event

Parameters

in	*instance*	The ERM instance number
in	*channel*	The examined memory channel
out	*addressPtr*	The pointer to address of the last ECC event in Memory n with ECC event

Returns

The last occurred ECC event

3.22.3.4 ERM_DRV_GetInterruptConfig()

```
void ERM_DRV_GetInterruptConfig (
            uint32_t instance,
            uint8_t channel,
            erm_interrupt_config_t *const interruptPtr )
```

Gets interrupt notification.

This function gets the current interrupt configuration of the available events (which interrupts are enabled/disabled)

Parameters

in	*instance*	The ERM instance number
in	*channel*	The examined memory channel
out	*interruptPtr*	The pointer to the ERM interrupt configuration structure

3.22.3.5 ERM_DRV_Init()

```
void ERM_DRV_Init (
            uint32_t instance,
            uint8_t channelCnt,
            const erm_user_config_t * userConfigArr )
```

Initializes the ERM module.

This function initializes ERM driver based on user configuration input, channelCnt takes values between 1 and the maximum channel count supported by the hardware

Parameters

in	*instance*	The ERM instance number
in	*channelCnt*	The number of channels
in	*userConfigArr*	The pointer to the array of ERM user configure structure

3.22.3.6 ERM_DRV_SetInterruptConfig()

```
void ERM_DRV_SetInterruptConfig (
            uint32_t instance,
            uint8_t channel,
            erm_interrupt_config_t interruptCfg )
```

Sets interrupt notification.

This function sets interrupt notification based on interrupt notification configuration input

Parameters

in	*instance*	The ERM instance number
in	*channel*	The configured memory channel
in	*interruptCfg*	The ERM interrupt configuration structure

3.22.4 Variable Documentation

3.22.4.1 g_ermBase

```
ERM_Type* const g_ermBase[ERM_INSTANCE_COUNT]
```

Table of base addresses for ERM instances.

3.23 ERM HAL

3.23.1 Detailed Description

Error Reporting Module Hardware Abstraction Layer.

This section describes the programming interface of the ERM HAL.

Macros

- #define ERM_CHANNELS_OFFSET_SIZE (4U)

  The distance between channels.
- #define ERM_NCE_START (30U)

  Start bit of non correctable error.
- #define ERM_SBC_START (31U)

  Start bit of single bit correction.

Enumerations

- enum erm_ecc_event_t { ERM_EVENT_NONE = 0U, ERM_EVENT_SINGLE_BIT = 1U, ERM_EVENT_N↩ON_CORRECTABLE = 2U }

  ERM types of ECC events Implements : erm_ecc_event_t_Class.

ERM HAL API

- void ERM_HAL_Init (ERM_Type ∗const base)

  Initializes the ERM module.
- static void ERM_HAL_EnableEventInterrupt (ERM_Type ∗const base, uint8_t channel, erm_ecc_event_↩t eccEvent, bool enable)

  Enables Memory n interrupt event.
- static bool ERM_HAL_IsEventInterruptEnabled (const ERM_Type ∗const base, uint8_t channel, erm_ecc_↩event_t eccEvent)

  Checks if the Memory n interrupt event is enabled.
- static bool ERM_HAL_IsEventDetected (const ERM_Type ∗const base, uint8_t channel, erm_ecc_event_t eccEvent)

  Checks if the Memory n error event is detected.
- static void ERM_HAL_ClearEvent (ERM_Type ∗const base, uint8_t channel, erm_ecc_event_t eccEvent)

  Clears error event and the corresponding interrupt notification.
- static uint32_t ERM_HAL_GetLastErrorAddress (const ERM_Type ∗const base, uint8_t channel)

  Gets the address of the last ECC event in Memory n.
- erm_ecc_event_t ERM_HAL_GetErrorDetail (const ERM_Type ∗const base, uint8_t channel, uint32_↩t ∗addressPtr)

  Gets the address of the last ECC event in Memory n and ECC event.

3.23.2 Macro Definition Documentation

3.23.2.1 ERM_CHANNELS_OFFSET_SIZE

```
#define ERM_CHANNELS_OFFSET_SIZE (4U)
```

The distance between channels.

3.23.2.2 ERM_NCE_START

```
#define ERM_NCE_START (30U)
```

Start bit of non correctable error.

3.23.2.3 ERM_SBC_START

```
#define ERM_SBC_START (31U)
```

Start bit of single bit correction.

3.23.3 Enumeration Type Documentation

3.23.3.1 erm_ecc_event_t

```
enum erm_ecc_event_t
```

ERM types of ECC events Implements : erm_ecc_event_t_Class.

Enumerator

ERM_EVENT_NONE	None events
ERM_EVENT_SINGLE_BIT	Single-bit correction ECC events
ERM_EVENT_NON_CORRECTABLE	Non-correctable ECC events

3.23.4 Function Documentation

3.23.4.1 ERM_HAL_ClearEvent()

```
static void ERM_HAL_ClearEvent (
             ERM_Type *const base,
             uint8_t channel,
             erm_ecc_event_t eccEvent )  [inline], [static]
```

Clears error event and the corresponding interrupt notification.

This function clears error event and the corresponding interrupt notification

Parameters

in	*base*	The ERM peripheral base address
in	*channel*	The configured memory channel
in	*eccEvent*	The examined event Implements : ERM_HAL_ClearEvent_Activity

3.23.4.2 ERM_HAL_EnableEventInterrupt()

```
static void ERM_HAL_EnableEventInterrupt (
             ERM_Type *const base,
             uint8_t channel,
             erm_ecc_event_t eccEvent,
             bool enable )  [inline], [static]
```

Enables Memory n interrupt event.

This function enables Memory n interrupt event

Parameters

in	*base*	The ERM peripheral base address
in	*channel*	The configured memory channel
in	*eccEvent*	The configured event
in	*enable*	Enable interrupt event • true: Interrupt event is enabled • false: Interrupt event is disabled Implements : ERM_HAL_EnableEventInterrupt_Activity

3.23.4.3 ERM_HAL_GetErrorDetail()

```
erm_ecc_event_t ERM_HAL_GetErrorDetail (
              const ERM_Type *const base,
              uint8_t channel,
              uint32_t * addressPtr )
```

Gets the address of the last ECC event in Memory n and ECC event.

This function gets the address of the last ECC event in Memory n and ECC event

Parameters

in	*base*	The ERM peripheral base address
in	*channel*	The examined memory channel
out	*addressPtr*	The pointer to address of the last ECC event in Memory n with ECC event

Returns

The last occurred ECC event

3.23.4.4 ERM_HAL_GetLastErrorAddress()

```
static uint32_t ERM_HAL_GetLastErrorAddress (
              const ERM_Type *const base,
              uint8_t channel )  [inline], [static]
```

Gets the address of the last ECC event in Memory n.

This function gets the address of the last ECC event in Memory n

Parameters

in	*base*	The ERM peripheral base address
in	*channel*	The examined memory channel

Returns

Address of the last ECC event Implements : ERM_HAL_GetLastErrorAddress_Activity

3.23.4.5 ERM_HAL_Init()

```
void ERM_HAL_Init (
            ERM_Type *const base )
```

Initializes the ERM module.

This function initializes the module to default configuration, the configuration register is initialized with interrupt notification disabled for all channels and the status register events are cleared

Parameters

in	*base*	The ERM peripheral base address

3.23.4.6 ERM_HAL_IsEventDetected()

```
static bool ERM_HAL_IsEventDetected (
            const ERM_Type *const base,
            uint8_t channel,
            erm_ecc_event_t eccEvent ) [inline], [static]
```

Checks if the Memory n error event is detected.

This function checks if the Memory n error event is detected

Parameters

in	*base*	The ERM peripheral base address
in	*channel*	The examined memory channel
in	*eccEvent*	The examined event

Returns

The status of Memory n error event

- true: Error event on Memory n detected
- false: No error event on Memory n detected Implements : ERM_HAL_IsEventDetected_Activity

3.23.4.7 ERM_HAL_IsEventInterruptEnabled()

```
static bool ERM_HAL_IsEventInterruptEnabled (
            const ERM_Type *const base,
            uint8_t channel,
            erm_ecc_event_t eccEvent )  [inline], [static]
```

Checks if the Memory n interrupt event is enabled.

This function checks if the Memory n interrupt event is enabled

Parameters

in	*base*	The ERM peripheral base address
in	*channel*	The examined memory channel
in	*eccEvent*	The examined event

Returns

Interrupt event

- true: Interrupt event is enabled
- false: Interrupt event is disabled Implements : ERM_HAL_IsEventInterruptEnabled_Activity

3.24 EWM Driver

3.24.1 Detailed Description

External Watchdog Monitor Peripheral Driver.

Hardware background

Features:

- Independent LPO clock source
- Programmable time-out period specified in terms of number of EWM LPO clock cycles.
- Windowed refresh option
  - Provides robust check that program flow is faster than expected.
  - Programmable window.
  - Refresh outside window leads to assertion of EWM_out.
- Robust refresh mechanism
  - Write values of **0xB4** and **0x2C** to EWM Refresh Register within 15 (**EWM_service_time**) peripheral bus clock cycles.
- One output port, **EWM_out**, when asserted is used to reset or place the external circuit into safe mode
- One Input port, **EWM_in**, allows an external circuit to control the **EWM_out** signal.

The EWM can be initialized only once as all the configuration registers are write once per reset

Clocking and pin configuration

The EWM Driver does not handle clock setup (from PCC) or any kind of pin configuration (done by PORT module). This is handled by the Clock Manager and PORT module, respectively. The driver assumes that correct clock configurations have been made, so it is the user's responsibility to set up clocking and pin configurations correctly.

Interrupts

The EWM module can generate interrupts, if enabled on `EWM_DRV_Init()` but they are not handled by the driver. The EWM shares the interrupt vector with the Watchdog Timer. The following code snippet is an example of how enable the interrupt and assign a handler:

```
/* EWM and watchdog interrupt service routine */
void EWM_Watchdog_ISR()
{
     /* Do something(e.g perform a clean reset) */
     ...
}
int main()
 {
     /* Init clocks, pins, other modules */
     ...
     /* Install interrupt handler for EWM and Watchdog */
     INT_SYS_InstallHandler(WDOG_EWM_IRQn, &EWM_Watchdog_ISR, (
      isr_t *)0);
     /* Enable the interrupt */
     INT_SYS_EnableIRQ(WDOG_EWM_IRQn);

     /* Init EWM */
     ...
     /* Infinite loop*/
     while(1)
     {
         /* Do something until the counter needs to be refreshed */
         ...
         /* Refresh the counter */
         EWM_DRV_Refresh(EWM_INSTANCE);
     }
 }
```

Using the EWM driver in your application

```
/* Declare the EWM instance you want to use */
#define EWM_INSTANCE    0UL

int main()
{
    /* Declare the EWM configuration structure */
    ewm_init_config_t ewmConfig;
    /* Variable where to store the init status */
    status_t ewmStatus;
    /* Init clocks, pins, other modules */
    ...

    /* Get the default configuration values */
    EWM_DRV_GetDefaultConfig(&ewmConfig);
    /* Init the module instance */
    ewmStatus = EWM_DRV_Init(EWM_INSTANCE, &ewmConfig);

    /* Infinite loop*/
    while(1)
    {
        /* Do something until the counter needs to be refreshed */
        ...
        /* Refresh the counter */
        EWM_DRV_Refresh(EWM_INSTANCE);
    }
}
```

Data Structures

- struct ewm_init_config_t

EWM Driver API

- status_t EWM_DRV_Init (uint32_t instance, const ewm_init_config_t ∗config)

  Init EWM. This method initializes EWM instance to the configuration from the passed structure. The user must make sure that the clock is enabled. This is the only method needed to be called to start the module.
- void EWM_DRV_GetDefaultConfig (ewm_init_config_t ∗config)

  Init configuration structure to default values.
- void EWM_DRV_Refresh (uint32_t instance)

  Refresh EWM. This method needs to be called within the window period specified by the Compare Low and Compare High registers.

3.24.2 Data Structure Documentation

3.24.2.1 struct ewm_init_config_t

Data Fields

- ewm_in_assert_logic_t assertLogic
- bool interruptEnable
- uint8_t prescaler
- uint8_t compareLow
- uint8_t compareHigh

Field Documentation

3.24.2.1.1 assertLogic

```
ewm_in_assert_logic_t assertLogic
```

Assert logic for EWM input pin

3.24.2.1.2 compareHigh

```
uint8_t compareHigh
```

Compare high value

3.24.2.1.3 compareLow

```
uint8_t compareLow
```

Compare low value

3.24.2.1.4 interruptEnable

```
bool interruptEnable
```

Enable EWM interrupt

3.24.2.1.5 prescaler

```
uint8_t prescaler
```

EWM clock prescaler

3.24.3 Function Documentation

3.24.3.1 EWM_DRV_GetDefaultConfig()

```
void EWM_DRV_GetDefaultConfig (
            ewm_init_config_t * config )
```

Init configuration structure to default values.

Parameters

out	*config*	Pointer to the configuration structure to initialize

Returns

None

3.24.3.2 EWM_DRV_Init()

```
status_t EWM_DRV_Init (
            uint32_t instance,
            const ewm_init_config_t * config )
```

Init EWM. This method initializes EWM instance to the configuration from the passed structure. The user must make sure that the clock is enabled. This is the only method needed to be called to start the module.

Example configuration structure:

```
ewm_init_config_t ewmUserCfg = {
    .assertLogic         = EWM_IN_ASSERT_ON_LOGIC_ZERO,
    .interruptEnable     = true,
    .prescaler           = 128,
    .compareLow          = 0,
    .compareHigh         = 254
};
```

This configuration will enable the peripheral, with input pin configured to assert on logic low, interrupt enabled, prescaler 128 and maximum refresh window.

The EWM can be initialized only once per CPU reset as the registers are write once.

Parameters

in	*instance*	EWM instance number
in	*config*	Pointer to the module configuration structure.

Returns

status_t Will return the status of the operation:

- STATUS_SUCCESS if the operation is successful
- STATUS_ERROR if the windows values are not correct or if the instance is already enabled

3.24.3.3 EWM_DRV_Refresh()

```
void EWM_DRV_Refresh (
              uint32_t instance )
```

Refresh EWM. This method needs to be called within the window period specified by the Compare Low and Compare High registers.

Parameters

in	*instance*	EWM instance number

Returns

None

3.25 EWM HAL

3.25.1 Detailed Description

External Watchdog Monitor Hardware Abstraction Layer.

This HAL provides low-level access to all hardware features of the EWM.

Enumerations

- enum ewm_in_assert_logic_t { EWM_IN_ASSERT_DISABLED = 0x00U, EWM_IN_ASSERT_ON_LOGIC←_ZERO = 0x01U, EWM_IN_ASSERT_ON_LOGIC_ONE = 0x02U }

    EWM input pin configuration Configures if the input pin is enabled and when is asserted Implements : ewm_in_←assert_logic_t_Class.

External Watchdog Module HAL

- void EWM_HAL_Init (EWM_Type ∗const base, bool interruptEnable, ewm_in_assert_logic_t assertLogic, bool enable)

    Init EWM. This method configures the EWM instance Control Register fields such as interrupt enable, input pin, instance enablement. The user must make sure that the prescaler, compare high and compare low registers are configured prior to this function call.
- ewm_in_assert_logic_t EWM_HAL_GetInputPinAssertLogic (const EWM_Type ∗const base)

    Get the Input pin assert logic.
- static void EWM_HAL_Refresh (EWM_Type ∗const base)

    Refresh EWM. This method needs to be called within the window period specified by the Compare Low and Compare High registers.
- static bool EWM_HAL_IsInterruptEnabled (const EWM_Type ∗const base)

    Get the Interrupt Enable bit.
- static bool EWM_HAL_IsEnabled (const EWM_Type ∗const base)

    Get the EWM enable bit.
- static uint8_t EWM_HAL_GetControl (const EWM_Type ∗const base)

    Get the Control register Value.
- static void EWM_HAL_SetCompareLow (EWM_Type ∗const base, uint8_t value)

    Set the Compare Low Value. This register can be only written once after a CPU reset. The user must make sure that the Compare High is greater than Compare Low value.
- static uint8_t EWM_HAL_GetCompareLow (const EWM_Type ∗const base)

    Get the Compare Low Value.
- static void EWM_HAL_SetCompareHigh (EWM_Type ∗const base, uint8_t value)

    Set the Compare High Value. This register can be only written once after a CPU reset. The user must make sure that the Compare High is greater than Compare Low value Note: The maximum Compare High value is 0xFE.
- static uint8_t EWM_HAL_GetCompareHigh (const EWM_Type ∗const base)

    Get the Compare High Value.
- static void EWM_HAL_SetPrescaler (EWM_Type ∗const base, uint8_t value)

    Set the Clock Prescaler Value. This register can be only written once after a CPU reset and it must be written before enabling the EWM.
- static uint8_t EWM_HAL_GetPrescaler (const EWM_Type ∗const base)

    Get the Clock Prescaler Value.

3.25.2 Enumeration Type Documentation

3.25.2.1 ewm_in_assert_logic_t

`enum ewm_in_assert_logic_t`

EWM input pin configuration Configures if the input pin is enabled and when is asserted Implements : ewm_in_←assert_logic_t_Class.

Enumerator

EWM_IN_ASSERT_DISABLED	Input pin disabled
EWM_IN_ASSERT_ON_LOGIC_ZERO	Input pin asserts EWM when on logic 0
EWM_IN_ASSERT_ON_LOGIC_ONE	Input pin asserts EWM when on logic 1

3.25.3 Function Documentation

3.25.3.1 EWM_HAL_GetCompareHigh()

```
static uint8_t EWM_HAL_GetCompareHigh (
            const EWM_Type *const base )  [inline], [static]
```

Get the Compare High Value.

Parameters

in	*base*	EWM base pointer

Returns

Value stored in Compare High register Implements : EWM_HAL_GetCompareHigh_Activity

3.25.3.2 EWM_HAL_GetCompareLow()

```
static uint8_t EWM_HAL_GetCompareLow (
            const EWM_Type *const base )  [inline], [static]
```

Get the Compare Low Value.

Parameters

in	*base*	EWM base pointer

Returns

Value stored in Compare Low register Implements : EWM_HAL_GetCompareLow_Activity

3.25.3.3 EWM_HAL_GetControl()

```
static uint8_t EWM_HAL_GetControl (
            const EWM_Type *const base )  [inline], [static]
```

Get the Control register Value.

Parameters

in	*base*	EWM base pointer

Returns

Value stored in Control register Implements : EWM_HAL_GetControl_Activity

3.25.3.4 EWM_HAL_GetInputPinAssertLogic()

```
ewm_in_assert_logic_t EWM_HAL_GetInputPinAssertLogic (
            const EWM_Type *const base )
```

Get the Input pin assert logic.

Parameters

in	*base*	EWM base pointer

Returns

The input pin assert logic:

- EWM_IN_ASSERT_DISABLED - EWM in disabled
- EWM_IN_ASSERT_ON_LOGIC_ZERO - EWM is asserted when EWM_in is logic 0
- EWM_IN_ASSERT_ON_LOGIC_ONE - EWM is asserted when EWM_in is logic 1

3.25.3.5 EWM_HAL_GetPrescaler()

```
static uint8_t EWM_HAL_GetPrescaler (
            const EWM_Type *const base )  [inline], [static]
```

Get the Clock Prescaler Value.

Parameters

in	*base*	EWM base pointer

Returns

Value stored in Clock Prescaler register Implements : EWM_HAL_GetPrescaler_Activity

3.25.3.6 EWM_HAL_Init()

```
void EWM_HAL_Init (
            EWM_Type *const base,
            bool interruptEnable,
            ewm_in_assert_logic_t assertLogic,
            bool enable )
```

Init EWM. This method configures the EWM instance Control Register fields such as interrupt enable, input pin, instance enablement. The user must make sure that the prescaler, compare high and compare low registers are configured prior to this function call.

Parameters

in	*base*	EWM base pointer
in	*interruptEnable*	Enable or disable EWM interrupt
in	*assertLogic*	Configure when the EWM input pin asserts
in	*enable*	Enable or disable the EWM instance

3.25.3.7 EWM_HAL_IsEnabled()

```
static bool EWM_HAL_IsEnabled (
            const EWM_Type *const base )  [inline], [static]
```

Get the EWM enable bit.

Parameters

in	*base*	EWM base pointer

Returns

The state of the device enable bit:

- false - EWM disabled
- true - EWM enabled Implements : EWM_HAL_IsEnabled_Activity

3.25.3.8 EWM_HAL_IsInterruptEnabled()

```
static bool EWM_HAL_IsInterruptEnabled (
            const EWM_Type *const base )  [inline], [static]
```

Get the Interrupt Enable bit.

Parameters

in	*base*	EWM base pointer

Returns

The state of the interrupt enable bit:

- false - interrupt disabled
- true - interrupt enabled Implements : EWM_HAL_IsInterruptEnabled_Activity

3.25.3.9 EWM_HAL_Refresh()

```
static void EWM_HAL_Refresh (
            EWM_Type *const base )  [inline], [static]
```

Refresh EWM. This method needs to be called within the window period specified by the Compare Low and Compare High registers.

Parameters

in	*base*	EWM base pointer Implements : EWM_HAL_Refresh_Activity

3.25.3.10 EWM_HAL_SetCompareHigh()

```
static void EWM_HAL_SetCompareHigh (
            EWM_Type *const base,
            uint8_t value ) [inline], [static]
```

Set the Compare High Value. This register can be only written once after a CPU reset. The user must make sure that the Compare High is greater than Compare Low value Note: The maximum Compare High value is 0xFE.

Parameters

in	*base*	EWM base pointer
in	*value*	Value to write into Compare High register Implements : EWM_HAL_SetCompareHigh_Activity

3.25.3.11 EWM_HAL_SetCompareLow()

```
static void EWM_HAL_SetCompareLow (
            EWM_Type *const base,
            uint8_t value ) [inline], [static]
```

Set the Compare Low Value. This register can be only written once after a CPU reset. The user must make sure that the Compare High is greater than Compare Low value.

Parameters

in	*base*	EWM base pointer
in	*value*	Value to write into Compare Low register Implements : EWM_HAL_SetCompareLow_Activity

3.25.3.12 EWM_HAL_SetPrescaler()

```
static void EWM_HAL_SetPrescaler (
            EWM_Type *const base,
            uint8_t value ) [inline], [static]
```

Set the Clock Prescaler Value. This register can be only written once after a CPU reset and it must be written before enabling the EWM.

Parameters

in	*base*	EWM base pointer
in	*value*	Prescaler Value Implements : EWM_HAL_SetPrescaler_Activity

3.26 Error Injection Module (EIM)

3.26.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Drivers for the Error Injection Module (EIM) of S32 MCU
.

The Error Injection Module is mainly used for diagnostic purposes. It provides a method for diagnostic coverage of the peripheral memories.
The Error Injection Module (EIM) provides support for inducing single-bit and multi-bit inversions on read data when accessing peripheral RAMs. Injecting faults on memory accesses can be used to exercise the SEC-DED ECC function of the related system.

Important Note:

1. Make sure that STACK memory is located in RAM different than where EIM will inject a non-correctable error.
2. For single bit error generation, flip only one bit out of DATA_MASK or CHKBIT_MASK bit-fields in EIM control registers.
3. For Double bit error generation, Flip only two bits out of DATA_MASK or CHKBIT_MASK bit-fields in EIM control registers.
4. If more than 2 bits are flipped that there is no guarantee in design that what type of error get generated.

Modules

- EIM Driver

  Error Injection Module Peripheral Driver.
  EIM PD provides a set of high-level APIs/services to configure the Error Injection Module (EIM) module.
- EIM HAL

  Error Injection Module Hardware Abstraction Level. EIM HAL provides low level APIs for reading and writing register bit-fields belonging to the EIM module.

3.27 Error Reporting Module (ERM)

3.27.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Drivers for the Error Reporting Module (ERM) module of S32 SDK devices.

The Error Reporting Module (ERM) provides information and optional interrupt notification on memory errors events associated with ECC (Error Correction Code).
The ERM includes these features:
Capture address information on single-bit correction and non-correctable ECC events.
Optional interrupt notification on captured ECC events.
Support for ECC event capturing for memory sources, with individual reporting fields and interrupt configuration per memory channel.

3.27.2 ERM Driver Initialization

In order to be able to use the error reporting in your application, the first thing to do is initializing it with user configuration input. This is done by calling the **ERM_DRV_Init** function. Note that: channelCnt takes values between 1 and the maximum channel count supported by the hardware.

3.27.3 ERM Driver Operation

After ERM initialization, the ERM_DRV_SetInterruptConfig() shall be used to set interrupt notification based on interrupt notification configuration.
The ERM_DRV_GetInterruptConfig() shall be used to get the current interrupt configuration of the available events (which interrupts are enabled/disabled).

The ERM_DRV_GetErrorDetail() shall be used to get the address of the last ECC event in Memory n and ECC event.

The ERM_DRV_ClearEvent() shall be used to clear both the record of an event and the corresponding interrupt notification.

This is example code to configure the ERM driver:

```
/* Device instance number */
#define INST_ERM1 (0U)

/* The number of configured channel(s) */
#define ERM_NUM_OF_CFG_CHANNEL (2U)

/* Interrupt configuration 0 */
const erm_interrupt_config_t erm1_Interrupt1 =
{
    .enableSingleCorrection = false,
    .enableNonCorrectable   = true,
};

/* Interrupt configuration 1 */
const erm_interrupt_config_t erm1_Interrupt3 =
{
    .enableSingleCorrection = true,
    .enableNonCorrectable   = true,
};


/* User configuration */
const erm_user_config_t erm1_InitConfig[] =
{
    /* Channel 0U */
    {
        .channel      = 0U,
        .interruptCfg = &erm1_Interrupt1,
    },

    /* Channel 1U */
    {
        .channel      = 1U,
        .interruptCfg = &erm1_Interrupt3,
    }
};

int main()
{
    /* Initializes the ERM module */
    ERM_DRV_Init(INST_ERM1, ERM_NUM_OF_CFG_CHANNEL, erm1_InitConfig);
    ...
    /* De-Initializes the ERM module */
    ERM_DRV_Deinit(INST_ERM1);
    ...
    return 0;
}

/* Interrupt handler */
/* Interrupt handler for single bit */
void ERM_single_fault_IRQHandler()
{
    /* Clears the event for channel 1 */
    ERM_DRV_ClearEvent(INST_ERM1, 1U, ERM_EVENT_SINGLE_BIT);
    ...
}

/* Interrupt handler for non correctable */
void ERM_double_fault_IRQHandler()
{
    /* Clears the event for channel 0 */
    ERM_DRV_ClearEvent(INST_ERM1, 0U,
      ERM_EVENT_NON_CORRECTABLE);
    /* Clears the event for channel 1 */
    ERM_DRV_ClearEvent(INST_ERM1, 1U,
      ERM_EVENT_NON_CORRECTABLE);
    ...
}
```

Modules

- ERM Driver

  Error Reporting Module Peripheral Driver.
- ERM HAL

  Error Reporting Module Hardware Abstraction Layer.

3.28 External Watchdog Monitor (EWM)

3.28.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Drivers for the External Watchdog Monitor (EWM) module of S32K devices.

For safety, a redundant watchdog system, External Watchdog Monitor (EWM), is designed to monitor external circuits, as well as the MCU software flow. This provides a back-up mechanism to the internal watchdog that resets the MCU's CPU and peripherals.

The EWM differs from the internal watchdog in that it does not reset the MCU's CPU and peripherals. The EWM if allowed to time-out, provides an independent EWM_out pin that when asserted resets or places an external circuit into a safe mode. The CPU resets the EWM counter that is logically ANDed with an external digital input pin. This pin allows an external circuit to influence the reset_out signal.

Modules

- EWM Driver

    External Watchdog Monitor Peripheral Driver.
- EWM HAL

    External Watchdog Monitor Hardware Abstraction Layer.

3.29 FTM Driver

3.29.1 Detailed Description

FlexTimer Peripheral Driver.

Data Structures

- struct ftm_state_t

    FlexTimer state structure of the driver. More...
- struct ftm_pwm_sync_t

    FlexTimer Registers sync parameters Please don't use software and hardware trigger simultaneously Implements : ftm_pwm_sync_t_Class. More...
- struct ftm_user_config_t

    Configuration structure that the user needs to set. More...
- struct ftm_timer_param_t

    FlexTimer driver timer mode configuration structure. More...
- struct ftm_pwm_ch_fault_param_t

    FlexTimer driver PWM Fault channel parameters. More...
- struct ftm_pwm_fault_param_t

    FlexTimer driver PWM Fault parameter. More...
- struct ftm_independent_ch_param_t

    FlexTimer driver independent PWM parameter. More...
- struct ftm_combined_ch_param_t

    FlexTimer driver combined PWM parameter. More...
- struct ftm_pwm_param_t

    FlexTimer driver PWM parameters. More...
- struct ftm_input_ch_param_t

    FlexTimer driver Input capture parameters for each channel. More...
- struct ftm_input_param_t

    FlexTimer driver input capture parameters. More...
- struct ftm_output_cmp_ch_param_t

    FlexTimer driver PWM parameters. More...
- struct ftm_output_cmp_param_t

    FlexTimer driver PWM parameters. More...
- struct ftm_phase_params_t

    FlexTimer quadrature decoder channel parameters. More...
- struct ftm_quad_decode_config_t

    FTM quadrature configure structure. More...
- struct ftm_quad_decoder_state_t

    FTM quadrature state(counter value and flags) More...

Macros

- #define FTM_MAX_DUTY_CYCLE (0x8000U)

    Maximum value for PWM duty cycle.
- #define FTM_DUTY_TO_TICKS_SHIFT (15U)

    Shift value which converts duty to ticks.

Typedefs

- typedef void(∗ ftm_channel_event_callback_t) (void ∗userData)

  Channel event callback function.

Enumerations

- enum ftm_input_op_mode_t { FTM_EDGE_DETECT = 0U, FTM_SIGNAL_MEASUREMENT = 1U, FTM_↩ NO_OPERATION = 2U }

  FTM status.
- enum ftm_signal_measurement_mode_t {
  FTM_NO_MEASUREMENT = 0x00U, FTM_RISING_EDGE_PERIOD_MEASUREMENT = 0x01U, FTM_F↩ ALLING_EDGE_PERIOD_MEASUREMENT = 0x02U, FTM_PERIOD_ON_MEASUREMENT = 0x03U,
  FTM_PERIOD_OFF_MEASUREMENT = 0x04U }

  FlexTimer input capture measurement type for dual edge input capture.
- enum ftm_output_compare_update_t { FTM_RELATIVE_VALUE = 0x00U, FTM_ABSOLUTE_VALUE = 0x01U }

  FlexTimer input capture type of the next output compare value.
- enum ftm_pwm_update_option_t { FTM_PWM_UPDATE_IN_DUTY_CYCLE = 0x00U, FTM_PWM_UPDA↩ TE_IN_TICKS = 0x01U }

  FlexTimer Configure type of PWM update in the duty cycle or in ticks.
- enum ftm_config_mode_t {
  FTM_MODE_NOT_INITIALIZED = 0x00U, FTM_MODE_INPUT_CAPTURE = 0x01U, FTM_MODE_OUT↩ PUT_COMPARE = 0x02U, FTM_MODE_EDGE_ALIGNED_PWM = 0x03U,
  FTM_MODE_CEN_ALIGNED_PWM = 0x04U, FTM_MODE_QUADRATURE_DECODER = 0x05U, FTM_↩ MODE_UP_TIMER = 0x06U, FTM_MODE_UP_DOWN_TIMER = 0x07U }

  FlexTimer operation mode.
- enum ftm_output_compare_mode_t { FTM_DISABLE_OUTPUT = 0x00U, FTM_TOGGLE_ON_MATCH = 0x01U, FTM_CLEAR_ON_MATCH = 0x02U, FTM_SET_ON_MATCH = 0x03U }

  FlexTimer Mode configuration for output compare mode.
- enum ftm_edge_alignment_mode_t { FTM_NO_PIN_CONTROL = 0x00U, FTM_RISING_EDGE = 0x01U, FTM_FALLING_EDGE = 0x02U, FTM_BOTH_EDGES = 0x03U }

  FlexTimer input capture edge mode, rising edge, or falling edge.

Functions

- status_t FTM_DRV_Init (uint32_t instance, const ftm_user_config_t ∗info, ftm_state_t ∗state)

  Initializes the FTM driver.
- status_t FTM_DRV_Deinit (uint32_t instance)

  Shuts down the FTM driver.
- status_t FTM_DRV_InitCounter (uint32_t instance, const ftm_timer_param_t ∗timer)

  Initialize the FTM counter.
- status_t FTM_DRV_CounterStart (uint32_t instance)

  Starts the FTM counter.
- status_t FTM_DRV_CounterStop (uint32_t instance)

  Stops the FTM counter.
- uint32_t FTM_DRV_CounterRead (uint32_t instance)

  Reads back the current value of the FTM counter.
- status_t FTM_DRV_DeinitPwm (uint32_t instance)

  Stops all PWM channels .
- status_t FTM_DRV_InitPwm (uint32_t instance, const ftm_pwm_param_t ∗param)

Configures the duty cycle and frequency and starts outputting the PWM on all channels configured in param.
- status_t FTM_DRV_UpdatePwmChannel (uint32_t instance, uint8_t channel, ftm_pwm_update_option_↩ t typeOfUpdate, uint16_t firstEdge, uint16_t secondEdge, bool softwareTrigger)

  This function updates the waveform output in PWM mode (duty cycle and phase).
- status_t FTM_DRV_UpdatePwmPeriod (uint32_t instance, ftm_pwm_update_option_t typeOfUpdate, uint32_t newValue, bool softwareTrigger)

  This function will update the new period in the frequency or in the counter value into mode register which modify the period of PWM signal on the channel output.
- status_t FTM_DRV_MaskOutputChannels (uint32_t instance, uint32_t channelsMask, bool softwareTrigger)

  This function will mask the output of the channels and at match events will be ignored by the masked channels.
- status_t FTM_DRV_SetInitialCounterValue (uint32_t instance, uint16_t counterValue, bool softwareTrigger)

  This function configure the initial counter value. The counter will get this value after an overflow event.
- status_t FTM_DRV_SetHalfCycleReloadPoint (uint32_t instance, uint16_t reloadPoint, bool softwareTrigger)

  This function configure the value of the counter which will generates an reload point.
- status_t FTM_DRV_SetSoftOutChnValue (uint32_t instance, uint8_t channelsValues, bool softwareTrigger)

  This function will force the output value of a channel to a specific value. Before using this function it's mandatory to mask the match events using FTM_DRV_MaskOutputChannels and to enable software output control using FTM_↩ DRV_SetSoftwareOutputChannelControl.
- status_t FTM_DRV_SetSoftwareOutputChannelControl (uint32_t instance, uint8_t channelsMask, bool softwareTrigger)

  This function will configure which output channel can be software controlled.
- status_t FTM_DRV_SetInvertingControl (uint32_t instance, uint8_t channelsPairMask, bool softwareTrigger)

  This function will configure if the second channel of a pair will be inverted or not.
- status_t FTM_DRV_SetModuloCounterValue (uint32_t instance, uint16_t counterValue, bool softwareTrigger)

  This function configure the maximum counter value.
- status_t FTM_DRV_SetSync (uint32_t instance, const ftm_pwm_sync_t ∗param)

  This function configures sync mechanism for some FTM registers (MOD, CNINT, HCR, CnV, OUTMASK, INVCTRL, SWOCTRL).
- status_t FTM_DRV_InitOutputCompare (uint32_t instance, const ftm_output_cmp_param_t ∗param)

  Configures the FTM to generate timed pulses(Output compare mode).
- status_t FTM_DRV_DeinitOutputCompare (uint32_t instance, const ftm_output_cmp_param_t ∗param)

  Disables compare match output control and clears FTM timer configuration.
- status_t FTM_DRV_UpdateOutputCompareChannel (uint32_t instance, uint8_t channel, uint16_t next↩ ComparematchValue, ftm_output_compare_update_t update, bool softwareTrigger)

  Sets the next compare match value based on the current counter value.
- status_t FTM_DRV_InitInputCapture (uint32_t instance, const ftm_input_param_t ∗param)

  Configures Channel Input Capture for either getting time-stamps on edge detection or on signal measurement . When the edge specified in the captureMode argument occurs on the channel the FTM counter is captured into the CnV register. The user will have to read the CnV register separately to get this value. The filter function is disabled if the filterVal argument passed in is 0. The filter function is available only on channels 0,1,2,3.
- status_t FTM_DRV_DeinitInputCapture (uint32_t instance, const ftm_input_param_t ∗param)

  Disables input capture mode and clears FTM timer configuration.
- uint16_t FTM_DRV_GetInputCaptureMeasurement (uint32_t instance, uint8_t channel)

  This function is used to calculate the measurement and/or time stamps values which are read from the C(n, n+1)V registers and stored to the static buffers.
- status_t FTM_DRV_StartNewSignalMeasurement (uint32_t instance, uint8_t channel)

  Starts new single-shot signal measurement of the given channel.
- status_t FTM_DRV_QuadDecodeStart (uint32_t instance, const ftm_quad_decode_config_t ∗config)

  Configures the quadrature mode and starts measurement.
- status_t FTM_DRV_QuadDecodeStop (uint32_t instance)

  De-activates the quadrature decode mode.
- ftm_quad_decoder_state_t FTM_DRV_QuadGetState (uint32_t instance)

  Return the current quadrature decoder state (counter value, overflow flag and overflow direction)

- uint32_t FTM_DRV_GetFrequency (uint32_t instance)

    Retrieves the frequency of the clock source feeding the FTM counter.
- uint16_t FTM_DRV_ConvertFreqToPeriodTicks (uint32_t instance, uint32_t freqencyHz)

    This function is used to covert the given frequency to period in ticks.

Variables

- FTM_Type ∗const g_ftmBase [FTM_INSTANCE_COUNT]

    Table of base addresses for FTM instances.
- const IRQn_Type g_ftmIrqId [FTM_INSTANCE_COUNT][FEATURE_FTM_CHANNEL_COUNT]

    Interrupt vectors for the FTM peripheral.
- const IRQn_Type g_ftmFaultIrqId [FTM_INSTANCE_COUNT]
- const IRQn_Type g_ftmOverflowIrqId [FTM_INSTANCE_COUNT]
- const IRQn_Type g_ftmReloadIrqId [FTM_INSTANCE_COUNT]

3.29.2 Data Structure Documentation

3.29.2.1 struct ftm_state_t

FlexTimer state structure of the driver.

Implements : ftm_state_t_Class

Data Fields

- ftm_clock_source_t ftmClockSource
- ftm_config_mode_t ftmMode
- uint16_t ftmPeriod
- uint32_t ftmSourceClockFrequency
- uint16_t measurementResults [FEATURE_FTM_CHANNEL_COUNT]
- void ∗ channelsCallbacksParams [FEATURE_FTM_CHANNEL_COUNT]
- ftm_channel_event_callback_t channelsCallbacks [FEATURE_FTM_CHANNEL_COUNT]

Field Documentation

3.29.2.1.1 channelsCallbacks

```
ftm_channel_event_callback_t channelsCallbacks[FEATURE_FTM_CHANNEL_COUNT]
```

Vector of callbacks for channels events

3.29.2.1.2 channelsCallbacksParams

```
void∗ channelsCallbacksParams[FEATURE_FTM_CHANNEL_COUNT]
```

Vector of callbacks parameters for channels events

3.29.2.1.3 ftmClockSource

```
ftm_clock_source_t ftmClockSource
```

Clock source used by FTM counter

3.29.2.1.4 ftmMode

```
ftm_config_mode_t ftmMode
```

Mode of operation for FTM

3.29.2.1.5 ftmPeriod

```
uint16_t ftmPeriod
```

This field is used only in PWM mode to store signal period

3.29.2.1.6 ftmSourceClockFrequency

```
uint32_t ftmSourceClockFrequency
```

The clock frequency is used for counting

3.29.2.1.7 measurementResults

```
uint16_t measurementResults[FEATURE_FTM_CHANNEL_COUNT]
```

This field is used only in input capture mode to store edges time stamps

3.29.2.2 struct ftm_pwm_sync_t

FlexTimer Registers sync parameters Please don't use software and hardware trigger simultaneously Implements : ftm_pwm_sync_t_Class.

Data Fields

- bool softwareSync
- bool hardwareSync0
- bool hardwareSync1
- bool hardwareSync2
- bool maxLoadingPoint
- bool minLoadingPoint
- ftm_reg_update_t inverterSync
- ftm_reg_update_t outRegSync
- ftm_reg_update_t maskRegSync
- ftm_reg_update_t initCounterSync
- bool autoClearTrigger
- ftm_pwm_sync_mode_t syncPoint

Field Documentation

3.29.2.2.1 autoClearTrigger

```
bool autoClearTrigger
```

Available only for hardware trigger

3.29.2.2.2 hardwareSync0

`bool hardwareSync0`

True - enable hardware 0 sync, False - disable hardware 0 sync

3.29.2.2.3 hardwareSync1

`bool hardwareSync1`

True - enable hardware 1 sync, False - disable hardware 1 sync

3.29.2.2.4 hardwareSync2

`bool hardwareSync2`

True - enable hardware 2 sync, False - disable hardware 2 sync

3.29.2.2.5 initCounterSync

`ftm_reg_update_t initCounterSync`

Configures CNTIN sync

3.29.2.2.6 inverterSync

`ftm_reg_update_t inverterSync`

Configures INVCTRL sync

3.29.2.2.7 maskRegSync

`ftm_reg_update_t maskRegSync`

Configures OUTMASK sync

3.29.2.2.8 maxLoadingPoint

`bool maxLoadingPoint`

True - enable maximum loading point, False - disable maximum loading point

3.29.2.2.9 minLoadingPoint

`bool minLoadingPoint`

True - enable minimum loading point, False - disable minimum loading point

3.29.2.2.10 outRegSync

`ftm_reg_update_t outRegSync`

Configures SWOCTRL sync

3.29.2.2.11 softwareSync

```
bool softwareSync
```

True - enable software sync, False - disable software sync

3.29.2.2.12 syncPoint

```
ftm_pwm_sync_mode_t syncPoint
```

Configure synchronization method (waiting next loading point or immediate)

3.29.2.3 struct ftm_user_config_t

Configuration structure that the user needs to set.

Implements : ftm_user_config_t_Class

Data Fields

- ftm_pwm_sync_t syncMethod
- ftm_config_mode_t ftmMode
- ftm_clock_ps_t ftmPrescaler
- ftm_clock_source_t ftmClockSource
- ftm_bdm_mode_t BDMMode
- bool isTofIsrEnabled
- bool enableInitializationTrigger

Field Documentation

3.29.2.3.1 BDMMode

```
ftm_bdm_mode_t BDMMode
```

Select FTM behavior in BDM mode

3.29.2.3.2 enableInitializationTrigger

```
bool enableInitializationTrigger
```

true: enable the generation of initialization trigger false: disable the generation of initialization trigger

3.29.2.3.3 ftmClockSource

```
ftm_clock_source_t ftmClockSource
```

Select clock source for FTM

3.29.2.3.4 ftmMode

```
ftm_config_mode_t ftmMode
```

Mode of operation for FTM

3.29.2.3.5 ftmPrescaler

```
ftm_clock_ps_t ftmPrescaler
```

Register pre-scaler options available in the ftm_clock_ps_t enumeration

3.29.2.3.6 isTofIsrEnabled

```
bool isTofIsrEnabled
```

true: enable interrupt, false: write interrupt is disabled

3.29.2.3.7 syncMethod

```
ftm_pwm_sync_t syncMethod
```

Register sync options available in the ftm_sync_method_t enumeration

3.29.2.4 struct ftm_timer_param_t

FlexTimer driver timer mode configuration structure.

Implements : ftm_timer_param_t_Class

Data Fields

- ftm_config_mode_t mode
- uint16_t initialValue
- uint16_t finalValue

Field Documentation

3.29.2.4.1 finalValue

```
uint16_t finalValue
```

Final counter value

3.29.2.4.2 initialValue

```
uint16_t initialValue
```

Initial counter value

3.29.2.4.3 mode

```
ftm_config_mode_t mode
```

FTM mode

3.29.2.5 struct ftm_pwm_ch_fault_param_t

FlexTimer driver PWM Fault channel parameters.

Implements : ftm_pwm_ch_fault_param_t_Class

Data Fields

- bool faultChannelEnabled
- bool faultFilterEnabled
- ftm_polarity_t ftmFaultPinPolarity

Field Documentation

3.29.2.5.1 faultChannelEnabled

```
bool faultChannelEnabled
```

Fault channel state

3.29.2.5.2 faultFilterEnabled

```
bool faultFilterEnabled
```

Fault channel filter state

3.29.2.5.3 ftmFaultPinPolarity

```
ftm_polarity_t ftmFaultPinPolarity
```

Channel output state on fault

3.29.2.6 struct ftm_pwm_fault_param_t

FlexTimer driver PWM Fault parameter.

Implements : ftm_pwm_fault_param_t_Class

Data Fields

- bool pwmOutputStateOnFault
- bool pwmFaultInterrupt
- uint8_t faultFilterValue
- ftm_fault_mode_t faultMode
- ftm_pwm_ch_fault_param_t ftmFaultChannelParam [FTM_FEATURE_FAULT_CHANNELS]

Field Documentation

3.29.2.6.1 faultFilterValue

```
uint8_t faultFilterValue
```

Fault filter value

3.29.2.6.2 faultMode

```
ftm_fault_mode_t faultMode
```

Fault mode

3.29.2.6.3 ftmFaultChannelParam

```
ftm_pwm_ch_fault_param_t ftmFaultChannelParam[FTM_FEATURE_FAULT_CHANNELS]
```

Fault channels configuration

3.29.2.6.4 pwmFaultInterrupt

```
bool pwmFaultInterrupt
```

PWM fault interrupt state

3.29.2.6.5 pwmOutputStateOnFault

```
bool pwmOutputStateOnFault
```

Output pin state on fault

3.29.2.7 struct ftm_independent_ch_param_t

FlexTimer driver independent PWM parameter.

Implements : ftm_independent_ch_param_t_Class

Data Fields

- uint8_t hwChannelId
- ftm_polarity_t polarity
- uint16_t uDutyCyclePercent
- bool enableExternalTrigger

Field Documentation

3.29.2.7.1 enableExternalTrigger

```
bool enableExternalTrigger
```

true: enable the generation of a trigger is used for on-chip modules false: disable the generation of a trigger

3.29.2.7.2 hwChannelId

```
uint8_t hwChannelId
```

Physical hardware channel ID

3.29.2.7.3 polarity

```
ftm_polarity_t polarity
```

PWM output polarity

3.29.2.7.4 uDutyCyclePercent

```
uint16_t uDutyCyclePercent
```

PWM pulse width, value should be between 0 (0%) to FTM_MAX_DUTY_CYCLE (100%)

3.29.2.8 struct ftm_combined_ch_param_t

FlexTimer driver combined PWM parameter.

Implements : ftm_combined_ch_param_t_Class

Data Fields

- uint8_t hwChannelId
- uint16_t firstEdge
- uint16_t secondEdge
- bool deadTime
- bool enableModifiedCombine
- ftm_polarity_t mainChannelPolarity
- bool enableSecondChannelOutput
- ftm_second_channel_polarity_t secondChannelPolarity
- bool enableExternalTrigger
- bool enableExternalTriggerOnNextChn

Field Documentation

3.29.2.8.1 deadTime

```
bool deadTime
```

Enable/disable dead time for channel

3.29.2.8.2 enableExternalTrigger

```
bool enableExternalTrigger
```

The generation of the channel (n) trigger true: enable the generation of a trigger on the channel (n) false: disable the generation of a trigger on the channel (n)

3.29.2.8.3 enableExternalTriggerOnNextChn

```
bool enableExternalTriggerOnNextChn
```

The generation of the channel (n+1) trigger true: enable the generation of a trigger on the channel (n+1) false: disable the generation of a trigger on the channel (n+1)

3.29.2.8.4 enableModifiedCombine

```
bool enableModifiedCombine
```

Enable/disable the modified combine mode for channels (n) and (n+1)

3.29.2.8.5 enableSecondChannelOutput

```
bool enableSecondChannelOutput
```

Select if channel (n+1) output is enabled/disabled

3.29.2.8.6 firstEdge

```
uint16_t firstEdge
```

First edge time. This time is relative to signal period. The value for this parameter is between 0 and FTM_MAX_↩
DUTY_CYCLE(0 = 0% from period and FTM_MAX_DUTY_CYCLE = 100% from period)

3.29.2.8.7 hwChannelId

```
uint8_t hwChannelId
```

Physical hardware channel ID for channel (n)

3.29.2.8.8 mainChannelPolarity

```
ftm_polarity_t mainChannelPolarity
```

Main channel polarity. For FTM_POLARITY_HIGH first output value is 0 and for FTM_POLAIRTY first output value is 1

3.29.2.8.9 secondChannelPolarity

```
ftm_second_channel_polarity_t secondChannelPolarity
```

Select channel (n+1) polarity relative to channel (n)

3.29.2.8.10 secondEdge

```
uint16_t secondEdge
```

Second edge time. This time is relative to signal period. The value for this parameter is between 0 and FTM_MA↩
X_DUTY_CYCLE(0 = 0% from period and FTM_MAX_DUTY_CYCLE = 100% from period)

3.29.2.9 struct ftm_pwm_param_t

FlexTimer driver PWM parameters.

Implements : ftm_pwm_param_t_Class

Data Fields

- uint8_t nNumIndependentPwmChannels
- uint8_t nNumCombinedPwmChannels
- ftm_config_mode_t mode
- uint8_t deadTimeValue
- ftm_deadtime_ps_t deadTimePrescaler
- uint32_t uFrequencyHZ
- const ftm_independent_ch_param_t ∗ pwmIndependentChannelConfig
- const ftm_combined_ch_param_t ∗ pwmCombinedChannelConfig
- const ftm_pwm_fault_param_t ∗ faultConfig

Field Documentation

3.29.2.9.1 deadTimePrescaler

```
ftm_deadtime_ps_t deadTimePrescaler
```

Dead time pre-scaler value[ticks]

3.29.2.9.2 deadTimeValue

```
uint8_t deadTimeValue
```

Dead time value in [ticks]

3.29.2.9.3 faultConfig

```
const ftm_pwm_fault_param_t∗ faultConfig
```

Configuration for PWM fault

3.29.2.9.4 mode

```
ftm_config_mode_t mode
```

FTM mode

3.29.2.9.5 nNumCombinedPwmChannels

```
uint8_t nNumCombinedPwmChannels
```

Number of combined PWM channels

3.29.2.9.6 nNumIndependentPwmChannels

```
uint8_t nNumIndependentPwmChannels
```

Number of independent PWM channels

3.29.2.9.7 pwmCombinedChannelConfig

```
const ftm_combined_ch_param_t* pwmCombinedChannelConfig
```

Configuration for combined PWM channels

3.29.2.9.8 pwmIndependentChannelConfig

```
const ftm_independent_ch_param_t* pwmIndependentChannelConfig
```

Configuration for independent PWM channels

3.29.2.9.9 uFrequencyHZ

```
uint32_t uFrequencyHZ
```

PWM period in Hz

3.29.2.10 struct ftm_input_ch_param_t

FlexTimer driver Input capture parameters for each channel.

Implements : ftm_input_ch_param_t_Class

Data Fields

- uint8_t hwChannelId
- ftm_input_op_mode_t inputMode
- ftm_edge_alignment_mode_t edgeAlignement
- ftm_signal_measurement_mode_t measurementType
- uint16_t filterValue
- bool filterEn
- bool continuousModeEn
- void * channelsCallbacksParams
- ftm_channel_event_callback_t channelsCallbacks

Field Documentation

3.29.2.10.1 channelsCallbacks

```
ftm_channel_event_callback_t channelsCallbacks
```

Vector of callbacks for channels events

3.29.2.10.2 channelsCallbacksParams

```
void* channelsCallbacksParams
```

Vector of callbacks parameters for channels events

3.29.2.10.3 continuousModeEn

```
bool continuousModeEn
```

Continuous measurement state

3.29.2.10.4 edgeAlignement

```
ftm_edge_alignment_mode_t edgeAlignement
```

Edge alignment Mode for signal measurement

3.29.2.10.5 filterEn

```
bool filterEn
```

Input capture filter state

3.29.2.10.6 filterValue

```
uint16_t filterValue
```

Filter Value

3.29.2.10.7 hwChannelId

```
uint8_t hwChannelId
```

Physical hardware channel ID

3.29.2.10.8 inputMode

```
ftm_input_op_mode_t inputMode
```

FlexTimer module mode of operation

3.29.2.10.9 measurementType

```
ftm_signal_measurement_mode_t measurementType
```

Measurement Mode for signal measurement

3.29.2.11 struct ftm_input_param_t

FlexTimer driver input capture parameters.

Implements : ftm_input_param_t_Class

Data Fields

- uint8_t nNumChannels
- uint16_t nMaxCountValue
- const ftm_input_ch_param_t ∗ inputChConfig

Field Documentation

3.29.2.11.1 inputChConfig

```
const ftm_input_ch_param_t* inputChConfig
```

Input capture channels configuration

3.29.2.11.2 nMaxCountValue

```
uint16_t nMaxCountValue
```

Maximum counter value. Minimum value is 0 for this mode

3.29.2.11.3 nNumChannels

```
uint8_t nNumChannels
```

Number of input capture channel used

3.29.2.12 struct ftm_output_cmp_ch_param_t

FlexTimer driver PWM parameters.

Implements : ftm_output_cmp_ch_param_t_Class

Data Fields

- uint8_t hwChannelId
- ftm_output_compare_mode_t chMode
- uint16_t comparedValue
- bool enableExternalTrigger

Field Documentation

3.29.2.12.1 chMode

```
ftm_output_compare_mode_t chMode
```

Channel output mode

3.29.2.12.2 comparedValue

```
uint16_t comparedValue
```

The compared value

3.29.2.12.3 enableExternalTrigger

```
bool enableExternalTrigger
```

true: enable the generation of a trigger is used for on-chip modules false: disable the generation of a trigger

3.29.2.12.4 hwChannelId

```
uint8_t hwChannelId
```

Physical hardware channel ID

3.29.2.13 struct ftm_output_cmp_param_t

FlexTimer driver PWM parameters.

Implements : ftm_output_cmp_param_t_Class

Data Fields

- uint8_t nNumOutputChannels
- ftm_config_mode_t mode
- uint16_t maxCountValue
- const ftm_output_cmp_ch_param_t ∗ outputChannelConfig

Field Documentation

3.29.2.13.1 maxCountValue

```
uint16_t maxCountValue
```

Maximum count value in ticks

3.29.2.13.2 mode

```
ftm_config_mode_t mode
```

FlexTimer PWM operation mode

3.29.2.13.3 nNumOutputChannels

```
uint8_t nNumOutputChannels
```

Number of output compare channels

3.29.2.13.4 outputChannelConfig

```
const ftm_output_cmp_ch_param_t* outputChannelConfig
```

Output compare channels configuration

3.29.2.14 struct ftm_phase_params_t

FlexTimer quadrature decoder channel parameters.

Implements : ftm_phase_params_t_Class

Data Fields

- bool phaseInputFilter
- uint8_t phaseFilterVal
- ftm_quad_phase_polarity_t phasePolarity

Field Documentation

3.29.2.14.1 phaseFilterVal

```
uint8_t phaseFilterVal
```

Filter value (if input filter is enabled)

3.29.2.14.2 phaseInputFilter

```
bool phaseInputFilter
```

True: disable phase filter, False: enable phase filter

3.29.2.14.3 phasePolarity

```
ftm_quad_phase_polarity_t phasePolarity
```

Phase polarity

3.29.2.15 struct ftm_quad_decode_config_t

FTM quadrature configure structure.

Implements : ftm_quad_decode_config_t_Class

Data Fields

- ftm_quad_decode_mode_t mode
- uint16_t initialVal
- uint16_t maxVal
- ftm_phase_params_t phaseAConfig
- ftm_phase_params_t phaseBConfig

Field Documentation

3.29.2.15.1 initialVal

```
uint16_t initialVal
```

Initial counter value

3.29.2.15.2 maxVal

```
uint16_t maxVal
```

Maximum counter value

3.29.2.15.3 mode

```
ftm_quad_decode_mode_t mode
```

FTM_QUAD_PHASE_ENCODE or FTM_QUAD_COUNT_AND_DIR

3.29.2.15.4 phaseAConfig

```
ftm_phase_params_t phaseAConfig
```

Configuration for the input phase a

3.29.2.15.5 phaseBConfig

```
ftm_phase_params_t phaseBConfig
```

Configuration for the input phase b

3.29.2.16 struct ftm_quad_decoder_state_t

FTM quadrature state(counter value and flags)

Implements : ftm_quad_decoder_state_t_Class

Data Fields

- uint16_t counter
- bool overflowFlag
- bool overflowDirection
- bool counterDirection

Field Documentation

3.29.2.16.1 counter

```
uint16_t counter
```

Counter value

3.29.2.16.2 counterDirection

```
bool counterDirection
```

False FTM counter is decreasing, True FTM counter is increasing

3.29.2.16.3 overflowDirection

```
bool overflowDirection
```

False if overflow occurred at minimum value, True if overflow occurred at maximum value

3.29.2.16.4 overflowFlag

```
bool overflowFlag
```

True if overflow occurred, False if overflow doesn't occurred

3.29.3 Macro Definition Documentation

3.29.3.1 FTM_DUTY_TO_TICKS_SHIFT

```
#define FTM_DUTY_TO_TICKS_SHIFT (15U)
```

Shift value which converts duty to ticks.

3.29.3.2 FTM_MAX_DUTY_CYCLE

```
#define FTM_MAX_DUTY_CYCLE (0x8000U)
```

Maximum value for PWM duty cycle.

3.29.4 Typedef Documentation

3.29.4.1 ftm_channel_event_callback_t

```
typedef void(* ftm_channel_event_callback_t) (void *userData)
```

Channel event callback function.

Callback functions are called by the FTM driver in Input Capture mode when an event is detected(change in logical state of a pin or measurement complete)

3.29.5 Enumeration Type Documentation

3.29.5.1 ftm_config_mode_t

```
enum ftm_config_mode_t
```

FlexTimer operation mode.

Implements : ftm_config_mode_t_Class

Enumerator

FTM_MODE_NOT_INITIALIZED	The driver is not initialized
FTM_MODE_INPUT_CAPTURE	Input capture
FTM_MODE_OUTPUT_COMPARE	Output compare
FTM_MODE_EDGE_ALIGNED_PWM	Edge aligned PWM
FTM_MODE_CEN_ALIGNED_PWM	Center aligned PWM
FTM_MODE_QUADRATURE_DECODER	Quadrature decoder
FTM_MODE_UP_TIMER	Timer with up counter
FTM_MODE_UP_DOWN_TIMER	timer with up down counter

3.29.5.2 ftm_edge_alignment_mode_t

```
enum ftm_edge_alignment_mode_t
```

FlexTimer input capture edge mode, rising edge, or falling edge.

Implements : ftm_edge_alignment_mode_t_Class

Enumerator

FTM_NO_PIN_CONTROL	No trigger
FTM_RISING_EDGE	Rising edge trigger
FTM_FALLING_EDGE	Falling edge trigger
FTM_BOTH_EDGES	Rising and falling edge trigger

3.29.5.3 ftm_input_op_mode_t

```
enum ftm_input_op_mode_t
```

FTM status.

Implements : ftm_input_op_mode_t_Class

Enumerator

FTM_EDGE_DETECT	FTM edge detect
FTM_SIGNAL_MEASUREMENT	FTM signal measurement
FTM_NO_OPERATION	FTM no operation

3.29.5.4 ftm_output_compare_mode_t

```
enum ftm_output_compare_mode_t
```

FlexTimer Mode configuration for output compare mode.

Implements : ftm_output_compare_mode_t_Class

Enumerator

FTM_DISABLE_OUTPUT	No action on output pin
FTM_TOGGLE_ON_MATCH	Toggle on match
FTM_CLEAR_ON_MATCH	Clear on match
FTM_SET_ON_MATCH	Set on match

3.29.5.5 ftm_output_compare_update_t

```
enum ftm_output_compare_update_t
```

FlexTimer input capture type of the next output compare value.

Implements : ftm_output_compare_update_t_Class

Enumerator

FTM_RELATIVE_VALUE	Next compared value is relative to current value
FTM_ABSOLUTE_VALUE	Next compared value is absolute

3.29.5.6 ftm_pwm_update_option_t

```
enum ftm_pwm_update_option_t
```

FlexTimer Configure type of PWM update in the duty cycle or in ticks.

Implements : ftm_pwm_update_option_t_Class

Enumerator

FTM_PWM_UPDATE_IN_DUTY_CYCLE	The type of PWM update in the duty cycle/pulse or also use in frequency update
FTM_PWM_UPDATE_IN_TICKS	The type of PWM update in ticks

3.29.5.7 ftm_signal_measurement_mode_t

```
enum ftm_signal_measurement_mode_t
```

FlexTimer input capture measurement type for dual edge input capture.

Implements : ftm_signal_measurement_mode_t_Class

Enumerator

FTM_NO_MEASUREMENT	No measurement
FTM_RISING_EDGE_PERIOD_MEASUREMENT	Period measurement between two consecutive rising edges
FTM_FALLING_EDGE_PERIOD_MEASUREMENT	Period measurement between two consecutive falling edges
FTM_PERIOD_ON_MEASUREMENT	The time measurement taken for the pulse to remain ON or HIGH state
FTM_PERIOD_OFF_MEASUREMENT	The time measurement taken for the pulse to remain OFF or LOW state

3.29.6 Function Documentation

3.29.6.1 FTM_DRV_ConvertFreqToPeriodTicks()

```
uint16_t FTM_DRV_ConvertFreqToPeriodTicks (
            uint32_t instance,
            uint32_t freqencyHz )
```

This function is used to covert the given frequency to period in ticks.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*freqencyHz*	Frequency value in Hz.

Returns

uint16_t

3.29.6.2 FTM_DRV_CounterRead()

```
uint32_t FTM_DRV_CounterRead (
              uint32_t instance )
```

Reads back the current value of the FTM counter.

Parameters

in	*instance*	The FTM peripheral instance number.

Returns

The current counter value

3.29.6.3 FTM_DRV_CounterStart()

```
status_t FTM_DRV_CounterStart (
              uint32_t instance )
```

Starts the FTM counter.

Parameters

in	*instance*	The FTM peripheral instance number.

Returns

operation status

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.4 FTM_DRV_CounterStop()

```
status_t FTM_DRV_CounterStop (
              uint32_t instance )
```

Stops the FTM counter.

Parameters

in	*instance*	The FTM peripheral instance number.

Returns

operation status

- STATUS_SUCCESS : Completed successfully.

3.29.6.5 FTM_DRV_Deinit()

```
status_t FTM_DRV_Deinit (
            uint32_t instance )
```

Shuts down the FTM driver.

Parameters

in	*instance*	The FTM peripheral instance number.

Returns

operation status

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.6 FTM_DRV_DeinitInputCapture()

```
status_t FTM_DRV_DeinitInputCapture (
            uint32_t instance,
            const ftm_input_param_t * param )
```

Disables input capture mode and clears FTM timer configuration.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*param*	Configuration of the output compare channel.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.7 FTM_DRV_DeinitOutputCompare()

```
status_t FTM_DRV_DeinitOutputCompare (
            uint32_t instance,
            const ftm_output_cmp_param_t * param )
```

Disables compare match output control and clears FTM timer configuration.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*param*	Configuration of the output compare channel

Returns

success
- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.8 FTM_DRV_DeinitPwm()

```
status_t FTM_DRV_DeinitPwm (
            uint32_t instance )
```

Stops all PWM channels .

Parameters

in	*instance*	The FTM peripheral instance number.

Returns

counter the current counter value

3.29.6.9 FTM_DRV_GetFrequency()

```
uint32_t FTM_DRV_GetFrequency (
            uint32_t instance )
```

Retrieves the frequency of the clock source feeding the FTM counter.

Function will return a 0 if no clock source is selected and the FTM counter is disabled

Parameters

in	*instance*	The FTM peripheral instance number.

Returns

The frequency of the clock source running the FTM counter (0 if counter is disabled)

3.29.6.10 FTM_DRV_GetInputCaptureMeasurement()

```
uint16_t FTM_DRV_GetInputCaptureMeasurement (
            uint32_t instance,
            uint8_t channel )
```

This function is used to calculate the measurement and/or time stamps values which are read from the C(n, n+1)V registers and stored to the static buffers.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*channel*	For getting the time stamp of the last edge (in normal input capture) this parameter represents the channel number. For getting the last measured value (in dual edge input capture) this parameter is the lowest channel number of the pair (EX: 0, 2, 4, 6).

Returns

value The measured value

3.29.6.11 FTM_DRV_Init()

```
status_t FTM_DRV_Init (
            uint32_t instance,
            const ftm_user_config_t * info,
            ftm_state_t * state )
```

Initializes the FTM driver.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*info*	The FTM user configuration structure, see ftm_user_config_t.
out	*state*	The FTM state structure of the driver.

Returns

operation status

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.12 FTM_DRV_InitCounter()

```
status_t FTM_DRV_InitCounter (
            uint32_t instance,
            const ftm_timer_param_t * timer )
```

Initialize the FTM counter.

Starts the FTM counter. This function provides access to the FTM counter settings. The counter can be run in Up counting and Up-down counting modes. To run the counter in Free running mode, choose Up counting option and provide 0x0 for the countStartVal and 0xFFFF for countFinalVal. Please call this function only when FTM is used as timer/counter.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*timer*	Timer configuration structure.

Returns

operation status

- STATUS_SUCCESS : Initialized successfully.

3.29.6.13 FTM_DRV_InitInputCapture()

```
status_t FTM_DRV_InitInputCapture (
             uint32_t instance,
             const ftm_input_param_t * param )
```

Configures Channel Input Capture for either getting time-stamps on edge detection or on signal measurement . When the edge specified in the captureMode argument occurs on the channel the FTM counter is captured into the CnV register. The user will have to read the CnV register separately to get this value. The filter function is disabled if the filterVal argument passed in is 0. The filter function is available only on channels 0,1,2,3.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*param*	Configuration of the input capture channel.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.14 FTM_DRV_InitOutputCompare()

```
status_t FTM_DRV_InitOutputCompare (
             uint32_t instance,
             const ftm_output_cmp_param_t * param )
```

Configures the FTM to generate timed pulses(Output compare mode).

When the FTM counter matches the value of CnV, the channel output is changed based on what is specified in the compareMode argument. The signal period can be modified using param->MaxCountValue. After this function max counter value and CnV are equal. FTM_DRV_SetNextComparematchValue function can be used to change CnV value.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*param*	configuration of the output compare channels

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.15 FTM_DRV_InitPwm()

```
status_t FTM_DRV_InitPwm (
            uint32_t instance,
            const ftm_pwm_param_t * param )
```

Configures the duty cycle and frequency and starts outputting the PWM on all channels configured in param.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*param*	FTM driver PWM parameter to configure PWM options.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.16 FTM_DRV_MaskOutputChannels()

```
status_t FTM_DRV_MaskOutputChannels (
            uint32_t instance,
            uint32_t channelsMask,
            bool softwareTrigger )
```

This function will mask the output of the channels and at match events will be ignored by the masked channels.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*channelsMask*	The mask which will select which channels will ignore match events.
in	*softwareTrigger*	If true a software trigger is generate to update PWM parameters.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.17 FTM_DRV_QuadDecodeStart()

```
status_t FTM_DRV_QuadDecodeStart (
            uint32_t instance,
            const ftm_quad_decode_config_t * config )
```

Configures the quadrature mode and starts measurement.

Parameters

in	*instance*	Instance number of the FTM module.
in	*config*	Configuration structure(quadrature decode mode, polarity for both phases, initial and maximum value for the counter, filter configuration).

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.18 FTM_DRV_QuadDecodeStop()

```
status_t FTM_DRV_QuadDecodeStop (
            uint32_t instance )
```

De-activates the quadrature decode mode.

Parameters

in	*instance*	Instance number of the FTM module.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.19 FTM_DRV_QuadGetState()

```
ftm_quad_decoder_state_t FTM_DRV_QuadGetState (
            uint32_t instance )
```

Return the current quadrature decoder state (counter value, overflow flag and overflow direction)

Parameters

in	*instance*	Instance number of the FTM module.

Returns

The current state of quadrature decoder

3.29.6.20 FTM_DRV_SetHalfCycleReloadPoint()

```
status_t FTM_DRV_SetHalfCycleReloadPoint (
            uint32_t instance,
            uint16_t reloadPoint,
            bool softwareTrigger )
```

This function configure the value of the counter which will generates an reload point.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*reloadPoint*	Counter value which generates the reload point.
in	*softwareTrigger*	If true a software trigger is generate to update parameters.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.21 FTM_DRV_SetInitialCounterValue()

```
status_t FTM_DRV_SetInitialCounterValue (
            uint32_t instance,
            uint16_t counterValue,
            bool softwareTrigger )
```

This function configure the initial counter value. The counter will get this value after an overflow event.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*counterValue*	Initial counter value.
in	*softwareTrigger*	If true a software trigger is generate to update parameters.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.22 FTM_DRV_SetInvertingControl()

```
status_t FTM_DRV_SetInvertingControl (
            uint32_t instance,
            uint8_t channelsPairMask,
            bool softwareTrigger )
```

This function will configure if the second channel of a pair will be inverted or not.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*pairOfChannelsMask*	The mask which will configure which channel pair will invert the second channel.
in	*softwareTrigger*	If true a software trigger is generate to update registers.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.23 FTM_DRV_SetModuloCounterValue()

```
status_t FTM_DRV_SetModuloCounterValue (
            uint32_t instance,
            uint16_t counterValue,
            bool softwareTrigger )
```

This function configure the maximum counter value.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*counterValue*	Maximum counter value
in	*softwareTrigger*	If true a software trigger is generate to update parameters

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.24 FTM_DRV_SetSoftOutChnValue()

```
status_t FTM_DRV_SetSoftOutChnValue (
            uint32_t instance,
            uint8_t channelsValues,
            bool softwareTrigger )
```

This function will force the output value of a channel to a specific value. Before using this function it's mandatory to mask the match events using FTM_DRV_MaskOutputChannels and to enable software output control using FTM↩
_DRV_SetSoftwareOutputChannelControl.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*channelsValues*	The values which will be software configured for channels.
in	*softwareTrigger*	If true a software trigger is generate to update registers

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.25 FTM_DRV_SetSoftwareOutputChannelControl()

```
status_t FTM_DRV_SetSoftwareOutputChannelControl (
            uint32_t instance,
            uint8_t channelsMask,
            bool softwareTrigger )
```

This function will configure which output channel can be software controlled.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*channelsMask*	The mask which will configure the channels which can be software controlled.
in	*softwareTrigger*	If true a software trigger is generate to update registers

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.26 FTM_DRV_SetSync()

```
status_t FTM_DRV_SetSync (
            uint32_t instance,
            const ftm_pwm_sync_t * param )
```

This function configures sync mechanism for some FTM registers (MOD, CNINT, HCR, CnV, OUTMASK, INVCTRL, SWOCTRL).

Parameters

in	*instance*	The FTM peripheral instance number.
in	*param*	The sync configuration structure.

Returns

operation status

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.27 FTM_DRV_StartNewSignalMeasurement()

```
status_t FTM_DRV_StartNewSignalMeasurement (
            uint32_t instance,
            uint8_t channel )
```

Starts new single-shot signal measurement of the given channel.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*channel*	Configuration of the output compare channel.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.28 FTM_DRV_UpdateOutputCompareChannel()

```
status_t FTM_DRV_UpdateOutputCompareChannel (
            uint32_t instance,
            uint8_t channel,
            uint16_t nextComparematchValue,
            ftm_output_compare_update_t update,
            bool softwareTrigger )
```

Sets the next compare match value based on the current counter value.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*channel*	Configuration of the output compare channel
in	*nextComparematchValue*	Timer value in ticks until the next compare match event should appear
in	*update*	• FTM_RELATIVE_VALUE : nextComparemantchValue will be added to current counter value • FTM_ABSOLUTE_VALUE : nextComparemantchValue will be written in counter register as it is
in	*softwareTrigger*	This parameter will be true if software trigger sync is enabled and the user want to generate a software trigger (the value from buffer will be moved to register immediate or at next loading point depending on the sync configuration). Otherwise this parameter must be false and the next compared value will be stored in buffer until a trigger signal will be received.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.29 FTM_DRV_UpdatePwmChannel()

```
status_t FTM_DRV_UpdatePwmChannel (
            uint32_t instance,
            uint8_t channel,
            ftm_pwm_update_option_t typeOfUpdate,
            uint16_t firstEdge,
            uint16_t secondEdge,
            bool softwareTrigger )
```

This function updates the waveform output in PWM mode (duty cycle and phase).

Parameters

in	*instance*	The FTM peripheral instance number.
in	*channel*	The channel number. In combined mode, the code finds the channel.
in	*typeOfUpdate*	The type of PWM update in the duty cycle/pulse or in ticks.
in	*firstEdge*	Duty cycle or first edge time for PWM mode. Can take value between 0 - FTM_MAX_DUTY_CYCLE(0 = 0% from period and FTM_MAX_DUTY_CYCLE = 100% from period) Or value in ticks for the first of the PWM mode in which can have value between 0 and ftmPeriod is stored in the state structure.
in	*secondEdge*	Second edge time - only for combined mode. Can take value between 0 - FTM_MAX_DUTY_CYCLE(0 = 0% from period and FTM_MAX_DUTY_CYCLE = 100% from period). Or value in ticks for the second of the PWM mode in which can have value between 0 and ftmPeriod is stored in the state structure.
in	*softwareTrigger*	If true a software trigger is generate to update PWM parameters.

Returns

success

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.6.30 FTM_DRV_UpdatePwmPeriod()

```
status_t FTM_DRV_UpdatePwmPeriod (
             uint32_t instance,
             ftm_pwm_update_option_t typeOfUpdate,
             uint32_t newValue,
             bool softwareTrigger )
```

This function will update the new period in the frequency or in the counter value into mode register which modify the period of PWM signal on the channel output.

Parameters

in	*instance*	The FTM peripheral instance number.
in	*typeOfUpdate*	The type of PWM update is a period in Hz or in ticks. • For FTM_PWM_UPDATE_IN_DUTY_CYCLE which reuse in FTM_DRV_UpdatePwmChannel function will update in Hz. • For FTM_PWM_UPDATE_IN_TICKS will update in ticks.
in	*newValue*	The frequency or the counter value which will select with modified value for PWM signal. If the type of update in the duty cycle, the newValue parameter must be value between 1U and maximum is the frequency of the FTM counter. If the type of update in ticks, the newValue parameter must be value between 1U and 0xFFFFU.
in	*softwareTrigger*	If true a software trigger is generate to update PWM parameters.

Returns

operation status

- STATUS_SUCCESS : Completed successfully.
- STATUS_ERROR : Error occurred.

3.29.7 Variable Documentation

3.29.7.1 g_ftmBase

```
FTM_Type* const g_ftmBase[FTM_INSTANCE_COUNT]
```

Table of base addresses for FTM instances.

3.29.7.2 g_ftmFaultIrqId

```
const IRQn_Type g_ftmFaultIrqId[FTM_INSTANCE_COUNT]
```

3.29.7.3 g_ftmIrqId

```
const IRQn_Type g_ftmIrqId[FTM_INSTANCE_COUNT][FEATURE_FTM_CHANNEL_COUNT]
```

Interrupt vectors for the FTM peripheral.

3.29.7.4 g_ftmOverflowIrqId

```
const IRQn_Type g_ftmOverflowIrqId[FTM_INSTANCE_COUNT]
```

3.29.7.5 g_ftmReloadIrqId

```
const IRQn_Type g_ftmReloadIrqId[FTM_INSTANCE_COUNT]
```

3.30 FTM HAL

3.30.1 Detailed Description

FlexTimer Module Hardware Abstraction Level. FTM HAL provides low level APIs for reading and writing to all hardware features of the FlexTimer module.

Macros

- #define FTM_RMW_SC(base, mask, value) (((base)->SC) = ((((base)->SC) & ~(mask)) | (value)))

  FTM_SC - Read and modify and write to Status And Control (RW)
- #define FTM_RMW_CNT(base, mask, value) (((base)->CNT) = ((((base)->CNT) & ~(mask)) | (value)))

  FTM_CNT - Read and modify and write to Counter (RW)
- #define FTM_RMW_MOD(base, mask, value) (((base)->MOD) = ((((base)->MOD) & ~(mask)) | (value)))

  FTM_MOD - Read and modify and write Modulo (RW)
- #define FTM_RMW_CNTIN(base, mask, value) (((base)->CNTIN) = ((((base)->CNTIN) & ~(mask)) | (value)))

  FTM_CNTIN - Read and modify and write Counter Initial Value (RW)
- #define FTM_RMW_STATUS(base, mask, value) (((base)->STATUS) = ((((base)->STATUS) & ~(mask)) | (value)))

  FTM_STATUS - Read and modify and write Capture And Compare Status (RW)
- #define FTM_RMW_MODE(base, mask, value) (((base)->MODE) = ((((base)->MODE) & ~(mask)) | (value)))

  FTM_MODE - Read and modify and write Counter Features Mode Selection (RW)
- #define FTM_RMW_CnSCV_REG(base, channel, mask, value) (((base)->CONTROLS[channel].CnSC) = ((((base)->CONTROLS[channel].CnSC) & ~(mask)) | (value)))

  FTM_CnSCV - Read and modify and write Channel (n) Status And Control (RW)
- #define FTM_RMW_DEADTIME(base, mask, value) (((base)->DEADTIME) = ((((base)->DEADTIME) & ~(mask)) | (value)))

  FTM_DEADTIME - Read and modify and write Dead-time Insertion Control (RW)
- #define FTM_RMW_EXTTRIG_REG(base, mask, value) (((base)->EXTTRIG) = ((((base)->EXTTRIG) & ~(mask)) | (value)))

  FTM_EXTTRIG - Read and modify and write External Trigger Control (RW)
- #define FTM_RMW_FLTCTRL(base, mask, value) (((base)->FLTCTRL) = ((((base)->FLTCTRL) & ~(mask)) | (value)))

  FTM_FLTCTRL - Read and modify and write Fault Control (RW)
- #define FTM_RMW_FMS(base, mask, value) (((base)->FMS) = ((((base)->FMS) & ~(mask)) | (value)))

  FTM_FMS - Read and modify and write Fault Mode Status (RW)
- #define FTM_RMW_CONF(base, mask, value) (((base)->CONF) = ((((base)->CONF) & ~(mask)) | (value)))

  FTM_CONF - Read and modify and write Configuration (RW)
- #define FTM_RMW_POL(base, mask, value) (((base)->POL) = ((((base)->POL) & ~(mask)) | (value)))

  POL - Read and modify and write Polarity (RW)
- #define FTM_RMW_FILTER(base, mask, value) (((base)->FILTER) = ((((base)->FILTER) & ~(mask)) | (value)))

  FILTER - Read and modify and write Filter (RW)
- #define FTM_RMW_SYNC(base, mask, value) (((base)->SYNC) = ((((base)->SYNC) & ~(mask)) | (value)))

  SYNC - Read and modify and write Synchronization (RW)
- #define FTM_RMW_QDCTRL(base, mask, value) (((base)->QDCTRL) = ((((base)->QDCTRL) & ~(mask)) | (value)))

  QDCTRL - Read and modify and write Quadrature Decoder Control And Status (RW)

- #define FTM_RMW_PAIR0DEADTIME(base, mask, value) (((base)->PAIR0DEADTIME) = ((((base)->PA←IR0DEADTIME) & ∼(mask)) | (value)))

    FTM_PAIR0DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 0 (RW)
- #define FTM_RMW_PAIR1DEADTIME(base, mask, value) (((base)->PAIR1DEADTIME) = ((((base)->PA←IR1DEADTIME) & ∼(mask)) | (value)))

    FTM_PAIR1DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 1 (RW)
- #define FTM_RMW_PAIR2DEADTIME(base, mask, value) (((base)->PAIR2DEADTIME) = ((((base)->PA←IR2DEADTIME) & ∼(mask)) | (value)))

    FTM_PAIR2DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 2 (RW)
- #define FTM_RMW_PAIR3DEADTIME(base, mask, value) (((base)->PAIR3DEADTIME) = ((((base)->PA←IR3DEADTIME) & ∼(mask)) | (value)))

    FTM_PAIR3DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 3 (RW)
- #define CHAN0_IDX (0U)

    Channel number for CHAN1.
- #define CHAN1_IDX (1U)

    Channel number for CHAN2.
- #define CHAN2_IDX (2U)

    Channel number for CHAN3.
- #define CHAN3_IDX (3U)

    Channel number for CHAN4.
- #define CHAN4_IDX (4U)

    Channel number for CHAN5.
- #define CHAN5_IDX (5U)

    Channel number for CHAN6.
- #define CHAN6_IDX (6U)

    Channel number for CHAN7.
- #define CHAN7_IDX (7U)

Enumerations

- enum ftm_clock_source_t { FTM_CLOCK_SOURCE_NONE = 0x00U, FTM_CLOCK_SOURCE_SYSTEM←CLK = 0x01U, FTM_CLOCK_SOURCE_FIXEDCLK = 0x02U, FTM_CLOCK_SOURCE_EXTERNALCLK = 0x03U }

    FlexTimer clock source selection.
- enum ftm_clock_ps_t {
  FTM_CLOCK_DIVID_BY_1 = 0x00U, FTM_CLOCK_DIVID_BY_2 = 0x01U, FTM_CLOCK_DIVID_BY_4 = 0x02U, FTM_CLOCK_DIVID_BY_8 = 0x03U,
  FTM_CLOCK_DIVID_BY_16 = 0x04U, FTM_CLOCK_DIVID_BY_32 = 0x05U, FTM_CLOCK_DIVID_BY_64 = 0x06U, FTM_CLOCK_DIVID_BY_128 = 0x07U }

    FlexTimer pre-scaler factor selection for the clock source. In quadrature decoder mode set FTM_CLOCK_DIVID_←BY_1.
- enum ftm_deadtime_ps_t { FTM_DEADTIME_DIVID_BY_1 = 0x01U, FTM_DEADTIME_DIVID_BY_4 = 0x02U, FTM_DEADTIME_DIVID_BY_16 = 0x03U }

    FlexTimer pre-scaler factor for the dead-time insertion.
- enum ftm_polarity_t { FTM_POLARITY_LOW = 0x00U, FTM_POLARITY_HIGH = 0x01U }

    FlexTimer PWM output pulse mode, high-true or low-true on match up.
- enum ftm_second_channel_polarity_t { FTM_MAIN_INVERTED = 0x01U, FTM_MAIN_DUPLICATED = 0x00U }

    FlexTimer PWM channel (n+1) polarity for combine mode.
- enum ftm_quad_decode_mode_t { FTM_QUAD_PHASE_ENCODE = 0x00U, FTM_QUAD_COUNT_AND←_DIR = 0x01U }

    FlexTimer quadrature decode modes, phase encode or count and direction mode.

- enum ftm_quad_phase_polarity_t { FTM_QUAD_PHASE_NORMAL = 0x00U, FTM_QUAD_PHASE_INVE↩ RT = 0x01U }

  FlexTimer quadrature phase polarities, normal or inverted polarity.
- enum ftm_bdm_mode_t { FTM_BDM_MODE_00 = 0x00U, FTM_BDM_MODE_01 = 0x01U, FTM_BDM_M↩ ODE_10 = 0x02U, FTM_BDM_MODE_11 = 0x03U }

  Options for the FlexTimer behavior in BDM Mode.
- enum ftm_fault_mode_t { FTM_FAULT_CONTROL_DISABLED = 0x00U, FTM_FAULT_CONTROL_MA↩ N_EVEN = 0x01U, FTM_FAULT_CONTROL_MAN_ALL = 0x02U, FTM_FAULT_CONTROL_AUTO_ALL = 0x03U }

  FlexTimer fault control.
- enum ftm_reg_update_t { FTM_SYSTEM_CLOCK = 0U, FTM_PWM_SYNC = 1U }

  FTM sync source.
- enum ftm_pwm_sync_mode_t { FTM_WAIT_LOADING_POINTS = 0U, FTM_UPDATE_NOW = 1U }

  FTM update register.

Functions

- static void FTM_HAL_SetHalfCycleValue (FTM_Type ∗const ftmBase, uint16_t value)

  Sets the value for the half cycle reload register.
- static void FTM_HAL_SetClockFilterPs (FTM_Type ∗const ftmBase, uint8_t filterPrescale)

  Sets the filter Pre-scaler divider.
- static uint8_t FTM_HAL_GetClockFilterPs (const FTM_Type ∗ftmBase)

  Reads the FTM filter clock divider.
- static void FTM_HAL_SetClockSource (FTM_Type ∗const ftmBase, ftm_clock_source_t clock)

  Sets the FTM clock source.
- static uint8_t FTM_HAL_GetClockSource (const FTM_Type ∗ftmBase)

  Reads the FTM clock source.
- static void FTM_HAL_SetClockPs (FTM_Type ∗const ftmBase, ftm_clock_ps_t ps)

  Sets the FTM clock divider.
- static uint8_t FTM_HAL_GetClockPs (const FTM_Type ∗ftmBase)

  Reads the FTM clock divider.
- static void FTM_HAL_SetTimerOverflowInt (FTM_Type ∗const ftmBase, bool state)

  Enables the FTM peripheral timer overflow interrupt.
- static bool FTM_HAL_IsOverflowIntEnabled (const FTM_Type ∗ftmBase)

  Reads the bit that controls enabling the FTM timer overflow interrupt.
- static void FTM_HAL_EnablePwmChannelOutputs (FTM_Type ∗const ftmBase, uint8_t channel)

  Enable PWM channel Outputs.
- static void FTM_HAL_DisablePwmChannelOutputs (FTM_Type ∗const ftmBase, uint8_t channel)

  Disable PWM channel Outputs.
- static void FTM_HAL_ClearTimerOverflow (FTM_Type ∗const ftmBase)

  Clears the timer overflow interrupt flag.
- static bool FTM_HAL_HasTimerOverflowed (const FTM_Type ∗ftmBase)

  Returns the FTM peripheral timer overflow interrupt flag.
- static void FTM_HAL_SetCpwms (FTM_Type ∗const ftmBase, bool mode)

  Sets the FTM count direction bit.
- static bool FTM_HAL_GetCpwms (const FTM_Type ∗ftmBase)

  Gets the FTM count direction bit.
- static void FTM_HAL_SetReIntEnabledCmd (FTM_Type ∗const ftmBase, bool enable)

  Set the FTM reload interrupt enable.
- static bool FTM_HAL_GetReloadFlag (const FTM_Type ∗ftmBase)

Get the state whether the FTM counter reached a reload point.
- static void FTM_HAL_ClearReloadFlag (FTM_Type ∗const ftmBase)

  Clears the reload flag bit.
- static void FTM_HAL_SetCounter (FTM_Type ∗const ftmBase, uint16_t value)

  Sets the FTM peripheral current counter value.
- static uint16_t FTM_HAL_GetCounter (const FTM_Type ∗ftmBase)

  Returns the FTM peripheral current counter value.
- static void FTM_HAL_SetMod (FTM_Type ∗const ftmBase, uint16_t value)

  Sets the FTM peripheral timer modulo value.
- static uint16_t FTM_HAL_GetMod (const FTM_Type ∗ftmBase)

  Returns the FTM peripheral counter modulo value.
- static void FTM_HAL_SetCounterInitVal (FTM_Type ∗const ftmBase, uint16_t value)

  Sets the FTM peripheral timer counter initial value.
- static uint16_t FTM_HAL_GetCounterInitVal (const FTM_Type ∗ftmBase)

  Returns the FTM peripheral counter initial value.
- static void FTM_HAL_SetChnMSnBAMode (FTM_Type ∗const ftmBase, uint8_t channel, uint8_t selection)

  Sets the FTM peripheral timer channel mode.
- static void FTM_HAL_SetChnEdgeLevel (FTM_Type ∗const ftmBase, uint8_t channel, uint8_t level)

  Sets the FTM peripheral timer channel edge level.
- static void FTM_HAL_ClearChSC (FTM_Type ∗const ftmBase, uint8_t channel)

  Clears the content of Channel (n) Status And Control.
- static uint8_t FTM_HAL_GetChnMode (const FTM_Type ∗ftmBase, uint8_t channel)

  Gets the FTM peripheral timer channel mode.
- static uint8_t FTM_HAL_GetChnEdgeLevel (const FTM_Type ∗ftmBase, uint8_t channel)

  Gets the FTM peripheral timer channel edge level.
- static void FTM_HAL_SetChnIcrstCmd (FTM_Type ∗const ftmBase, uint8_t channel, bool enable)

  Configure the feature of FTM counter reset by the selected input capture event.
- static bool FTM_HAL_IsChnIcrst (const FTM_Type ∗ftmBase, uint8_t channel)

  Returns whether the FTM FTM counter is reset.
- static void FTM_HAL_SetChnDmaCmd (FTM_Type ∗const ftmBase, uint8_t channel, bool enable)

  Enables or disables the FTM peripheral timer channel DMA.
- static bool FTM_HAL_IsChnDma (const FTM_Type ∗ftmBase, uint8_t channel)

  Returns whether the FTM peripheral timer channel DMA is enabled.
- static bool FTM_HAL_IsChnIntEnabled (const FTM_Type ∗ftmBase, uint8_t channel)

  Get FTM channel(n) interrupt enabled or not.
- static void FTM_HAL_EnableChnInt (FTM_Type ∗const ftmBase, uint8_t channel)

  Enables the FTM peripheral timer channel(n) interrupt.
- static void FTM_HAL_DisableChnInt (FTM_Type ∗const ftmBase, uint8_t channel)

  Disables the FTM peripheral timer channel(n) interrupt.
- static bool FTM_HAL_HasChnEventOccurred (const FTM_Type ∗ftmBase, uint8_t channel)

  Returns whether any event for the FTM peripheral timer channel has occurred.
- static void FTM_HAL_ClearChnEventFlag (FTM_Type ∗const ftmBase, uint8_t channel)

  Clear the channel flag by writing a 0 to the CHF bit.
- static void FTM_HAL_SetTrigModeControlCmd (FTM_Type ∗const ftmBase, uint8_t channel, bool enable)

  Enables or disables the trigger generation on FTM channel outputs.
- static bool FTM_HAL_GetTriggerControled (const FTM_Type ∗ftmBase, uint8_t channel)

  Returns whether the trigger mode is enabled.
- static bool FTM_HAL_GetChInputState (const FTM_Type ∗ftmBase, uint8_t channel)

  Get the state of channel input.
- static bool FTM_HAL_GetChOutputValue (const FTM_Type ∗ftmBase, uint8_t channel)

  Get the value of channel output.

- static void FTM_HAL_SetChnCountVal (FTM_Type ∗const ftmBase, uint8_t channel, uint16_t value)

    Sets the FTM peripheral timer channel counter value.
- static uint16_t FTM_HAL_GetChnCountVal (const FTM_Type ∗ftmBase, uint8_t channel)

    Gets the FTM peripheral timer channel counter value.
- static bool FTM_HAL_GetChnEventStatus (const FTM_Type ∗ftmBase, uint8_t channel)

    Gets the FTM peripheral timer channel event status.
- static uint32_t FTM_HAL_GetEventStatus (const FTM_Type ∗ftmBase)

    Gets the FTM peripheral timer status info for all channels.
- static void FTM_HAL_ClearChnEventStatus (FTM_Type ∗const ftmBase, uint8_t channel)

    Clears the FTM peripheral timer all channel event status.
- static void FTM_HAL_SetOutmaskReg (FTM_Type ∗const ftmBase, uint32_t regVal)

    Writes the provided value to the OUTMASK register.
- static void FTM_HAL_SetChnOutputMask (FTM_Type ∗const ftmBase, uint8_t channel, bool mask)

    Sets the FTM peripheral timer channel output mask.
- static void FTM_HAL_SetChnOutputInitStateCmd (FTM_Type ∗const ftmBase, uint8_t channel, bool state)

    Sets the FTM peripheral timer channel output initial state 0 or 1.
- static void FTM_HAL_SetChnOutputPolarityCmd (FTM_Type ∗const ftmBase, uint8_t channel, ftm_polarity↩_t polarity)

    Sets the FTM peripheral timer channel output polarity.
- static void FTM_HAL_SetChnFaultInputPolarityCmd (FTM_Type ∗const ftmBase, uint8_t fltChannel, ftm_↩polarity_t polarity)

    Sets the FTM peripheral timer channel fault input polarity.
- static void FTM_HAL_SetFaultInt (FTM_Type ∗const ftmBase, bool state)

    Enables/disables the FTM peripheral timer fault interrupt.
- static void FTM_HAL_DisableFaultInt (FTM_Type ∗const ftmBase)

    Disables the FTM peripheral timer fault interrupt.
- static bool FTM_HAL_IsFaultIntEnabled (const FTM_Type ∗ftmBase)

    Return true/false whether the Fault interrupt was enabled or not.
- static void FTM_HAL_ClearFaultsIsr (FTM_Type ∗const ftmBase)

    Clears all fault interrupt flags that are active.
- static void FTM_HAL_SetFaultControlMode (FTM_Type ∗const ftmBase, ftm_fault_mode_t mode)

    Defines the FTM fault control mode.
- static void FTM_HAL_SetCaptureTestCmd (FTM_Type ∗const ftmBase, bool enable)

    Enables or disables the FTM peripheral timer capture test mode.
- static void FTM_HAL_SetWriteProtectionCmd (FTM_Type ∗const ftmBase, bool enable)

    Enables or disables the FTM write protection.
- static void FTM_HAL_Enable (FTM_Type ∗const ftmBase, bool enable)

    Enables the FTM peripheral timer group.
- static bool FTM_HAL_IsFtmEnable (const FTM_Type ∗ftmBase)

    Get status of the FTMEN bit in the FTM_MODE register.
- static void FTM_HAL_SetInitChnOutputCmd (FTM_Type ∗const ftmBase, bool enable)

    Initializes the channels output.
- static void FTM_HAL_SetPwmSyncMode (FTM_Type ∗const ftmBase, bool enable)

    Sets the FTM peripheral timer sync mode.
- static void FTM_HAL_SetSoftwareTriggerCmd (FTM_Type ∗const ftmBase, bool enable)

    Enables or disables the FTM peripheral timer software trigger.
- static void FTM_HAL_SetHardwareSyncTriggerSrc (FTM_Type ∗const ftmBase, uint8_t trigger_num, bool enable)

    Sets the FTM hardware synchronization trigger.
- static void FTM_HAL_SetOutmaskPwmSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

    Determines when the OUTMASK register is updated with the value of its buffer.

- static void FTM_HAL_SetCountReinitSyncCmd (FTM_Type ∗const ftmBase, bool enable)

  Determines if the FTM counter is re-initialized when the selected trigger for synchronization is detected.
- static void FTM_HAL_SetMaxLoadingCmd (FTM_Type ∗const ftmBase, bool enable)

  Enables or disables the FTM peripheral timer maximum loading points.
- static void FTM_HAL_SetMinLoadingCmd (FTM_Type ∗const ftmBase, bool enable)

  Enables or disables the FTM peripheral timer minimum loading points.
- static void FTM_HAL_SetDualChnMofCombineCmd (FTM_Type ∗const ftmBase, uint8_t chnlPairNum, bool enable)

  Enables the FTM peripheral timer channel modified combine mode.
- static void FTM_HAL_SetDualChnFaultCmd (FTM_Type ∗const ftmBase, uint8_t chnlPairNum, bool enable)

  Enables the FTM peripheral timer channel pair fault control.
- static void FTM_HAL_SetDualChnPwmSyncCmd (FTM_Type ∗const ftmBase, uint8_t chnlPairNum, bool enable)

  Enables or disables the FTM peripheral timer channel pair counter PWM sync.
- static void FTM_HAL_SetDualChnDeadtimeCmd (FTM_Type ∗const ftmBase, uint8_t chnlPairNum, bool enable)

  Enables or disabled the FTM peripheral timer channel pair deadtime insertion.
- static void FTM_HAL_SetDualChnDecapCmd (FTM_Type ∗const ftmBase, uint8_t chnlPairNum, bool enable)

  Enables or disables the FTM peripheral timer channel dual edge capture.
- static void FTM_HAL_SetDualEdgeCaptureCmd (FTM_Type ∗const ftmBase, uint8_t chnlPairNum, bool enable)

  Enables the FTM peripheral timer dual edge capture mode.
- static bool FTM_HAL_GetDualEdgeCaptureBit (const FTM_Type ∗ftmBase, uint8_t chnlPairNum)

  Enables the FTM peripheral timer dual edge capture mode.
- static void FTM_HAL_SetDualChnCompCmd (FTM_Type ∗const ftmBase, uint8_t chnlPairNum, ftm_second_channel_polarity_t polarity)

  Enables or disables the FTM peripheral timer channel pair output complement mode.
- static void FTM_HAL_SetDualChnCombineCmd (FTM_Type ∗const ftmBase, uint8_t chnlPairNum, bool enable)

  Enables or disables the FTM peripheral timer channel pair output combine mode.
- static bool FTM_HAL_GetDualChnCombineCmd (const FTM_Type ∗ftmBase, uint8_t chnlPairNum)

  Verify if an channels pair is used in combine mode or not.
- static bool FTM_HAL_GetDualChnMofCombineCmd (const FTM_Type ∗ftmBase, uint8_t chnlPairNum)

  Verify if an channels pair is used in the modified combine mode or not.
- static void FTM_HAL_SetExtDeadtimeValue (FTM_Type ∗const ftmBase, uint8_t value)

  Sets the FTM extended dead-time value.
- static void FTM_HAL_SetDeadtimePrescale (FTM_Type ∗const ftmBase, ftm_deadtime_ps_t divider)

  Sets the FTM dead time divider.
- static void FTM_HAL_SetDeadtimeCount (FTM_Type ∗const ftmBase, uint8_t count)

  Sets the FTM deadtime value.
- static void FTM_HAL_SetInitTriggerCmd (FTM_Type ∗const ftmBase, bool enable)

  Enables or disables the generation of the trigger when the FTM counter is equal to the CNTIN register.
- static bool FTM_HAL_IsChnTriggerGenerated (const FTM_Type ∗ftmBase)

  Checks whether any channel trigger event has occurred.
- static void FTM_HAL_ClearChnTriggerFlag (FTM_Type ∗const ftmBase)

  Clear the channel trigger flag.
- static bool FTM_HAL_GetDetectedFaultInput (const FTM_Type ∗ftmBase)

  Gets the FTM detected fault input.
- static bool FTM_HAL_IsWriteProtectionEnabled (const FTM_Type ∗ftmBase)

  Checks whether the write protection is enabled.
- static bool FTM_HAL_IsFaultInputEnabled (const FTM_Type ∗ftmBase)

Checks whether the logic OR of the fault inputs is enabled.

- static bool FTM_HAL_IsFaultFlagDetected (const FTM_Type ∗ftmBase, uint8_t channel)

  Checks whether a fault condition is detected at the fault input.
- static void FTM_HAL_ClearFaultFlagDetected (FTM_Type ∗const ftmBase, uint8_t channel)

  Clear a fault condition is detected at the fault input.
- static void FTM_HAL_SetQuadDecoderCmd (FTM_Type ∗const ftmBase, bool enable)

  Enables the channel quadrature decoder.
- static void FTM_HAL_SetQuadPhaseAFilterCmd (FTM_Type ∗const ftmBase, bool enable)

  Enables or disables the phase A input filter.
- static void FTM_HAL_SetQuadPhaseBFilterCmd (FTM_Type ∗const ftmBase, bool enable)

  Enables or disables the phase B input filter.
- static void FTM_HAL_SetQuadPhaseAPolarity (FTM_Type ∗const ftmBase, ftm_quad_phase_polarity_↩ t mode)

  Selects polarity for the quadrature decode phase A input.
- static void FTM_HAL_SetQuadPhaseBPolarity (FTM_Type ∗const ftmBase, ftm_quad_phase_polarity_↩ t mode)

  Selects polarity for the quadrature decode phase B input.
- static void FTM_HAL_SetQuadMode (FTM_Type ∗const ftmBase, ftm_quad_decode_mode_t quadMode)

  Sets the encoding mode used in quadrature decoding mode.
- static bool FTM_HAL_GetQuadDir (const FTM_Type ∗ftmBase)

  Gets the FTM counter direction in quadrature mode.
- static bool FTM_HAL_GetQuadTimerOverflowDir (const FTM_Type ∗ftmBase)

  Gets the Timer overflow direction in quadrature mode.
- static void FTM_HAL_SetFaultInputFilterVal (FTM_Type ∗const ftmBase, uint32_t value)

  Sets the fault input filter value.
- static void FTM_HAL_SetFaultInputFilterCmd (FTM_Type ∗const ftmBase, uint8_t inputNum, bool enable)

  Enables or disables the fault input filter.
- static void FTM_HAL_ClearFaultControl (FTM_Type ∗const ftmBase)

  Clears the entire content value of the Fault control register.
- static void FTM_HAL_SetFaultInputCmd (FTM_Type ∗const ftmBase, uint8_t inputNum, bool enable)

  Enables or disables the fault input.
- static void FTM_HAL_SetPwmFaultBehavior (FTM_Type ∗const ftmBase, bool enable)

  Configures the behavior of the PWM outputs when a fault is detected.
- static void FTM_HAL_SetDualChnInvertCmd (FTM_Type ∗const ftmBase, uint8_t chnlPairNum, bool enable)

  Enables or disables the channel invert for a channel pair.
- static void FTM_HAL_SetInvctrlReg (FTM_Type ∗const ftmBase, uint32_t regVal)

  Writes the provided value to the Inverting control register.
- static void FTM_HAL_SetChnSoftwareCtrlCmd (FTM_Type ∗const ftmBase, uint8_t channel, bool enable)

  Enables or disables the channel software output control.
- static void FTM_HAL_SetAllChnSoftwareCtrlCmd (FTM_Type ∗const ftmBase, uint8_t channelsMask)

  Enables or disables the channel software output control.The main difference between this function and FTM_HAL↩ _SetChnSoftwareCtrlCmd is that this can configure all the channels in the same time.
- static void FTM_HAL_SetChnSoftwareCtrlVal (FTM_Type ∗const ftmBase, uint8_t channel, bool enable)

  Sets the channel software output control value.
- static void FTM_HAL_SetAllChnSoftwareCtrlVal (FTM_Type ∗const ftmBase, uint8_t channelsValues)

  Sets the channel software output control value.
- static void FTM_HAL_SetGlobalLoadCmd (FTM_Type ∗const ftmBase)

  Set the global load mechanism.
- static void FTM_HAL_SetLoadCmd (FTM_Type ∗const ftmBase, bool enable)

  Enable the global load.
- static void FTM_HAL_SetHalfCycleCmd (FTM_Type ∗const ftmBase, bool enable)

Enable the half cycle reload.
- static void FTM_HAL_SetPwmLoadCmd (FTM_Type ∗const ftmBase, bool enable)

  Enables or disables the loading of MOD, CNTIN and CV with values of their write buffer.
- static void FTM_HAL_SetPwmLoadChnSelCmd (FTM_Type ∗const ftmBase, uint8_t channel, bool enable)

  Includes or excludes the channel in the matching process.
- static void FTM_HAL_SetInitTrigOnReloadCmd (FTM_Type ∗const ftmBase, bool enable)

  Enables or disables the FTM initialization trigger on Reload Point.
- static void FTM_HAL_SetGlobalTimeBaseOutputCmd (FTM_Type ∗const ftmBase, bool enable)

  Enables or disables the FTM global time base signal generation to other FTM's.
- static void FTM_HAL_SetGlobalTimeBaseCmd (FTM_Type ∗const ftmBase, bool enable)

  Enables or disables the FTM timer global time base.
- static void FTM_HAL_SetBdmMode (FTM_Type ∗const ftmBase, ftm_bdm_mode_t val)

  Sets the BDM mode.
- static void FTM_HAL_SetLoadFreq (FTM_Type ∗const ftmBase, uint8_t val)

  Sets the FTM timer TOF Frequency.
- static void FTM_HAL_SetSwoctrlHardwareSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

  Sets the sync mode for the FTM SWOCTRL register when using a hardware trigger.
- static void FTM_HAL_SetInvctrlHardwareSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

  Sets sync mode for FTM INVCTRL register when using a hardware trigger.
- static void FTM_HAL_SetOutmaskHardwareSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

  Sets sync mode for FTM OUTMASK register when using a hardware trigger.
- static void FTM_HAL_SetModCntinCvHardwareSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

  Sets sync mode for FTM MOD, CNTIN and CV registers when using a hardware trigger.
- static void FTM_HAL_SetCounterHardwareSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

  Sets sync mode for FTM counter register when using a hardware trigger.
- static void FTM_HAL_SetSwoctrlSoftwareSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

  Sets sync mode for FTM SWOCTRL register when using a software trigger.
- static void FTM_HAL_SetInvctrlSoftwareSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

  Sets sync mode for FTM INVCTRL register when using a software trigger.
- static void FTM_HAL_SetOutmaskSoftwareSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

  Sets sync mode for FTM OUTMASK register when using a software trigger.
- static void FTM_HAL_SetModCntinCvSoftwareSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

  Sets sync mode for FTM MOD, CNTIN and CV registers when using a software trigger.
- static void FTM_HAL_SetHwTriggerSyncModeCmd (FTM_Type ∗const ftmBase, bool enable)

  Sets hardware trigger mode.
- static void FTM_HAL_SetCounterSoftwareSyncModeCmd (FTM_Type ∗const ftmBase, ftm_pwm_sync_↩ mode_t update_mode)

  Sets sync mode for FTM counter register when using a software trigger.
- static void FTM_HAL_SetPwmSyncModeCmd (FTM_Type ∗const ftmBase, bool mode)

  Sets the PWM synchronization mode to enhanced or legacy.
- static void FTM_HAL_SetSwoctrlPwmSyncModeCmd (FTM_Type ∗const ftmBase, ftm_reg_update_t mode)

  Sets the SWOCTRL register PWM synchronization mode.
- static void FTM_HAL_SetInvctrlPwmSyncModeCmd (FTM_Type ∗const ftmBase, ftm_reg_update_t mode)

  Sets the INVCTRL register PWM synchronization mode.
- static void FTM_HAL_SetCntinPwmSyncModeCmd (FTM_Type ∗const ftmBase, ftm_reg_update_t mode)

  Sets the CNTIN register PWM synchronization mode.
- static void FTM_HAL_SetExtPairDeadtimeValue (FTM_Type ∗const ftmBase, uint8_t channelPair, uint8_↩ t value)

  Sets the FTM extended dead-time value for the channel pair.
- static void FTM_HAL_SetPairDeadtimePrescale (FTM_Type ∗const ftmBase, uint8_t channelPair, ftm_↩ deadtime_ps_t divider)

Sets the FTM dead time divider for the channel pair.
- static void FTM_HAL_SetPairDeadtimeCount (FTM_Type ∗const ftmBase, uint8_t channelPair, uint8_t count)

  Sets the FTM dead-time value for the channel pair.
- void FTM_HAL_Reset (FTM_Type ∗const ftmBase)

  Resets the FTM registers. All the register use in the driver should be reset to default value of each register.
- void FTM_HAL_Init (FTM_Type ∗const ftmBase, ftm_clock_ps_t ftmClockPrescaler)

  Initializes the FTM. This function will enable the flexTimer module and selects one pre-scale factor for the FTM clock source.
- void FTM_HAL_SetChnTriggerCmd (FTM_Type ∗const ftmBase, uint8_t channel, bool enable)

  Enables or disables the generation of the FTM peripheral timer channel trigger when the FTM counter is equal to its initial value.
- void FTM_HAL_SetChnInputCaptureFilter (FTM_Type ∗const ftmBase, uint8_t channel, uint8_t value)

  Sets the FTM peripheral timer channel input capture filter value.

FTM instance number

- #define FTM0_IDX (0U)

  Instance number for FTM0.
- #define FTM1_IDX (1U)

  Instance number for FTM2.
- #define FTM2_IDX (2U)

  Instance number for FTM3.
- #define FTM3_IDX (3U)

3.30.2 Macro Definition Documentation

3.30.2.1 CHAN0_IDX

```
#define CHAN0_IDX (0U)
```

Channel number for CHAN1.

3.30.2.2 CHAN1_IDX

```
#define CHAN1_IDX (1U)
```

Channel number for CHAN2.

3.30.2.3 CHAN2_IDX

```
#define CHAN2_IDX (2U)
```

Channel number for CHAN3.

3.30.2.4 CHAN3_IDX

```
#define CHAN3_IDX (3U)
```

Channel number for CHAN4.

3.30.2.5 CHAN4_IDX

```
#define CHAN4_IDX (4U)
```

Channel number for CHAN5.

3.30.2.6 CHAN5_IDX

```
#define CHAN5_IDX (5U)
```

Channel number for CHAN6.

3.30.2.7 CHAN6_IDX

```
#define CHAN6_IDX (6U)
```

Channel number for CHAN7.

3.30.2.8 CHAN7_IDX

```
#define CHAN7_IDX (7U)
```

3.30.2.9 FTM0_IDX

```
#define FTM0_IDX (0U)
```

Instance number for FTM0.

<Instance number for FTM1.

3.30.2.10 FTM1_IDX

```
#define FTM1_IDX (1U)
```

Instance number for FTM2.

3.30.2.11 FTM2_IDX

```
#define FTM2_IDX (2U)
```

Instance number for FTM3.

3.30.2.12 FTM3_IDX

```
#define FTM3_IDX (3U)
```

3.30.2.13 FTM_RMW_CnSCV_REG

```
#define FTM_RMW_CnSCV_REG(
              base,
              channel,
              mask,
              value ) (((base)->CONTROLS[channel].CnSC) = ((((base)->CONTROLS[channel].CnSC) &
~(mask)) | (value)))
```

FTM_CnSCV - Read and modify and write Channel (n) Status And Control (RW)

3.30.2.14 FTM_RMW_CNT

```
#define FTM_RMW_CNT(
              base,
              mask,
              value ) (((base)->CNT) = ((((base)->CNT) & ∼(mask)) | (value)))
```

FTM_CNT - Read and modify and write to Counter (RW)

3.30.2.15 FTM_RMW_CNTIN

```
#define FTM_RMW_CNTIN(
              base,
              mask,
              value ) (((base)->CNTIN) = ((((base)->CNTIN) & ∼(mask)) | (value)))
```

FTM_CNTIN - Read and modify and write Counter Initial Value (RW)

3.30.2.16 FTM_RMW_CONF

```
#define FTM_RMW_CONF(
              base,
              mask,
              value ) (((base)->CONF) = ((((base)->CONF) & ∼(mask)) | (value)))
```

FTM_CONF - Read and modify and write Configuration (RW)

3.30.2.17 FTM_RMW_DEADTIME

```
#define FTM_RMW_DEADTIME(
              base,
              mask,
              value ) (((base)->DEADTIME) = ((((base)->DEADTIME) & ∼(mask)) | (value)))
```

FTM_DEADTIME - Read and modify and write Dead-time Insertion Control (RW)

3.30.2.18 FTM_RMW_EXTTRIG_REG

```
#define FTM_RMW_EXTTRIG_REG(
              base,
              mask,
              value ) (((base)->EXTTRIG) = ((((base)->EXTTRIG) & ∼(mask)) | (value)))
```

FTM_EXTTRIG - Read and modify and write External Trigger Control (RW)

3.30.2.19 FTM_RMW_FILTER

```
#define FTM_RMW_FILTER(
              base,
              mask,
              value ) (((base)->FILTER) = ((((base)->FILTER) & ∼(mask)) | (value)))
```

FILTER - Read and modify and write Filter (RW)

3.30.2.20 FTM_RMW_FLTCTRL

```
#define FTM_RMW_FLTCTRL(
              base,
              mask,
              value ) (((base)->FLTCTRL) = ((((base)->FLTCTRL) & ~(mask)) | (value)))
```

FTM_FLTCTRL - Read and modify and write Fault Control (RW)

3.30.2.21 FTM_RMW_FMS

```
#define FTM_RMW_FMS(
              base,
              mask,
              value ) (((base)->FMS) = ((((base)->FMS) & ~(mask)) | (value)))
```

FTM_FMS - Read and modify and write Fault Mode Status (RW)

3.30.2.22 FTM_RMW_MOD

```
#define FTM_RMW_MOD(
              base,
              mask,
              value ) (((base)->MOD) = ((((base)->MOD) & ~(mask)) | (value)))
```

FTM_MOD - Read and modify and write Modulo (RW)

3.30.2.23 FTM_RMW_MODE

```
#define FTM_RMW_MODE(
              base,
              mask,
              value ) (((base)->MODE) = ((((base)->MODE) & ~(mask)) | (value)))
```

FTM_MODE - Read and modify and write Counter Features Mode Selection (RW)

3.30.2.24 FTM_RMW_PAIR0DEADTIME

```
#define FTM_RMW_PAIR0DEADTIME(
              base,
              mask,
              value ) (((base)->PAIR0DEADTIME) = ((((base)->PAIR0DEADTIME) & ~(mask)) | (value)))
```

FTM_PAIR0DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 0 (RW)

3.30.2.25 FTM_RMW_PAIR1DEADTIME

```
#define FTM_RMW_PAIR1DEADTIME(
              base,
              mask,
              value ) (((base)->PAIR1DEADTIME) = ((((base)->PAIR1DEADTIME) & ~(mask)) | (value)))
```

FTM_PAIR1DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 1 (RW)

3.30.2.26 FTM_RMW_PAIR2DEADTIME

```
#define FTM_RMW_PAIR2DEADTIME(
              base,
              mask,
              value ) (((base)->PAIR2DEADTIME) = ((((base)->PAIR2DEADTIME) & ~(mask)) | (value)))
```

FTM_PAIR2DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 2 (RW)

3.30.2.27 FTM_RMW_PAIR3DEADTIME

```
#define FTM_RMW_PAIR3DEADTIME(
              base,
              mask,
              value ) (((base)->PAIR3DEADTIME) = ((((base)->PAIR3DEADTIME) & ~(mask)) | (value)))
```

FTM_PAIR3DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 3 (RW)

Channel number for CHAN0.

3.30.2.28 FTM_RMW_POL

```
#define FTM_RMW_POL(
              base,
              mask,
              value ) (((base)->POL) = ((((base)->POL) & ~(mask)) | (value)))
```

POL - Read and modify and write Polarity (RW)

3.30.2.29 FTM_RMW_QDCTRL

```
#define FTM_RMW_QDCTRL(
              base,
              mask,
              value ) (((base)->QDCTRL) = ((((base)->QDCTRL) & ~(mask)) | (value)))
```

QDCTRL - Read and modify and write Quadrature Decoder Control And Status (RW)

3.30.2.30 FTM_RMW_SC

```
#define FTM_RMW_SC(
              base,
              mask,
              value ) (((base)->SC) = ((((base)->SC) & ~(mask)) | (value)))
```

FTM_SC - Read and modify and write to Status And Control (RW)

3.30.2.31 FTM_RMW_STATUS

```
#define FTM_RMW_STATUS(
              base,
              mask,
              value ) (((base)->STATUS) = ((((base)->STATUS) & ~(mask)) | (value)))
```

FTM_STATUS - Read and modify and write Capture And Compare Status (RW)

3.30.2.32 FTM_RMW_SYNC

```
#define FTM_RMW_SYNC(
              base,
              mask,
              value ) (((base)->SYNC) = ((((base)->SYNC) & ~(mask)) | (value)))
```

SYNC - Read and modify and write Synchronization (RW)

3.30.3 Enumeration Type Documentation

3.30.3.1 ftm_bdm_mode_t

```
enum ftm_bdm_mode_t
```

Options for the FlexTimer behavior in BDM Mode.

Implements : ftm_bdm_mode_t_Class

Enumerator

FTM_BDM_MODE_00	FTM counter stopped, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers
FTM_BDM_MODE_01	FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are forced to their safe value , writes to MOD,CNTIN and C(n)V registers bypass the register buffers
FTM_BDM_MODE_10	FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are frozen when chip enters in BDM mode, writes to MOD, CNTIN and C(n)V registers bypass the register buffers
FTM_BDM_MODE_11	FTM counter in functional mode, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers is in fully functional mode

3.30.3.2 ftm_clock_ps_t

```
enum ftm_clock_ps_t
```

FlexTimer pre-scaler factor selection for the clock source. In quadrature decoder mode set FTM_CLOCK_DIVID↩
_BY_1.

Implements : ftm_clock_ps_t_Class

Enumerator

FTM_CLOCK_DIVID_BY_1	Divide by 1
FTM_CLOCK_DIVID_BY_2	Divide by 2
FTM_CLOCK_DIVID_BY_4	Divide by 4
FTM_CLOCK_DIVID_BY_8	Divide by 8
FTM_CLOCK_DIVID_BY_16	Divide by 16
FTM_CLOCK_DIVID_BY_32	Divide by 32
FTM_CLOCK_DIVID_BY_64	Divide by 64
FTM_CLOCK_DIVID_BY_128	Divide by 128

3.30.3.3 ftm_clock_source_t

```
enum ftm_clock_source_t
```

FlexTimer clock source selection.

Implements : ftm_clock_source_t_Class

Enumerator

FTM_CLOCK_SOURCE_NONE	None use clock for FTM
FTM_CLOCK_SOURCE_SYSTEMCLK	System clock
FTM_CLOCK_SOURCE_FIXEDCLK	Fixed clock
FTM_CLOCK_SOURCE_EXTERNALCLK	External clock

3.30.3.4 ftm_deadtime_ps_t

```
enum ftm_deadtime_ps_t
```

FlexTimer pre-scaler factor for the dead-time insertion.

Implements : ftm_deadtime_ps_t_Class

Enumerator

FTM_DEADTIME_DIVID_BY_1	Divide by 1
FTM_DEADTIME_DIVID_BY_4	Divide by 4
FTM_DEADTIME_DIVID_BY_16	Divide by 16

3.30.3.5 ftm_fault_mode_t

```
enum ftm_fault_mode_t
```

FlexTimer fault control.

Implements : ftm_fault_mode_t_Class

Enumerator

FTM_FAULT_CONTROL_DISABLED	Fault control is disabled for all channels
FTM_FAULT_CONTROL_MAN_EVEN	Fault control is enabled for even channels only (channels 0, 2, 4, and 6), and the selected mode is the manual fault clearing
FTM_FAULT_CONTROL_MAN_ALL	Fault control is enabled for all channels, and the selected mode is the manual fault clearing
FTM_FAULT_CONTROL_AUTO_ALL	Fault control is enabled for all channels, and the selected mode is the automatic fault clearing

3.30.3.6 ftm_polarity_t

```
enum ftm_polarity_t
```

FlexTimer PWM output pulse mode, high-true or low-true on match up.

Implements : ftm_polarity_t_Class

Enumerator

FTM_POLARITY_LOW	When counter > CnV output signal is LOW
FTM_POLARITY_HIGH	When counter > CnV output signal is HIGH

3.30.3.7 ftm_pwm_sync_mode_t

```
enum ftm_pwm_sync_mode_t
```

FTM update register.

Implements : ftm_pwm_sync_mode_t_Class

Enumerator

FTM_WAIT_LOADING_POINTS	FTM register is updated at first loading point
FTM_UPDATE_NOW	FTM register is updated immediately

3.30.3.8 ftm_quad_decode_mode_t

```
enum ftm_quad_decode_mode_t
```

FlexTimer quadrature decode modes, phase encode or count and direction mode.

Implements : ftm_quad_decode_mode_t_Class

Enumerator

FTM_QUAD_PHASE_ENCODE	Phase encoding mode
FTM_QUAD_COUNT_AND_DIR	Counter and direction encoding mode

3.30.3.9 ftm_quad_phase_polarity_t

```
enum ftm_quad_phase_polarity_t
```

FlexTimer quadrature phase polarities, normal or inverted polarity.

Implements : ftm_quad_phase_polarity_t_Class

Enumerator

FTM_QUAD_PHASE_NORMAL	Phase input signal is not inverted before identifying the rising and falling edges of this signal
FTM_QUAD_PHASE_INVERT	Phase input signal is inverted before identifying the rising and falling edges of this signal

3.30.3.10 ftm_reg_update_t

```
enum ftm_reg_update_t
```

FTM sync source.

Implements : ftm_reg_update_t_Class

Enumerator

FTM_SYSTEM_CLOCK	Register is updated with its buffer value at all rising edges of system clock
FTM_PWM_SYNC	Register is updated with its buffer value at the FTM synchronization

3.30.3.11 ftm_second_channel_polarity_t

```
enum ftm_second_channel_polarity_t
```

FlexTimer PWM channel (n+1) polarity for combine mode.

Implements : ftm_second_channel_polarity_t_Class

Enumerator

FTM_MAIN_INVERTED	The channel (n+1) output is the inverse of the channel (n) output
FTM_MAIN_DUPLICATED	The channel (n+1) output is the same as the channel (n) output

3.30.4 Function Documentation

3.30.4.1 FTM_HAL_ClearChnEventFlag()

```
static void FTM_HAL_ClearChnEventFlag (
             FTM_Type *const ftmBase,
             uint8_t channel )  [inline], [static]
```

Clear the channel flag by writing a 0 to the CHF bit.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Implements : FTM_HAL_ClearChnEventFlag_Activity

3.30.4.2 FTM_HAL_ClearChnEventStatus()

```
static void FTM_HAL_ClearChnEventStatus (
             FTM_Type *const ftmBase,
             uint8_t channel )  [inline], [static]
```

Clears the FTM peripheral timer all channel event status.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Implements : FTM_HAL_ClearChnEventStatus_Activity

3.30.4.3 FTM_HAL_ClearChnTriggerFlag()

```
static void FTM_HAL_ClearChnTriggerFlag (
             FTM_Type *const ftmBase )  [inline], [static]
```

Clear the channel trigger flag.

Parameters

in	*ftmBase*	The FTM base address pointer

Implements : FTM_HAL_ClearChnTriggerFlag_Activity

3.30.4.4 FTM_HAL_ClearChSC()

```
static void FTM_HAL_ClearChSC (
             FTM_Type *const ftmBase,
             uint8_t channel )  [inline], [static]
```

Clears the content of Channel (n) Status And Control.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Implements : FTM_HAL_ClearChSC_Activity

3.30.4.5 FTM_HAL_ClearFaultControl()

```
static void FTM_HAL_ClearFaultControl (
             FTM_Type *const ftmBase )  [inline], [static]
```

Clears the entire content value of the Fault control register.

Parameters

in	*ftmBase*	The FTM base address pointer

Implements : FTM_HAL_ClearFaultControl_Activity

3.30.4.6 FTM_HAL_ClearFaultFlagDetected()

```
static void FTM_HAL_ClearFaultFlagDetected (
            FTM_Type *const ftmBase,
            uint8_t channel )  [inline], [static]
```

Clear a fault condition is detected at the fault input.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel

Implements : FTM_HAL_ClearFaultFlagDetected_Activity

3.30.4.7 FTM_HAL_ClearFaultsIsr()

```
static void FTM_HAL_ClearFaultsIsr (
            FTM_Type *const ftmBase )  [inline], [static]
```

Clears all fault interrupt flags that are active.

Parameters

in	*ftmBase*	The FTM base address pointer

Implements : FTM_HAL_ClearFaultsIsr_Activity

3.30.4.8 FTM_HAL_ClearReloadFlag()

```
static void FTM_HAL_ClearReloadFlag (
            FTM_Type *const ftmBase )  [inline], [static]
```

Clears the reload flag bit.

Parameters

in	*ftmBase*	The FTM base address pointer

Implements : FTM_HAL_ClearReloadFlag_Activity

3.30.4.9 FTM_HAL_ClearTimerOverflow()

```
static void FTM_HAL_ClearTimerOverflow (
            FTM_Type *const ftmBase )  [inline], [static]
```

Clears the timer overflow interrupt flag.

Parameters

in	*ftmBase*	The FTM base address pointer

Implements : FTM_HAL_ClearTimerOverflow_Activity

3.30.4.10 FTM_HAL_DisableChnInt()

```
static void FTM_HAL_DisableChnInt (
            FTM_Type *const ftmBase,
            uint8_t channel )  [inline], [static]
```

Disables the FTM peripheral timer channel(n) interrupt.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Implements : FTM_HAL_DisableChnInt_Activity

3.30.4.11 FTM_HAL_DisableFaultInt()

```
static void FTM_HAL_DisableFaultInt (
            FTM_Type *const ftmBase )  [inline], [static]
```

Disables the FTM peripheral timer fault interrupt.

Parameters

in	*ftmBase*	The FTM base address pointer

Implements : FTM_HAL_DisableFaultInt_Activity

3.30.4.12 FTM_HAL_DisablePwmChannelOutputs()

```
static void FTM_HAL_DisablePwmChannelOutputs (
            FTM_Type *const ftmBase,
            uint8_t channel )  [inline], [static]
```

Disable PWM channel Outputs.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM channel

Implements : FTM_HAL_DisablePwmChannelOutputs_Activity

3.30.4.13 FTM_HAL_Enable()

```
static void FTM_HAL_Enable (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Enables the FTM peripheral timer group.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	FTM mode selection • true : All registers including FTM-specific registers are available • false: Only the TPM-compatible registers are available

Implements : FTM_HAL_Enable_Activity

3.30.4.14 FTM_HAL_EnableChnInt()

```
static void FTM_HAL_EnableChnInt (
            FTM_Type *const ftmBase,
            uint8_t channel )  [inline], [static]
```

Enables the FTM peripheral timer channel(n) interrupt.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Implements : FTM_HAL_EnableChnInt_Activity

3.30.4.15 FTM_HAL_EnablePwmChannelOutputs()

```
static void FTM_HAL_EnablePwmChannelOutputs (
            FTM_Type *const ftmBase,
            uint8_t channel )  [inline], [static]
```

Enable PWM channel Outputs.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM channel

Implements : FTM_HAL_EnablePwmChannelOutputs_Activity

3.30.4.16 FTM_HAL_GetChInputState()

```
static bool FTM_HAL_GetChInputState (
            const FTM_Type * ftmBase,
            uint8_t channel )  [inline], [static]
```

Get the state of channel input.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Returns

State of the channel inputs

- true : The channel input is one
- false: The channel input is zero

Implements : FTM_HAL_GetChInputState_Activity

3.30.4.17 FTM_HAL_GetChnCountVal()

```
static uint16_t FTM_HAL_GetChnCountVal (
            const FTM_Type * ftmBase,
            uint8_t channel )  [inline], [static]
```

Gets the FTM peripheral timer channel counter value.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Returns

Channel counter value

Implements : FTM_HAL_GetChnCountVal_Activity

3.30.4.18 FTM_HAL_GetChnEdgeLevel()

```
static uint8_t FTM_HAL_GetChnEdgeLevel (
            const FTM_Type * ftmBase,
            uint8_t channel )  [inline], [static]
```

Gets the FTM peripheral timer channel edge level.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Returns

The ELSnB:ELSnA mode value, will be 00, 01, 10, 11

Implements : FTM_HAL_GetChnEdgeLevel_Activity

3.30.4.19 FTM_HAL_GetChnEventStatus()

```
static bool FTM_HAL_GetChnEventStatus (
            const FTM_Type * ftmBase,
            uint8_t channel ) [inline], [static]
```

Gets the FTM peripheral timer channel event status.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Returns

Channel event status

- true : A channel event has occurred
- false : No channel event has occurred

Implements : FTM_HAL_GetChnEventStatus_Activity

3.30.4.20 FTM_HAL_GetChnMode()

```
static uint8_t FTM_HAL_GetChnMode (
            const FTM_Type * ftmBase,
            uint8_t channel ) [inline], [static]
```

Gets the FTM peripheral timer channel mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Returns

The MSnB:MSnA mode value, will be 00, 01, 10, 11

Implements : FTM_HAL_GetChnMode_Activity

3.30.4.21 FTM_HAL_GetChOutputValue()

```
static bool FTM_HAL_GetChOutputValue (
            const FTM_Type * ftmBase,
            uint8_t channel ) [inline], [static]
```

Get the value of channel output.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Returns

Value of the channel outputs

- true : The channel output is one
- false: The channel output is zero

Implements : FTM_HAL_GetChOutputValue_Activity

3.30.4.22 FTM_HAL_GetClockFilterPs()

```
static uint8_t FTM_HAL_GetClockFilterPs (
            const FTM_Type * ftmBase )  [inline], [static]
```

Reads the FTM filter clock divider.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

The FTM filter clock pre-scale divider

Implements : FTM_HAL_GetClockFilterPs_Activity

3.30.4.23 FTM_HAL_GetClockPs()

```
static uint8_t FTM_HAL_GetClockPs (
            const FTM_Type * ftmBase )  [inline], [static]
```

Reads the FTM clock divider.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

The FTM clock pre-scale divider

Implements : FTM_HAL_GetClockPs_Activity

3.30.4.24 FTM_HAL_GetClockSource()

```
static uint8_t FTM_HAL_GetClockSource (
            const FTM_Type * ftmBase )  [inline], [static]
```

Reads the FTM clock source.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

The FTM clock source selection

- 00: No clock
- 01: system clock
- 10: fixed clock
- 11: External clock

Implements : FTM_HAL_GetClockSource_Activity

3.30.4.25 FTM_HAL_GetCounter()

```
static uint16_t FTM_HAL_GetCounter (
            const FTM_Type * ftmBase )  [inline], [static]
```

Returns the FTM peripheral current counter value.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

The current FTM timer counter value

Implements : FTM_HAL_GetCounter_Activity

3.30.4.26 FTM_HAL_GetCounterInitVal()

```
static uint16_t FTM_HAL_GetCounterInitVal (
            const FTM_Type * ftmBase )  [inline], [static]
```

Returns the FTM peripheral counter initial value.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

FTM timer counter initial value

Implements : FTM_HAL_GetCounterInitVal_Activity

3.30.4.27 FTM_HAL_GetCpwms()

```
static bool FTM_HAL_GetCpwms (
            const FTM_Type * ftmBase )  [inline], [static]
```

Gets the FTM count direction bit.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

The Center-Aligned PWM selection

- 1U: Up counting mode
- 0U: Up down counting mode

Implements : FTM_HAL_GetCpwms_Activity

3.30.4.28 FTM_HAL_GetDetectedFaultInput()

```
static bool FTM_HAL_GetDetectedFaultInput (
            const FTM_Type * ftmBase )  [inline], [static]
```

Gets the FTM detected fault input.

This function reads the status for all fault inputs

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

The fault byte

- 0 : No fault condition was detected.
- 1 : A fault condition was detected.

Implements : FTM_HAL_GetDetectedFaultInput_Activity

3.30.4.29 FTM_HAL_GetDualChnCombineCmd()

```
static bool FTM_HAL_GetDualChnCombineCmd (
            const FTM_Type * ftmBase,
            uint8_t chnlPairNum )  [inline], [static]
```

Verify if an channels pair is used in combine mode or not.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number

Returns

Channel pair output combine mode status
- true : Channels pair are combined
- false: Channels pair are independent

Implements : FTM_HAL_GetDualChnCombineCmd_Activity

3.30.4.30 FTM_HAL_GetDualChnMofCombineCmd()

```
static bool FTM_HAL_GetDualChnMofCombineCmd (
            const FTM_Type * ftmBase,
            uint8_t chnlPairNum )  [inline], [static]
```

Verify if an channels pair is used in the modified combine mode or not.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number

Returns

Channel pair output combine mode status
- true : Channels pair are combined
- false: Channels pair are independent

Implements : FTM_HAL_GetDualChnMofCombineCmd_Activity

3.30.4.31 FTM_HAL_GetDualEdgeCaptureBit()

```
static bool FTM_HAL_GetDualEdgeCaptureBit (
            const FTM_Type * ftmBase,
            uint8_t chnlPairNum )  [inline], [static]
```

Enables the FTM peripheral timer dual edge capture mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number

Returns

Dual edge capture mode status
- true : To enable dual edge capture
- false: To disable

Implements : FTM_HAL_GetDualEdgeCaptureBit_Activity

3.30.4.32 FTM_HAL_GetEventStatus()

```
static uint32_t FTM_HAL_GetEventStatus (
            const FTM_Type * ftmBase )  [inline], [static]
```

Gets the FTM peripheral timer status info for all channels.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

Channel event status value

Implements : FTM_HAL_GetEventStatus_Activity

3.30.4.33 FTM_HAL_GetMod()

```
static uint16_t FTM_HAL_GetMod (
            const FTM_Type * ftmBase )  [inline], [static]
```

Returns the FTM peripheral counter modulo value.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

FTM timer modulo value

Implements : FTM_HAL_GetMod_Activity

3.30.4.34 FTM_HAL_GetQuadDir()

```
static bool FTM_HAL_GetQuadDir (
            const FTM_Type * ftmBase )  [inline], [static]
```

Gets the FTM counter direction in quadrature mode.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

The counting direction

- 1U: if counting direction is increasing
- 0U: if counting direction is decreasing

Implements : FTM_HAL_GetQuadDir_Activity

3.30.4.35 FTM_HAL_GetQuadTimerOverflowDir()

```
static bool FTM_HAL_GetQuadTimerOverflowDir (
            const FTM_Type * ftmBase ) [inline], [static]
```

Gets the Timer overflow direction in quadrature mode.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

The timer overflow direction

- 1U: if TOF bit was set on the top of counting
- 0U: if TOF bit was set on the bottom of counting

Implements : FTM_HAL_GetQuadTimerOverflowDir_Activity

3.30.4.36 FTM_HAL_GetReloadFlag()

```
static bool FTM_HAL_GetReloadFlag (
            const FTM_Type * ftmBase ) [inline], [static]
```

Get the state whether the FTM counter reached a reload point.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

State of reload point

- true : FTM counter reached a reload point
- false: FTM counter did not reach a reload point

Implements : FTM_HAL_GetReloadFlag_Activity

3.30.4.37 FTM_HAL_GetTriggerControled()

```
static bool FTM_HAL_GetTriggerControled (
            const FTM_Type * ftmBase,
            uint8_t channel ) [inline], [static]
```

Returns whether the trigger mode is enabled.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Returns

State of the channel outputs

- true : Enabled a trigger generation on channel output
- false: PWM outputs without generating a pulse

Implements : FTM_HAL_GetTriggerControled_Activity

3.30.4.38 FTM_HAL_HasChnEventOccurred()

```
static bool FTM_HAL_HasChnEventOccurred (
             const FTM_Type * ftmBase,
             uint8_t channel )  [inline], [static]
```

Returns whether any event for the FTM peripheral timer channel has occurred.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Returns

State of channel flag

- true : Event occurred
- false: No event occurred.

Implements : FTM_HAL_HasChnEventOccurred_Activity

3.30.4.39 FTM_HAL_HasTimerOverflowed()

```
static bool FTM_HAL_HasTimerOverflowed (
             const FTM_Type * ftmBase )  [inline], [static]
```

Returns the FTM peripheral timer overflow interrupt flag.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

State of Timer Overflow Flag

- true : FTM counter has overflowed

- false: FTM counter has not overflowed

Implements : FTM_HAL_HasTimerOverflowed_Activity

3.30.4.40 FTM_HAL_Init()

```
void FTM_HAL_Init (
             FTM_Type *const ftmBase,
             ftm_clock_ps_t ftmClockPrescaler )
```

Initializes the FTM. This function will enable the flexTimer module and selects one pre-scale factor for the FTM clock source.

Parameters

in	*ftmBase*	The FTM base address pointer

3.30.4.41 FTM_HAL_IsChnDma()

```
static bool FTM_HAL_IsChnDma (
             const FTM_Type * ftmBase,
             uint8_t channel )  [inline], [static]
```

Returns whether the FTM peripheral timer channel DMA is enabled.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Returns

State of the FTM peripheral timer channel DMA

- true : Enabled DMA transfers
- false: Disabled DMA transfers

Implements : FTM_HAL_IsChnDma_Activity

3.30.4.42 FTM_HAL_IsChnIcrst()

```
static bool FTM_HAL_IsChnIcrst (
             const FTM_Type * ftmBase,
             uint8_t channel )  [inline], [static]
```

Returns whether the FTM FTM counter is reset.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number

Returns

State of the FTM peripheral timer channel ICRST

- true : Enabled the FTM counter reset
- false: Disabled the FTM counter reset

Implements : FTM_HAL_IsChnIcrst_Activity

3.30.4.43 FTM_HAL_IsChnIntEnabled()

```
static bool FTM_HAL_IsChnIntEnabled (
            const FTM_Type * ftmBase,
            uint8_t channel ) [inline], [static]
```

Get FTM channel(n) interrupt enabled or not.

Parameters

in	*ftmBase*	FTM module base address
in	*channel*	The FTM peripheral channel number

Implements : FTM_HAL_IsChnIntEnabled_Activity

3.30.4.44 FTM_HAL_IsChnTriggerGenerated()

```
static bool FTM_HAL_IsChnTriggerGenerated (
            const FTM_Type * ftmBase ) [inline], [static]
```

Checks whether any channel trigger event has occurred.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

Channel trigger status

- true : If there is a channel trigger event
- false: If not.

Implements : FTM_HAL_IsChnTriggerGenerated_Activity

3.30.4.45 FTM_HAL_IsFaultFlagDetected()

```
static bool FTM_HAL_IsFaultFlagDetected (
            const FTM_Type * ftmBase,
            uint8_t channel ) [inline], [static]
```

Checks whether a fault condition is detected at the fault input.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel

Returns

the fault condition status

- true : A fault condition was detected at the fault input
- false: No fault condition was detected at the fault input

Implements : FTM_HAL_IsFaultFlagDetected_Activity

3.30.4.46 FTM_HAL_IsFaultInputEnabled()

```
static bool FTM_HAL_IsFaultInputEnabled (
            const FTM_Type * ftmBase )  [inline], [static]
```

Checks whether the logic OR of the fault inputs is enabled.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

the enabled fault inputs status

- true : The logic OR of the enabled fault inputs is 1
- false: The logic OR of the enabled fault inputs is 0

Implements : FTM_HAL_IsFaultInputEnabled_Activity

3.30.4.47 FTM_HAL_IsFaultIntEnabled()

```
static bool FTM_HAL_IsFaultIntEnabled (
            const FTM_Type * ftmBase )  [inline], [static]
```

Return true/false whether the Fault interrupt was enabled or not.

Parameters

in	*ftmBase*	The FTM base address pointer

Implements : FTM_HAL_IsFaultIntEnabled_Activity

3.30.4.48 FTM_HAL_IsFtmEnable()

```
static bool FTM_HAL_IsFtmEnable (
            const FTM_Type * ftmBase )  [inline], [static]
```

Get status of the FTMEN bit in the FTM_MODE register.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

the FTM Enable status

- true : TPM compatibility. Free running counter and synchronization compatible with TPM
- false: Free running counter and synchronization are different from TPM behaviour

Implements : FTM_HAL_IsFtmEnable_Activity

3.30.4.49 FTM_HAL_IsOverflowIntEnabled()

```
static bool FTM_HAL_IsOverflowIntEnabled (
            const FTM_Type * ftmBase )  [inline], [static]
```

Reads the bit that controls enabling the FTM timer overflow interrupt.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

State of Timer Overflow Interrupt

- true : Overflow interrupt is enabled
- false: Overflow interrupt is disabled

Implements : FTM_HAL_IsOverflowIntEnabled_Activity

3.30.4.50 FTM_HAL_IsWriteProtectionEnabled()

```
static bool FTM_HAL_IsWriteProtectionEnabled (
            const FTM_Type * ftmBase )  [inline], [static]
```

Checks whether the write protection is enabled.

Parameters

in	*ftmBase*	The FTM base address pointer

Returns

Write-protection status

- true : If enabled
- false: If not

Implements : FTM_HAL_IsWriteProtectionEnabled_Activity

3.30.4.51 FTM_HAL_Reset()

```
void FTM_HAL_Reset (
            FTM_Type *const ftmBase )
```

Resets the FTM registers. All the register use in the driver should be reset to default value of each register.

Parameters

in	*ftmBase*	The FTM base address pointer

3.30.4.52 FTM_HAL_SetAllChnSoftwareCtrlCmd()

```
static void FTM_HAL_SetAllChnSoftwareCtrlCmd (
            FTM_Type *const ftmBase,
            uint8_t channelsMask )  [inline], [static]
```

Enables or disables the channel software output control.The main difference between this function and FTM_HA↩
L_SetChnSoftwareCtrlCmd is that this can configure all the channels in the same time.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channelsMask*	Channels to be enabled or disabled

Implements : FTM_HAL_SetAllChnSoftwareCtrlCmd_Activity

3.30.4.53 FTM_HAL_SetAllChnSoftwareCtrlVal()

```
static void FTM_HAL_SetAllChnSoftwareCtrlVal (
            FTM_Type *const ftmBase,
            uint8_t channelsValues )  [inline], [static]
```

Sets the channel software output control value.

Parameters

in	*ftmBase*	The FTM base address pointer.
in	*channelsValues*	The values which will overwrite the output channels

Implements : FTM_HAL_SetAllChnSoftwareCtrlVal_Activity

3.30.4.54 FTM_HAL_SetBdmMode()

```
static void FTM_HAL_SetBdmMode (
            FTM_Type *const ftmBase,
            ftm_bdm_mode_t val )  [inline], [static]
```

Sets the BDM mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*val*	The FTM behavior in BDM mode • FTM_BDM_MODE_00: FTM counter stopped, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers • FTM_BDM_MODE_01: FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are forced to their safe value , writes to MOD,CNTIN and C(n)V registers bypass the register buffers • FTM_BDM_MODE_10: FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are frozen when chip enters in BDM mode, writes to MOD, CNTIN and C(n)V registers bypass the register buffers • FTM_BDM_MODE_11: FTM counter in functional mode, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers is in fully functional mode

Implements : FTM_HAL_SetBdmMode_Activity

3.30.4.55 FTM_HAL_SetCaptureTestCmd()

```
static void FTM_HAL_SetCaptureTestCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Enables or disables the FTM peripheral timer capture test mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	Capture Test Mode Enable • true : Capture test mode is enabled • false: Capture test mode is disabled

Implements : FTM_HAL_SetCaptureTestCmd_Activity

3.30.4.56 FTM_HAL_SetChnCountVal()

```
static void FTM_HAL_SetChnCountVal (
             FTM_Type *const ftmBase,
             uint8_t channel,
             uint16_t value )  [inline], [static]
```

Sets the FTM peripheral timer channel counter value.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number
in	*value*	Counter value to be set

Implements : FTM_HAL_SetChnCountVal_Activity

3.30.4.57 FTM_HAL_SetChnDmaCmd()

```
static void FTM_HAL_SetChnDmaCmd (
             FTM_Type *const ftmBase,
             uint8_t channel,
             bool enable )  [inline], [static]
```

Enables or disables the FTM peripheral timer channel DMA.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number
in	*enable*	Enable DMA transfers for the channel • true : Enabled DMA transfers • false: Disabled DMA transfers

Implements : FTM_HAL_SetChnDmaCmd_Activity

3.30.4.58 FTM_HAL_SetChnEdgeLevel()

```
static void FTM_HAL_SetChnEdgeLevel (
             FTM_Type *const ftmBase,
             uint8_t channel,
             uint8_t level )  [inline], [static]
```

Sets the FTM peripheral timer channel edge level.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number
in	*level*	ELSnB:ELSnA :00, 01, 10, 11

Implements : FTM_HAL_SetChnEdgeLevel_Activity

3.30.4.59 FTM_HAL_SetChnFaultInputPolarityCmd()

```
static void FTM_HAL_SetChnFaultInputPolarityCmd (
             FTM_Type *const ftmBase,
             uint8_t fltChannel,
             ftm_polarity_t polarity )  [inline], [static]
```

Sets the FTM peripheral timer channel fault input polarity.

Parameters

in	*ftmBase*	The FTM base address pointer

Parameters

in	*channel*	The FTM peripheral channel number
in	*polarity*	The polarity to be set • FTM_POLARITY_LOW : The fault input polarity is active high • FTM_POLARITY_HIGH: The fault input polarity is active low

Implements : FTM_HAL_SetChnFaultInputPolarityCmd_Activity

3.30.4.60 FTM_HAL_SetChnIcrstCmd()

```
static void FTM_HAL_SetChnIcrstCmd (
            FTM_Type *const ftmBase,
            uint8_t channel,
            bool enable ) [inline], [static]
```

Configure the feature of FTM counter reset by the selected input capture event.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number
in	*enable*	Enable the FTM counter reset • true : FTM counter is reset • false: FTM counter is not reset

Implements : FTM_HAL_SetChnIcrstCmd_Activity

3.30.4.61 FTM_HAL_SetChnInputCaptureFilter()

```
void FTM_HAL_SetChnInputCaptureFilter (
            FTM_Type *const ftmBase,
            uint8_t channel,
            uint8_t value )
```

Sets the FTM peripheral timer channel input capture filter value.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number, only 0,1,2,3, channel 4, 5,6, 7 don't have
in	*value*	Filter value to be set

3.30.4.62 FTM_HAL_SetChnMSnBAMode()

```
static void FTM_HAL_SetChnMSnBAMode (
            FTM_Type *const ftmBase,
            uint8_t channel,
            uint8_t selection )  [inline], [static]
```

Sets the FTM peripheral timer channel mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number
in	*selection*	The mode to be set valid value MSnB:MSnA :00, 01, 10, 11

Implements : FTM_HAL_SetChnMSnBAMode_Activity

3.30.4.63 FTM_HAL_SetChnOutputInitStateCmd()

```
static void FTM_HAL_SetChnOutputInitStateCmd (
            FTM_Type *const ftmBase,
            uint8_t channel,
            bool state )  [inline], [static]
```

Sets the FTM peripheral timer channel output initial state 0 or 1.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number
in	*state*	Initial state for channels output • true : The initialization value is 1 • false: The initialization value is 0

Implements : FTM_HAL_SetChnOutputInitStateCmd_Activity

3.30.4.64 FTM_HAL_SetChnOutputMask()

```
static void FTM_HAL_SetChnOutputMask (
            FTM_Type *const ftmBase,
            uint8_t channel,
            bool mask )  [inline], [static]
```

Sets the FTM peripheral timer channel output mask.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number
in	*mask*	Value to set Output Mask • true : Channel output is masked • false: Channel output is not masked

Implements : FTM_HAL_SetChnOutputMask_Activity

3.30.4.65 FTM_HAL_SetChnOutputPolarityCmd()

```
static void FTM_HAL_SetChnOutputPolarityCmd (
             FTM_Type *const ftmBase,
             uint8_t channel,
             ftm_polarity_t polarity )  [inline], [static]
```

Sets the FTM peripheral timer channel output polarity.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number
in	*polarity*	The polarity to be set • FTM_POLARITY_HIGH : The channel polarity is active low • FTM_POLARITY_LOW : The channel polarity is active high

Implements : FTM_HAL_SetChnOutputPolarityCmd_Activity

3.30.4.66 FTM_HAL_SetChnSoftwareCtrlCmd()

```
static void FTM_HAL_SetChnSoftwareCtrlCmd (
             FTM_Type *const ftmBase,
             uint8_t channel,
             bool enable )  [inline], [static]
```

Enables or disables the channel software output control.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	Channel to be enabled or disabled
in	*enable*	State of channel software output control • true : To enable, channel output will be affected by software output control • false: To disable, channel output is unaffected

Implements : FTM_HAL_SetChnSoftwareCtrlCmd_Activity

3.30.4.67 FTM_HAL_SetChnSoftwareCtrlVal()

```
static void FTM_HAL_SetChnSoftwareCtrlVal (
             FTM_Type *const ftmBase,
             uint8_t channel,
             bool enable )  [inline], [static]
```

Sets the channel software output control value.

Parameters

in	*ftmBase*	The FTM base address pointer.
in	*channel*	Channel to be configured
in	*enable*	State of software output control value • true : to force 1 to the channel output • false: to force 0 to the channel output

Implements : FTM_HAL_SetChnSoftwareCtrlVal_Activity

3.30.4.68 FTM_HAL_SetChnTriggerCmd()

```
void FTM_HAL_SetChnTriggerCmd (
            FTM_Type *const ftmBase,
            uint8_t channel,
            bool enable )
```

Enables or disables the generation of the FTM peripheral timer channel trigger when the FTM counter is equal to its initial value.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number
in	*enable*	Enables the generation of the channel trigger • true : The generation of the channel trigger is enabled • false: The generation of the channel trigger is disabled

3.30.4.69 FTM_HAL_SetClockFilterPs()

```
static void FTM_HAL_SetClockFilterPs (
            FTM_Type *const ftmBase,
            uint8_t filterPrescale )  [inline], [static]
```

Sets the filter Pre-scaler divider.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*filterPrescale*	The FTM peripheral clock pre-scale divider

Implements : FTM_HAL_SetClockFilterPs_Activity

3.30.4.70 FTM_HAL_SetClockPs()

```
static void FTM_HAL_SetClockPs (
            FTM_Type *const ftmBase,
            ftm_clock_ps_t ps )  [inline], [static]
```

Sets the FTM clock divider.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*ps*	The FTM peripheral clock pre-scale divider

Implements : FTM_HAL_SetClockPs_Activity

3.30.4.71 FTM_HAL_SetClockSource()

```
static void FTM_HAL_SetClockSource (
            FTM_Type *const ftmBase,
            ftm_clock_source_t clock )  [inline], [static]
```

Sets the FTM clock source.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*clock*	The FTM peripheral clock selection • 00: No clock • 01: system clock • 10: fixed clock • 11: External clock

Implements : FTM_HAL_SetClockSource_Activity

3.30.4.72 FTM_HAL_SetCntinPwmSyncModeCmd()

```
static void FTM_HAL_SetCntinPwmSyncModeCmd (
            FTM_Type *const ftmBase,
            ftm_reg_update_t mode )  [inline], [static]
```

Sets the CNTIN register PWM synchronization mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*mode*	State of register synchronization • true : CNTIN register is updated by PWM sync • false: CNTIN register is updated at all rising edges of system clock

Implements : FTM_HAL_SetCntinPwmSyncModeCmd_Activity

3.30.4.73 FTM_HAL_SetCounter()

```
static void FTM_HAL_SetCounter (
            FTM_Type *const ftmBase,
            uint16_t value )  [inline], [static]
```

Sets the FTM peripheral current counter value.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*val*	The FTM timer counter value to be set

Implements : FTM_HAL_SetCounter_Activity

3.30.4.74 FTM_HAL_SetCounterHardwareSyncModeCmd()

```
static void FTM_HAL_SetCounterHardwareSyncModeCmd (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Sets sync mode for FTM counter register when using a hardware trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of FTM counter synchronization • true : The hardware trigger activates FTM counter sync • false: The hardware trigger does not activate FTM counter sync

Implements : FTM_HAL_SetCounterHardwareSyncModeCmd_Activity

3.30.4.75 FTM_HAL_SetCounterInitVal()

```
static void FTM_HAL_SetCounterInitVal (
            FTM_Type *const ftmBase,
            uint16_t value )  [inline], [static]
```

Sets the FTM peripheral timer counter initial value.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*value*	initial value to be set

Implements : FTM_HAL_SetCounterInitVal_Activity

3.30.4.76 FTM_HAL_SetCounterSoftwareSyncModeCmd()

```
static void FTM_HAL_SetCounterSoftwareSyncModeCmd (
            FTM_Type *const ftmBase,
            ftm_pwm_sync_mode_t update_mode )  [inline], [static]
```

Sets sync mode for FTM counter register when using a software trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*update_mode*	State of FTM counter synchronization • true : The software trigger activates FTM counter sync • false: The software trigger does not activate FTM counter sync

Implements : FTM_HAL_SetCounterSoftwareSyncModeCmd_Activity

3.30.4.77 FTM_HAL_SetCountReinitSyncCmd()

```
static void FTM_HAL_SetCountReinitSyncCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Determines if the FTM counter is re-initialized when the selected trigger for synchronization is detected.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	FTM counter re-initialization selection • true : To update FTM counter when triggered • false: To count normally

Implements : FTM_HAL_SetCountReinitSyncCmd_Activity

3.30.4.78 FTM_HAL_SetCpwms()

```
static void FTM_HAL_SetCpwms (
             FTM_Type *const ftmBase,
             bool mode )  [inline], [static]
```

Sets the FTM count direction bit.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*mode*	The Center-Aligned PWM selection • 1U: Up counting mode • 0U: Up down counting mode

Implements : FTM_HAL_SetCpwms_Activity

3.30.4.79 FTM_HAL_SetDeadtimeCount()

```
static void FTM_HAL_SetDeadtimeCount (
            FTM_Type *const ftmBase,
            uint8_t count )  [inline], [static]
```

Sets the FTM deadtime value.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*count*	The FTM peripheral pre-scaler divider • 0U : no counts inserted • 1U : 1 count is inserted • 2U : 2 count is inserted • ... up to a possible 63 counts

Implements : FTM_HAL_SetDeadtimeCount_Activity

3.30.4.80 FTM_HAL_SetDeadtimePrescale()

```
static void FTM_HAL_SetDeadtimePrescale (
            FTM_Type *const ftmBase,
            ftm_deadtime_ps_t divider )  [inline], [static]
```

Sets the FTM dead time divider.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*divider*	The FTM peripheral pre-scaler divider • FTM_DEADTIME_DIVID_BY_1 : Divide by 1 • FTM_DEADTIME_DIVID_BY_4 : Divide by 4 • FTM_DEADTIME_DIVID_BY_16: Divide by 16

Implements : FTM_HAL_SetDeadtimePrescale_Activity

3.30.4.81 FTM_HAL_SetDualChnCombineCmd()

```
static void FTM_HAL_SetDualChnCombineCmd (
            FTM_Type *const ftmBase,
            uint8_t chnlPairNum,
            bool enable )  [inline], [static]
```

Enables or disables the FTM peripheral timer channel pair output combine mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number

Parameters

in	*enable*	State of channel pair output combine mode • true : Channels pair are combined • false: Channels pair are independent

Implements : FTM_HAL_SetDualChnCombineCmd_Activity

3.30.4.82 FTM_HAL_SetDualChnCompCmd()

```
static void FTM_HAL_SetDualChnCompCmd (
            FTM_Type *const ftmBase,
            uint8_t chnlPairNum,
            ftm_second_channel_polarity_t polarity )  [inline], [static]
```

Enables or disables the FTM peripheral timer channel pair output complement mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number
in	*polarity*	State of channel pair output complement mode • FTM_MAIN_INVERTED : The channel (n+1) output is the complement of the channel (n) output • FTM_MAIN_DUPLICATED: The channel (n+1) output is the same as the channel (n) output

Implements : FTM_HAL_SetDualChnCompCmd_Activity

3.30.4.83 FTM_HAL_SetDualChnDeadtimeCmd()

```
static void FTM_HAL_SetDualChnDeadtimeCmd (
            FTM_Type *const ftmBase,
            uint8_t chnlPairNum,
            bool enable )  [inline], [static]
```

Enables or disabled the FTM peripheral timer channel pair deadtime insertion.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number
in	*enable*	State of channel pair deadtime insertion • true : To enable deadtime insertion • false: To disable

Implements : FTM_HAL_SetDualChnDeadtimeCmd_Activity

3.30.4.84 FTM_HAL_SetDualChnDecapCmd()

```
static void FTM_HAL_SetDualChnDecapCmd (
            FTM_Type *const ftmBase,
            uint8_t chnlPairNum,
            bool enable )  [inline], [static]
```

Enables or disables the FTM peripheral timer channel dual edge capture.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number
in	*enable*	State of channel dual edge capture • true : To enable dual edge capture mode • false: To disable

Implements : FTM_HAL_SetDualChnDecapCmd_Activity

3.30.4.85 FTM_HAL_SetDualChnFaultCmd()

```
static void FTM_HAL_SetDualChnFaultCmd (
            FTM_Type *const ftmBase,
            uint8_t chnlPairNum,
            bool enable )  [inline], [static]
```

Enables the FTM peripheral timer channel pair fault control.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number
in	*enable*	State of channel pair fault control • true : To enable fault control • false: To disable

Implements : FTM_HAL_SetDualChnFaultCmd_Activity

3.30.4.86 FTM_HAL_SetDualChnInvertCmd()

```
static void FTM_HAL_SetDualChnInvertCmd (
            FTM_Type *const ftmBase,
            uint8_t chnlPairNum,
            bool enable )  [inline], [static]
```

Enables or disables the channel invert for a channel pair.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number
in	*enable*	State of channel invert for a channel pair • true : To enable channel inverting • false: To disable channel inversion

Implements : FTM_HAL_SetDualChnInvertCmd_Activity

3.30.4.87 FTM_HAL_SetDualChnMofCombineCmd()

```
static void FTM_HAL_SetDualChnMofCombineCmd (
             FTM_Type *const ftmBase,
             uint8_t chnlPairNum,
             bool enable )  [inline], [static]
```

Enables the FTM peripheral timer channel modified combine mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number
in	*enable*	State of channel pair outputs modified combine • true : To enable modified combine • false: To disable modified combine

Implements : FTM_HAL_SetDualChnMofCombineCmd_Activity

3.30.4.88 FTM_HAL_SetDualChnPwmSyncCmd()

```
static void FTM_HAL_SetDualChnPwmSyncCmd (
             FTM_Type *const ftmBase,
             uint8_t chnlPairNum,
             bool enable )  [inline], [static]
```

Enables or disables the FTM peripheral timer channel pair counter PWM sync.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number
in	*enable*	State of channel pair counter PWM sync • true : To enable PWM synchronization • false: To disable

Implements : FTM_HAL_SetDualChnPwmSyncCmd_Activity

3.30.4.89 FTM_HAL_SetDualEdgeCaptureCmd()

```
static void FTM_HAL_SetDualEdgeCaptureCmd (
             FTM_Type *const ftmBase,
             uint8_t chnlPairNum,
             bool enable )  [inline], [static]
```

Enables the FTM peripheral timer dual edge capture mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*chnlPairNum*	The FTM peripheral channel pair number
in	*enable*	State of dual edge capture mode • true : To enable dual edge capture • false: To disable

Implements : FTM_HAL_SetDualEdgeCaptureCmd_Activity

3.30.4.90 FTM_HAL_SetExtDeadtimeValue()

```
static void FTM_HAL_SetExtDeadtimeValue (
             FTM_Type *const ftmBase,
             uint8_t value )  [inline], [static]
```

Sets the FTM extended dead-time value.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*value*	The FTM peripheral extend pre-scale divider

Implements : FTM_HAL_SetExtDeadtimeValue_Activity

3.30.4.91 FTM_HAL_SetExtPairDeadtimeValue()

```
static void FTM_HAL_SetExtPairDeadtimeValue (
             FTM_Type *const ftmBase,
             uint8_t channelPair,
             uint8_t value )  [inline], [static]
```

Sets the FTM extended dead-time value for the channel pair.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channelPair*	The FTM peripheral channel pair (n)
in	*value*	The FTM peripheral extend pre-scale divider using the concatenation with the dead-time value

Implements : FTM_HAL_SetExtPairDeadtimeValue_Activity

3.30.4.92 FTM_HAL_SetFaultControlMode()

```
static void FTM_HAL_SetFaultControlMode (
            FTM_Type *const ftmBase,
            ftm_fault_mode_t mode )  [inline], [static]
```

Defines the FTM fault control mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*mode*	Fault control mode value • FTM_FAULT_CONTROL_DISABLED: Fault control disabled • FTM_FAULT_CONTROL_MAN_EVEN: Fault control enabled for even channel (0, 2, 4, 6) and manual fault clearing. • FTM_FAULT_CONTROL_MAN_ALL : Fault control enabled for all channels and manual fault clearing is enabled. • FTM_FAULT_CONTROL_AUTO_ALL: Fault control enabled for all channels and automatic fault clearing is enabled.

Implements : FTM_HAL_SetFaultControlMode_Activity

3.30.4.93 FTM_HAL_SetFaultInputCmd()

```
static void FTM_HAL_SetFaultInputCmd (
            FTM_Type *const ftmBase,
            uint8_t inputNum,
            bool enable )  [inline], [static]
```

Enables or disables the fault input.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*inputNum*	fault input to be configured, valid value 0, 1, 2, 3
in	*enable*	State of fault input • true : To enable fault input • false: To disable fault input

Implements : FTM_HAL_SetFaultInputCmd_Activity

3.30.4.94 FTM_HAL_SetFaultInputFilterCmd()

```
static void FTM_HAL_SetFaultInputFilterCmd (
            FTM_Type *const ftmBase,
            uint8_t inputNum,
            bool enable )  [inline], [static]
```

Enables or disables the fault input filter.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*inputNum*	Fault input to be configured, valid value 0, 1, 2, 3
in	*enable*	State of fault input filter • true : To enable fault input filter • false: To disable fault input filter

Implements : FTM_HAL_SetFaultInputFilterCmd_Activity

3.30.4.95 FTM_HAL_SetFaultInputFilterVal()

```
static void FTM_HAL_SetFaultInputFilterVal (
            FTM_Type *const ftmBase,
            uint32_t value )  [inline], [static]
```

Sets the fault input filter value.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*value*	Fault input filter value

Implements : FTM_HAL_SetFaultInputFilterVal_Activity

3.30.4.96 FTM_HAL_SetFaultInt()

```
static void FTM_HAL_SetFaultInt (
            FTM_Type *const ftmBase,
            bool state )  [inline], [static]
```

Enables/disables the FTM peripheral timer fault interrupt.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*state*	Timer fault interrupt state • true : Fault control interrupt is enable • false: Fault control interrupt is disabled

Implements : FTM_HAL_SetFaultInt_Activity

3.30.4.97 FTM_HAL_SetGlobalLoadCmd()

```
static void FTM_HAL_SetGlobalLoadCmd (
            FTM_Type *const ftmBase )  [inline], [static]
```

Set the global load mechanism.

Parameters

in	*ftmBase*	The FTM base address pointer • true : LDOK bit is set • false: No action

Implements : FTM_HAL_SetGlobalLoadCmd_Activity

3.30.4.98 FTM_HAL_SetGlobalTimeBaseCmd()

```
static void FTM_HAL_SetGlobalTimeBaseCmd (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Enables or disables the FTM timer global time base.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of global time base • true : To enable • false: To disable

Implements : FTM_HAL_SetGlobalTimeBaseCmd_Activity

3.30.4.99 FTM_HAL_SetGlobalTimeBaseOutputCmd()

```
static void FTM_HAL_SetGlobalTimeBaseOutputCmd (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Enables or disables the FTM global time base signal generation to other FTM's.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of global time base signal • true : To enable • false: To disable

Implements : FTM_HAL_SetGlobalTimeBaseOutputCmd_Activity

3.30.4.100 FTM_HAL_SetHalfCycleCmd()

```
static void FTM_HAL_SetHalfCycleCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Enable the half cycle reload.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of the half cycle match as a reload opportunity • true : Half cycle reload is enabled • false: Half cycle reload is disabled

Implements : FTM_HAL_SetHalfCycleCmd_Activity

3.30.4.101 FTM_HAL_SetHalfCycleValue()

```
static void FTM_HAL_SetHalfCycleValue (
             FTM_Type *const ftmBase,
             uint16_t value )  [inline], [static]
```

Sets the value for the half cycle reload register.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*value*	The 16 bit counter value

Implements : FTM_HAL_SetHalfCycleValue_Activity

3.30.4.102 FTM_HAL_SetHardwareSyncTriggerSrc()

```
static void FTM_HAL_SetHardwareSyncTriggerSrc (
             FTM_Type *const ftmBase,
             uint8_t trigger_num,
             bool enable )  [inline], [static]
```

Sets the FTM hardware synchronization trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*trigger_num*	Number of trigger • 0U: trigger 0 • 1U: trigger 1 • 2U: trigger 2

Parameters

in	*enable*	State of trigger • true : Enable hardware trigger from field trigger_num for PWM synchronization • false: Disable hardware trigger from field trigger_num for PWM synchronization

Implements : FTM_HAL_SetHardwareSyncTriggerSrc_Activity

3.30.4.103 FTM_HAL_SetHwTriggerSyncModeCmd()

```
static void FTM_HAL_SetHwTriggerSyncModeCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Sets hardware trigger mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of hardware trigger mode • true : FTM does not clear the TRIGx bit when the hardware trigger j is detected • false: FTM clears the TRIGx bit when the hardware trigger j is detected

Implements : FTM_HAL_SetHwTriggerSyncModeCmd_Activity

3.30.4.104 FTM_HAL_SetInitChnOutputCmd()

```
static void FTM_HAL_SetInitChnOutputCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Initializes the channels output.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	Initialize the channels output • true : The channels output is initialized according to the state of OUTINIT reg • false: No effect

Implements : FTM_HAL_SetInitChnOutputCmd_Activity

3.30.4.105 FTM_HAL_SetInitTriggerCmd()

```
static void FTM_HAL_SetInitTriggerCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Enables or disables the generation of the trigger when the FTM counter is equal to the CNTIN register.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of initialization trigger • true : To enable • false: To disable

Implements : FTM_HAL_SetInitTriggerCmd_Activity

3.30.4.106 FTM_HAL_SetInitTrigOnReloadCmd()

```
static void FTM_HAL_SetInitTrigOnReloadCmd (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Enables or disables the FTM initialization trigger on Reload Point.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	bit controls whether an initialization trigger is generated • true : Trigger is generated when a reload point is reached • false: Trigger is generated on counter wrap events

Implements : FTM_HAL_SetInitTrigOnReloadCmd_Activity

3.30.4.107 FTM_HAL_SetInvctrlHardwareSyncModeCmd()

```
static void FTM_HAL_SetInvctrlHardwareSyncModeCmd (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Sets sync mode for FTM INVCTRL register when using a hardware trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of inverting control synchronization • true : The hardware trigger activates INVCTRL register sync • false: The hardware trigger does not activate INVCTRL register sync

Implements : FTM_HAL_SetInvctrlHardwareSyncModeCmd_Activity

3.30.4.108 FTM_HAL_SetInvctrlPwmSyncModeCmd()

```
static void FTM_HAL_SetInvctrlPwmSyncModeCmd (
            FTM_Type *const ftmBase,
            ftm_reg_update_t mode )  [inline], [static]
```

Sets the INVCTRL register PWM synchronization mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*mode*	State of register synchronization • true : INVCTRL register is updated by PWM sync • false: INVCTRL register is updated at all rising edges of system clock

Implements : FTM_HAL_SetInvctrlPwmSyncModeCmd_Activity

3.30.4.109 FTM_HAL_SetInvctrlReg()

```
static void FTM_HAL_SetInvctrlReg (
            FTM_Type *const ftmBase,
            uint32_t regVal )  [inline], [static]
```

Writes the provided value to the Inverting control register.

This function is enable/disable inverting control on multiple channel pairs.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*regVal*	Value to be written to the register

Implements : FTM_HAL_SetInvctrlReg_Activity

3.30.4.110 FTM_HAL_SetInvctrlSoftwareSyncModeCmd()

```
static void FTM_HAL_SetInvctrlSoftwareSyncModeCmd (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Sets sync mode for FTM INVCTRL register when using a software trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of State of inverting control synchronization • true : The software trigger activates INVCTRL register sync • false: The software trigger does not activate INVCTRL register sync

Implements : FTM_HAL_SetInvctrlSoftwareSyncModeCmd_Activity

3.30.4.111 FTM_HAL_SetLoadCmd()

```
static void FTM_HAL_SetLoadCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Enable the global load.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of the global load mechanism • true : Global Load OK enabled • false: Global Load OK disabled

Implements : FTM_HAL_SetLoadCmd_Activity

3.30.4.112 FTM_HAL_SetLoadFreq()

```
static void FTM_HAL_SetLoadFreq (
             FTM_Type *const ftmBase,
             uint8_t val )  [inline], [static]
```

Sets the FTM timer TOF Frequency.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*val*	Value of the TOF bit set frequency

Implements : FTM_HAL_SetLoadFreq_Activity

3.30.4.113 FTM_HAL_SetMaxLoadingCmd()

```
static void FTM_HAL_SetMaxLoadingCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Enables or disables the FTM peripheral timer maximum loading points.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	Maximum loading point selection • true : To enable maximum loading point • false: To disable

Implements : FTM_HAL_SetMaxLoadingCmd_Activity

3.30.4.114 FTM_HAL_SetMinLoadingCmd()

```
static void FTM_HAL_SetMinLoadingCmd (
            FTM_Type *const ftmBase,
            bool enable ) [inline], [static]
```

Enables or disables the FTM peripheral timer minimum loading points.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	Minimum loading point selection • true : To enable minimum loading point • false: To disable

Implements : FTM_HAL_SetMinLoadingCmd_Activity

3.30.4.115 FTM_HAL_SetMod()

```
static void FTM_HAL_SetMod (
            FTM_Type *const ftmBase,
            uint16_t value ) [inline], [static]
```

Sets the FTM peripheral timer modulo value.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*value*	The value to be set to the timer modulo

Implements : FTM_HAL_SetMod_Activity

3.30.4.116 FTM_HAL_SetModCntinCvHardwareSyncModeCmd()

```
static void FTM_HAL_SetModCntinCvHardwareSyncModeCmd (
            FTM_Type *const ftmBase,
            bool enable ) [inline], [static]
```

Sets sync mode for FTM MOD, CNTIN and CV registers when using a hardware trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of registers synchronization • true : The hardware trigger activates MOD, HCR, CNTIN, and CV registers sync • false: The hardware trigger does not activate MOD, HCR, CNTIN, and CV registers sync

Implements : FTM_HAL_SetModCntinCvHardwareSyncModeCmd_Activity

3.30.4.117 FTM_HAL_SetModCntinCvSoftwareSyncModeCmd()

```
static void FTM_HAL_SetModCntinCvSoftwareSyncModeCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Sets sync mode for FTM MOD, CNTIN and CV registers when using a software trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of registers synchronization • true : The software trigger activates FTM MOD, CNTIN and CV registers sync • false: The software trigger does not activate FTM MOD, CNTIN and CV registers sync

Implements : FTM_HAL_SetModCntinCvSoftwareSyncModeCmd_Activity

3.30.4.118 FTM_HAL_SetOutmaskHardwareSyncModeCmd()

```
static void FTM_HAL_SetOutmaskHardwareSyncModeCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Sets sync mode for FTM OUTMASK register when using a hardware trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of output mask synchronization • true : The hardware trigger activates OUTMASK register sync • false: The hardware trigger does not activate OUTMASK register sync

Implements : FTM_HAL_SetOutmaskHardwareSyncModeCmd_Activity

3.30.4.119 FTM_HAL_SetOutmaskPwmSyncModeCmd()

```
static void FTM_HAL_SetOutmaskPwmSyncModeCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Determines when the OUTMASK register is updated with the value of its buffer.

Parameters

in	*ftmBase*	The FTM base address pointer

Parameters

in	*enable*	Output Mask synchronization selection • true : OUTMASK register is updated only by PWM synchronization • false: OUTMASK register is updated in all rising edges of the system clock

Implements : FTM_HAL_SetOutmaskPwmSyncModeCmd_Activity

3.30.4.120 FTM_HAL_SetOutmaskReg()

```
static void FTM_HAL_SetOutmaskReg (
            FTM_Type *const ftmBase,
            uint32_t regVal )  [inline], [static]
```

Writes the provided value to the OUTMASK register.

This function will mask/unmask multiple channels.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*regVal*	Value to be written to the register

Implements : FTM_HAL_SetOutmaskReg_Activity

3.30.4.121 FTM_HAL_SetOutmaskSoftwareSyncModeCmd()

```
static void FTM_HAL_SetOutmaskSoftwareSyncModeCmd (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Sets sync mode for FTM OUTMASK register when using a software trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of output mask synchronization • true : The software trigger activates OUTMASK register sync • false: The software trigger does not activate OUTMASK register sync

Implements : FTM_HAL_SetOutmaskSoftwareSyncModeCmd_Activity

3.30.4.122 FTM_HAL_SetPairDeadtimeCount()

```
static void FTM_HAL_SetPairDeadtimeCount (
            FTM_Type *const ftmBase,
            uint8_t channelPair,
            uint8_t count )  [inline], [static]
```

Sets the FTM dead-time value for the channel pair.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channelPair*	The FTM peripheral channel pair (n)
in	*count*	The FTM peripheral selects the dead-time value • 0U : no counts inserted • 1U : 1 count is inserted • 2U : 2 count is inserted • ... up to a possible 63 counts

Implements : FTM_HAL_SetPairDeadtimeCount_Activity

3.30.4.123 FTM_HAL_SetPairDeadtimePrescale()

```
static void FTM_HAL_SetPairDeadtimePrescale (
            FTM_Type *const ftmBase,
            uint8_t channelPair,
            ftm_deadtime_ps_t divider )  [inline], [static]
```

Sets the FTM dead time divider for the channel pair.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channelPair*	The FTM peripheral channel pair (n)
in	*divider*	The FTM peripheral pre-scaler divider • FTM_DEADTIME_DIVID_BY_1 : Divide by 1 • FTM_DEADTIME_DIVID_BY_4 : Divide by 4 • FTM_DEADTIME_DIVID_BY_16: Divide by 16

Implements : FTM_HAL_SetPairDeadtimePrescale_Activity

3.30.4.124 FTM_HAL_SetPwmFaultBehavior()

```
static void FTM_HAL_SetPwmFaultBehavior (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Configures the behavior of the PWM outputs when a fault is detected.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of fault output • true : Output pins are set tri-state, • false: Pins are set to a safe state determined by POL bits

Implements : FTM_HAL_SetPwmFaultBehavior_Activity

3.30.4.125 FTM_HAL_SetPwmLoadChnSelCmd()

```
static void FTM_HAL_SetPwmLoadChnSelCmd (
             FTM_Type *const ftmBase,
             uint8_t channel,
             bool enable )  [inline], [static]
```

Includes or excludes the channel in the matching process.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	Channel to be configured
in	*enable*	State of channel • true : means include the channel in the matching process • false: means do not include channel in the matching process

Implements : FTM_HAL_SetPwmLoadChnSelCmd_Activity

3.30.4.126 FTM_HAL_SetPwmLoadCmd()

```
static void FTM_HAL_SetPwmLoadCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Enables or disables the loading of MOD, CNTIN and CV with values of their write buffer.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of loading updated values • true : To enable • false: To disable

Implements : FTM_HAL_SetPwmLoadCmd_Activity

3.30.4.127 FTM_HAL_SetPwmSyncMode()

```
static void FTM_HAL_SetPwmSyncMode (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Sets the FTM peripheral timer sync mode.

Parameters

in	*ftmBase*	The FTM base address pointer

Parameters

in	*enable*	PWM synchronization mode • false: No restriction both software and hardware triggers can be used • true : Software trigger can only be used for MOD and CnV synchronization, hardware trigger only for OUTMASK and FTM counter synchronization.

Implements : FTM_HAL_SetPwmSyncMode_Activity

3.30.4.128 FTM_HAL_SetPwmSyncModeCmd()

```
static void FTM_HAL_SetPwmSyncModeCmd (
             FTM_Type *const ftmBase,
             bool mode )  [inline], [static]
```

Sets the PWM synchronization mode to enhanced or legacy.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*mode*	State of synchronization mode • true : Enhanced PWM synchronization is selected • false: Legacy PWM synchronization is selected

Implements : FTM_HAL_SetPwmSyncModeCmd_Activity

3.30.4.129 FTM_HAL_SetQuadDecoderCmd()

```
static void FTM_HAL_SetQuadDecoderCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Enables the channel quadrature decoder.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of channel quadrature decoder • true : To enable • false: To disable

Implements : FTM_HAL_SetQuadDecoderCmd_Activity

3.30.4.130 FTM_HAL_SetQuadMode()

```
static void FTM_HAL_SetQuadMode (
            FTM_Type *const ftmBase,
            ftm_quad_decode_mode_t quadMode )  [inline], [static]
```

Sets the encoding mode used in quadrature decoding mode.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*quadMode*	Quadrature decoder mode selection • FTM_QUAD_PHASE_ENCODE: Phase A and Phase B encoding mode • FTM_QUAD_COUNT_AND_DIR: Count and direction encoding mode

Implements : FTM_HAL_SetQuadMode_Activity

3.30.4.131 FTM_HAL_SetQuadPhaseAFilterCmd()

```
static void FTM_HAL_SetQuadPhaseAFilterCmd (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Enables or disables the phase A input filter.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of phase A input filter • true : Enables the phase input filter • false: Disables the filter

Implements : FTM_HAL_SetQuadPhaseAFilterCmd_Activity

3.30.4.132 FTM_HAL_SetQuadPhaseAPolarity()

```
static void FTM_HAL_SetQuadPhaseAPolarity (
            FTM_Type *const ftmBase,
            ftm_quad_phase_polarity_t mode )  [inline], [static]
```

Selects polarity for the quadrature decode phase A input.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*mode*	Phase A input polarity selection • FTM_QUAD_PHASE_NORMAL: Normal polarity • FTM_QUAD_PHASE_INVERT: Inverted polarity

Implements : FTM_HAL_SetQuadPhaseAPolarity_Activity

3.30.4.133 FTM_HAL_SetQuadPhaseBFilterCmd()

```
static void FTM_HAL_SetQuadPhaseBFilterCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Enables or disables the phase B input filter.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of phase B input filter • true : Enables the phase input filter • false: Disables the filter

Implements : FTM_HAL_SetQuadPhaseBFilterCmd_Activity

3.30.4.134 FTM_HAL_SetQuadPhaseBPolarity()

```
static void FTM_HAL_SetQuadPhaseBPolarity (
             FTM_Type *const ftmBase,
             ftm_quad_phase_polarity_t mode )  [inline], [static]
```

Selects polarity for the quadrature decode phase B input.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*mode*	Phase B input polarity selection • FTM_QUAD_PHASE_NORMAL: Normal polarity • FTM_QUAD_PHASE_INVERT: Inverted polarity

Implements : FTM_HAL_SetQuadPhaseBPolarity_Activity

3.30.4.135 FTM_HAL_SetReIntEnabledCmd()

```
static void FTM_HAL_SetReIntEnabledCmd (
             FTM_Type *const ftmBase,
             bool enable )  [inline], [static]
```

Set the FTM reload interrupt enable.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	- true : Reload interrupt is enabled • false: Reload interrupt is disabled

Implements : FTM_HAL_SetReIntEnabledCmd_Activity

3.30.4.136 FTM_HAL_SetSoftwareTriggerCmd()

```
static void FTM_HAL_SetSoftwareTriggerCmd (
             FTM_Type *const ftmBase,
             bool enable ) [inline], [static]
```

Enables or disables the FTM peripheral timer software trigger.

Parameters

in	*ftmBase*	The FTM base address pointer.
in	*enable*	Software trigger selection • true : Software trigger is selected • false: Software trigger is not selected

Implements : FTM_HAL_SetSoftwareTriggerCmd_Activity

3.30.4.137 FTM_HAL_SetSwoctrlHardwareSyncModeCmd()

```
static void FTM_HAL_SetSwoctrlHardwareSyncModeCmd (
             FTM_Type *const ftmBase,
             bool enable ) [inline], [static]
```

Sets the sync mode for the FTM SWOCTRL register when using a hardware trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of software output control synchronization • true : The hardware trigger activates SWOCTRL register sync • false: The hardware trigger does not activate SWOCTRL register sync

Implements : FTM_HAL_SetSwoctrlHardwareSyncModeCmd_Activity

3.30.4.138 FTM_HAL_SetSwoctrlPwmSyncModeCmd()

```
static void FTM_HAL_SetSwoctrlPwmSyncModeCmd (
             FTM_Type *const ftmBase,
             ftm_reg_update_t mode ) [inline], [static]
```

Sets the SWOCTRL register PWM synchronization mode.

Parameters

in	*ftmBase*	The FTM base address pointer

Parameters

in	*mode*	State of register synchronization • true : SWOCTRL register is updated by PWM sync • false: SWOCTRL register is updated at all rising edges of system clock

Implements : FTM_HAL_SetSwoctrlPwmSyncModeCmd_Activity

3.30.4.139 FTM_HAL_SetSwoctrlSoftwareSyncModeCmd()

```
static void FTM_HAL_SetSwoctrlSoftwareSyncModeCmd (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Sets sync mode for FTM SWOCTRL register when using a software trigger.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	State of software output control synchronization • true : The software trigger activates SWOCTRL register sync • false: The software trigger does not activate SWOCTRL register sync

Implements : FTM_HAL_SetSwoctrlSoftwareSyncModeCmd_Activity

3.30.4.140 FTM_HAL_SetTimerOverflowInt()

```
static void FTM_HAL_SetTimerOverflowInt (
            FTM_Type *const ftmBase,
            bool state )  [inline], [static]
```

Enables the FTM peripheral timer overflow interrupt.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*state*	- true : Overflow interrupt enabled • false: Overflow interrupt disabled

Implements : FTM_HAL_SetTimerOverflowInt_Activity

3.30.4.141 FTM_HAL_SetTrigModeControlCmd()

```
static void FTM_HAL_SetTrigModeControlCmd (
            FTM_Type *const ftmBase,
            uint8_t channel,
            bool enable )  [inline], [static]
```

Enables or disables the trigger generation on FTM channel outputs.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*channel*	The FTM peripheral channel number
in	*enable*	Trigger mode control • false : Enable PWM output without generating a pulse • true : Disable a trigger generation on channel output

Implements : FTM_HAL_SetTrigModeControlCmd_Activity

3.30.4.142 FTM_HAL_SetWriteProtectionCmd()

```
static void FTM_HAL_SetWriteProtectionCmd (
            FTM_Type *const ftmBase,
            bool enable )  [inline], [static]
```

Enables or disables the FTM write protection.

Parameters

in	*ftmBase*	The FTM base address pointer
in	*enable*	The FTM write protection selection • true : Write-protection is enabled • false: Write-protection is disabled

Implements : FTM_HAL_SetWriteProtectionCmd_Activity

3.31 Flash Memory (Flash)

3.31.1 Detailed Description

This section describes the programming interface of the Flash Peripheral Driver.

Data Structures

- struct flash_user_config_t

  Flash User Configuration Structure. More...
- struct flash_ssd_config_t

  Flash SSD Configuration Structure. More...
- struct flash_eeprom_status_t

  EEPROM status structure. More...

Macros

- #define FTFx_BASE FTFC_BASE
- #define FTFx_FSTAT FTFC->FSTAT
- #define FTFx_FCNFG FTFC->FCNFG
- #define FTFx_FSEC FTFC->FSEC
- #define FTFx_FOPT FTFC->FOPT
- #define FTFx_FCCOB3 FTFC->FCCOB[0]
- #define FTFx_FCCOB2 FTFC->FCCOB[1]
- #define FTFx_FCCOB1 FTFC->FCCOB[2]
- #define FTFx_FCCOB0 FTFC->FCCOB[3]
- #define FTFx_FCCOB7 FTFC->FCCOB[4]
- #define FTFx_FCCOB6 FTFC->FCCOB[5]
- #define FTFx_FCCOB5 FTFC->FCCOB[6]
- #define FTFx_FCCOB4 FTFC->FCCOB[7]
- #define FTFx_FCCOBB FTFC->FCCOB[8]
- #define FTFx_FCCOBA FTFC->FCCOB[9]
- #define FTFx_FCCOB9 FTFC->FCCOB[10]
- #define FTFx_FCCOB8 FTFC->FCCOB[11]
- #define FTFx_FPROT3 FTFC->FPROT[0]
- #define FTFx_FPROT2 FTFC->FPROT[1]
- #define FTFx_FPROT1 FTFC->FPROT[2]
- #define FTFx_FPROT0 FTFC->FPROT[3]
- #define FTFx_FEPROT FTFC->FEPROT
- #define FTFx_FDPROT FTFC->FDPROT
- #define FTFx_FCSESTAT FTFC->FCSESTAT
- #define FTFx_FSTAT_MGSTAT0_MASK FTFC_FSTAT_MGSTAT0_MASK
- #define FTFx_FSTAT_MGSTAT0_SHIFT FTFC_FSTAT_MGSTAT0_SHIFT
- #define FTFx_FSTAT_MGSTAT0_WIDTH FTFC_FSTAT_MGSTAT0_WIDTH
- #define FTFx_FSTAT_MGSTAT0(x) FTFC_FSTAT_MGSTAT0(x)
- #define FTFx_FSTAT_FPVIOL_MASK FTFC_FSTAT_FPVIOL_MASK
- #define FTFx_FSTAT_FPVIOL_SHIFT FTFC_FSTAT_FPVIOL_SHIFT
- #define FTFx_FSTAT_FPVIOL_WIDTH FTFC_FSTAT_FPVIOL_WIDTH
- #define FTFx_FSTAT_FPVIOL(x) FTFC_FSTAT_FPVIOL(x)
- #define FTFx_FSTAT_ACCERR_MASK FTFC_FSTAT_ACCERR_MASK
- #define FTFx_FSTAT_ACCERR_SHIFT FTFC_FSTAT_ACCERR_SHIFT
- #define FTFx_FSTAT_ACCERR_WIDTH FTFC_FSTAT_ACCERR_WIDTH

- #define FTFx_FSTAT_ACCERR(x) FTFC_FSTAT_ACCERR(x)
- #define FTFx_FSTAT_RDCOLERR_MASK FTFC_FSTAT_RDCOLERR_MASK
- #define FTFx_FSTAT_RDCOLERR_SHIFT FTFC_FSTAT_RDCOLERR_SHIFT
- #define FTFx_FSTAT_RDCOLERR_WIDTH FTFC_FSTAT_RDCOLERR_WIDTH
- #define FTFx_FSTAT_RDCOLERR(x) FTFC_FSTAT_RDCOLERR(x)
- #define FTFx_FSTAT_CCIF_MASK FTFC_FSTAT_CCIF_MASK
- #define FTFx_FSTAT_CCIF_SHIFT FTFC_FSTAT_CCIF_SHIFT
- #define FTFx_FSTAT_CCIF_WIDTH FTFC_FSTAT_CCIF_WIDTH
- #define FTFx_FSTAT_CCIF(x) FTFC_FSTAT_CCIF(x)
- #define FTFx_FCNFG_EEERDY_MASK FTFC_FCNFG_EEERDY_MASK
- #define FTFx_FCNFG_EEERDY_SHIFT FTFC_FCNFG_EEERDY_SHIFT
- #define FTFx_FCNFG_EEERDY_WIDTH FTFC_FCNFG_EEERDY_WIDTH
- #define FTFx_FCNFG_EEERDY(x) FTFC_FCNFG_EEERDY(x)
- #define FTFx_FCNFG_RAMRDY_MASK FTFC_FCNFG_RAMRDY_MASK
- #define FTFx_FCNFG_RAMRDY_SHIFT FTFC_FCNFG_RAMRDY_SHIFT
- #define FTFx_FCNFG_RAMRDY_WIDTH FTFC_FCNFG_RAMRDY_WIDTH
- #define FTFx_FCNFG_RAMRDY(x) FTFC_FCNFG_RAMRDY(x)
- #define FTFx_FCNFG_ERSSUSP_MASK FTFC_FCNFG_ERSSUSP_MASK
- #define FTFx_FCNFG_ERSSUSP_SHIFT FTFC_FCNFG_ERSSUSP_SHIFT
- #define FTFx_FCNFG_ERSSUSP_WIDTH FTFC_FCNFG_ERSSUSP_WIDTH
- #define FTFx_FCNFG_ERSSUSP(x) FTFC_FCNFG_ERSSUSP(x)
- #define FTFx_FCNFG_ERSAREQ_MASK FTFC_FCNFG_ERSAREQ_MASK
- #define FTFx_FCNFG_ERSAREQ_SHIFT FTFC_FCNFG_ERSAREQ_SHIFT
- #define FTFx_FCNFG_ERSAREQ_WIDTH FTFC_FCNFG_ERSAREQ_WIDTH
- #define FTFx_FCNFG_ERSAREQ(x) FTFC_FCNFG_ERSAREQ(x)
- #define FTFx_FCNFG_RDCOLLIE_MASK FTFC_FCNFG_RDCOLLIE_MASK
- #define FTFx_FCNFG_RDCOLLIE_SHIFT FTFC_FCNFG_RDCOLLIE_SHIFT
- #define FTFx_FCNFG_RDCOLLIE_WIDTH FTFC_FCNFG_RDCOLLIE_WIDTH
- #define FTFx_FCNFG_RDCOLLIE(x) FTFC_FCNFG_RDCOLLIE(x)
- #define FTFx_FCNFG_CCIE_MASK FTFC_FCNFG_CCIE_MASK
- #define FTFx_FCNFG_CCIE_SHIFT FTFC_FCNFG_CCIE_SHIFT
- #define FTFx_FCNFG_CCIE_WIDTH FTFC_FCNFG_CCIE_WIDTH
- #define FTFx_FCNFG_CCIE(x) FTFC_FCNFG_CCIE(x)
- #define FTFx_FSEC_SEC_MASK FTFC_FSEC_SEC_MASK
- #define FTFx_FSEC_SEC_SHIFT FTFC_FSEC_SEC_SHIFT
- #define FTFx_FSEC_SEC_WIDTH FTFC_FSEC_SEC_WIDTH
- #define FTFx_FSEC_SEC(x) FTFC_FSEC_SEC(x)
- #define FTFx_FSEC_FSLACC_MASK FTFC_FSEC_FSLACC_MASK
- #define FTFx_FSEC_FSLACC_SHIFT FTFC_FSEC_FSLACC_SHIFT
- #define FTFx_FSEC_FSLACC_WIDTH FTFC_FSEC_FSLACC_WIDTH
- #define FTFx_FSEC_FSLACC(x) FTFC_FSEC_FSLACC(x)
- #define FTFx_FSEC_MEEN_MASK FTFC_FSEC_MEEN_MASK
- #define FTFx_FSEC_MEEN_SHIFT FTFC_FSEC_MEEN_SHIFT
- #define FTFx_FSEC_MEEN_WIDTH FTFC_FSEC_MEEN_WIDTH
- #define FTFx_FSEC_MEEN(x) FTFC_FSEC_MEEN(x)
- #define FTFx_FSEC_KEYEN_MASK FTFC_FSEC_KEYEN_MASK
- #define FTFx_FSEC_KEYEN_SHIFT FTFC_FSEC_KEYEN_SHIFT
- #define FTFx_FSEC_KEYEN_WIDTH FTFC_FSEC_KEYEN_WIDTH
- #define FTFx_FSEC_KEYEN(x) FTFC_FSEC_KEYEN(x)
- #define FTFx_FOPT_OPT_MASK FTFC_FOPT_OPT_MASK
- #define FTFx_FOPT_OPT_SHIFT FTFC_FOPT_OPT_SHIFT
- #define FTFx_FOPT_OPT_WIDTH FTFC_FOPT_OPT_WIDTH
- #define FTFx_FOPT_OPT(x) FTFC_FOPT_OPT(x)
- #define FTFx_FCCOB_CCOBn_MASK FTFC_FCCOB_CCOBn_MASK
- #define FTFx_FCCOB_CCOBn_SHIFT FTFC_FCCOB_CCOBn_SHIFT

- #define FTFx_FCCOB_CCOBn_WIDTH FTFC_FCCOB_CCOBn_WIDTH
- #define FTFx_FCCOB_CCOBn(x) FTFC_FCCOB_CCOBn(x)
- #define FTFx_FPROT_PROT_MASK FTFC_FPROT_PROT_MASK
- #define FTFx_FPROT_PROT_SHIFT FTFC_FPROT_PROT_SHIFT
- #define FTFx_FPROT_PROT_WIDTH FTFC_FPROT_PROT_WIDTH
- #define FTFx_FPROT_PROT(x) FTFC_FPROT_PROT(x)
- #define FTFx_FEPROT_EPROT_MASK FTFC_FEPROT_EPROT_MASK
- #define FTFx_FEPROT_EPROT_SHIFT FTFC_FEPROT_EPROT_SHIFT
- #define FTFx_FEPROT_EPROT_WIDTH FTFC_FEPROT_EPROT_WIDTH
- #define FTFx_FEPROT_EPROT(x) FTFC_FEPROT_EPROT(x)
- #define FTFx_FDPROT_DPROT_MASK FTFC_FDPROT_DPROT_MASK
- #define FTFx_FDPROT_DPROT_SHIFT FTFC_FDPROT_DPROT_SHIFT
- #define FTFx_FDPROT_DPROT_WIDTH FTFC_FDPROT_DPROT_WIDTH
- #define FTFx_FDPROT_DPROT(x) FTFC_FDPROT_DPROT(x)
- #define FTFX_FCSESTAT_IDB_MASK FTFC_FCSESTAT_IDB_MASK
- #define FTFX_FCSESTAT_IDB_SHIFT FTFC_FCSESTAT_IDB_SHIFT
- #define FTFX_FCSESTAT_IDB_WIDTH FTFC_FCSESTAT_IDB_WIDTH
- #define FTFX_FCSESTAT_IDB(x) FTFC_FCSESTAT_IDB(x)
- #define FTFX_FCSESTAT_EDB_MASK FTFC_FCSESTAT_EDB_MASK
- #define FTFX_FCSESTAT_EDB_SHIFT FTFC_FCSESTAT_EDB_SHIFT
- #define FTFX_FCSESTAT_EDB_WIDTH FTFC_FCSESTAT_EDB_WIDTH
- #define FTFX_FCSESTAT_EDB(x) FTFC_FCSESTAT_EDB(x)
- #define FTFX_FCSESTAT_RIN_MASK FTFC_FCSESTAT_RIN_MASK
- #define FTFX_FCSESTAT_RIN_SHIFT FTFC_FCSESTAT_RIN_SHIFT
- #define FTFX_FCSESTAT_RIN_WIDTH FTFC_FCSESTAT_RIN_WIDTH
- #define FTFX_FCSESTAT_RIN(x) FTFC_FCSESTAT_RIN(x)
- #define FTFX_FCSESTAT_BOK_MASK FTFC_FCSESTAT_BOK_MASK
- #define FTFX_FCSESTAT_BOK_SHIFT FTFC_FCSESTAT_BOK_SHIFT
- #define FTFX_FCSESTAT_BOK_WIDTH FTFC_FCSESTAT_BOK_WIDTH
- #define FTFX_FCSESTAT_BOK(x) FTFC_FCSESTAT_BOK(x)
- #define FTFX_FCSESTAT_BFN_MASK FTFC_FCSESTAT_BFN_MASK
- #define FTFX_FCSESTAT_BFN_SHIFT FTFC_FCSESTAT_BFN_SHIFT
- #define FTFX_FCSESTAT_BFN_WIDTH FTFC_FCSESTAT_BFN_WIDTH
- #define FTFX_FCSESTAT_BFN(x) FTFC_FCSESTAT_BFN(x)
- #define FTFX_FCSESTAT_BIN_MASK FTFC_FCSESTAT_BIN_MASK
- #define FTFX_FCSESTAT_BIN_SHIFT FTFC_FCSESTAT_BIN_SHIFT
- #define FTFX_FCSESTAT_BIN_WIDTH FTFC_FCSESTAT_BIN_WIDTH
- #define FTFX_FCSESTAT_BIN(x) FTFC_FCSESTAT_BIN(x)
- #define FTFX_FCSESTAT_SB_MASK FTFC_FCSESTAT_SB_MASK
- #define FTFX_FCSESTAT_SB_SHIFT FTFC_FCSESTAT_SB_SHIFT
- #define FTFX_FCSESTAT_SB_WIDTH FTFC_FCSESTAT_SB_WIDTH
- #define FTFX_FCSESTAT_SB(x) FTFC_FCSESTAT_SB(x)
- #define FTFX_FCSESTAT_BSY_MASK FTFC_FCSESTAT_BSY_MASK
- #define FTFX_FCSESTAT_BSY_SHIFT FTFC_FCSESTAT_BSY_SHIFT
- #define FTFX_FCSESTAT_BSY_WIDTH FTFC_FCSESTAT_BSY_WIDTH
- #define FTFX_FCSESTAT_BSY(x) FTFC_FCSESTAT_BSY(x)
- #define CLEAR_FTFx_FSTAT_ERROR_BITS FTFx_FSTAT = (uint8_t)(FTFx_FSTAT_FPVIOL_MASK | F↩
TFx_FSTAT_ACCERR_MASK | FTFx_FSTAT_RDCOLERR_MASK)
- #define FTFx_WORD_SIZE 0x0002U
- #define FTFx_LONGWORD_SIZE 0x0004U
- #define FTFx_PHRASE_SIZE 0x0008U
- #define FTFx_DPHRASE_SIZE 0x0010U
- #define FTFx_RSRC_CODE_REG FTFx_FCCOB4
- #define FTFx_VERIFY_BLOCK 0x00U
- #define FTFx_VERIFY_SECTION 0x01U

- #define FTFx_PROGRAM_CHECK 0x02U
- #define FTFx_READ_RESOURCE 0x03U
- #define FTFx_PROGRAM_LONGWORD 0x06U
- #define FTFx_PROGRAM_PHRASE 0x07U
- #define FTFx_ERASE_BLOCK 0x08U
- #define FTFx_ERASE_SECTOR 0x09U
- #define FTFx_PROGRAM_SECTION 0x0BU
- #define FTFx_VERIFY_ALL_BLOCK 0x40U
- #define FTFx_READ_ONCE 0x41U
- #define FTFx_PROGRAM_ONCE 0x43U
- #define FTFx_ERASE_ALL_BLOCK 0x44U
- #define FTFx_SECURITY_BY_PASS 0x45U
- #define FTFx_PFLASH_SWAP 0x46U
- #define FTFx_ERASE_ALL_BLOCK_UNSECURE 0x49U
- #define FTFx_PROGRAM_PARTITION 0x80U
- #define FTFx_SET_EERAM 0x81U
- #define RESUME_WAIT_CNT 0x20U

    Resume wait count used in FlashResume function.
- #define DFLASH_IFR_READRESOURCE_ADDRESS 0x8000FCU
- #define GET_BIT_0_7(value) ((uint8_t)(((uint32_t)(value)) & 0xFFU))
- #define GET_BIT_8_15(value) ((uint8_t)((((uint32_t)(value)) >> 8) & 0xFFU))
- #define GET_BIT_16_23(value) ((uint8_t)((((uint32_t)(value)) >> 16) & 0xFFU))
- #define GET_BIT_24_31(value) ((uint8_t)(((uint32_t)(value)) >> 24))
- #define FLASH_SECURITY_STATE_KEYEN 0x80U
- #define FLASH_SECURITY_STATE_UNSECURED 0x02U
- #define CSE_KEY_SIZE_CODE_MAX 0x03U
- #define FLASH_CALLBACK_CS 0x0AU

    Callback period count for FlashCheckSum.

Typedefs

- typedef void(∗ flash_callback_t) (void)

    Call back function pointer data type.

Enumerations

- enum flash_flexRam_function_control_code_t {
  EEE_ENABLE = 0x00U, EEE_QUICK_WRITE = 0x55U, EEE_STATUS_QUERY = 0x77U, EEE_COMPLE←
  TE_INTERRUPT_QUICK_WRITE = 0xAAU,
  EEE_DISABLE = 0xFFU }

    FlexRAM Function control Code.

Variables

- uint32_t PFlashBase
- uint32_t PFlashSize
- uint32_t DFlashBase
- uint32_t EERAMBase
- flash_callback_t CallBack
- uint32_t PFlashBase
- uint32_t PFlashSize
- uint32_t DFlashBase

- uint32_t DFlashSize
- uint32_t EERAMBase
- uint32_t EEESize
- flash_callback_t CallBack
- uint8_t brownOutCode
- uint16_t numOfRecordReqMaintain
- uint16_t sectorEraseCount

PFlash swap control codes

- #define FTFx_SWAP_SET_INDICATOR_ADDR 0x01U

  Initialize Swap System control code.
- #define FTFx_SWAP_SET_IN_PREPARE 0x02U

  Set Swap in Update State.
- #define FTFx_SWAP_SET_IN_COMPLETE 0x04U

  Set Swap in Complete State.
- #define FTFx_SWAP_REPORT_STATUS 0x08U

  Report Swap Status.

PFlash swap states

- #define FTFx_SWAP_UNINIT 0x00U

  Uninitialized swap mode.
- #define FTFx_SWAP_READY 0x01U

  Ready swap mode.
- #define FTFx_SWAP_UPDATE 0x02U

  Update swap mode.
- #define FTFx_SWAP_UPDATE_ERASED 0x03U

  Update-Erased swap mode.
- #define FTFx_SWAP_COMPLETE 0x04U

  Complete swap mode.

Flash security status

- #define FLASH_NOT_SECURE 0x01U

  Flash currently not in secure state.
- #define FLASH_SECURE_BACKDOOR_ENABLED 0x02U

  Flash is secured and backdoor key access enabled.
- #define FLASH_SECURE_BACKDOOR_DISABLED 0x04U

  Flash is secured and backdoor key access disabled.

Null Callback function definition

- #define NULL_CALLBACK ((flash_callback_t)0xFFFFFFFFU)

  Null callback.

Flash driver APIs

- status_t FLASH_DRV_Init (const flash_user_config_t ∗const pUserConf, flash_ssd_config_t ∗const pSSD←Config)

  Initializes Flash.
- void FLASH_DRV_GetPFlashProtection (uint32_t ∗protectStatus)

  P-Flash get protection.
- status_t FLASH_DRV_SetPFlashProtection (uint32_t protectStatus)

  P-Flash set protection.
- void FLASH_DRV_GetSecurityState (uint8_t ∗securityState)

  Flash get security state.
- status_t FLASH_DRV_SecurityBypass (const flash_ssd_config_t ∗pSSDConfig, const uint8_t ∗keyBuffer)

  Flash security bypass.
- status_t FLASH_DRV_EraseAllBlock (const flash_ssd_config_t ∗pSSDConfig)

  Flash erase all blocks.
- status_t FLASH_DRV_VerifyAllBlock (const flash_ssd_config_t ∗pSSDConfig, uint8_t marginLevel)

  Flash verify all blocks.
- status_t FLASH_DRV_EraseSector (const flash_ssd_config_t ∗pSSDConfig, uint32_t dest, uint32_t size)

  Flash erase sector.
- status_t FLASH_DRV_VerifySection (const flash_ssd_config_t ∗pSSDConfig, uint32_t dest, uint16_t number, uint8_t marginLevel)

  Flash verify section.
- void FLASH_DRV_EraseSuspend (void)

  Flash erase suspend.
- void FLASH_DRV_EraseResume (void)

  Flash erase resume.
- status_t FLASH_DRV_ReadOnce (const flash_ssd_config_t ∗pSSDConfig, uint8_t recordIndex, uint8_t ∗p←DataArray)

  Flash read once.
- status_t FLASH_DRV_ProgramOnce (const flash_ssd_config_t ∗pSSDConfig, uint8_t recordIndex, const uint8_t ∗pDataArray)

  Flash program once.
- status_t FLASH_DRV_Program (const flash_ssd_config_t ∗pSSDConfig, uint32_t dest, uint32_t size, const uint8_t ∗pData)

  Flash program.
- status_t FLASH_DRV_ProgramCheck (const flash_ssd_config_t ∗pSSDConfig, uint32_t dest, uint32_t size, const uint8_t ∗pExpectedData, uint32_t ∗pFailAddr, uint8_t marginLevel)

  Flash program check.
- status_t FLASH_DRV_CheckSum (const flash_ssd_config_t ∗pSSDConfig, uint32_t dest, uint32_t size, uint32_t ∗pSum)

  Calculates check sum.
- status_t FLASH_DRV_ProgramSection (const flash_ssd_config_t ∗pSSDConfig, uint32_t dest, uint16_t number)

  Flash program section.
- status_t FLASH_DRV_EraseBlock (const flash_ssd_config_t ∗pSSDConfig, uint32_t dest)

  Flash erase block.
- status_t FLASH_DRV_VerifyBlock (const flash_ssd_config_t ∗pSSDConfig, uint32_t dest, uint8_t margin←Level)

  Flash verify block.
- status_t FLASH_DRV_GetEERAMProtection (uint8_t ∗protectStatus)

  EERAM get protection.
- status_t FLASH_DRV_SetEERAMProtection (uint8_t protectStatus)

EERAM set protection.

- status_t FLASH_DRV_SetFlexRamFunction (const flash_ssd_config_t ∗pSSDConfig, flash_flexRam↩_function_control_code_t flexRamFuncCode, uint16_t byteOfQuickWrite, flash_eeprom_status_t ∗const pEEPROMStatus)

    Set FlexRAM function.
- status_t FLASH_DRV_EEEWrite (const flash_ssd_config_t ∗pSSDConfig, uint32_t dest, uint32_t size, const uint8_t ∗pData)

    EEPROM Emulator Write.
- status_t FLASH_DRV_DEFlashPartition (const flash_ssd_config_t ∗pSSDConfig, uint8_t uEEEDataSize↩Code, uint8_t uDEPartitionCode, uint8_t uCSEcKeySize, bool uSFE)

    Flash D/E-Flash Partition.
- status_t FLASH_DRV_GetDFlashProtection (const flash_ssd_config_t ∗pSSDConfig, uint8_t ∗protectStatus)

    D-Flash get protection.
- status_t FLASH_DRV_SetDFlashProtection (const flash_ssd_config_t ∗pSSDConfig, uint8_t protectStatus)

    D-Flash set protection.
- status_t FLASH_DRV_EraseAllBlockUnsecure (const flash_ssd_config_t ∗pSSDConfig)

    Flash erase all blocks unsecure.

3.31.2 Data Structure Documentation

3.31.2.1 struct flash_user_config_t

Flash User Configuration Structure.

Implements : flash_user_config_t_Class

Data Fields

- uint32_t PFlashBase
- uint32_t PFlashSize
- uint32_t DFlashBase
- uint32_t EERAMBase
- flash_callback_t CallBack

3.31.2.2 struct flash_ssd_config_t

Flash SSD Configuration Structure.

The structure includes the static parameters for C90TFS/FTFx which are device-dependent. The fields including PFlashBlockBase, PFlashBlockSize, DFlashBlockBase, EERAMBlockBase, and CallBack are passed via flash_↩user_config_t. The rest of parameters such as DFlashBlockSize, and EEEBlockSize will be initialized in FlashInit() automatically.

Implements : flash_ssd_config_t_Class

Data Fields

- uint32_t PFlashBase
- uint32_t PFlashSize
- uint32_t DFlashBase
- uint32_t DFlashSize
- uint32_t EERAMBase
- uint32_t EEESize
- flash_callback_t CallBack

3.31.2.3 struct flash_eeprom_status_t

EEPROM status structure.

Implements : flash_eeprom_status_t_Class

Data Fields

- uint8_t brownOutCode
- uint16_t numOfRecordReqMaintain
- uint16_t sectorEraseCount

3.31.3 Macro Definition Documentation

3.31.3.1 CLEAR_FTFx_FSTAT_ERROR_BITS

```
#define CLEAR_FTFx_FSTAT_ERROR_BITS FTFx_FSTAT = (uint8_t)(FTFx_FSTAT_FPVIOL_MASK | FTFx_FSTA←
T_ACCERR_MASK | FTFx_FSTAT_RDCOLERR_MASK)
```

3.31.3.2 CSE_KEY_SIZE_CODE_MAX

```
#define CSE_KEY_SIZE_CODE_MAX 0x03U
```

3.31.3.3 DFLASH_IFR_READRESOURCE_ADDRESS

```
#define DFLASH_IFR_READRESOURCE_ADDRESS 0x8000FCU
```

3.31.3.4 FLASH_CALLBACK_CS

```
#define FLASH_CALLBACK_CS 0x0AU
```

Callback period count for FlashCheckSum.

This value is only relevant for FlashCheckSum operation, where a high rate of calling back can impair performance. The rest of the flash operations invoke the callback as often as possible while waiting for the flash controller to finish the requested operation.

3.31.3.5 FLASH_NOT_SECURE

```
#define FLASH_NOT_SECURE 0x01U
```

Flash currently not in secure state.

3.31.3.6 FLASH_SECURE_BACKDOOR_DISABLED

```
#define FLASH_SECURE_BACKDOOR_DISABLED 0x04U
```

Flash is secured and backdoor key access disabled.

3.31.3.7 FLASH_SECURE_BACKDOOR_ENABLED

```
#define FLASH_SECURE_BACKDOOR_ENABLED 0x02U
```

Flash is secured and backdoor key access enabled.

3.31.3.8 FLASH_SECURITY_STATE_KEYEN

```
#define FLASH_SECURITY_STATE_KEYEN 0x80U
```

3.31.3.9 FLASH_SECURITY_STATE_UNSECURED

```
#define FLASH_SECURITY_STATE_UNSECURED 0x02U
```

3.31.3.10 FTFx_BASE

```
#define FTFx_BASE FTFC_BASE
```

3.31.3.11 FTFx_DPHRASE_SIZE

```
#define FTFx_DPHRASE_SIZE 0x0010U
```

3.31.3.12 FTFx_ERASE_ALL_BLOCK

```
#define FTFx_ERASE_ALL_BLOCK 0x44U
```

3.31.3.13 FTFx_ERASE_ALL_BLOCK_UNSECURE

```
#define FTFx_ERASE_ALL_BLOCK_UNSECURE 0x49U
```

3.31.3.14 FTFx_ERASE_BLOCK

```
#define FTFx_ERASE_BLOCK 0x08U
```

3.31.3.15 FTFx_ERASE_SECTOR

```
#define FTFx_ERASE_SECTOR 0x09U
```

3.31.3.16 FTFx_FCCOB0

```
#define FTFx_FCCOB0 FTFC->FCCOB[3]
```

3.31.3.17 FTFx_FCCOB1

```
#define FTFx_FCCOB1 FTFC->FCCOB[2]
```

3.31.3.18 FTFx_FCCOB2

```
#define FTFx_FCCOB2 FTFC->FCCOB[1]
```

3.31.3.19 FTFx_FCCOB3

```
#define FTFx_FCCOB3 FTFC->FCCOB[0]
```

3.31.3.20 FTFx_FCCOB4

```
#define FTFx_FCCOB4 FTFC->FCCOB[7]
```

3.31.3.21 FTFx_FCCOB5

```
#define FTFx_FCCOB5 FTFC->FCCOB[6]
```

3.31.3.22 FTFx_FCCOB6

```
#define FTFx_FCCOB6 FTFC->FCCOB[5]
```

3.31.3.23 FTFx_FCCOB7

```
#define FTFx_FCCOB7 FTFC->FCCOB[4]
```

3.31.3.24 FTFx_FCCOB8

```
#define FTFx_FCCOB8 FTFC->FCCOB[11]
```

3.31.3.25 FTFx_FCCOB9

```
#define FTFx_FCCOB9 FTFC->FCCOB[10]
```

3.31.3.26 FTFx_FCCOB_CCOBn

```
#define FTFx_FCCOB_CCOBn(
             x ) FTFC_FCCOB_CCOBn(x)
```

3.31.3.27 FTFx_FCCOB_CCOBn_MASK

```
#define FTFx_FCCOB_CCOBn_MASK FTFC_FCCOB_CCOBn_MASK
```

3.31.3.28 FTFx_FCCOB_CCOBn_SHIFT

```
#define FTFx_FCCOB_CCOBn_SHIFT FTFC_FCCOB_CCOBn_SHIFT
```

3.31.3.29 FTFx_FCCOB_CCOBn_WIDTH

```
#define FTFx_FCCOB_CCOBn_WIDTH FTFC_FCCOB_CCOBn_WIDTH
```

3.31.3.30 FTFx_FCCOBA

```
#define FTFx_FCCOBA FTFC->FCCOB[9]
```

3.31.3.31 FTFx_FCCOBB

```
#define FTFx_FCCOBB FTFC->FCCOB[8]
```

3.31.3.32 FTFx_FCNFG

```
#define FTFx_FCNFG FTFC->FCNFG
```

3.31.3.33 FTFx_FCNFG_CCIE

```
#define FTFx_FCNFG_CCIE(
              x ) FTFC_FCNFG_CCIE(x)
```

3.31.3.34 FTFx_FCNFG_CCIE_MASK

```
#define FTFx_FCNFG_CCIE_MASK FTFC_FCNFG_CCIE_MASK
```

3.31.3.35 FTFx_FCNFG_CCIE_SHIFT

```
#define FTFx_FCNFG_CCIE_SHIFT FTFC_FCNFG_CCIE_SHIFT
```

3.31.3.36 FTFx_FCNFG_CCIE_WIDTH

```
#define FTFx_FCNFG_CCIE_WIDTH FTFC_FCNFG_CCIE_WIDTH
```

3.31.3.37 FTFx_FCNFG_EEERDY

```
#define FTFx_FCNFG_EEERDY(
              x ) FTFC_FCNFG_EEERDY(x)
```

3.31.3.38 FTFx_FCNFG_EEERDY_MASK

```
#define FTFx_FCNFG_EEERDY_MASK FTFC_FCNFG_EEERDY_MASK
```

3.31.3.39 FTFx_FCNFG_EEERDY_SHIFT

```
#define FTFx_FCNFG_EEERDY_SHIFT FTFC_FCNFG_EEERDY_SHIFT
```

3.31.3.40 FTFx_FCNFG_EEERDY_WIDTH

```
#define FTFx_FCNFG_EEERDY_WIDTH FTFC_FCNFG_EEERDY_WIDTH
```

3.31.3.41 FTFx_FCNFG_ERSAREQ

```
#define FTFx_FCNFG_ERSAREQ(
              x ) FTFC_FCNFG_ERSAREQ(x)
```

3.31.3.42 FTFx_FCNFG_ERSAREQ_MASK

```
#define FTFx_FCNFG_ERSAREQ_MASK FTFC_FCNFG_ERSAREQ_MASK
```

3.31.3.43 FTFx_FCNFG_ERSAREQ_SHIFT

```
#define FTFx_FCNFG_ERSAREQ_SHIFT FTFC_FCNFG_ERSAREQ_SHIFT
```

3.31.3.44 FTFx_FCNFG_ERSAREQ_WIDTH

```
#define FTFx_FCNFG_ERSAREQ_WIDTH FTFC_FCNFG_ERSAREQ_WIDTH
```

3.31.3.45 FTFx_FCNFG_ERSSUSP

```
#define FTFx_FCNFG_ERSSUSP(
              x ) FTFC_FCNFG_ERSSUSP(x)
```

3.31.3.46 FTFx_FCNFG_ERSSUSP_MASK

```
#define FTFx_FCNFG_ERSSUSP_MASK FTFC_FCNFG_ERSSUSP_MASK
```

3.31.3.47 FTFx_FCNFG_ERSSUSP_SHIFT

```
#define FTFx_FCNFG_ERSSUSP_SHIFT FTFC_FCNFG_ERSSUSP_SHIFT
```

3.31.3.48 FTFx_FCNFG_ERSSUSP_WIDTH

```
#define FTFx_FCNFG_ERSSUSP_WIDTH FTFC_FCNFG_ERSSUSP_WIDTH
```

3.31.3.49 FTFx_FCNFG_RAMRDY

```
#define FTFx_FCNFG_RAMRDY(
              x ) FTFC_FCNFG_RAMRDY(x)
```

3.31.3.50 FTFx_FCNFG_RAMRDY_MASK

```
#define FTFx_FCNFG_RAMRDY_MASK FTFC_FCNFG_RAMRDY_MASK
```

3.31.3.51 FTFx_FCNFG_RAMRDY_SHIFT

```
#define FTFx_FCNFG_RAMRDY_SHIFT FTFC_FCNFG_RAMRDY_SHIFT
```

3.31.3.52 FTFx_FCNFG_RAMRDY_WIDTH

```
#define FTFx_FCNFG_RAMRDY_WIDTH FTFC_FCNFG_RAMRDY_WIDTH
```

3.31.3.53 FTFx_FCNFG_RDCOLLIE

```
#define FTFx_FCNFG_RDCOLLIE(
              x ) FTFC_FCNFG_RDCOLLIE(x)
```

3.31.3.54 FTFx_FCNFG_RDCOLLIE_MASK

```
#define FTFx_FCNFG_RDCOLLIE_MASK FTFC_FCNFG_RDCOLLIE_MASK
```

3.31.3.55 FTFx_FCNFG_RDCOLLIE_SHIFT

```
#define FTFx_FCNFG_RDCOLLIE_SHIFT FTFC_FCNFG_RDCOLLIE_SHIFT
```

3.31.3.56 FTFx_FCNFG_RDCOLLIE_WIDTH

```
#define FTFx_FCNFG_RDCOLLIE_WIDTH FTFC_FCNFG_RDCOLLIE_WIDTH
```

3.31.3.57 FTFx_FCSESTAT

```
#define FTFx_FCSESTAT FTFC->FCSESTAT
```

3.31.3.58 FTFX_FCSESTAT_BFN

```
#define FTFX_FCSESTAT_BFN(
             x ) FTFC_FCSESTAT_BFN(x)
```

3.31.3.59 FTFX_FCSESTAT_BFN_MASK

```
#define FTFX_FCSESTAT_BFN_MASK FTFC_FCSESTAT_BFN_MASK
```

3.31.3.60 FTFX_FCSESTAT_BFN_SHIFT

```
#define FTFX_FCSESTAT_BFN_SHIFT FTFC_FCSESTAT_BFN_SHIFT
```

3.31.3.61 FTFX_FCSESTAT_BFN_WIDTH

```
#define FTFX_FCSESTAT_BFN_WIDTH FTFC_FCSESTAT_BFN_WIDTH
```

3.31.3.62 FTFX_FCSESTAT_BIN

```
#define FTFX_FCSESTAT_BIN(
             x ) FTFC_FCSESTAT_BIN(x)
```

3.31.3.63 FTFX_FCSESTAT_BIN_MASK

```
#define FTFX_FCSESTAT_BIN_MASK FTFC_FCSESTAT_BIN_MASK
```

3.31.3.64 FTFX_FCSESTAT_BIN_SHIFT

```
#define FTFX_FCSESTAT_BIN_SHIFT FTFC_FCSESTAT_BIN_SHIFT
```

3.31.3.65 FTFX_FCSESTAT_BIN_WIDTH

```
#define FTFX_FCSESTAT_BIN_WIDTH FTFC_FCSESTAT_BIN_WIDTH
```

3.31.3.66 FTFX_FCSESTAT_BOK

```
#define FTFX_FCSESTAT_BOK(
             x ) FTFC_FCSESTAT_BOK(x)
```

3.31.3.67 FTFX_FCSESTAT_BOK_MASK

```
#define FTFX_FCSESTAT_BOK_MASK FTFC_FCSESTAT_BOK_MASK
```

3.31.3.68 FTFX_FCSESTAT_BOK_SHIFT

```
#define FTFX_FCSESTAT_BOK_SHIFT FTFC_FCSESTAT_BOK_SHIFT
```

3.31.3.69 FTFX_FCSESTAT_BOK_WIDTH

```
#define FTFX_FCSESTAT_BOK_WIDTH FTFC_FCSESTAT_BOK_WIDTH
```

3.31.3.70 FTFX_FCSESTAT_BSY

```
#define FTFX_FCSESTAT_BSY(
              x ) FTFC_FCSESTAT_BSY(x)
```

3.31.3.71 FTFX_FCSESTAT_BSY_MASK

```
#define FTFX_FCSESTAT_BSY_MASK FTFC_FCSESTAT_BSY_MASK
```

3.31.3.72 FTFX_FCSESTAT_BSY_SHIFT

```
#define FTFX_FCSESTAT_BSY_SHIFT FTFC_FCSESTAT_BSY_SHIFT
```

3.31.3.73 FTFX_FCSESTAT_BSY_WIDTH

```
#define FTFX_FCSESTAT_BSY_WIDTH FTFC_FCSESTAT_BSY_WIDTH
```

3.31.3.74 FTFX_FCSESTAT_EDB

```
#define FTFX_FCSESTAT_EDB(
              x ) FTFC_FCSESTAT_EDB(x)
```

3.31.3.75 FTFX_FCSESTAT_EDB_MASK

```
#define FTFX_FCSESTAT_EDB_MASK FTFC_FCSESTAT_EDB_MASK
```

3.31.3.76 FTFX_FCSESTAT_EDB_SHIFT

```
#define FTFX_FCSESTAT_EDB_SHIFT FTFC_FCSESTAT_EDB_SHIFT
```

3.31.3.77 FTFX_FCSESTAT_EDB_WIDTH

```
#define FTFX_FCSESTAT_EDB_WIDTH FTFC_FCSESTAT_EDB_WIDTH
```

3.31.3.78 FTFX_FCSESTAT_IDB

```
#define FTFX_FCSESTAT_IDB(
              x ) FTFC_FCSESTAT_IDB(x)
```

3.31.3.79 FTFX_FCSESTAT_IDB_MASK

```
#define FTFX_FCSESTAT_IDB_MASK FTFC_FCSESTAT_IDB_MASK
```

3.31.3.80 FTFX_FCSESTAT_IDB_SHIFT

```
#define FTFX_FCSESTAT_IDB_SHIFT FTFC_FCSESTAT_IDB_SHIFT
```

3.31.3.81 FTFX_FCSESTAT_IDB_WIDTH

```
#define FTFX_FCSESTAT_IDB_WIDTH FTFC_FCSESTAT_IDB_WIDTH
```

3.31.3.82 FTFX_FCSESTAT_RIN

```
#define FTFX_FCSESTAT_RIN(
              x ) FTFC_FCSESTAT_RIN(x)
```

3.31.3.83 FTFX_FCSESTAT_RIN_MASK

```
#define FTFX_FCSESTAT_RIN_MASK FTFC_FCSESTAT_RIN_MASK
```

3.31.3.84 FTFX_FCSESTAT_RIN_SHIFT

```
#define FTFX_FCSESTAT_RIN_SHIFT FTFC_FCSESTAT_RIN_SHIFT
```

3.31.3.85 FTFX_FCSESTAT_RIN_WIDTH

```
#define FTFX_FCSESTAT_RIN_WIDTH FTFC_FCSESTAT_RIN_WIDTH
```

3.31.3.86 FTFX_FCSESTAT_SB

```
#define FTFX_FCSESTAT_SB(
              x ) FTFC_FCSESTAT_SB(x)
```

3.31.3.87 FTFX_FCSESTAT_SB_MASK

```
#define FTFX_FCSESTAT_SB_MASK FTFC_FCSESTAT_SB_MASK
```

3.31.3.88 FTFX_FCSESTAT_SB_SHIFT

```
#define FTFX_FCSESTAT_SB_SHIFT FTFC_FCSESTAT_SB_SHIFT
```

3.31.3.89 FTFX_FCSESTAT_SB_WIDTH

```
#define FTFX_FCSESTAT_SB_WIDTH FTFC_FCSESTAT_SB_WIDTH
```

3.31.3.90 FTFx_FDPROT

```
#define FTFx_FDPROT FTFC->FDPROT
```

3.31.3.91 FTFx_FDPROT_DPROT

```
#define FTFx_FDPROT_DPROT(
              x ) FTFC_FDPROT_DPROT(x)
```

3.31.3.92 FTFx_FDPROT_DPROT_MASK

```
#define FTFx_FDPROT_DPROT_MASK FTFC_FDPROT_DPROT_MASK
```

3.31.3.93 FTFx_FDPROT_DPROT_SHIFT

```
#define FTFx_FDPROT_DPROT_SHIFT FTFC_FDPROT_DPROT_SHIFT
```

3.31.3.94 FTFx_FDPROT_DPROT_WIDTH

```
#define FTFx_FDPROT_DPROT_WIDTH FTFC_FDPROT_DPROT_WIDTH
```

3.31.3.95 FTFx_FEPROT

```
#define FTFx_FEPROT FTFC->FEPROT
```

3.31.3.96 FTFx_FEPROT_EPROT

```
#define FTFx_FEPROT_EPROT(
              x ) FTFC_FEPROT_EPROT(x)
```

3.31.3.97 FTFx_FEPROT_EPROT_MASK

```
#define FTFx_FEPROT_EPROT_MASK FTFC_FEPROT_EPROT_MASK
```

3.31.3.98 FTFx_FEPROT_EPROT_SHIFT

```
#define FTFx_FEPROT_EPROT_SHIFT FTFC_FEPROT_EPROT_SHIFT
```

3.31.3.99 FTFx_FEPROT_EPROT_WIDTH

```
#define FTFx_FEPROT_EPROT_WIDTH FTFC_FEPROT_EPROT_WIDTH
```

3.31.3.100 FTFx_FOPT

```
#define FTFx_FOPT FTFC->FOPT
```

3.31.3.101 FTFx_FOPT_OPT

```
#define FTFx_FOPT_OPT(
              x ) FTFC_FOPT_OPT(x)
```

3.31.3.102 FTFx_FOPT_OPT_MASK

```
#define FTFx_FOPT_OPT_MASK FTFC_FOPT_OPT_MASK
```

3.31.3.103 FTFx_FOPT_OPT_SHIFT

```
#define FTFx_FOPT_OPT_SHIFT FTFC_FOPT_OPT_SHIFT
```

3.31.3.104 FTFx_FOPT_OPT_WIDTH

```
#define FTFx_FOPT_OPT_WIDTH FTFC_FOPT_OPT_WIDTH
```

3.31.3.105 FTFx_FPROT0

```
#define FTFx_FPROT0 FTFC->FPROT[3]
```

3.31.3.106 FTFx_FPROT1

```
#define FTFx_FPROT1 FTFC->FPROT[2]
```

3.31.3.107 FTFx_FPROT2

```
#define FTFx_FPROT2 FTFC->FPROT[1]
```

3.31.3.108 FTFx_FPROT3

```
#define FTFx_FPROT3 FTFC->FPROT[0]
```

3.31.3.109 FTFx_FPROT_PROT

```
#define FTFx_FPROT_PROT(
              x ) FTFC_FPROT_PROT(x)
```

3.31.3.110 FTFx_FPROT_PROT_MASK

```
#define FTFx_FPROT_PROT_MASK FTFC_FPROT_PROT_MASK
```

3.31.3.111 FTFx_FPROT_PROT_SHIFT

```
#define FTFx_FPROT_PROT_SHIFT FTFC_FPROT_PROT_SHIFT
```

3.31.3.112 FTFx_FPROT_PROT_WIDTH

```
#define FTFx_FPROT_PROT_WIDTH FTFC_FPROT_PROT_WIDTH
```

3.31.3.113 FTFx_FSEC

```
#define FTFx_FSEC FTFC->FSEC
```

3.31.3.114 FTFx_FSEC_FSLACC

```
#define FTFx_FSEC_FSLACC(
              x ) FTFC_FSEC_FSLACC(x)
```

3.31.3.115 FTFx_FSEC_FSLACC_MASK

```
#define FTFx_FSEC_FSLACC_MASK FTFC_FSEC_FSLACC_MASK
```

3.31.3.116 FTFx_FSEC_FSLACC_SHIFT

```
#define FTFx_FSEC_FSLACC_SHIFT FTFC_FSEC_FSLACC_SHIFT
```

3.31.3.117 FTFx_FSEC_FSLACC_WIDTH

```
#define FTFx_FSEC_FSLACC_WIDTH FTFC_FSEC_FSLACC_WIDTH
```

3.31.3.118 FTFx_FSEC_KEYEN

```
#define FTFx_FSEC_KEYEN(
              x ) FTFC_FSEC_KEYEN(x)
```

3.31.3.119 FTFx_FSEC_KEYEN_MASK

```
#define FTFx_FSEC_KEYEN_MASK FTFC_FSEC_KEYEN_MASK
```

3.31.3.120 FTFx_FSEC_KEYEN_SHIFT

```
#define FTFx_FSEC_KEYEN_SHIFT FTFC_FSEC_KEYEN_SHIFT
```

3.31.3.121 FTFx_FSEC_KEYEN_WIDTH

```
#define FTFx_FSEC_KEYEN_WIDTH FTFC_FSEC_KEYEN_WIDTH
```

3.31.3.122 FTFx_FSEC_MEEN

```
#define FTFx_FSEC_MEEN(
              x ) FTFC_FSEC_MEEN(x)
```

3.31.3.123 FTFx_FSEC_MEEN_MASK

```
#define FTFx_FSEC_MEEN_MASK FTFC_FSEC_MEEN_MASK
```

3.31.3.124 FTFx_FSEC_MEEN_SHIFT

```
#define FTFx_FSEC_MEEN_SHIFT FTFC_FSEC_MEEN_SHIFT
```

3.31.3.125 FTFx_FSEC_MEEN_WIDTH

```
#define FTFx_FSEC_MEEN_WIDTH FTFC_FSEC_MEEN_WIDTH
```

3.31.3.126 FTFx_FSEC_SEC

```
#define FTFx_FSEC_SEC(
              x ) FTFC_FSEC_SEC(x)
```

3.31.3.127 FTFx_FSEC_SEC_MASK

```
#define FTFx_FSEC_SEC_MASK FTFC_FSEC_SEC_MASK
```

3.31.3.128 FTFx_FSEC_SEC_SHIFT

```
#define FTFx_FSEC_SEC_SHIFT FTFC_FSEC_SEC_SHIFT
```

3.31.3.129 FTFx_FSEC_SEC_WIDTH

```
#define FTFx_FSEC_SEC_WIDTH FTFC_FSEC_SEC_WIDTH
```

3.31.3.130 FTFx_FSTAT

```
#define FTFx_FSTAT FTFC->FSTAT
```

3.31.3.131 FTFx_FSTAT_ACCERR

```
#define FTFx_FSTAT_ACCERR(
              x ) FTFC_FSTAT_ACCERR(x)
```

3.31.3.132 FTFx_FSTAT_ACCERR_MASK

```
#define FTFx_FSTAT_ACCERR_MASK FTFC_FSTAT_ACCERR_MASK
```

3.31.3.133 FTFx_FSTAT_ACCERR_SHIFT

```
#define FTFx_FSTAT_ACCERR_SHIFT FTFC_FSTAT_ACCERR_SHIFT
```

3.31.3.134 FTFx_FSTAT_ACCERR_WIDTH

```
#define FTFx_FSTAT_ACCERR_WIDTH FTFC_FSTAT_ACCERR_WIDTH
```

3.31.3.135 FTFx_FSTAT_CCIF

```
#define FTFx_FSTAT_CCIF(
              x ) FTFC_FSTAT_CCIF(x)
```

3.31.3.136 FTFx_FSTAT_CCIF_MASK

```
#define FTFx_FSTAT_CCIF_MASK FTFC_FSTAT_CCIF_MASK
```

3.31.3.137 FTFx_FSTAT_CCIF_SHIFT

```
#define FTFx_FSTAT_CCIF_SHIFT FTFC_FSTAT_CCIF_SHIFT
```

3.31.3.138 FTFx_FSTAT_CCIF_WIDTH

```
#define FTFx_FSTAT_CCIF_WIDTH FTFC_FSTAT_CCIF_WIDTH
```

3.31.3.139 FTFx_FSTAT_FPVIOL

```
#define FTFx_FSTAT_FPVIOL(
              x ) FTFC_FSTAT_FPVIOL(x)
```

3.31.3.140 FTFx_FSTAT_FPVIOL_MASK

```
#define FTFx_FSTAT_FPVIOL_MASK FTFC_FSTAT_FPVIOL_MASK
```

3.31.3.141 FTFx_FSTAT_FPVIOL_SHIFT

```
#define FTFx_FSTAT_FPVIOL_SHIFT FTFC_FSTAT_FPVIOL_SHIFT
```

3.31.3.142 FTFx_FSTAT_FPVIOL_WIDTH

```
#define FTFx_FSTAT_FPVIOL_WIDTH FTFC_FSTAT_FPVIOL_WIDTH
```

3.31.3.143 FTFx_FSTAT_MGSTAT0

```
#define FTFx_FSTAT_MGSTAT0(
              x ) FTFC_FSTAT_MGSTAT0(x)
```

3.31.3.144 FTFx_FSTAT_MGSTAT0_MASK

```
#define FTFx_FSTAT_MGSTAT0_MASK FTFC_FSTAT_MGSTAT0_MASK
```

3.31.3.145 FTFx_FSTAT_MGSTAT0_SHIFT

```
#define FTFx_FSTAT_MGSTAT0_SHIFT FTFC_FSTAT_MGSTAT0_SHIFT
```

3.31.3.146 FTFx_FSTAT_MGSTAT0_WIDTH

```
#define FTFx_FSTAT_MGSTAT0_WIDTH FTFC_FSTAT_MGSTAT0_WIDTH
```

3.31.3.147 FTFx_FSTAT_RDCOLERR

```
#define FTFx_FSTAT_RDCOLERR(
              x ) FTFC_FSTAT_RDCOLERR(x)
```

3.31.3.148 FTFx_FSTAT_RDCOLERR_MASK

```
#define FTFx_FSTAT_RDCOLERR_MASK FTFC_FSTAT_RDCOLERR_MASK
```

3.31.3.149 FTFx_FSTAT_RDCOLERR_SHIFT

```
#define FTFx_FSTAT_RDCOLERR_SHIFT FTFC_FSTAT_RDCOLERR_SHIFT
```

3.31.3.150 FTFx_FSTAT_RDCOLERR_WIDTH

```
#define FTFx_FSTAT_RDCOLERR_WIDTH FTFC_FSTAT_RDCOLERR_WIDTH
```

3.31.3.151 FTFx_LONGWORD_SIZE

```
#define FTFx_LONGWORD_SIZE 0x0004U
```

3.31.3.152 FTFx_PFLASH_SWAP

```
#define FTFx_PFLASH_SWAP 0x46U
```

3.31.3.153 FTFx_PHRASE_SIZE

```
#define FTFx_PHRASE_SIZE 0x0008U
```

3.31.3.154 FTFx_PROGRAM_CHECK

```
#define FTFx_PROGRAM_CHECK 0x02U
```

3.31.3.155 FTFx_PROGRAM_LONGWORD

```
#define FTFx_PROGRAM_LONGWORD 0x06U
```

3.31.3.156 FTFx_PROGRAM_ONCE

```
#define FTFx_PROGRAM_ONCE 0x43U
```

3.31.3.157 FTFx_PROGRAM_PARTITION

```
#define FTFx_PROGRAM_PARTITION 0x80U
```

3.31.3.158 FTFx_PROGRAM_PHRASE

```
#define FTFx_PROGRAM_PHRASE 0x07U
```

3.31.3.159 FTFx_PROGRAM_SECTION

```
#define FTFx_PROGRAM_SECTION 0x0BU
```

3.31.3.160 FTFx_READ_ONCE

```
#define FTFx_READ_ONCE 0x41U
```

3.31.3.161 FTFx_READ_RESOURCE

```
#define FTFx_READ_RESOURCE 0x03U
```

3.31.3.162 FTFx_RSRC_CODE_REG

```
#define FTFx_RSRC_CODE_REG FTFx_FCCOB4
```

3.31.3.163 FTFx_SECURITY_BY_PASS

```
#define FTFx_SECURITY_BY_PASS 0x45U
```

3.31.3.164 FTFx_SET_EERAM

```
#define FTFx_SET_EERAM 0x81U
```

3.31.3.165 FTFx_SWAP_COMPLETE

```
#define FTFx_SWAP_COMPLETE 0x04U
```

Complete swap mode.

3.31.3.166 FTFx_SWAP_READY

```
#define FTFx_SWAP_READY 0x01U
```

Ready swap mode.

3.31.3.167 FTFx_SWAP_REPORT_STATUS

```
#define FTFx_SWAP_REPORT_STATUS 0x08U
```

Report Swap Status.

3.31.3.168 FTFx_SWAP_SET_IN_COMPLETE

```
#define FTFx_SWAP_SET_IN_COMPLETE 0x04U
```

Set Swap in Complete State.

3.31.3.169 FTFx_SWAP_SET_IN_PREPARE

```
#define FTFx_SWAP_SET_IN_PREPARE 0x02U
```

Set Swap in Update State.

3.31.3.170 FTFx_SWAP_SET_INDICATOR_ADDR

```
#define FTFx_SWAP_SET_INDICATOR_ADDR 0x01U
```

Initialize Swap System control code.

3.31.3.171 FTFx_SWAP_UNINIT

```
#define FTFx_SWAP_UNINIT 0x00U
```

Uninitialized swap mode.

3.31.3.172 FTFx_SWAP_UPDATE

```
#define FTFx_SWAP_UPDATE 0x02U
```

Update swap mode.

3.31.3.173 FTFx_SWAP_UPDATE_ERASED

```
#define FTFx_SWAP_UPDATE_ERASED 0x03U
```

Update-Erased swap mode.

3.31.3.174 FTFx_VERIFY_ALL_BLOCK

```
#define FTFx_VERIFY_ALL_BLOCK 0x40U
```

3.31.3.175 FTFx_VERIFY_BLOCK

```
#define FTFx_VERIFY_BLOCK 0x00U
```

3.31.3.176 FTFx_VERIFY_SECTION

```
#define FTFx_VERIFY_SECTION 0x01U
```

3.31.3.177 FTFx_WORD_SIZE

```
#define FTFx_WORD_SIZE 0x0002U
```

3.31.3.178 GET_BIT_0_7

```
#define GET_BIT_0_7(
              value ) ((uint8_t)(((uint32_t)(value)) & 0xFFU))
```

3.31.3.179 GET_BIT_16_23

```
#define GET_BIT_16_23(
              value ) ((uint8_t)((((uint32_t)(value)) >> 16) & 0xFFU))
```

3.31.3.180 GET_BIT_24_31

```
#define GET_BIT_24_31(
              value ) ((uint8_t)(((uint32_t)(value)) >> 24))
```

3.31.3.181 GET_BIT_8_15

```
#define GET_BIT_8_15(
              value ) ((uint8_t)((((uint32_t)(value)) >> 8) & 0xFFU))
```

3.31.3.182 NULL_CALLBACK

```
#define NULL_CALLBACK ((flash_callback_t)0xFFFFFFFFU)
```

Null callback.

3.31.3.183 RESUME_WAIT_CNT

```
#define RESUME_WAIT_CNT 0x20U
```

Resume wait count used in FlashResume function.

3.31.4 Typedef Documentation

3.31.4.1 flash_callback_t

```
typedef void(* flash_callback_t) (void)
```

Call back function pointer data type.

If using callback in the application, any code reachable from this function must not be placed in a Flash block targeted for a program/erase operation. Functions can be placed in RAM section by using the START/END_FUN↩ CTION_DEFINITION/DECLARATION_RAMSECTION macros.

3.31.5 Enumeration Type Documentation

3.31.5.1 flash_flexRam_function_control_code_t

```
enum flash_flexRam_function_control_code_t
```

FlexRAM Function control Code.

Implements : flash_flexRAM_function_control_code_t_Class

Enumerator

EEE_ENABLE	Make FlexRAM available for emulated EEPROM
EEE_QUICK_WRITE	Make FlexRAM available for EEPROM quick writes
EEE_STATUS_QUERY	EEPROM quick write status query
EEE_COMPLETE_INTERRUPT_QUICK_WRITE	Complete interrupted EEPROM quick write process
EEE_DISABLE	Make FlexRAM available as RAM

3.31.6 Function Documentation

3.31.6.1 FLASH_DRV_CheckSum()

```
status_t FLASH_DRV_CheckSum (
            const flash_ssd_config_t * pSSDConfig,
            uint32_t dest,
            uint32_t size,
            uint32_t * pSum )
```

Calculates check sum.

This API performs 32 bit sum of each byte data over a specified Flash memory range without carry which provides rapid method for checking data integrity. The callback time period of this API is determined via FLASH_CALLBA←CK_CS macro in flash_driver.h which is used as a counter value for the CallBack() function calling in this API. This value can be changed as per the user requirement. User can change this value to obtain the maximum permissible callback time period. This API always returns STATUS_SUCCESS if size provided by user is zero regardless of the input validation.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*dest*	Start address of the Flash range to be summed.
in	*size*	Size in byte of the Flash range to be summed.
in	*pSum*	To return the sum value.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.2 FLASH_DRV_DEFlashPartition()

```
status_t FLASH_DRV_DEFlashPartition (
            const flash_ssd_config_t * pSSDConfig,
            uint8_t uEEEDataSizeCode,
            uint8_t uDEPartitionCode,
            uint8_t uCSEcKeySize,
            bool uSFE )
```

Flash D/E-Flash Partition.

This API prepares the FlexNVM block for use as D-Flash, EEPROM backup, or a combination of both and initializes the FlexRAM. In addition, this function is used to configure number of user keys and user key 'Verify Only' attribute if CSEc is enabled.

The single partition choice should be used through entire life time of a given application to guarantee the Flash endurance and data retention of Flash module.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer
in	*EEEDataSizeCode*	EEPROM Data Size Code
in	*DEPartitionCode*	FlexNVM Partition Code

Parameters

in	*uCSEcKeySize*	CSEc Key Size Code, it should be 0 if the function is not used for configuring CSEc part. It can be: • 0x00U: Number of User Keys is Zero Number of Bytes (subtracts from the total 4K EEERAM space) is 0 • 0x01U: Number of User Keys is 1 to 6 keys Number of Bytes (subtracts from the total 4K EEERAM space) is 128 bytes • 0x02U: Number of User Keys is 1 to 12 keys Number of Bytes (subtracts from the total 4K EEERAM space) is 256 bytes • 0x03U: Number of User Keys is 1 to 24 keys Number of Bytes (subtracts from the total 4K EEERAM space) is 512 bytes
in	*uSFE*	Security Flag Extension, it should be false if the function is not used for configuring CSEc part.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.3 FLASH_DRV_EEEWrite()

```
status_t FLASH_DRV_EEEWrite (
            const flash_ssd_config_t * pSSDConfig,
            uint32_t dest,
            uint32_t size,
            const uint8_t * pData )
```

EEPROM Emulator Write.

This API is used to write data to FlexRAM section which is partitioned as EEPROM use for EEPROM operation. After data has been written to EEPROM use section of FlexRAM, the EEPROM file system will create new data record in EEPROM back-up area of FlexNVM in round-robin fashion. There is no alignment constraint for destination and size parameters provided by user. However, according to user's input provided, this API will set priority to write to FlexRAM with following rules: 32-bit writing is invoked if destination is 32 bit aligned and size is not less than 32 bits. 16-bit writing is invoked if destination is 16 bit aligned and size is not less than 16 bits. 8-bit writing is invoked if destination is 8 bit aligned and size is not less than 8 bits.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*dest*	Start address for the intended write operation.
in	*size*	Size in byte to be written.
in	*pData*	Pointer to source address from which data has to be taken for writing operation.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.4 FLASH_DRV_EraseAllBlock()

```
status_t FLASH_DRV_EraseAllBlock (
            const flash_ssd_config_t * pSSDConfig )
```

Flash erase all blocks.

This API erases all Flash memory, initializes the FlexRAM, verifies all memory contents, and then releases the MCU security.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.5 FLASH_DRV_EraseAllBlockUnsecure()

```
status_t FLASH_DRV_EraseAllBlockUnsecure (
            const flash_ssd_config_t * pSSDConfig )
```

Flash erase all blocks unsecure.

This API erases all Flash memory, initializes the FlexRAM, verifies all memory contents, and then releases the MCU security.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.6 FLASH_DRV_EraseBlock()

```
status_t FLASH_DRV_EraseBlock (
            const flash_ssd_config_t * pSSDConfig,
            uint32_t dest )
```

Flash erase block.

This API erases all addresses in an individual P-Flash or D-Flash block. For the derivatives including multiply logical P-Flash or D-Flash blocks, this API erases a single block in a single call.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*dest*	Start address for the intended erase operation.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.7 FLASH_DRV_EraseResume()

```
void FLASH_DRV_EraseResume (
             void  )
```

Flash erase resume.

This API is used to resume a previous suspended operation of Flash erase sector command This function must be located in RAM memory or different Flash blocks which are targeted for writing to avoid RWW error.

3.31.6.8 FLASH_DRV_EraseSector()

```
status_t FLASH_DRV_EraseSector (
             const flash_ssd_config_t * pSSDConfig,
             uint32_t dest,
             uint32_t size )
```

Flash erase sector.

This API erases one or more sectors in P-Flash or D-Flash memory. This API always returns FTFx_OK if size provided by the user is zero regardless of the input validation.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*dest*	Address in the first sector to be erased.
in	*size*	Size to be erased in bytes. It is used to determine number of sectors to be erased.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.
- STATUS_UNSUPPORTED: Operation was unsupported.

3.31.6.9 FLASH_DRV_EraseSuspend()

```
void FLASH_DRV_EraseSuspend (
             void  )
```

Flash erase suspend.

This API is used to suspend a current operation of Flash erase sector command. This function must be located in RAM memory or different Flash blocks which are targeted for writing to avoid the RWW error.

3.31.6.10 FLASH_DRV_GetDFlashProtection()

```
status_t FLASH_DRV_GetDFlashProtection (
            const flash_ssd_config_t * pSSDConfig,
            uint8_t * protectStatus )
```

D-Flash get protection.

This API retrieves current P-Flash protection status. Considering the time consumption for getting protection is very low and even can be ignored, it is not necessary to utilize the Callback function to support the time-critical events.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer
out	*protectStatus*	To return the current value of the D-Flash Protection Register. Each bit is corresponding to protection status of 1/8 of the total D-Flash. The least significant bit is corresponding to the lowest address area of D-Flash. The most significant bit is corresponding to the highest address area of D-Flash and so on. There are two possible cases as below: • 0 : this area is protected. • 1 : this area is unprotected.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_UNSUPPORTED: Operation was unsupported.

3.31.6.11 FLASH_DRV_GetEERAMProtection()

```
status_t FLASH_DRV_GetEERAMProtection (
            uint8_t * protectStatus )
```

EERAM get protection.

This API retrieves which EEPROM sections of FlexRAM are protected against program and erase operations. Considering the time consumption for getting protection is very low and even can be ignored, it is not necessary to utilize the Callback function to support the time-critical events

Parameters

out	*protectStatus*	To return the current value of the EEPROM Protection Register. Each bit is corresponding to protection status of 1/8 of the total EEPROM use. The least significant bit is corresponding to the lowest address area of EEPROM. The most significant bit is corresponding to the highest address area of EEPROM and so on. There are two possible cases as below: • 0: this area is protected. • 1: this area is unprotected.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_UNSUPPORTED: Operation was unsupported.

3.31.6.12 FLASH_DRV_GetPFlashProtection()

```
void FLASH_DRV_GetPFlashProtection (
            uint32_t * protectStatus )
```

P-Flash get protection.

This API retrieves the current P-Flash protection status. Considering the time consumption for getting protection is very low and even can be ignored. It is not necessary to utilize the Callback function to support the time-critical events.

Parameters

out	*protectStatus*	To return the current value of the P-Flash Protection. Each bit is corresponding to protection of 1/32 of the total P-Flash. The least significant bit is corresponding to the lowest address area of P-Flash. The most significant bit is corresponding to the highest address area of P-Flash and so on. There are two possible cases as below: • 0: this area is protected. • 1: this area is unprotected.

3.31.6.13 FLASH_DRV_GetSecurityState()

```
void FLASH_DRV_GetSecurityState (
            uint8_t * securityState )
```

Flash get security state.

This API retrieves the current Flash security status, including the security enabling state and the back door key enabling state.

Parameters

out	*securityState*	To return the current security status code. • FLASH_NOT_SECURE (0x01U): Flash currently not in secure state • FLASH_SECURE_BACKDOOR_ENABLED (0x02U): Flash is secured and back door key access enabled • FLASH_SECURE_BACKDOOR_DISABLED (0x04U): Flash is secured and back door key access disabled.

3.31.6.14 FLASH_DRV_Init()

```
status_t FLASH_DRV_Init (
            const flash_user_config_t *const pUserConf,
            flash_ssd_config_t *const pSSDConfig )
```

Initializes Flash.

This API initializes Flash module by clearing status error bit and reporting the memory configuration via SSD configuration structure.

Parameters

in	*pUserConf*	The user configuration structure pointer.
in	*pSSDConfig*	The SSD configuration structure pointer.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.

3.31.6.15 FLASH_DRV_Program()

```
status_t FLASH_DRV_Program (
            const flash_ssd_config_t * pSSDConfig,
            uint32_t dest,
            uint32_t size,
            const uint8_t * pData )
```

Flash program.

This API is used to program 4 consecutive bytes (for program long word command) and 8 consecutive bytes (for program phrase command) on P-Flash or D-Flash block. This API always returns FTFx_OK if size provided by user is zero regardless of the input validation

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*dest*	Start address for the intended program operation.
in	*size*	Size in byte to be programmed
in	*pData*	Pointer of source address from which data has to be taken for program operation.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.
- STATUS_UNSUPPORTED: Operation was unsupported.

3.31.6.16 FLASH_DRV_ProgramCheck()

```
status_t FLASH_DRV_ProgramCheck (
            const flash_ssd_config_t * pSSDConfig,
            uint32_t dest,
            uint32_t size,
            const uint8_t * pExpectedData,
            uint32_t * pFailAddr,
            uint8_t marginLevel )
```

Flash program check.

This API tests a previously programmed P-Flash or D-Flash long word to see if it reads correctly at the specified margin level. This API always returns FTFx_OK if size provided by user is zero regardless of the input validation

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*dest*	Start address for the intended program check operation.
in	*size*	Size in byte to check accuracy of program operation
in	*pExpectedData*	The pointer to the expected data.
in	*pFailAddr*	Returned the first aligned failing address.
in	*marginLevel*	Read margin choice as follows: • marginLevel = 0x1U: read at User margin 1/0 level. • marginLevel = 0x2U: read at Factory margin 1/0 level.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.17 FLASH_DRV_ProgramOnce()

```
status_t FLASH_DRV_ProgramOnce (
            const flash_ssd_config_t * pSSDConfig,
            uint8_t recordIndex,
            const uint8_t * pDataArray )
```

Flash program once.

This API is used to program to a reserved 64 byte field located in the P-Flash IFR via given number of record. See the corresponding reference manual to get correct value of this number.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*recordIndex*	The record index will be read. It can be from 0x0U to 0x7U or from 0x0U to 0xF according to specific derivative.
in	*pDataArray*	Pointer to the array from which data will be taken for program once command.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.18 FLASH_DRV_ProgramSection()

```
status_t FLASH_DRV_ProgramSection (
             const flash_ssd_config_t * pSSDConfig,
             uint32_t dest,
             uint16_t number )
```

Flash program section.

This API will program the data found in the Section Program Buffer to previously erased locations in the Flash memory. Data is preloaded into the Section Program Buffer by writing to the acceleration Ram and FlexRam while it is set to function as a RAM. The Section Program Buffer is limited to the value of FlexRam divides by a ratio. Refer to the associate reference manual to get correct value of this ratio. For derivatives including swap feature, the swap indicator address is encountered during FlashProgramSection, it is bypassed without setting FPVIOL but the content are not be programmed. In addition, the content of source data used to program to swap indicator will be re-initialized to 0xFF after completion of this command.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*dest*	Start address for the intended program operation.
in	*number*	Number of alignment unit to be programmed. Refer to associate reference manual to get correct value of this alignment constrain.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.
- STATUS_UNSUPPORTED: Operation was unsupported.

3.31.6.19 FLASH_DRV_ReadOnce()

```
status_t FLASH_DRV_ReadOnce (
             const flash_ssd_config_t * pSSDConfig,
             uint8_t recordIndex,
             uint8_t * pDataArray )
```

Flash read once.

This API is used to read out a reserved 64 byte field located in the P-Flash IFR via given number of record. See the corresponding reference manual to get the correct value of this number.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*recordIndex*	The record index will be read. It can be from 0x0U to 0x7U or from 0x0U to 0xF according to specific derivative.
in	*pDataArray*	Pointer to the array to return the data read by the read once command.

Returns

operation status
- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.20 FLASH_DRV_SecurityBypass()

```
status_t FLASH_DRV_SecurityBypass (
            const flash_ssd_config_t * pSSDConfig,
            const uint8_t * keyBuffer )
```

Flash security bypass.

This API un-secures the device by comparing the user's provided back door key with the ones in the Flash Configuration Field. If they are matched, the security is released. Otherwise, an error code is returned.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*keyBuffer*	Point to the user buffer containing the back door key.

Returns

operation status
- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.21 FLASH_DRV_SetDFlashProtection()

```
status_t FLASH_DRV_SetDFlashProtection (
            const flash_ssd_config_t * pSSDConfig,
            uint8_t protectStatus )
```

D-Flash set protection.

This API sets the D-Flash protection to the intended protection status. Setting D-Flash protection status is subject to a protection transition restriction. If there is a setting violation, it returns failed information and the current protection status will not be changed.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer
in	*protectStatus*	The expected protect status user wants to set to D-Flash Protection Register. Each bit is corresponding to protection status of 1/8 of the total D-Flash. The least significant bit is corresponding to the lowest address area of D-Flash. The most significant bit is corresponding to the highest address area of D-Flash and so on. There are two possible cases as below: • 0 : this area is protected. • 1 : this area is unprotected.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.
- STATUS_UNSUPPORTED: Operation was unsupported.

3.31.6.22 FLASH_DRV_SetEERAMProtection()

```
status_t FLASH_DRV_SetEERAMProtection (
            uint8_t protectStatus )
```

EERAM set protection.

This API sets protection to the intended protection status for EEPROM us area of FlexRam. This is subject to a protection transition restriction. If there is a setting violation, it returns failed information and the current protection status will not be changed.

Parameters

in	*protectStatus*	The intended protection status value should be written to the EEPROM Protection Register. Each bit is corresponding to protection status of 1/8 of the total EEPROM use. The least significant bit is corresponding to the lowest address area of EEPROM. The most significant bit is corresponding to the highest address area of EEPROM and so on. There are two possible cases as below: • 0: this area is protected. • 1: this area is unprotected.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.
- STATUS_UNSUPPORTED: Operation was unsupported.

3.31.6.23 FLASH_DRV_SetFlexRamFunction()

```
status_t FLASH_DRV_SetFlexRamFunction (
            const flash_ssd_config_t * pSSDConfig,
            flash_flexRam_function_control_code_t flexRamFuncCode,
            uint16_t byteOfQuickWrite,
            flash_eeprom_status_t *const pEEPROMStatus )
```

Set FlexRAM function.

This function is used to change the function of the FlexRAM. When not partitioned for emulated EEPROM, the FlexRAM is typically used as traditional RAM. Otherwise, the FlexRam is typically used to store EEPROM data, the writing to EEPROM is normal write or quick write. In addition, this function may be used to get EEPROM status or complete interrupted EEPROM quick write process. For example, after partitioning to have EEPROM backup, FlexRAM is used for EEPROM use accordingly and if want to change FlexRAM to traditional RAM for FlashProgramSection() use, call this API with the function control code is 0xFFU.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*flexRamFuncCode*	FlexRam function control code. It can be: • 0x00U: Make FlexRAM available for emulated EEPROM. • 0x55U: Make FlexRAM available for EEPROM quick writes. • 0x77U: EEPROM quick write status query. • 0xAAU: Complete interrupted EEPROM quick write process. • 0xFFU: Make FlexRAM available as RAM.
in	*byteOfQuickWrite*	Number of FlexRAM bytes allocated for EEPROM quick writes. This parameter is only used for EEE_QUICK_WRITE command, it should be 0 if the function is not used for EEEPROM quick write.
out	*pEEPROMStatus*	Pointer to the EEPROM status. This parameter is only used for EEE_STATUS_QUERY command, it should be NULL if the function is not used for EEPROM status query.

Returns

operation status
- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.24 FLASH_DRV_SetPFlashProtection()

```
status_t FLASH_DRV_SetPFlashProtection (
            uint32_t protectStatus )
```

P-Flash set protection.

This API sets the P-Flash protection to the intended protection status. Setting P-Flash protection status is subject to a protection, transition restriction. If there is a setting violation, it returns an error code and the current protection status will not be changed.

Parameters

in	*protectStatus*	The expected protect status user wants to set to P-Flash protection register. Each bit is corresponding to protection of 1/32 of the total P-Flash. The least significant bit is corresponding to the lowest address area of P-Flash. The most significant bit is corresponding to the highest address area of P- Flash, and so on. There are two possible cases as shown below: • 0: this area is protected. • 1: this area is unprotected.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.25 FLASH_DRV_VerifyAllBlock()

```
status_t FLASH_DRV_VerifyAllBlock (
             const flash_ssd_config_t * pSSDConfig,
             uint8_t marginLevel )
```

Flash verify all blocks.

This function checks to see if the P-Flash and/or D-Flash, EEPROM backup area, and D-Flash IFR have been erased to the specified read margin level, if applicable, and releases security if the readout passes.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*marginLevel*	Read Margin Choice as follows: • marginLevel = 0x0U: use the Normal read level • marginLevel = 0x1U: use the User read • marginLevel = 0x2U: use the Factory read

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.26 FLASH_DRV_VerifyBlock()

```
status_t FLASH_DRV_VerifyBlock (
             const flash_ssd_config_t * pSSDConfig,
             uint32_t dest,
             uint8_t marginLevel )
```

Flash verify block.

This API checks to see if an entire P-Flash or D-Flash block has been erased to the specified margin level For the derivatives including multiply logical P-Flash or D-Flash blocks, this API erases a single block in a single call.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*dest*	Start address for the intended verify operation.
in	*marginLevel*	Read Margin Choice as follows: • marginLevel = 0x0U: use Normal read level • marginLevel = 0x1U: use the User read • marginLevel = 0x2U: use the Factory read

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failure was occurred.

3.31.6.27 FLASH_DRV_VerifySection()

```
status_t FLASH_DRV_VerifySection (
            const flash_ssd_config_t * pSSDConfig,
            uint32_t dest,
            uint16_t number,
            uint8_t marginLevel )
```

Flash verify section.

This API checks if a section of the P-Flash or the D-Flash memory is erased to the specified read margin level.

Parameters

in	*pSSDConfig*	The SSD configuration structure pointer.
in	*dest*	Start address for the intended verify operation.
in	*number*	Number of alignment unit to be verified. Refer to corresponding reference manual to get correct information of alignment constrain.
in	*marginLevel*	Read Margin Choice as follows: • marginLevel = 0x0U: use Normal read level • marginLevel = 0x1U: use the User read • marginLevel = 0x2U: use the Factory read

Returns

operation status

- STATUS_SUCCESS: Operation was successful.

- STATUS_ERROR: Operation failure was occurred.

3.31.7 Variable Documentation

3.31.7.1 brownOutCode

```
uint8_t brownOutCode
```

Brown-out detection code

3.31.7.2 CallBack [1/2]

```
flash_callback_t CallBack
```

Call back function to service the time critical events. Any code reachable from this function must not be placed in a Flash block targeted for a program/erase operation

3.31.7.3 CallBack [2/2]

```
flash_callback_t CallBack
```

Call back function to service the time critical events. Any code reachable from this function must not be placed in a Flash block targeted for a program/erase operation

3.31.7.4 DFlashBase [1/2]

```
uint32_t DFlashBase
```

For FlexNVM device, this is the base address of D-Flash memory (FlexNVM memory); For non-FlexNVM device, this field is unused

3.31.7.5 DFlashBase [2/2]

```
uint32_t DFlashBase
```

For FlexNVM device, this is the base address of D-Flash memory (FlexNVM memory); For non-FlexNVM device, this field is unused

3.31.7.6 DFlashSize

```
uint32_t DFlashSize
```

For FlexNVM device, this is the size in byte of area which is used as D-Flash from FlexNVM memory; For non-↩ FlexNVM device, this field is unused

3.31.7.7 EEESize

```
uint32_t EEESize
```

For FlexNVM device, this is the size in byte of EEPROM area which was partitioned from FlexRAM; For non-Flex↩ NVM device, this field is unused

3.31.7.8 EERAMBase [1/2]

```
uint32_t EERAMBase
```

The base address of FlexRAM (for FlexNVM device) or acceleration RAM memory (for non-FlexNVM device)

3.31.7.9 EERAMBase [2/2]

```
uint32_t EERAMBase
```

The base address of FlexRAM (for FlexNVM device) or acceleration RAM memory (for non-FlexNVM device)

3.31.7.10 numOfRecordReqMaintain

```
uint16_t numOfRecordReqMaintain
```

Number of EEPROM quick write records requiring maintenance

3.31.7.11 PFlashBase [1/2]

```
uint32_t PFlashBase
```

The base address of P-Flash memory

3.31.7.12 PFlashBase [2/2]

```
uint32_t PFlashBase
```

The base address of P-Flash memory

3.31.7.13 PFlashSize [1/2]

```
uint32_t PFlashSize
```

The size in byte of P-Flash memory

3.31.7.14 PFlashSize [2/2]

```
uint32_t PFlashSize
```

The size in byte of P-Flash memory

3.31.7.15 sectorEraseCount

```
uint16_t sectorEraseCount
```

EEPROM sector erase count

3.32 Flash Memory (Flash)

3.32.1 Detailed Description

Flash Memory Module provides the general flash APIs.

Flash memory is ideal for single-supply applications, permitting in-the-field erase and reprogramming operations without the need for any external high voltage power sources. The flash module includes a memory controller that executes commands to modify flash memory contents. An erased bit reads '1' and a programmed bit reads '0'. The programming operation is unidirectional; it can only move bits from the '1' state (erased) to the '0' state (programmed). Only the erase operation restores bits from '0' to '1'; bits cannot be programmed from a '0' to a '1'.

C90TFS Flash Driver

The C90TFS flash module includes the following accessible memory regions.

1. Program flash memory for vector space and code store.
2. FlexNVM for data store, additional code store and also non-volatile storage for the EEPROM filing system representing data written to the FlexRAM requiring highest endurance.
3. FlexRAM for high-endurance EEPROM data store or traditional RAM.

Some platforms may be designed to have only program flash memory or all of them.

The S32 SDK provides the C90TFS Flash driver of S32K platforms. The driver includes general APIs to handle specific operations on C90TFS Flash module. The user can use those APIs directly in the application.

EEPROM feature

For platforms with FlexNVM, the flash module provides a built-in hardware emulation scheme to emulate the characteristics of an EEPROM by effectively providing a high-endurance, byte write-able NVM. The EEPROM system is shown in the following figure.

Figure 1. EEPROM Architecture

To handle with various customer's requirements, the FlexRAM and FlexNVM blocks can be split into partitions:

1. EEPROM partition(EEESIZE) — The amount of FlexRAM used for EEPROM can be set from 0 Bytes (no EEPROM) to the maximum FlexRAM size. The remainder of the FlexRAM not used for EEPROM is not accessible while the FlexRAM is configured for EEPROM.The EEPROM partition grows upward from the bottom of the FlexRAM address space.
2. Data flash partition(DEPART) — The amount of FlexNVM memory used for data flash can be programmed from 0 bytes (all of the FlexNVM block is available for EEPROM backup) to the maximum size of the FlexNVM block.
3. FlexNVM EEPROM partition — The amount of FlexNVM memory used for EEPROM backup, which is equal to the FlexNVM block size minus the data flash memory partition size. The EEPROM backup size must be at least 16 times the EEPROM partition size in FlexRAM.

The partition information (EEESIZE, DEPART) is programmed using the **FLASH_DRV_DEFlashPartition** API.

The function of FlexRAM can be changed from EEPROM usage to traditional RAM for accelerate programming in **FLASH_DRV_ProgramSection** API and vice versa by **FLASH_DRV_SetFlexRamFunction** API.

This is example code of EEE usage sequence:

```
    /* Provide information about the flash blocks. */
    const flash_user_config_t flashUserConfig =
    {
        0x00000000u,                                /* Base address of Program Flash block */
        FEATURE_FLS_PF_BLOCK_SIZE,                  /* Size of Program Flash block */
        FEATURE_FLS_DF_START_ADDRESS,               /* Base address of Data Flash block */
        FEATURE_FLS_FLEX_RAM_START_ADDRESS,         /* Base address of FlexRAM block */
        NULL_CALLBACK                               /* Pointer to callback function */
    };

    /* Declare a FLASH configuration structure which is initialized by FlashInit, and will be used by all
       flash APIs */
    flash_ssd_config_t flashSSDConfig;

    /* Always initialize the driver before calling other functions */
    ret = FLASH_DRV_Init(&flashUserConfig, &flashSSDConfig);
    if (ret != STATUS_SUCCESS)
    {
        return ret;
    }

#if ((FEATURE_FLS_HAS_FLEX_NVM == 1u) & (FEATURE_FLS_HAS_FLEX_RAM == 1u))
    /* Configure FlexRAM as EEPROM if it is currently used as traditional RAM */
    if (flashSSDConfig.EEESize == 0u)
    {
        /* Configure FlexRAM as EEPROM and FlexNVM as EEPROM backup region,
           DEFlashPartition will be failed if the IFR region isn't blank.
           Refer to the device document for valid EEPROM Data Size Code
           and FlexNVM Partition Code. For example on S32K144:
           - EEEDataSizeCode = 0x02u: EEPROM size = 4 Kbytes
           - DEPartitionCode = 0x08u: EEPROM backup size = 64 Kbytes */
        ret = FLASH_DRV_DEFlashPartition(&flashSSDConfig, 0x02u, 0x08u, 0x0,
      false);
        if (ret != STATUS_SUCCESS)
        {
            return ret;
        }
        else
        {
            /* Re-initialize the driver to update the new EEPROM configuration */
            ret = FLASH_DRV_Init(&flashUserConfig, &flashSSDConfig);
            if (ret != STATUS_SUCCESS)
            {
                return ret;
            }

            /* Make FlexRAM available for EEPROM */
            ret = FLASH_DRV_SetFlexRamFunction(&flashSSDConfig,
      EEE_ENABLE, 0x0u, NULL);
            if (ret != STATUS_SUCCESS)
            {
                return ret;
            }
        }
    }
    else    /* FLexRAM is already configured as EEPROM */
    {
        /* Make FlexRAM available for EEPROM, make sure that FlexNVM and FlexRAM
           are already partitioned successfully before */
        ret = FLASH_DRV_SetFlexRamFunction(&flashSSDConfig,
      EEE_ENABLE, 0x0u, NULL);
        if (ret != STATUS_SUCCESS)
        {
            return ret;
        }
    }
#endif
```

Important Note

1. If using callback in the application, any code reachable from this function must not be placed in a Flash block targeted for a program/erase operation to avoid the RWW error. Functions can be placed in RAM section by using the START/END_FUNCTION_DEFINITION/DECLARATION_RAMSECTION macros.

2. To suspend the sector erase operation for a simple method, invoke the **FLASH_DRV_EraseSuspend** function within callback of **FLASH_DRV_EraseSector**. In this case, the **FLASH_DRV_EraseSuspend** must not be placed in the same block in which the Flash erase sector command is going on.
3. **#FLASH_DRV_CommandSequence**, **FLASH_DRV_EraseSuspend** and **FLASH_DRV_EraseResume** should be executed from RAM or different Flash blocks which are targeted for writing to avoid the RWW error. **FLASH_DRV_EraseSuspend** and **FLASH_DRV_EraseResume** functions should be called in pairs.
4. To guarantee the correct execution of this driver, the Flash cache in the Flash memory controller module should be disabled before invoking any API.
5. Partitioning FlexNVM and FlexRAM for EEPROM usage shall be executed only once in the lifetime of the device.
6. After successfully partitioning FlexNVM and FlexRAM for EEPROM usage, user needs to call **FLASH_DR↩V_Init** to update memory information in global structure.

Modules

- Flash Memory (Flash)

3.33 FlexCAN Driver

3.33.1 Detailed Description

How to use the FlexCAN driver in your application

In order to be able to use the FlexCAN in your application, the first thing to do is initializing it with the desired configuration. This is done by calling the **FLEXCAN_DRV_Init** function. One of the arguments passed to this function is the configuration which will be used for the FlexCAN, specified by the **flexcan_user_config_t** structure.

The **flexcan_user_config_t** structure allows you to configure the following:

- the number of message buffers needed;
- the number of RX FIFO ID filters needed;
- enable/disable the RxFIFO feature;
- the operation mode, which can be one of the following:
  - normal mode;
  - listen-only mode;
  - loopback mode;
  - freeze mode;
  - disable mode;
- the payload size of the message buffers:
  - 8 bytes;
  - 16 bytes (only available with the FD feature enabled);
  - 32 bytes (only available with the FD feature enabled);
  - 64 bytes (only available with the FD feature enabled);
- enable/disable the Flexible Data-rate feature;
- the clock source of the CAN Protocol Engine (PE);
- the bitrate used for standard frames or for the arbitration phase of FD frames;
- the bitrate used for the data phase of FD frames;
- the Rx transfer type, which can be one of the following:
  - using interrupts;
  - using DMA;
- the DMA channel number to be used for DMA transfers;

The bitrate is represented by a **flexcan_time_segment_t** structure, with the following fields:

- propagation segment;
- phase segment 1;
- phase segment 2;
- clock pre-divider;
- resync jump width.

Details about these fields can be found in the reference manual.

In order to use a mailbox for transmission/reception, it should be initialized using either **FLEXCAN_DRV_Config↩ RxMb**, **FLEXCAN_DRV_ConfigRxFifo** or **FLEXCAN_DRV_ConfigTxMb**.

After having the mailbox configured, you can start sending/receiving using it by calling one of the following functions:

- FLEXCAN_DRV_Send;
- FLEXCAN_DRV_SendBlocking;
- FLEXCAN_DRV_Receive;
- FLEXCAN_DRV_ReceiveBlocking;
- FLEXCAN_DRV_RxFifo;
- FLEXCAN_DRV_RxFifoBlocking.

Important Notes

- DMA module has to be initialized prior to FlexCAN RxFIFO usage in DMA mode; also, the DMA channel needs to be allocated by the application (the driver only takes care of configuring the DMA channel received in the configuration structure).

Example:

```
#define INST_CANCOM1 (0U)

flexcan_state_t canCom1_State;

const flexcan_user_config_t canCom1_InitConfig0 = {
    .fd_enable = true,
    .pe_clock = FLEXCAN_CLK_SOURCE_SOSCDIV2,
    .max_num_mb = 16,
    .num_id_filters = FLEXCAN_RX_FIFO_ID_FILTERS_8,
    .is_rx_fifo_needed = false,
    .flexcanMode = FLEXCAN_NORMAL_MODE,
    .payload = FLEXCAN_PAYLOAD_SIZE_8,
    .bitrate = {
        .propSeg = 7,
        .phaseSeg1 = 4,
        .phaseSeg2 = 1,
        .preDivider = 0,
        .rJumpwidth = 1
    },
    .bitrate_cbt = {
        .propSeg = 11,
        .phaseSeg1 = 1,
        .phaseSeg2 = 1,
        .preDivider = 0,
        .rJumpwidth = 1
    },
    .transfer_type = FLEXCAN_RXFIFO_USING_INTERRUPTS,
    .rxFifoDMAChannel = 0U
};

/* Initialize FlexCAN driver */
FLEXCAN_DRV_Init(INST_CANCOM1, &canCom1_State, &canCom1_InitConfig0);

/* Set information about the data to be received */
flexcan_data_info_t dataInfo =
{
    .data_length = 1U,
    .msg_id_type = FLEXCAN_MSG_ID_STD,
    .enable_brs  = true,
    .fd_enable   = true,
    .fd_padding  = 0U
};

/* Configure Rx message buffer with index 1 to receive frames with ID 1 */
FLEXCAN_DRV_ConfigRxMb(INST_CANCOM1, 1, &dataInfo, 1);
```

```
/* Receive a frame in the recvBuff variable */
flexcan_msgbuff_t recvBuff;
recvBuff.msgId = 1;
FLEXCAN_DRV_Receive(INST_CANCOM1, 1, &recvBuff);
/* Wait for the message to be received */
while (FLEXCAN_DRV_GetTransferStatus(INST_CANCOM1, 1) == STATUS_BUSY);

/* De-initialize driver */
FLEXCAN_DRV_Deinit(INST_CANCOM1);
```

Data Structures

- struct flexcan_mb_handle_t

  Information needed for internal handling of a given MB. Implements : flexcan_mb_handle_t_Class. More...
- struct FlexCANState

  Internal driver state information. More...
- struct flexcan_data_info_t

  FlexCAN data info from user Implements : flexcan_data_info_t_Class. More...
- struct flexcan_user_config_t

  FlexCAN configuration. More...

Typedefs

- typedef struct FlexCANState flexcan_state_t

  Internal driver state information.
- typedef void(∗ flexcan_callback_t) (uint8_t instance, flexcan_event_type_t eventType, flexcan_state_t ∗flexcanState)

  FlexCAN Driver callback function type Implements : flexcan_callback_t_Class.

Enumerations

- enum flexcan_rxfifo_transfer_type_t { FLEXCAN_RXFIFO_USING_INTERRUPTS, FLEXCAN_RXFIFO_USING_DMA }

  The type of the RxFIFO transfer (interrupts/DMA). Implements : flexcan_rxfifo_transfer_type_t_Class.
- enum flexcan_event_type_t { FLEXCAN_EVENT_RX_COMPLETE, FLEXCAN_EVENT_RXFIFO_COMPLETE, FLEXCAN_EVENT_TX_COMPLETE }

  The type of the event which occured when the callback was invoked. Implements : flexcan_event_type_t_Class.
- enum flexcan_mb_state_t { FLEXCAN_MB_IDLE, FLEXCAN_MB_RX_BUSY, FLEXCAN_MB_TX_BUSY }

  The state of a given MB (idle/Rx busy/Tx busy). Implements : flexcan_mb_state_t_Class.
- enum flexcan_rx_fifo_id_filter_num_t {
  FLEXCAN_RX_FIFO_ID_FILTERS_8 = 0x0, FLEXCAN_RX_FIFO_ID_FILTERS_16 = 0x1, FLEXCAN_RX_FIFO_ID_FILTERS_24 = 0x2, FLEXCAN_RX_FIFO_ID_FILTERS_32 = 0x3,
  FLEXCAN_RX_FIFO_ID_FILTERS_40 = 0x4, FLEXCAN_RX_FIFO_ID_FILTERS_48 = 0x5, FLEXCAN_RX_FIFO_ID_FILTERS_56 = 0x6, FLEXCAN_RX_FIFO_ID_FILTERS_64 = 0x7,
  FLEXCAN_RX_FIFO_ID_FILTERS_72 = 0x8, FLEXCAN_RX_FIFO_ID_FILTERS_80 = 0x9, FLEXCAN_RX_FIFO_ID_FILTERS_88 = 0xA, FLEXCAN_RX_FIFO_ID_FILTERS_96 = 0xB,
  FLEXCAN_RX_FIFO_ID_FILTERS_104 = 0xC, FLEXCAN_RX_FIFO_ID_FILTERS_112 = 0xD, FLEXCAN_RX_FIFO_ID_FILTERS_120 = 0xE, FLEXCAN_RX_FIFO_ID_FILTERS_128 = 0xF }

  FlexCAN Rx FIFO filters number Implements : flexcan_rx_fifo_id_filter_num_t_Class.

Functions

- void FLEXCAN_DRV_IRQHandler (uint8_t instance)
  Interrupt handler for a FlexCAN instance.
- status_t FLEXCAN_DRV_GetTransferStatus (uint32_t instance, uint8_t mb_idx)
  Returns whether the previous FLEXCAN transfer has finished.
- void FLEXCAN_DRV_InstallEventCallback (uint8_t instance, flexcan_callback_t callback, void ∗callback←Param)
  Installs a callback function for the IRQ handler.

Variables

- CAN_Type ∗const g_flexcanBase [CAN_INSTANCE_COUNT]
  Table of base addresses for FlexCAN instances.
- const IRQn_Type g_flexcanRxWarningIrqId [CAN_INSTANCE_COUNT]
  Table to save RX Warning IRQ numbers for FlexCAN instances.
- const IRQn_Type g_flexcanTxWarningIrqId [CAN_INSTANCE_COUNT]
  Table to save TX Warning IRQ numbers for FlexCAN instances.
- const IRQn_Type g_flexcanWakeUpIrqId [CAN_INSTANCE_COUNT]
  Table to save wakeup IRQ numbers for FlexCAN instances.
- const IRQn_Type g_flexcanErrorIrqId [CAN_INSTANCE_COUNT]
  Table to save error IRQ numbers for FlexCAN instances.
- const IRQn_Type g_flexcanBusOffIrqId [CAN_INSTANCE_COUNT]
  Table to save Bus off IRQ numbers for FlexCAN instances.
- const IRQn_Type g_flexcanOredMessageBufferIrqId [CAN_INSTANCE_COUNT][FEATURE_CAN_MB_IR←QS_MAX_COUNT]
  Table to save message buffer IRQ numbers for FlexCAN instances.

Bit rate

- void FLEXCAN_DRV_SetBitrate (uint8_t instance, const flexcan_time_segment_t ∗bitrate)
  Sets the FlexCAN bit rate.

Set baud rate for BRS FD

- void FLEXCAN_DRV_SetBitrateCbt (uint8_t instance, const flexcan_time_segment_t ∗bitrate)
  Sets the FlexCAN bit rate for FD BRS.
- void FLEXCAN_DRV_GetBitrate (uint8_t instance, flexcan_time_segment_t ∗bitrate)
  Gets the FlexCAN bit rate.

Global mask

- void FLEXCAN_DRV_SetRxMaskType (uint8_t instance, flexcan_rx_mask_type_t type)
  Sets the RX masking type.
- void FLEXCAN_DRV_SetRxFifoGlobalMask (uint8_t instance, flexcan_msgbuff_id_type_t id_type, uint32_t mask)
  Sets the FlexCAN RX FIFO global standard or extended mask.
- void FLEXCAN_DRV_SetRxMbGlobalMask (uint8_t instance, flexcan_msgbuff_id_type_t id_type, uint32_t mask)
  Sets the FlexCAN RX MB global standard or extended mask.
- status_t FLEXCAN_DRV_SetRxIndividualMask (uint8_t instance, flexcan_msgbuff_id_type_t id_type, uint8←_t mb_idx, uint32_t mask)
  Sets the FlexCAN RX individual standard or extended mask.

Initialization and Shutdown

- status_t FLEXCAN_DRV_Init (uint32_t instance, flexcan_state_t ∗state, const flexcan_user_config_t ∗data)
  Initializes the FlexCAN peripheral.
- status_t FLEXCAN_DRV_Deinit (uint8_t instance)
  Shuts down a FlexCAN instance.

Send configuration

- status_t FLEXCAN_DRV_ConfigTxMb (uint8_t instance, uint8_t mb_idx, const flexcan_data_info_t ∗tx_info, uint32_t msg_id)
  FlexCAN transmit message buffer field configuration.
- status_t FLEXCAN_DRV_SendBlocking (uint8_t instance, uint8_t mb_idx, const flexcan_data_info_t ∗tx_info, uint32_t msg_id, const uint8_t ∗mb_data, uint32_t timeout_ms)
  Sends a CAN frame using the specified message buffer, in a blocking manner.
- status_t FLEXCAN_DRV_Send (uint8_t instance, uint8_t mb_idx, const flexcan_data_info_t ∗tx_info, uint32_t msg_id, const uint8_t ∗mb_data)
  Sends a CAN frame using the specified message buffer.
- status_t FLEXCAN_DRV_AbortTransfer (uint32_t instance, uint8_t mb_idx)
  Ends a non-blocking FlexCAN transfer early.

Receive configuration

- status_t FLEXCAN_DRV_ConfigRxMb (uint8_t instance, uint8_t mb_idx, const flexcan_data_info_t ∗rx_info, uint32_t msg_id)
  FlexCAN receive message buffer field configuration.
- void FLEXCAN_DRV_ConfigRxFifo (uint8_t instance, flexcan_rx_fifo_id_element_format_t id_format, const flexcan_id_table_t ∗id_filter_table)
  FlexCAN RX FIFO field configuration.
- status_t FLEXCAN_DRV_ReceiveBlocking (uint8_t instance, uint8_t mb_idx, flexcan_msgbuff_t ∗data, uint32_t timeout_ms)
  Receives a CAN frame using the specified message buffer, in a blocking manner.
- status_t FLEXCAN_DRV_Receive (uint8_t instance, uint8_t mb_idx, flexcan_msgbuff_t ∗data)
  Receives a CAN frame using the specified message buffer.
- status_t FLEXCAN_DRV_RxFifoBlocking (uint8_t instance, flexcan_msgbuff_t ∗data, uint32_t timeout_ms)
  Receives a CAN frame using the message FIFO, in a blocking manner.
- status_t FLEXCAN_DRV_RxFifo (uint8_t instance, flexcan_msgbuff_t ∗data)
  Receives a CAN frame using the message FIFO.

3.33.2 Data Structure Documentation

3.33.2.1 struct flexcan_mb_handle_t

Information needed for internal handling of a given MB. Implements : flexcan_mb_handle_t_Class.

Data Fields

- flexcan_msgbuff_t ∗ mb_message
- semaphore_t mbSema
- flexcan_mb_state_t state
- bool isBlocking
- bool isRemote

Field Documentation

3.33.2.1.1 isBlocking

```
bool isBlocking
```

True if the transfer is blocking

3.33.2.1.2 isRemote

```
bool isRemote
```

True if the frame is a remote frame

3.33.2.1.3 mb_message

```
flexcan_msgbuff_t* mb_message
```

The FlexCAN receive MB data

3.33.2.1.4 mbSema

```
semaphore_t mbSema
```

Semaphore used for signaling completion of a blocking transfer

3.33.2.1.5 state

```
flexcan_mb_state_t state
```

The state of the current MB (idle/Rx busy/Tx busy)

3.33.2.2 struct FlexCANState

Internal driver state information.

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. Implements : flexcan_state_t_Class

Data Fields

- volatile flexcan_mb_handle_t mbs [FEATURE_CAN_MAX_MB_NUM]
- void(∗ callback)(uint8_t instance, flexcan_event_type_t eventType, struct FlexCANState ∗state)
- void ∗ callbackParam
- flexcan_rxfifo_transfer_type_t transferType

Field Documentation

3.33.2.2.1 callback

```
void(* callback) (uint8_t instance, flexcan_event_type_t eventType, struct FlexCANState *state)
```

IRQ handler callback function.

3.33.2.2.2 callbackParam

```
void* callbackParam
```

Parameter used to pass user data when invoking the callback function.

3.33.2.2.3 mbs

```
volatile flexcan_mb_handle_t mbs[FEATURE_CAN_MAX_MB_NUM]
```

Array containing information related to each MB

3.33.2.2.4 transferType

```
flexcan_rxfifo_transfer_type_t transferType
```

Type of RxFIFO transfer.

3.33.2.3 struct flexcan_data_info_t

FlexCAN data info from user Implements : flexcan_data_info_t_Class.

Data Fields

- flexcan_msgbuff_id_type_t msg_id_type
- uint32_t data_length
- bool fd_enable
- uint8_t fd_padding
- bool enable_brs
- bool is_remote

Field Documentation

3.33.2.3.1 data_length

```
uint32_t data_length
```

Length of Data in Bytes

3.33.2.3.2 enable_brs

```
bool enable_brs
```

is bigger than data_length to fill the MB Enable bit rate switch

3.33.2.3.3 fd_enable

```
bool fd_enable
```

Enable or disable FD

3.33.2.3.4 fd_padding

```
uint8_t fd_padding
```

Set a value for padding. It will be used when payload

3.33.2.3.5 is_remote

```
bool is_remote
```

Specifies if the frame is standard or remote

3.33.2.3.6 msg_id_type

```
flexcan_msgbuff_id_type_t msg_id_type
```

Type of message ID (standard or extended)

3.33.2.4 struct flexcan_user_config_t

FlexCAN configuration.

Data Fields

- uint32_t max_num_mb
- flexcan_rx_fifo_id_filter_num_t num_id_filters
- bool is_rx_fifo_needed
- flexcan_operation_modes_t flexcanMode
- flexcan_fd_payload_size_t payload
- bool fd_enable
- flexcan_time_segment_t bitrate
- flexcan_time_segment_t bitrate_cbt
- flexcan_rxfifo_transfer_type_t transfer_type
- uint8_t rxFifoDMAChannel

Field Documentation

3.33.2.4.1 bitrate

```
flexcan_time_segment_t bitrate
```

The bitrate used for standard frames or for the arbitration phase of FD frames.

3.33.2.4.2 bitrate_cbt

```
flexcan_time_segment_t bitrate_cbt
```

The bitrate used for the data phase of FD frames.

3.33.2.4.3 fd_enable

```
bool fd_enable
```

Enable/Disable the Flexible Data Rate feature.

3.33.2.4.4 flexcanMode

```
flexcan_operation_modes_t flexcanMode
```

User configurable FlexCAN operation modes.

3.33.2.4.5 is_rx_fifo_needed

```
bool is_rx_fifo_needed
```

1 if needed; 0 if not. This controls whether the Rx FIFO feature is enabled or not.

3.33.2.4.6 max_num_mb

```
uint32_t max_num_mb
```

The maximum number of Message Buffers

3.33.2.4.7 num_id_filters

```
flexcan_rx_fifo_id_filter_num_t num_id_filters
```

The number of RX FIFO ID filters needed

3.33.2.4.8 payload

```
flexcan_fd_payload_size_t payload
```

The payload size of the mailboxes.

3.33.2.4.9 rxFifoDMAChannel

```
uint8_t rxFifoDMAChannel
```

Specifies the DMA channel number to be used for DMA transfers.

3.33.2.4.10 transfer_type

```
flexcan_rxfifo_transfer_type_t transfer_type
```

Specifies if the RxFIFO uses interrupts or DMA.

3.33.3 Typedef Documentation

3.33.3.1 flexcan_callback_t

```
typedef void(* flexcan_callback_t) (uint8_t instance, flexcan_event_type_t eventType, flexcan↩
_state_t *flexcanState)
```

FlexCAN Driver callback function type Implements : flexcan_callback_t_Class.

3.33.3.2 flexcan_state_t

```
typedef struct FlexCANState flexcan_state_t
```

Internal driver state information.

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. Implements : flexcan_state_t_Class

3.33.4 Enumeration Type Documentation

3.33.4.1 flexcan_event_type_t

```
enum flexcan_event_type_t
```

The type of the event which occured when the callback was invoked. Implements : flexcan_event_type_t_Class.

Enumerator

FLEXCAN_EVENT_RX_COMPLETE	A frame was received in the configured Rx MB.
FLEXCAN_EVENT_RXFIFO_COMPLETE	A frame was received in the RxFIFO.
FLEXCAN_EVENT_TX_COMPLETE	A frame was sent from the configured Tx MB.

3.33.4.2 flexcan_mb_state_t

```
enum flexcan_mb_state_t
```

The state of a given MB (idle/Rx busy/Tx busy). Implements : flexcan_mb_state_t_Class.

Enumerator

FLEXCAN_MB_IDLE	The MB is not used by any transfer.
FLEXCAN_MB_RX_BUSY	The MB is used for a reception.
FLEXCAN_MB_TX_BUSY	The MB is used for a transmission.

3.33.4.3 flexcan_rx_fifo_id_filter_num_t

```
enum flexcan_rx_fifo_id_filter_num_t
```

FlexCAN Rx FIFO filters number Implements : flexcan_rx_fifo_id_filter_num_t_Class.

Enumerator

FLEXCAN_RX_FIFO_ID_FILTERS_8	8 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_16	16 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_24	24 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_32	32 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_40	40 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_48	48 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_56	56 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_64	64 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_72	72 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_80	80 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_88	88 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_96	96 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_104	104 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_112	112 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_120	120 Rx FIFO Filters.
FLEXCAN_RX_FIFO_ID_FILTERS_128	128 Rx FIFO Filters.

3.33.4.4 flexcan_rxfifo_transfer_type_t

```
enum flexcan_rxfifo_transfer_type_t
```

The type of the RxFIFO transfer (interrupts/DMA). Implements : flexcan_rxfifo_transfer_type_t_Class.

Enumerator

FLEXCAN_RXFIFO_USING_INTERRUPTS	Use interrupts for RxFIFO.
FLEXCAN_RXFIFO_USING_DMA	Use DMA for RxFIFO.

3.33.5 Function Documentation

3.33.5.1 FLEXCAN_DRV_AbortTransfer()

```
status_t FLEXCAN_DRV_AbortTransfer (
            uint32_t instance,
            uint8_t mb_idx )
```

Ends a non-blocking FlexCAN transfer early.

Parameters

instance	A FlexCAN instance number
mb_idx	The index of the message buffer

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_NO_TRANSFER_IN_PROGRESS if no transfer was running

3.33.5.2 FLEXCAN_DRV_ConfigRxFifo()

```
void FLEXCAN_DRV_ConfigRxFifo (
            uint8_t instance,
            flexcan_rx_fifo_id_element_format_t id_format,
            const flexcan_id_table_t * id_filter_table )
```

FlexCAN RX FIFO field configuration.

Parameters

instance	A FlexCAN instance number
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

3.33.5.3 FLEXCAN_DRV_ConfigRxMb()

```
status_t FLEXCAN_DRV_ConfigRxMb (
            uint8_t instance,
            uint8_t mb_idx,
            const flexcan_data_info_t * rx_info,
            uint32_t msg_id )
```

FlexCAN receive message buffer field configuration.

Parameters

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

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of a message buffer is invalid; STATUS_ERROR if other error occurred

3.33.5.4 FLEXCAN_DRV_ConfigTxMb()

```
status_t FLEXCAN_DRV_ConfigTxMb (
            uint8_t instance,
            uint8_t mb_idx,
            const flexcan_data_info_t * tx_info,
            uint32_t msg_id )
```

FlexCAN transmit message buffer field configuration.

Parameters

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

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of the message buffer is invalid

3.33.5.5 FLEXCAN_DRV_Deinit()

```
status_t FLEXCAN_DRV_Deinit (
            uint8_t instance )
```

Shuts down a FlexCAN instance.

Parameters

instance	A FlexCAN instance number

Returns

STATUS_SUCCESS if successful; STATUS_ERROR if failed

3.33.5.6 FLEXCAN_DRV_GetBitrate()

```
void FLEXCAN_DRV_GetBitrate (
            uint8_t instance,
            flexcan_time_segment_t * bitrate )
```

Gets the FlexCAN bit rate.

Parameters

instance	A FlexCAN instance number
bitrate	A pointer to a variable for returning the FlexCAN bit rate settings

3.33.5.7 FLEXCAN_DRV_GetTransferStatus()

```
status_t FLEXCAN_DRV_GetTransferStatus (
            uint32_t instance,
            uint8_t mb_idx )
```

Returns whether the previous FLEXCAN transfer has finished.

When performing an async transfer, call this function to ascertain the state of the current transfer: in progress (or busy) or complete (success).

Parameters

instance	The FLEXCAN module base address.
mb_idx	The index of the message buffer.

Returns

STATUS_SUCCESS if successful; STATUS_BUSY if a resource is busy;

3.33.5.8 FLEXCAN_DRV_Init()

```
status_t FLEXCAN_DRV_Init (
            uint32_t instance,
            flexcan_state_t * state,
            const flexcan_user_config_t * data )
```

Initializes the FlexCAN peripheral.

This function initializes

Parameters

instance	A FlexCAN instance number
state	Pointer to the FlexCAN driver state structure.
data	The FlexCAN platform data

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of a message buffer is invalid; STATUS_ERROR if other error occurred

3.33.5.9 FLEXCAN_DRV_InstallEventCallback()

```
void FLEXCAN_DRV_InstallEventCallback (
            uint8_t instance,
            flexcan_callback_t callback,
            void * callbackParam )
```

Installs a callback function for the IRQ handler.

Parameters

instance	The FLEXCAN module base address.
callback	The callback function.
callbackParam	User parameter passed to the callback function through the state parameter.

3.33.5.10 FLEXCAN_DRV_IRQHandler()

```
void FLEXCAN_DRV_IRQHandler (
            uint8_t instance )
```

Interrupt handler for a FlexCAN instance.

Parameters

instance	The FlexCAN instance number.

3.33.5.11 FLEXCAN_DRV_Receive()

```
status_t FLEXCAN_DRV_Receive (
            uint8_t instance,
            uint8_t mb_idx,
            flexcan_msgbuff_t * data )
```

Receives a CAN frame using the specified message buffer.

This function receives a CAN frame using a configured message buffer. The function returns immediately. If a callback is installed, it will be invoked after the frame was received and read into the specified buffer.

Parameters

instance	A FlexCAN instance number
mb_idx	Index of the message buffer
data	The FlexCAN receive message buffer data.

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of a message buffer is invalid; STATUS_BUSY if a resource is busy; STATUS_ERROR if other error occurred

3.33.5.12 FLEXCAN_DRV_ReceiveBlocking()

```
status_t FLEXCAN_DRV_ReceiveBlocking (
            uint8_t instance,
            uint8_t mb_idx,
            flexcan_msgbuff_t * data,
            uint32_t timeout_ms )
```

Receives a CAN frame using the specified message buffer, in a blocking manner.

This function receives a CAN frame using a configured message buffer. The function blocks until either a frame was received, or the specified timeout expired.

Parameters

instance	A FlexCAN instance number
mb_idx	Index of the message buffer
data	The FlexCAN receive message buffer data.
timeout_ms	A timeout for the transfer in milliseconds.

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of a message buffer is invalid; STATUS_BUSY if a resource is busy; STATUS_TIMEOUT if the timeout is reached; STAT↩
US_ERROR if other error occurred

3.33.5.13 FLEXCAN_DRV_RxFifo()

```
status_t FLEXCAN_DRV_RxFifo (
            uint8_t instance,
            flexcan_msgbuff_t * data )
```

Receives a CAN frame using the message FIFO.

This function receives a CAN frame using the Rx FIFO. The function returns immediately. If a callback is installed, it will be invoked after the frame was received and read into the specified buffer.

Parameters

instance	A FlexCAN instance number
data	The FlexCAN receive message buffer data.

Returns

STATUS_SUCCESS if successful; STATUS_BUSY if a resource is busy; STATUS_ERROR if other error occurred

3.33.5.14 FLEXCAN_DRV_RxFifoBlocking()

```
status_t FLEXCAN_DRV_RxFifoBlocking (
            uint8_t instance,
            flexcan_msgbuff_t * data,
            uint32_t timeout_ms )
```

Receives a CAN frame using the message FIFO, in a blocking manner.

This function receives a CAN frame using the Rx FIFO. The function blocks until either a frame was received, or the specified timeout expired.

Parameters

instance	A FlexCAN instance number
data	The FlexCAN receive message buffer data.
timeout_ms	A timeout for the transfer in milliseconds.

Returns

STATUS_SUCCESS if successful; STATUS_BUSY if a resource is busy; STATUS_TIMEOUT if the timeout is reached; STATUS_ERROR if other error occurred

3.33.5.15 FLEXCAN_DRV_Send()

```
status_t FLEXCAN_DRV_Send (
            uint8_t instance,
            uint8_t mb_idx,
            const flexcan_data_info_t * tx_info,
            uint32_t msg_id,
            const uint8_t * mb_data )
```

Sends a CAN frame using the specified message buffer.

This function sends a CAN frame using a configured message buffer. The function returns immediately. If a callback is installed, it will be invoked after the frame was sent.

Parameters

instance	A FlexCAN instance number
mb_idx	Index of the message buffer
tx_info	Data info
msg_id	ID of the message to transmit
mb_data	Bytes of the FlexCAN message.

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of a message buffer is invalid; STATUS_BUSY if a resource is busy; STATUS_ERROR if other error occurred

3.33.5.16 FLEXCAN_DRV_SendBlocking()

```
status_t FLEXCAN_DRV_SendBlocking (
             uint8_t instance,
             uint8_t mb_idx,
             const flexcan_data_info_t * tx_info,
             uint32_t msg_id,
             const uint8_t * mb_data,
             uint32_t timeout_ms )
```

Sends a CAN frame using the specified message buffer, in a blocking manner.

This function sends a CAN frame using a configured message buffer. The function blocks until either the frame was sent, or the specified timeout expired.

Parameters

instance	A FlexCAN instance number
mb_idx	Index of the message buffer
tx_info	Data info
msg_id	ID of the message to transmit
mb_data	Bytes of the FlexCAN message
timeout_ms	A timeout for the transfer in milliseconds.

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of a message buffer is invalid; STATUS_BUSY if a resource is busy; STATUS_TIMEOUT if the timeout is reached; STAT↩ US_ERROR if other error occurred

3.33.5.17 FLEXCAN_DRV_SetBitrate()

```
void FLEXCAN_DRV_SetBitrate (
             uint8_t instance,
             const flexcan_time_segment_t * bitrate )
```

Sets the FlexCAN bit rate.

Parameters

instance	A FlexCAN instance number
bitrate	A pointer to the FlexCAN bit rate settings.

3.33.5.18 FLEXCAN_DRV_SetBitrateCbt()

```
void FLEXCAN_DRV_SetBitrateCbt (
             uint8_t instance,
             const flexcan_time_segment_t * bitrate )
```

Sets the FlexCAN bit rate for FD BRS.

Parameters

instance	A FlexCAN instance number
bitrate	A pointer to the FlexCAN bit rate settings.

3.33.5.19 FLEXCAN_DRV_SetRxFifoGlobalMask()

```
void FLEXCAN_DRV_SetRxFifoGlobalMask (
             uint8_t instance,
             flexcan_msgbuff_id_type_t id_type,
             uint32_t mask )
```

Sets the FlexCAN RX FIFO global standard or extended mask.

Parameters

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

3.33.5.20 FLEXCAN_DRV_SetRxIndividualMask()

```
status_t FLEXCAN_DRV_SetRxIndividualMask (
             uint8_t instance,
             flexcan_msgbuff_id_type_t id_type,
             uint8_t mb_idx,
             uint32_t mask )
```

Sets the FlexCAN RX individual standard or extended 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

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of the message buffer is invalid

3.33.5.21 FLEXCAN_DRV_SetRxMaskType()

```
void FLEXCAN_DRV_SetRxMaskType (
             uint8_t instance,
             flexcan_rx_mask_type_t type )
```

Sets the RX masking type.

Parameters

instance	A FlexCAN instance number
type	The FlexCAN RX mask type

3.33.5.22 FLEXCAN_DRV_SetRxMbGlobalMask()

```
void FLEXCAN_DRV_SetRxMbGlobalMask (
             uint8_t instance,
             flexcan_msgbuff_id_type_t id_type,
             uint32_t mask )
```

Sets the FlexCAN RX MB global standard or extended mask.

Parameters

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

3.33.6 Variable Documentation

3.33.6.1 g_flexcanBase

```
CAN_Type* const g_flexcanBase[CAN_INSTANCE_COUNT]
```

Table of base addresses for FlexCAN instances.

3.33.6.2 g_flexcanBusOffIrqId

```
const IRQn_Type g_flexcanBusOffIrqId[CAN_INSTANCE_COUNT]
```

Table to save Bus off IRQ numbers for FlexCAN instances.

3.33.6.3 g_flexcanErrorIrqId

```
const IRQn_Type g_flexcanErrorIrqId[CAN_INSTANCE_COUNT]
```

Table to save error IRQ numbers for FlexCAN instances.

3.33.6.4 g_flexcanOredMessageBufferIrqId

```
const IRQn_Type g_flexcanOredMessageBufferIrqId[CAN_INSTANCE_COUNT][FEATURE_CAN_MB_IRQS_MAX_↩
COUNT]
```

Table to save message buffer IRQ numbers for FlexCAN instances.

3.33.6.5 g_flexcanRxWarningIrqId

```
const IRQn_Type g_flexcanRxWarningIrqId[CAN_INSTANCE_COUNT]
```

Table to save RX Warning IRQ numbers for FlexCAN instances.

3.33.6.6 g_flexcanTxWarningIrqId

```
const IRQn_Type g_flexcanTxWarningIrqId[CAN_INSTANCE_COUNT]
```

Table to save TX Warning IRQ numbers for FlexCAN instances.

3.33.6.7 g_flexcanWakeUpIrqId

```
const IRQn_Type g_flexcanWakeUpIrqId[CAN_INSTANCE_COUNT]
```

Table to save wakeup IRQ numbers for FlexCAN instances.

3.34 FlexCAN HAL

3.34.1 Detailed Description

Data Structures

- struct flexcan_id_table_t

  FlexCAN RX FIFO ID filter table structure Implements : flexcan_id_table_t_Class. More...
- struct flexcan_buserr_counter_t

  FlexCAN bus error counters Implements : flexcan_buserr_counter_t_Class. More...
- struct flexcan_msgbuff_code_status_t

  FlexCAN Message Buffer code and status for transmit and receive Implements : flexcan_msgbuff_code_status_t_↩ Class. More...
- struct flexcan_msgbuff_t

  FlexCAN message buffer structure Implements : flexcan_msgbuff_t_Class. More...
- struct flexcan_time_segment_t

  FlexCAN timing related structures Implements : flexcan_time_segment_t_Class. More...
- struct flexcan_pn_id_filter_t

  Pretended Networking ID filter. More...
- struct flexcan_pn_payload_filter_t

  Pretended Networking payload filter. More...
- struct flexcan_pn_config_t

  Pretended Networking configuration structure Implements : flexcan_pn_config_t_Class. More...

Macros

- #define CAN_ID_EXT_MASK 0x3FFFFu
- #define CAN_ID_EXT_SHIFT 0
- #define CAN_ID_EXT_WIDTH 18
- #define CAN_ID_STD_MASK 0x1FFC0000u
- #define CAN_ID_STD_SHIFT 18
- #define CAN_ID_STD_WIDTH 11
- #define CAN_ID_PRIO_MASK 0xE0000000u
- #define CAN_ID_PRIO_SHIFT 29
- #define CAN_ID_PRIO_WIDTH 3
- #define CAN_CS_TIME_STAMP_MASK 0xFFFFu
- #define CAN_CS_TIME_STAMP_SHIFT 0
- #define CAN_CS_TIME_STAMP_WIDTH 16
- #define CAN_CS_DLC_MASK 0xF0000u
- #define CAN_CS_DLC_SHIFT 16
- #define CAN_CS_DLC_WIDTH 4
- #define CAN_CS_RTR_MASK 0x100000u
- #define CAN_CS_RTR_SHIFT 20
- #define CAN_CS_RTR_WIDTH 1
- #define CAN_CS_IDE_MASK 0x200000u
- #define CAN_CS_IDE_SHIFT 21
- #define CAN_CS_IDE_WIDTH 1
- #define CAN_CS_SRR_MASK 0x400000u
- #define CAN_CS_SRR_SHIFT 22
- #define CAN_CS_SRR_WIDTH 1
- #define CAN_CS_CODE_MASK 0xF000000u
- #define CAN_CS_CODE_SHIFT 24
- #define CAN_CS_CODE_WIDTH 4
- #define CAN_MB_EDL_MASK 0x80000000u
- #define CAN_MB_BRS_MASK 0x40000000u

Enumerations

- enum flexcan_operation_modes_t {
FLEXCAN_NORMAL_MODE, FLEXCAN_LISTEN_ONLY_MODE, FLEXCAN_LOOPBACK_MODE, FLEX↩
CAN_FREEZE_MODE,
FLEXCAN_DISABLE_MODE }

  FlexCAN operation modes Implements : flexcan_operation_modes_t_Class.
- enum {
FLEXCAN_RX_INACTIVE = 0x0, FLEXCAN_RX_FULL = 0x2, FLEXCAN_RX_EMPTY = 0x4, FLEXCAN_↩
RX_OVERRUN = 0x6,
FLEXCAN_RX_BUSY = 0x8, FLEXCAN_RX_RANSWER = 0xA, FLEXCAN_RX_NOT_USED = 0xF }

  FlexCAN message buffer CODE for Rx buffers.
- enum {
FLEXCAN_TX_INACTIVE = 0x08, FLEXCAN_TX_ABORT = 0x09, FLEXCAN_TX_DATA = 0x0C, FLEXC↩
AN_TX_REMOTE = 0x1C,
FLEXCAN_TX_TANSWER = 0x0E, FLEXCAN_TX_NOT_USED = 0xF }

  FlexCAN message buffer CODE FOR Tx buffers.
- enum {
FLEXCAN_MB_STATUS_TYPE_TX, FLEXCAN_MB_STATUS_TYPE_TX_REMOTE, FLEXCAN_MB_ST↩
ATUS_TYPE_RX, FLEXCAN_MB_STATUS_TYPE_RX_REMOTE,
FLEXCAN_MB_STATUS_TYPE_RX_TX_REMOTE }

  FlexCAN message buffer transmission types.
- enum flexcan_fd_payload_size_t { FLEXCAN_PAYLOAD_SIZE_8 = 0, FLEXCAN_PAYLOAD_SIZE_16, F↩
LEXCAN_PAYLOAD_SIZE_32, FLEXCAN_PAYLOAD_SIZE_64 }

  FlexCAN payload sizes Implements : flexcan_fd_payload_size_t_Class.
- enum flexcan_rx_fifo_id_element_format_t { FLEXCAN_RX_FIFO_ID_FORMAT_A, FLEXCAN_RX_FIFO↩
_ID_FORMAT_B, FLEXCAN_RX_FIFO_ID_FORMAT_C, FLEXCAN_RX_FIFO_ID_FORMAT_D }

  ID formats for RxFIFO Implements : flexcan_rx_fifo_id_element_format_t_Class.
- enum flexcan_rx_mask_type_t { FLEXCAN_RX_MASK_GLOBAL, FLEXCAN_RX_MASK_INDIVIDUAL }

  FlexCAN RX mask type. Implements : flexcan_rx_mask_type_t_Class.
- enum flexcan_msgbuff_id_type_t { FLEXCAN_MSG_ID_STD, FLEXCAN_MSG_ID_EXT }

  FlexCAN Message Buffer ID type Implements : flexcan_msgbuff_id_type_t_Class.
- enum flexcan_clk_source_t { FLEXCAN_CLK_SOURCE_SOSCDIV2, FLEXCAN_CLK_SOURCE_SYS }

  FlexCAN clock source Implements : flexcan_clk_source_t_Class.
- enum flexcan_int_type_t { FLEXCAN_INT_RX_WARNING = CAN_CTRL1_RWRNMSK_MASK, FLEXCA↩
N_INT_TX_WARNING = CAN_CTRL1_TWRNMSK_MASK, FLEXCAN_INT_ERR = CAN_CTRL1_ERRM↩
SK_MASK, FLEXCAN_INT_BUSOFF = CAN_CTRL1_BOFFMSK_MASK }

  FlexCAN error interrupt types Implements : flexcan_int_type_t_Class.
- enum flexcan_pn_filter_combination_t { FLEXCAN_FILTER_ID, FLEXCAN_FILTER_ID_PAYLOAD, FLEX↩
CAN_FILTER_ID_NTIMES, FLEXCAN_FILTER_ID_PAYLOAD_NTIMES }

  Pretended Networking filtering combinations.
- enum flexcan_pn_filter_selection_t { FLEXCAN_FILTER_MATCH_EXACT, FLEXCAN_FILTER_MATCH_↩
GEQ, FLEXCAN_FILTER_MATCH_LEQ, FLEXCAN_FILTER_MATCH_RANGE }

  Pretended Networking matching schemes.

Configuration

- void FLEXCAN_HAL_Enable (CAN_Type ∗base)

  Enables FlexCAN controller.
- void FLEXCAN_HAL_Disable (CAN_Type ∗base)

  Disables FlexCAN controller.
- void FLEXCAN_HAL_SelectClock (CAN_Type ∗base, flexcan_clk_source_t clk)

  Selects the clock source for FlexCAN.

- static bool FLEXCAN_HAL_GetClock (const CAN_Type ∗base)

  Reads the clock source for FlexCAN Protocol Engine (PE).
- void FLEXCAN_HAL_Init (CAN_Type ∗base)

  Initializes the FlexCAN controller.
- void FLEXCAN_HAL_SetTimeSegments (CAN_Type ∗base, const flexcan_time_segment_t ∗timeSeg)

  Sets the FlexCAN time segments for setting up bit rate.
- void FLEXCAN_HAL_SetTimeSegmentsCbt (CAN_Type ∗base, const flexcan_time_segment_t ∗timeSeg)

  Sets the FlexCAN time segments for setting up bit rate for FD BRS.
- void FLEXCAN_HAL_GetTimeSegments (const CAN_Type ∗base, flexcan_time_segment_t ∗timeSeg)

  Gets the FlexCAN time segments to calculate the bit rate.
- void FLEXCAN_HAL_ExitFreezeMode (CAN_Type ∗base)

  Un freezes the FlexCAN module.
- void FLEXCAN_HAL_EnterFreezeMode (CAN_Type ∗base)

  Freezes the FlexCAN module.
- void FLEXCAN_HAL_SetOperationMode (CAN_Type ∗base, flexcan_operation_modes_t mode)

  Set operation mode.
- void FLEXCAN_HAL_ExitOperationMode (CAN_Type ∗base, flexcan_operation_modes_t mode)

  Exit operation mode.
- void FLEXCAN_HAL_SetFDEnabled (CAN_Type ∗base, bool enable)

  Enables/Disables Flexible Data rate (if supported).
- bool FLEXCAN_HAL_IsFDEnabled (const CAN_Type ∗base)

  Checks if the Flexible Data rate feature is enabled.
- void FLEXCAN_HAL_SetPayloadSize (CAN_Type ∗base, flexcan_fd_payload_size_t payloadSize)

  Sets the payload size of the MBs.
- uint8_t FLEXCAN_HAL_GetPayloadSize (const CAN_Type ∗base)

  Gets the payload size of the MBs.

Data transfer

- status_t FLEXCAN_HAL_SetTxMsgBuff (CAN_Type ∗base, uint32_t msgBuffIdx, const flexcan_msgbuff_↩
code_status_t ∗cs, uint32_t msgId, const uint8_t ∗msgData)

  Sets the FlexCAN message buffer fields for transmitting.
- status_t FLEXCAN_HAL_SetRxMsgBuff (CAN_Type ∗base, uint32_t msgBuffIdx, const flexcan_msgbuff_↩
code_status_t ∗cs, uint32_t msgId)

  Sets the FlexCAN message buffer fields for receiving.
- status_t FLEXCAN_HAL_GetMsgBuff (CAN_Type ∗base, uint32_t msgBuffIdx, flexcan_msgbuff_t ∗msgBuff)

  Gets the FlexCAN message buffer fields.
- status_t FLEXCAN_HAL_LockRxMsgBuff (CAN_Type ∗base, uint32_t msgBuffIdx)

  Locks the FlexCAN Rx message buffer.
- static void FLEXCAN_HAL_UnlockRxMsgBuff (const CAN_Type ∗base)

  Unlocks the FlexCAN Rx message buffer.
- status_t FLEXCAN_HAL_EnableRxFifo (CAN_Type ∗base, uint32_t numOfFilters)

  Enables the Rx FIFO.
- void FLEXCAN_HAL_DisableRxFifo (CAN_Type ∗base)

  Disables the Rx FIFO.
- static bool FLEXCAN_HAL_IsRxFifoEnabled (const CAN_Type ∗base)

  Checks if Rx FIFO is enabled.
- void FLEXCAN_HAL_SetRxFifoFilterNum (CAN_Type ∗base, uint32_t number)

  Sets the number of the Rx FIFO filters.
- status_t FLEXCAN_HAL_SetMaxMsgBuffNum (CAN_Type ∗base, uint32_t maxMsgBuffNum)

Sets the maximum number of Message Buffers.

- void FLEXCAN_HAL_SetRxFifoFilter (CAN_Type ∗base, flexcan_rx_fifo_id_element_format_t idFormat, const flexcan_id_table_t ∗idFilterTable)

  Sets the FlexCAN Rx FIFO fields.
- void FLEXCAN_HAL_ReadRxFifo (const CAN_Type ∗base, flexcan_msgbuff_t ∗rxFifo)

  Gets the FlexCAN Rx FIFO data.

Interrupts

- status_t FLEXCAN_HAL_SetMsgBuffIntCmd (CAN_Type ∗base, uint32_t msgBuffIdx, bool enable)

  Enables/Disables the FlexCAN Message Buffer interrupt.
- void FLEXCAN_HAL_SetErrIntCmd (CAN_Type ∗base, flexcan_int_type_t errType, bool enable)

  Enables error interrupt of the FlexCAN module.

Status

- static uint32_t FLEXCAN_HAL_GetFreezeAck (const CAN_Type ∗base)

  Gets the value of FlexCAN freeze ACK.
- uint8_t FLEXCAN_HAL_GetMsgBuffIntStatusFlag (const CAN_Type ∗base, uint32_t msgBuffIdx)

  Gets the individual FlexCAN MB interrupt flag.
- static uint32_t FLEXCAN_HAL_GetAllMsgBuffIntStatusFlag (const CAN_Type ∗base)

  Gets all FlexCAN Message Buffer interrupt flags.
- static void FLEXCAN_HAL_ClearMsgBuffIntStatusFlag (CAN_Type ∗base, uint32_t flag)

  Clears the interrupt flag of the message buffers.
- void FLEXCAN_HAL_GetErrCounter (const CAN_Type ∗base, flexcan_buserr_counter_t ∗errCount)

  Gets the transmit error counter and receives the error counter.
- static uint32_t FLEXCAN_HAL_GetErrStatus (const CAN_Type ∗base)

  Gets error and status.
- void FLEXCAN_HAL_ClearErrIntStatusFlag (CAN_Type ∗base)

  Clears all other interrupts in ERRSTAT register (Error, Busoff, Wakeup).

Mask

- void FLEXCAN_HAL_SetRxMaskType (CAN_Type ∗base, flexcan_rx_mask_type_t type)

  Sets the Rx masking type.
- void FLEXCAN_HAL_SetRxFifoGlobalStdMask (CAN_Type ∗base, uint32_t stdMask)

  Sets the FlexCAN RX FIFO global standard mask.
- void FLEXCAN_HAL_SetRxFifoGlobalExtMask (CAN_Type ∗base, uint32_t extMask)

  Sets the FlexCAN Rx FIFO global extended mask.
- status_t FLEXCAN_HAL_SetRxIndividualStdMask (CAN_Type ∗base, uint32_t msgBuffIdx, uint32_t std↩ Mask)

  Sets the FlexCAN Rx individual standard mask for ID filtering in the Rx MBs and the Rx FIFO.
- status_t FLEXCAN_HAL_SetRxIndividualExtMask (CAN_Type ∗base, uint32_t msgBuffIdx, uint32_t ext↩ Mask)

  Sets the FlexCAN Rx individual extended mask for ID filtering in the Rx Message Buffers and the Rx FIFO.
- void FLEXCAN_HAL_SetRxMsgBuffGlobalStdMask (CAN_Type ∗base, uint32_t stdMask)

  Sets the FlexCAN Rx Message Buffer global standard mask.
- void FLEXCAN_HAL_SetRxMsgBuff14StdMask (CAN_Type ∗base, uint32_t stdMask)

  Sets the FlexCAN RX Message Buffer BUF14 standard mask.

- void FLEXCAN_HAL_SetRxMsgBuff15StdMask (CAN_Type ∗base, uint32_t stdMask)

  Sets the FlexCAN Rx Message Buffer BUF15 standard mask.
- void FLEXCAN_HAL_SetRxMsgBuffGlobalExtMask (CAN_Type ∗base, uint32_t extMask)

  Sets the FlexCAN RX Message Buffer global extended mask.
- void FLEXCAN_HAL_SetRxMsgBuff14ExtMask (CAN_Type ∗base, uint32_t extMask)

  Sets the FlexCAN RX Message Buffer BUF14 extended mask.
- void FLEXCAN_HAL_SetRxMsgBuff15ExtMask (CAN_Type ∗base, uint32_t extMask)

  Sets the FlexCAN RX MB BUF15 extended mask.
- static uint32_t FLEXCAN_HAL_GetRxFifoHitIdAcceptanceFilter (const CAN_Type ∗base)

  Gets the FlexCAN ID acceptance filter hit indicator on Rx FIFO.
- void FLEXCAN_HAL_SetStuffBitCount (CAN_Type ∗base, bool enable)

  Enables/Disables the Stuff Bit Count for CAN FD frames.
- void FLEXCAN_HAL_SetSelfReception (CAN_Type ∗base, bool enable)

  Enables/Disables the Self Reception feature.
- status_t FLEXCAN_HAL_SetRxFifoDMA (CAN_Type ∗base, bool enable)

  Enables/Disables the DMA support for RxFIFO.
- void FLEXCAN_HAL_SetTDCOffset (CAN_Type ∗base, bool enable, uint8_t offset)

  Enables/Disables the Transceiver Delay Compensation feature and sets the Transceiver Delay Compensation Offset (offset value to be added to the measured transceiver's loop delay in order to define the position of the delayed comparison point when bit rate switching is active).
- static uint8_t FLEXCAN_HAL_GetTDCValue (const CAN_Type ∗base)

  Gets the value of the Transceiver Delay Compensation.
- static bool FLEXCAN_HAL_GetTDCFail (const CAN_Type ∗base)

  Gets the value of the TDC Fail flag.
- static void FLEXCAN_HAL_ClearTDCFail (CAN_Type ∗base)

  Clears the TDC Fail flag.
- void FLEXCAN_HAL_ConfigPN (CAN_Type ∗base, const flexcan_pn_config_t ∗pnConfig)

  Configures the Pretended Networking mode.
- void FLEXCAN_HAL_GetWMB (const CAN_Type ∗base, uint8_t wmbIndex, flexcan_msgbuff_t ∗wmb)

  Extracts one of the frames which triggered the wake up event.
- static void FLEXCAN_HAL_SetPN (CAN_Type ∗base, bool enable)

  Enables/Disables the Pretended Networking mode.

3.34.2 Data Structure Documentation

3.34.2.1 struct flexcan_id_table_t

FlexCAN RX FIFO ID filter table structure Implements : flexcan_id_table_t_Class.

Data Fields

- bool isRemoteFrame
- bool isExtendedFrame
- uint32_t ∗ idFilter

Field Documentation

3.34.2.1.1 idFilter

```
uint32_t* idFilter
```

Rx FIFO ID filter elements

3.34.2.1.2 isExtendedFrame

```
bool isExtendedFrame
```

Extended frame

3.34.2.1.3 isRemoteFrame

```
bool isRemoteFrame
```

Remote frame

3.34.2.2 struct flexcan_buserr_counter_t

FlexCAN bus error counters Implements : flexcan_buserr_counter_t_Class.

Data Fields

- uint16_t txerr
- uint16_t rxerr

Field Documentation

3.34.2.2.1 rxerr

```
uint16_t rxerr
```

Receive error counter

3.34.2.2.2 txerr

```
uint16_t txerr
```

Transmit error counter

3.34.2.3 struct flexcan_msgbuff_code_status_t

FlexCAN Message Buffer code and status for transmit and receive Implements : flexcan_msgbuff_code_status_t↩
_Class.

Data Fields

- uint32_t code
- flexcan_msgbuff_id_type_t msgIdType
- uint32_t dataLen
- bool fd_enable
- uint8_t fd_padding
- bool enable_brs

Field Documentation

3.34.2.3.1 code

```
uint32_t code
```

MB code for TX or RX buffers.

3.34.2.3.2 dataLen

```
uint32_t dataLen
```

Length of Data in Bytes

3.34.2.3.3 enable_brs

```
bool enable_brs
```

3.34.2.3.4 fd_enable

```
bool fd_enable
```

3.34.2.3.5 fd_padding

```
uint8_t fd_padding
```

3.34.2.3.6 msgIdType

```
flexcan_msgbuff_id_type_t msgIdType
```

Defined by flexcan_mb_code_rx_t and flexcan_mb_code_tx_t Type of message ID (standard or extended)

3.34.2.4 struct flexcan_msgbuff_t

FlexCAN message buffer structure Implements : flexcan_msgbuff_t_Class.

Data Fields

- uint32_t cs
- uint32_t msgId
- uint8_t data [64]
- uint8_t dataLen

Field Documentation

3.34.2.4.1 cs

```
uint32_t cs
```

Code and Status

3.34.2.4.2 data

```
uint8_t data[64]
```

Bytes of the FlexCAN message

3.34.2.4.3 dataLen

```
uint8_t dataLen
```

Length of data in bytes

3.34.2.4.4 msgId

```
uint32_t msgId
```

Message Buffer ID

3.34.2.5 struct flexcan_time_segment_t

FlexCAN timing related structures Implements : flexcan_time_segment_t_Class.

Data Fields

- uint32_t propSeg
- uint32_t phaseSeg1
- uint32_t phaseSeg2
- uint32_t preDivider
- uint32_t rJumpwidth

Field Documentation

3.34.2.5.1 phaseSeg1

```
uint32_t phaseSeg1
```

Phase segment 1

3.34.2.5.2 phaseSeg2

```
uint32_t phaseSeg2
```

Phase segment 2

3.34.2.5.3 preDivider

```
uint32_t preDivider
```

Clock pre divider

3.34.2.5.4 propSeg

```
uint32_t propSeg
```

Propagation segment

3.34.2.5.5 rJumpwidth

```
uint32_t rJumpwidth
```

Resync jump width

3.34.2.6 struct flexcan_pn_id_filter_t

Pretended Networking ID filter.

Data Fields

- bool extendedId
- bool remoteFrame
- uint32_t id

Field Documentation

3.34.2.6.1 extendedId

```
bool extendedId
```

Specifies if the ID is standard or extended.

3.34.2.6.2 id

```
uint32_t id
```

Specifies the ID value.

3.34.2.6.3 remoteFrame

```
bool remoteFrame
```

Specifies if the frame is standard or remote.

3.34.2.7 struct flexcan_pn_payload_filter_t

Pretended Networking payload filter.

Data Fields

- uint8_t dlcLow
- uint8_t dlcHigh
- uint8_t payload1 [8U]
- uint8_t payload2 [8U]

Field Documentation

3.34.2.7.1 dlcHigh

```
uint8_t dlcHigh
```

Specifies the upper limit of the payload size.

3.34.2.7.2 dlcLow

```
uint8_t dlcLow
```

Specifies the lower limit of the payload size.

3.34.2.7.3 payload1

```
uint8_t payload1[8U]
```

Specifies the payload to be matched (for MATCH_EXACT), the lower limit (for MATCH_GEQ and MATCH_RANGE) or the upper limit (for MATCH_LEQ).

3.34.2.7.4 payload2

```
uint8_t payload2[8U]
```

Specifies the mask (for MATCH_EXACT) or the upper limit (for MATCH_RANGE).

3.34.2.8 struct flexcan_pn_config_t

Pretended Networking configuration structure Implements : flexcan_pn_config_t_Class.

Data Fields

- bool wakeUpTimeout
- bool wakeUpMatch
- uint16_t numMatches
- uint16_t matchTimeout
- flexcan_pn_filter_combination_t filterComb
- flexcan_pn_id_filter_t idFilter1
- flexcan_pn_id_filter_t idFilter2
- flexcan_pn_filter_selection_t idFilterType
- flexcan_pn_filter_selection_t payloadFilterType
- flexcan_pn_payload_filter_t payloadFilter

Field Documentation

3.34.2.8.1 filterComb

```
flexcan_pn_filter_combination_t filterComb
```

Defines the filtering scheme used.

3.34.2.8.2 idFilter1

```
flexcan_pn_id_filter_t idFilter1
```

The configuration of the first ID filter (match exact / lower limit / upper limit).

3.34.2.8.3 idFilter2

```
flexcan_pn_id_filter_t idFilter2
```

The configuration of the second ID filter (mask / upper limit).

3.34.2.8.4 idFilterType

```
flexcan_pn_filter_selection_t idFilterType
```

Defines the ID filtering scheme.

3.34.2.8.5 matchTimeout

```
uint16_t matchTimeout
```

Defines a timeout value that generates a wake up event if wakeUpTimeout is true.

3.34.2.8.6 numMatches

```
uint16_t numMatches
```

The number of matches needed before generating a wake up event.

3.34.2.8.7 payloadFilter

```
flexcan_pn_payload_filter_t payloadFilter
```

The configuration of the payload filter.

3.34.2.8.8 payloadFilterType

```
flexcan_pn_filter_selection_t payloadFilterType
```

Defines the payload filtering scheme.

3.34.2.8.9 wakeUpMatch

```
bool wakeUpMatch
```

Specifies if an wake up event is triggered on match.

3.34.2.8.10 wakeUpTimeout

```
bool wakeUpTimeout
```

Specifies if an wake up event is triggered on timeout.

3.34.3 Macro Definition Documentation

3.34.3.1 CAN_CS_CODE_MASK

```
#define CAN_CS_CODE_MASK 0xF000000u
```

3.34.3.2 CAN_CS_CODE_SHIFT

```
#define CAN_CS_CODE_SHIFT 24
```

3.34.3.3 CAN_CS_CODE_WIDTH

```
#define CAN_CS_CODE_WIDTH 4
```

3.34.3.4 CAN_CS_DLC_MASK

```
#define CAN_CS_DLC_MASK 0xF0000u
```

3.34.3.5 CAN_CS_DLC_SHIFT

```
#define CAN_CS_DLC_SHIFT 16
```

3.34.3.6 CAN_CS_DLC_WIDTH

```
#define CAN_CS_DLC_WIDTH 4
```

3.34.3.7 CAN_CS_IDE_MASK

```
#define CAN_CS_IDE_MASK 0x200000u
```

3.34.3.8 CAN_CS_IDE_SHIFT

```
#define CAN_CS_IDE_SHIFT 21
```

3.34.3.9 CAN_CS_IDE_WIDTH

```
#define CAN_CS_IDE_WIDTH 1
```

3.34.3.10 CAN_CS_RTR_MASK

```
#define CAN_CS_RTR_MASK 0x100000u
```

3.34.3.11 CAN_CS_RTR_SHIFT

```
#define CAN_CS_RTR_SHIFT 20
```

3.34.3.12 CAN_CS_RTR_WIDTH

```
#define CAN_CS_RTR_WIDTH 1
```

3.34.3.13 CAN_CS_SRR_MASK

```
#define CAN_CS_SRR_MASK 0x400000u
```

3.34.3.14 CAN_CS_SRR_SHIFT

```
#define CAN_CS_SRR_SHIFT 22
```

3.34.3.15 CAN_CS_SRR_WIDTH

```
#define CAN_CS_SRR_WIDTH 1
```

3.34.3.16 CAN_CS_TIME_STAMP_MASK

```
#define CAN_CS_TIME_STAMP_MASK 0xFFFFu
```

3.34.3.17 CAN_CS_TIME_STAMP_SHIFT

```
#define CAN_CS_TIME_STAMP_SHIFT 0
```

3.34.3.18 CAN_CS_TIME_STAMP_WIDTH

```
#define CAN_CS_TIME_STAMP_WIDTH 16
```

3.34.3.19 CAN_ID_EXT_MASK

```
#define CAN_ID_EXT_MASK 0x3FFFFu
```

3.34.3.20 CAN_ID_EXT_SHIFT

```
#define CAN_ID_EXT_SHIFT 0
```

3.34.3.21 CAN_ID_EXT_WIDTH

```
#define CAN_ID_EXT_WIDTH 18
```

3.34.3.22 CAN_ID_PRIO_MASK

```
#define CAN_ID_PRIO_MASK 0xE0000000u
```

3.34.3.23 CAN_ID_PRIO_SHIFT

```
#define CAN_ID_PRIO_SHIFT 29
```

3.34.3.24 CAN_ID_PRIO_WIDTH

```
#define CAN_ID_PRIO_WIDTH 3
```

3.34.3.25 CAN_ID_STD_MASK

```
#define CAN_ID_STD_MASK 0x1FFC0000u
```

3.34.3.26 CAN_ID_STD_SHIFT

```
#define CAN_ID_STD_SHIFT 18
```

3.34.3.27 CAN_ID_STD_WIDTH

```
#define CAN_ID_STD_WIDTH 11
```

3.34.3.28 CAN_MB_BRS_MASK

```
#define CAN_MB_BRS_MASK 0x40000000u
```

3.34.3.29 CAN_MB_EDL_MASK

```
#define CAN_MB_EDL_MASK 0x80000000u
```

3.34.4 Enumeration Type Documentation

3.34.4.1 anonymous enum

```
anonymous enum
```

FlexCAN message buffer CODE for Rx buffers.

Enumerator

FLEXCAN_RX_INACTIVE	MB is not active.
FLEXCAN_RX_FULL	MB is full.
FLEXCAN_RX_EMPTY	MB is active and empty.
FLEXCAN_RX_OVERRUN	MB is overwritten into a full buffer.
FLEXCAN_RX_BUSY	FlexCAN is updating the contents of the MB.
FLEXCAN_RX_RANSWER	The CPU must not access the MB. A frame was configured to recognize a Remote Request Frame
FLEXCAN_RX_NOT_USED	and transmit a Response Frame in return. Not used

3.34.4.2 anonymous enum

```
anonymous enum
```

FlexCAN message buffer CODE FOR Tx buffers.

Enumerator

FLEXCAN_TX_INACTIVE	MB is not active.
FLEXCAN_TX_ABORT	MB is aborted.

Enumerator

FLEXCAN_TX_DATA	MB is a TX Data Frame(MB RTR must be 0).
FLEXCAN_TX_REMOTE	MB is a TX Remote Request Frame (MB RTR must be 1).
FLEXCAN_TX_TANSWER	MB is a TX Response Request Frame from.
FLEXCAN_TX_NOT_USED	an incoming Remote Request Frame. Not used

3.34.4.3 anonymous enum

`anonymous enum`

FlexCAN message buffer transmission types.

Enumerator

FLEXCAN_MB_STATUS_TYPE_TX	Transmit MB
FLEXCAN_MB_STATUS_TYPE_TX_REMOTE	Transmit remote request MB
FLEXCAN_MB_STATUS_TYPE_RX	Receive MB
FLEXCAN_MB_STATUS_TYPE_RX_REMOTE	Receive remote request MB
FLEXCAN_MB_STATUS_TYPE_RX_TX_REMOTE	FlexCAN remote frame receives remote request and transmits MB.

3.34.4.4 flexcan_clk_source_t

`enum flexcan_clk_source_t`

FlexCAN clock source Implements : flexcan_clk_source_t_Class.

Enumerator

FLEXCAN_CLK_SOURCE_SOSCDIV2	Clock divider 2 for System OSC
FLEXCAN_CLK_SOURCE_SYS	Sys clock

3.34.4.5 flexcan_fd_payload_size_t

`enum flexcan_fd_payload_size_t`

FlexCAN payload sizes Implements : flexcan_fd_payload_size_t_Class.

Enumerator

FLEXCAN_PAYLOAD_SIZE_8	FlexCAN message buffer payload size in bytes
FLEXCAN_PAYLOAD_SIZE_16	FlexCAN message buffer payload size in bytes
FLEXCAN_PAYLOAD_SIZE_32	FlexCAN message buffer payload size in bytes
FLEXCAN_PAYLOAD_SIZE_64	FlexCAN message buffer payload size in bytes

3.34.4.6 flexcan_int_type_t

```
enum flexcan_int_type_t
```

FlexCAN error interrupt types Implements : flexcan_int_type_t_Class.

Enumerator

FLEXCAN_INT_RX_WARNING	RX warning interrupt
FLEXCAN_INT_TX_WARNING	TX warning interrupt
FLEXCAN_INT_ERR	Error interrupt
FLEXCAN_INT_BUSOFF	Bus off interrupt

3.34.4.7 flexcan_msgbuff_id_type_t

```
enum flexcan_msgbuff_id_type_t
```

FlexCAN Message Buffer ID type Implements : flexcan_msgbuff_id_type_t_Class.

Enumerator

FLEXCAN_MSG_ID_STD	Standard ID
FLEXCAN_MSG_ID_EXT	Extended ID

3.34.4.8 flexcan_operation_modes_t

```
enum flexcan_operation_modes_t
```

FlexCAN operation modes Implements : flexcan_operation_modes_t_Class.

Enumerator

FLEXCAN_NORMAL_MODE	Normal mode or user mode
FLEXCAN_LISTEN_ONLY_MODE	Listen-only mode
FLEXCAN_LOOPBACK_MODE	Loop-back mode
FLEXCAN_FREEZE_MODE	Freeze mode
FLEXCAN_DISABLE_MODE	Module disable mode

3.34.4.9 flexcan_pn_filter_combination_t

```
enum flexcan_pn_filter_combination_t
```

Pretended Networking filtering combinations.

Enumerator

FLEXCAN_FILTER_ID	Message ID filtering only
FLEXCAN_FILTER_ID_PAYLOAD	Message ID filtering and payload filtering
FLEXCAN_FILTER_ID_NTIMES	Message ID filtering occurring a specified number of times
FLEXCAN_FILTER_ID_PAYLOAD_NTIMES	Message ID filtering and payload filtering a specified number of times

3.34.4.10 flexcan_pn_filter_selection_t

```
enum flexcan_pn_filter_selection_t
```

Pretended Networking matching schemes.

Enumerator

FLEXCAN_FILTER_MATCH_EXACT	Match an exact target value.
FLEXCAN_FILTER_MATCH_GEQ	Match greater than or equal to a specified target value.
FLEXCAN_FILTER_MATCH_LEQ	Match less than or equal to a specified target value.
FLEXCAN_FILTER_MATCH_RANGE	Match inside a range, greater than or equal to a specified lower limit and smaller than or equal a specified upper limit.

3.34.4.11 flexcan_rx_fifo_id_element_format_t

```
enum flexcan_rx_fifo_id_element_format_t
```

ID formats for RxFIFO Implements : flexcan_rx_fifo_id_element_format_t_Class.

Enumerator

FLEXCAN_RX_FIFO_ID_FORMAT_A	One full ID (standard and extended) per ID Filter Table
FLEXCAN_RX_FIFO_ID_FORMAT_B	element. Two full standard IDs or two partial 14-bit (standard and
FLEXCAN_RX_FIFO_ID_FORMAT_C	extended) IDs per ID Filter Table element. Four partial 8-bit Standard IDs per ID Filter Table
FLEXCAN_RX_FIFO_ID_FORMAT_D	element. All frames rejected.

3.34.4.12 flexcan_rx_mask_type_t

```
enum flexcan_rx_mask_type_t
```

FlexCAN RX mask type. Implements : flexcan_rx_mask_type_t_Class.

Enumerator

FLEXCAN_RX_MASK_GLOBAL	Rx global mask
FLEXCAN_RX_MASK_INDIVIDUAL	Rx individual mask

3.34.5 Function Documentation

3.34.5.1 FLEXCAN_HAL_ClearErrIntStatusFlag()

```
void FLEXCAN_HAL_ClearErrIntStatusFlag (
            CAN_Type * base )
```

Clears all other interrupts in ERRSTAT register (Error, Busoff, Wakeup).

Parameters

base	The FlexCAN base address

3.34.5.2 FLEXCAN_HAL_ClearMsgBuffIntStatusFlag()

```
static void FLEXCAN_HAL_ClearMsgBuffIntStatusFlag (
            CAN_Type * base,
            uint32_t flag )  [inline], [static]
```

Clears the interrupt flag of the message buffers.

Parameters

base	The FlexCAN base address
flag	The value to be written to the interrupt flag1 register. Implements : FLEXCAN_HAL_ClearMsgBuffIntStatusFlag_Activity

3.34.5.3 FLEXCAN_HAL_ClearTDCFail()

```
static void FLEXCAN_HAL_ClearTDCFail (
            CAN_Type * base )  [inline], [static]
```

Clears the TDC Fail flag.

Parameters

base	The FlexCAN base address Implements : FLEXCAN_HAL_ClearTDCFail_Activity

3.34.5.4 FLEXCAN_HAL_ConfigPN()

```
void FLEXCAN_HAL_ConfigPN (
            CAN_Type * base,
            const flexcan_pn_config_t * pnConfig )
```

Configures the Pretended Networking mode.

Parameters

base	The FlexCAN base address
pnConfig	The pretended networking configuration

3.34.5.5 FLEXCAN_HAL_Disable()

```
void FLEXCAN_HAL_Disable (
            CAN_Type * base )
```

Disables FlexCAN controller.

Parameters

base	The FlexCAN base address

3.34.5.6 FLEXCAN_HAL_DisableRxFifo()

```
void FLEXCAN_HAL_DisableRxFifo (
             CAN_Type * base )
```

Disables the Rx FIFO.

Parameters

base	The FlexCAN base address

3.34.5.7 FLEXCAN_HAL_Enable()

```
void FLEXCAN_HAL_Enable (
             CAN_Type * base )
```

Enables FlexCAN controller.

Parameters

base	The FlexCAN base address

3.34.5.8 FLEXCAN_HAL_EnableRxFifo()

```
status_t FLEXCAN_HAL_EnableRxFifo (
             CAN_Type * base,
             uint32_t numOfFilters )
```

Enables the Rx FIFO.

Parameters

base	The FlexCAN base address
numOfFilters	The number of Rx FIFO filters

Returns

The status of the operation

Return values

STATUS_SUCCESS	RxFIFO was successfully enabled
STATUS_ERROR	RxFIFO could not be enabled (e.g. the FD feature was enabled, and these two features are not compatible)

3.34.5.9 FLEXCAN_HAL_EnterFreezeMode()

```
void FLEXCAN_HAL_EnterFreezeMode (
             CAN_Type * base )
```

Freezes the FlexCAN module.

Parameters

base	The FlexCAN base address

3.34.5.10 FLEXCAN_HAL_ExitFreezeMode()

```
void FLEXCAN_HAL_ExitFreezeMode (
             CAN_Type * base )
```

Un freezes the FlexCAN module.

Parameters

base	The FlexCAN base address

3.34.5.11 FLEXCAN_HAL_ExitOperationMode()

```
void FLEXCAN_HAL_ExitOperationMode (
             CAN_Type * base,
             flexcan_operation_modes_t mode )
```

Exit operation mode.

Parameters

base	The FlexCAN base address
mode	Exit An operation mode

3.34.5.12 FLEXCAN_HAL_GetAllMsgBuffIntStatusFlag()

```
static uint32_t FLEXCAN_HAL_GetAllMsgBuffIntStatusFlag (
             const CAN_Type * base )  [inline], [static]
```

Gets all FlexCAN Message Buffer interrupt flags.

Parameters

base	The FlexCAN base address

Returns

all MB interrupt flags Implements : FLEXCAN_HAL_GetAllMsgBuffIntStatusFlag_Activity

3.34.5.13 FLEXCAN_HAL_GetClock()

```
static bool FLEXCAN_HAL_GetClock (
             const CAN_Type * base )  [inline], [static]
```

Reads the clock source for FlexCAN Protocol Engine (PE).

Parameters

base	The FlexCAN base address

Returns

0: if clock source is oscillator clock, 1: if clock source is peripheral clock Implements : FLEXCAN_HAL_Get←Clock_Activity

3.34.5.14 FLEXCAN_HAL_GetErrCounter()

```
void FLEXCAN_HAL_GetErrCounter (
             const CAN_Type * base,
             flexcan_buserr_counter_t * errCount )
```

Gets the transmit error counter and receives the error counter.

Parameters

base	The FlexCAN base address
errCount	Transmit error counter and receive error counter

3.34.5.15 FLEXCAN_HAL_GetErrStatus()

```
static uint32_t FLEXCAN_HAL_GetErrStatus (
             const CAN_Type * base )  [inline], [static]
```

Gets error and status.

Parameters

base	The FlexCAN base address

Returns

The current error and status Implements : FLEXCAN_HAL_GetErrStatus_Activity

3.34.5.16 FLEXCAN_HAL_GetFreezeAck()

```
static uint32_t FLEXCAN_HAL_GetFreezeAck (
             const CAN_Type * base )  [inline], [static]
```

Gets the value of FlexCAN freeze ACK.

Parameters

base	The FlexCAN base address

Returns

freeze ACK state (1-freeze mode, 0-not in freeze mode). Implements : FLEXCAN_HAL_GetFreezeAck_↩ Activity

3.34.5.17 FLEXCAN_HAL_GetMsgBuff()

```
status_t FLEXCAN_HAL_GetMsgBuff (
             CAN_Type * base,
             uint32_t msgBuffIdx,
             flexcan_msgbuff_t * msgBuff )
```

Gets the FlexCAN message buffer fields.

Parameters

base	The FlexCAN base address
msgBuffIdx	Index of the message buffer
msgBuff	The fields of the message buffer

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of the message buffer is invalid

3.34.5.18 FLEXCAN_HAL_GetMsgBuffIntStatusFlag()

```
uint8_t FLEXCAN_HAL_GetMsgBuffIntStatusFlag (
             const CAN_Type * base,
             uint32_t msgBuffIdx )
```

Gets the individual FlexCAN MB interrupt flag.

Parameters

base	The FlexCAN base address
msgBuffIdx	Index of the message buffer

Returns

the individual Message Buffer interrupt flag (0 and 1 are the flag value)

3.34.5.19 FLEXCAN_HAL_GetPayloadSize()

```
uint8_t FLEXCAN_HAL_GetPayloadSize (
             const CAN_Type * base )
```

Gets the payload size of the MBs.

Parameters

base	The FlexCAN base address

Returns

The payload size in bytes

3.34.5.20 FLEXCAN_HAL_GetRxFifoHitIdAcceptanceFilter()

```
static uint32_t FLEXCAN_HAL_GetRxFifoHitIdAcceptanceFilter (
            const CAN_Type * base )  [inline], [static]
```

Gets the FlexCAN ID acceptance filter hit indicator on Rx FIFO.

Parameters

base	The FlexCAN base address

Returns

RX FIFO information Implements : FLEXCAN_HAL_GetRxFifoHitIdAcceptanceFilter_Activity

3.34.5.21 FLEXCAN_HAL_GetTDCFail()

```
static bool FLEXCAN_HAL_GetTDCFail (
            const CAN_Type * base )  [inline], [static]
```

Gets the value of the TDC Fail flag.

Parameters

base	The FlexCAN base address

Returns

If true, indicates that the TDC mechanism is out of range, unable to compensate the transceiver's loop delay and successfully compare the delayed received bits to the transmitted ones. Implements : FLEXCAN_HAL↩ _GetTDCFail_Activity

3.34.5.22 FLEXCAN_HAL_GetTDCValue()

```
static uint8_t FLEXCAN_HAL_GetTDCValue (
            const CAN_Type * base )  [inline], [static]
```

Gets the value of the Transceiver Delay Compensation.

Parameters

base	The FlexCAN base address

Returns

The value of the transceiver loop delay measured from the transmitted EDL to R0 transition edge to the respective received one added to the TDCOFF value specified by FLEXCAN_HAL_SetTDCOffset. Implements : FLEXCAN_HAL_GetTDCValue_Activity

3.34.5.23 FLEXCAN_HAL_GetTimeSegments()

```
void FLEXCAN_HAL_GetTimeSegments (
            const CAN_Type * base,
            flexcan_time_segment_t * timeSeg )
```

Gets the FlexCAN time segments to calculate the bit rate.

Parameters

base	The FlexCAN base address
timeSeg	FlexCAN time segments read for bit rate

3.34.5.24 FLEXCAN_HAL_GetWMB()

```
void FLEXCAN_HAL_GetWMB (
            const CAN_Type * base,
            uint8_t wmbIndex,
            flexcan_msgbuff_t * wmb )
```

Extracts one of the frames which triggered the wake up event.

Parameters

base	The FlexCAN base address
wmbIndex	The index of the message buffer to be extracted.
wmb	Pointer to the message buffer structure where the frame will be saved.

3.34.5.25 FLEXCAN_HAL_Init()

```
void FLEXCAN_HAL_Init (
            CAN_Type * base )
```

Initializes the FlexCAN controller.

Parameters

base	The FlexCAN base address

3.34.5.26 FLEXCAN_HAL_IsFDEnabled()

```
bool FLEXCAN_HAL_IsFDEnabled (
            const CAN_Type * base )
```

Checks if the Flexible Data rate feature is enabled.

Parameters

base	The FlexCAN base address

Returns

true if enabled; false if disabled

3.34.5.27 FLEXCAN_HAL_IsRxFifoEnabled()

```
static bool FLEXCAN_HAL_IsRxFifoEnabled (
            const CAN_Type * base )  [inline], [static]
```

Checks if Rx FIFO is enabled.

Parameters

base	The FlexCAN base address

Returns

RxFifo status (true = enabled / false = disabled) Implements : FLEXCAN_HAL_IsRxFifoEnabled_Activity

3.34.5.28 FLEXCAN_HAL_LockRxMsgBuff()

```
status_t FLEXCAN_HAL_LockRxMsgBuff (
            CAN_Type * base,
            uint32_t msgBuffIdx )
```

Locks the FlexCAN Rx message buffer.

Parameters

base	The FlexCAN base address
msgBuffIdx	Index of the message buffer

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of the message buffer is invalid

3.34.5.29 FLEXCAN_HAL_ReadRxFifo()

```
void FLEXCAN_HAL_ReadRxFifo (
            const CAN_Type * base,
            flexcan_msgbuff_t * rxFifo )
```

Gets the FlexCAN Rx FIFO data.

Parameters

base	The FlexCAN base address
rxFifo	The FlexCAN receive FIFO data

3.34.5.30 FLEXCAN_HAL_SelectClock()

```
void FLEXCAN_HAL_SelectClock (
            CAN_Type * base,
            flexcan_clk_source_t clk )
```

Selects the clock source for FlexCAN.

Parameters

base	The FlexCAN base address
clk	The FlexCAN clock source

3.34.5.31 FLEXCAN_HAL_SetErrIntCmd()

```
void FLEXCAN_HAL_SetErrIntCmd (
            CAN_Type * base,
            flexcan_int_type_t errType,
            bool enable )
```

Enables error interrupt of the FlexCAN module.

Parameters

base	The FlexCAN base address
errType	The interrupt type
enable	choose enable or disable

3.34.5.32 FLEXCAN_HAL_SetFDEnabled()

```
void FLEXCAN_HAL_SetFDEnabled (
            CAN_Type * base,
            bool enable )
```

Enables/Disables Flexible Data rate (if supported).

Parameters

base	The FlexCAN base address
enable	true to enable; false to disable

3.34.5.33 FLEXCAN_HAL_SetMaxMsgBuffNum()

```
status_t FLEXCAN_HAL_SetMaxMsgBuffNum (
            CAN_Type * base,
            uint32_t maxMsgBuffNum )
```

Sets the maximum number of Message Buffers.

Parameters

base	The FlexCAN base address
maxMsgBuffNum	Maximum number of message buffers

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of the message buffer is invalid

3.34.5.34 FLEXCAN_HAL_SetMsgBuffIntCmd()

```
status_t FLEXCAN_HAL_SetMsgBuffIntCmd (
            CAN_Type * base,
            uint32_t msgBuffIdx,
            bool enable )
```

Enables/Disables the FlexCAN Message Buffer interrupt.

Parameters

base	The FlexCAN base address
msgBuffIdx	Index of the message buffer
enable	choose enable or disable

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of the message buffer is invalid

3.34.5.35 FLEXCAN_HAL_SetOperationMode()

```
void FLEXCAN_HAL_SetOperationMode (
            CAN_Type * base,
            flexcan_operation_modes_t mode )
```

Set operation mode.

Parameters

base	The FlexCAN base address
mode	Set an operation mode

3.34.5.36 FLEXCAN_HAL_SetPayloadSize()

```
void FLEXCAN_HAL_SetPayloadSize (
            CAN_Type * base,
            flexcan_fd_payload_size_t payloadSize )
```

Sets the payload size of the MBs.

Parameters

base	The FlexCAN base address
payloadSize	The payload size

3.34.5.37 FLEXCAN_HAL_SetPN()

```
static void FLEXCAN_HAL_SetPN (
            CAN_Type * base,
            bool enable ) [inline], [static]
```

Enables/Disables the Pretended Networking mode.

Parameters

base	The FlexCAN base address
enable	Enable/Disable Pretending Networking Implements : FLEXCAN_HAL_SetPN_Activity

3.34.5.38 FLEXCAN_HAL_SetRxFifoDMA()

```
status_t FLEXCAN_HAL_SetRxFifoDMA (
            CAN_Type * base,
            bool enable )
```

Enables/Disables the DMA support for RxFIFO.

Parameters

base	The FlexCAN base address
enable	Enable/Disable DMA support

Returns

STATUS_SUCCESS if successfully enabled; STATUS_ERROR if not enabled (due to the RxFIFO feature being disabled).

3.34.5.39 FLEXCAN_HAL_SetRxFifoFilter()

```
void FLEXCAN_HAL_SetRxFifoFilter (
            CAN_Type * base,
            flexcan_rx_fifo_id_element_format_t idFormat,
            const flexcan_id_table_t * idFilterTable )
```

Sets the FlexCAN Rx FIFO fields.

Parameters

base	The FlexCAN base address
idFormat	The format of the Rx FIFO ID Filter Table Elements
idFilterTable	The ID filter table elements which contain RTR bit, IDE bit, and RX message ID.

3.34.5.40 FLEXCAN_HAL_SetRxFifoFilterNum()

```
void FLEXCAN_HAL_SetRxFifoFilterNum (
            CAN_Type * base,
            uint32_t number )
```

Sets the number of the Rx FIFO filters.

Parameters

base	The FlexCAN base address
number	The number of Rx FIFO filters

3.34.5.41 FLEXCAN_HAL_SetRxFifoGlobalExtMask()

```
void FLEXCAN_HAL_SetRxFifoGlobalExtMask (
            CAN_Type * base,
            uint32_t extMask )
```

Sets the FlexCAN Rx FIFO global extended mask.

Parameters

base	The FlexCAN base address
extMask	Extended mask

3.34.5.42 FLEXCAN_HAL_SetRxFifoGlobalStdMask()

```
void FLEXCAN_HAL_SetRxFifoGlobalStdMask (
            CAN_Type * base,
            uint32_t stdMask )
```

Sets the FlexCAN RX FIFO global standard mask.

Parameters

base	The FlexCAN base address
stdMask	Standard mask

3.34.5.43 FLEXCAN_HAL_SetRxIndividualExtMask()

```
status_t FLEXCAN_HAL_SetRxIndividualExtMask (
            CAN_Type * base,
            uint32_t msgBuffIdx,
            uint32_t extMask )
```

Sets the FlexCAN Rx individual extended mask for ID filtering in the Rx Message Buffers and the Rx FIFO.

Parameters

base	The FlexCAN base address
msgBuffIdx	Index of the message buffer
extMask	Individual extended mask

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of the message buffer is invalid

3.34.5.44 FLEXCAN_HAL_SetRxIndividualStdMask()

```
status_t FLEXCAN_HAL_SetRxIndividualStdMask (
            CAN_Type * base,
            uint32_t msgBuffIdx,
            uint32_t stdMask )
```

Sets the FlexCAN Rx individual standard mask for ID filtering in the Rx MBs and the Rx FIFO.

Parameters

base	The FlexCAN base address
msgBuffIdx	Index of the message buffer
stdMask	Individual standard mask

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of the message buffer is invalid

3.34.5.45 FLEXCAN_HAL_SetRxMaskType()

```
void FLEXCAN_HAL_SetRxMaskType (
            CAN_Type * base,
            flexcan_rx_mask_type_t type )
```

Sets the Rx masking type.

Parameters

base	The FlexCAN base address
type	The FlexCAN Rx mask type

3.34.5.46 FLEXCAN_HAL_SetRxMsgBuff()

```
status_t FLEXCAN_HAL_SetRxMsgBuff (
             CAN_Type * base,
             uint32_t msgBuffIdx,
             const flexcan_msgbuff_code_status_t * cs,
             uint32_t msgId )
```

Sets the FlexCAN message buffer fields for receiving.

Parameters

base	The FlexCAN base address
msgBuffIdx	Index of the message buffer
cs	CODE/status values (RX)
msgId	ID of the message to receive

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of the message buffer is invalid

3.34.5.47 FLEXCAN_HAL_SetRxMsgBuff14ExtMask()

```
void FLEXCAN_HAL_SetRxMsgBuff14ExtMask (
             CAN_Type * base,
             uint32_t extMask )
```

Sets the FlexCAN RX Message Buffer BUF14 extended mask.

Parameters

base	The FlexCAN base address
extMask	Extended mask

3.34.5.48 FLEXCAN_HAL_SetRxMsgBuff14StdMask()

```
void FLEXCAN_HAL_SetRxMsgBuff14StdMask (
             CAN_Type * base,
             uint32_t stdMask )
```

Sets the FlexCAN RX Message Buffer BUF14 standard mask.

Parameters

base	The FlexCAN base address
stdMask	Standard mask

3.34.5.49 FLEXCAN_HAL_SetRxMsgBuff15ExtMask()

```
void FLEXCAN_HAL_SetRxMsgBuff15ExtMask (
            CAN_Type * base,
            uint32_t extMask )
```

Sets the FlexCAN RX MB BUF15 extended mask.

Parameters

base	The FlexCAN base address
extMask	Extended mask

3.34.5.50 FLEXCAN_HAL_SetRxMsgBuff15StdMask()

```
void FLEXCAN_HAL_SetRxMsgBuff15StdMask (
            CAN_Type * base,
            uint32_t stdMask )
```

Sets the FlexCAN Rx Message Buffer BUF15 standard mask.

Parameters

base	The FlexCAN base address
stdMask	Standard mask

3.34.5.51 FLEXCAN_HAL_SetRxMsgBuffGlobalExtMask()

```
void FLEXCAN_HAL_SetRxMsgBuffGlobalExtMask (
            CAN_Type * base,
            uint32_t extMask )
```

Sets the FlexCAN RX Message Buffer global extended mask.

Parameters

base	The FlexCAN base address
extMask	Extended mask

3.34.5.52 FLEXCAN_HAL_SetRxMsgBuffGlobalStdMask()

```
void FLEXCAN_HAL_SetRxMsgBuffGlobalStdMask (
            CAN_Type * base,
            uint32_t stdMask )
```

Sets the FlexCAN Rx Message Buffer global standard mask.

Parameters

base	The FlexCAN base address
stdMask	Standard mask

3.34.5.53 FLEXCAN_HAL_SetSelfReception()

```
void FLEXCAN_HAL_SetSelfReception (
            CAN_Type * base,
            bool enable )
```

Enables/Disables the Self Reception feature.

If enabled, FlexCAN is allowed to receive frames transmitted by itself.

Parameters

base	The FlexCAN base address
enable	Enable/Disable Self Reception

3.34.5.54 FLEXCAN_HAL_SetStuffBitCount()

```
void FLEXCAN_HAL_SetStuffBitCount (
            CAN_Type * base,
            bool enable )
```

Enables/Disables the Stuff Bit Count for CAN FD frames.

If enabled, the modulo 8 count of variable stuff bits inserted plus the respective parity bit (even parity calculated over the 3-bit modulo 8 count) are combined as the 4-bit Stuff Count field and inserted before the CRC Sequence field. CRC calculation extends beyond the end of Data field and takes the Stuff Count field bits into account.

Parameters

base	The FlexCAN base address
enable	Enable/Disable Stuff Bit Count

3.34.5.55 FLEXCAN_HAL_SetTDCOffset()

```
void FLEXCAN_HAL_SetTDCOffset (
            CAN_Type * base,
            bool enable,
            uint8_t offset )
```

Enables/Disables the Transceiver Delay Compensation feature and sets the Transceiver Delay Compensation Offset (offset value to be added to the measured transceiver's loop delay in order to define the position of the delayed comparison point when bit rate switching is active).

Parameters

base	The FlexCAN base address
enable	Enable/Disable Transceiver Delay Compensation
offset	Transceiver Delay Compensation Offset

3.34.5.56 FLEXCAN_HAL_SetTimeSegments()

```
void FLEXCAN_HAL_SetTimeSegments (
            CAN_Type * base,
            const flexcan_time_segment_t * timeSeg )
```

Sets the FlexCAN time segments for setting up bit rate.

Parameters

base	The FlexCAN base address
timeSeg	FlexCAN time segments, which need to be set for the bit rate.

3.34.5.57 FLEXCAN_HAL_SetTimeSegmentsCbt()

```
void FLEXCAN_HAL_SetTimeSegmentsCbt (
            CAN_Type * base,
            const flexcan_time_segment_t * timeSeg )
```

Sets the FlexCAN time segments for setting up bit rate for FD BRS.

Parameters

base	The FlexCAN base address
timeSeg	FlexCAN time segments, which need to be set for the bit rate.

3.34.5.58 FLEXCAN_HAL_SetTxMsgBuff()

```
status_t FLEXCAN_HAL_SetTxMsgBuff (
            CAN_Type * base,
            uint32_t msgBuffIdx,
            const flexcan_msgbuff_code_status_t * cs,
            uint32_t msgId,
            const uint8_t * msgData )
```

Sets the FlexCAN message buffer fields for transmitting.

Parameters

base	The FlexCAN base address
msgBuffIdx	Index of the message buffer
cs	CODE/status values (TX)
msgId	ID of the message to transmit
msgData	Bytes of the FlexCAN message

Returns

STATUS_SUCCESS if successful; STATUS_FLEXCAN_MB_OUT_OF_RANGE if the index of the message buffer is invalid

3.34.5.59 FLEXCAN_HAL_UnlockRxMsgBuff()

```
static void FLEXCAN_HAL_UnlockRxMsgBuff (
            const CAN_Type * base ) [inline], [static]
```

Unlocks the FlexCAN Rx message buffer.

Parameters

base	The FlexCAN base address Implements : FLEXCAN_HAL_UnlockRxMsgBuff_Activity

3.35 FlexIO Common Driver

3.35.1 Detailed Description

Common services for FlexIO drivers.

The Flexio Common driver layer contains services used by all Flexio drivers. The need for this layer derives from the requirement to allow multiple Flexio drivers to run in parallel on the same device, to the extent that enough hardware resources (shifters and timers) are available.

Functionality

The Flexio Common driver layer provides functions for device initialization and reset. Before using any Flexio driver the device must first be initialized using function FLEXIO_DRV_InitDevice(). Then any number of Flexio drivers can be initialized on the same device, to the extent that enough hardware resources (shifters and timers) are available. Driver initialization functions will return STATUS_ERROR if not enough resources are available for a new driver.

Important Notes

Calling any Flexio common function will destroy any driver that is active on that device. Normally these functions should be called only when there are no active driver instances on the device.

Typedefs

- typedef void(∗ flexio_callback_t) (void ∗driverState, flexio_event_t event, void ∗userData)

  flexio callback function

Enumerations

- enum flexio_driver_type_t { FLEXIO_DRIVER_TYPE_INTERRUPTS = 0U, FLEXIO_DRIVER_TYPE_POL←LING = 1U, FLEXIO_DRIVER_TYPE_DMA = 2U }

  Driver type: interrupts/polling/DMA Implements : flexio_driver_type_t_Class.
- enum flexio_event_t { FLEXIO_EVENT_RX_FULL = 0x00U, FLEXIO_EVENT_TX_EMPTY = 0x01U, FLE←XIO_EVENT_END_TRANSFER = 0x02U }

  flexio events Implements : flexio_event_t_Class

FLEXIO_I2C Driver

- status_t FLEXIO_DRV_InitDevice (uint32_t instance, flexio_device_state_t ∗deviceState)

  Initializes the FlexIO device.
- status_t FLEXIO_DRV_DeinitDevice (uint32_t instance)

  De-initializes the FlexIO device.
- status_t FLEXIO_DRV_Reset (uint32_t instance)

  Resets the FlexIO device.

3.35.2 Typedef Documentation

3.35.2.1 flexio_callback_t

```
typedef void(* flexio_callback_t) (void *driverState, flexio_event_t event, void *userData)
```

flexio callback function

Callback functions are called by flexio drivers when relevant events must be reported. See type flexio_event_t for a list of events. The callback can then react to the event, for example providing the buffers for transmission or reception, or waking a task to use the received data. Note that callback functions are called from interrupts, so the callback execution time should be as small as possible.

3.35.3 Enumeration Type Documentation

3.35.3.1 flexio_driver_type_t

```
enum flexio_driver_type_t
```

Driver type: interrupts/polling/DMA Implements : flexio_driver_type_t_Class.

Enumerator

FLEXIO_DRIVER_TYPE_INTERRUPTS	Driver uses interrupts for data transfers
FLEXIO_DRIVER_TYPE_POLLING	Driver is based on polling
FLEXIO_DRIVER_TYPE_DMA	Driver uses DMA for data transfers

3.35.3.2 flexio_event_t

```
enum flexio_event_t
```

flexio events Implements : flexio_event_t_Class

Enumerator

FLEXIO_EVENT_RX_FULL	Rx buffer is full
FLEXIO_EVENT_TX_EMPTY	Tx buffer is empty
FLEXIO_EVENT_END_TRANSFER	The current transfer is ending

3.35.4 Function Documentation

3.35.4.1 FLEXIO_DRV_DeinitDevice()

```
status_t FLEXIO_DRV_DeinitDevice (
              uint32_t instance )
```

De-initializes the FlexIO device.

This function de-initializes the FlexIO device.

Parameters

instance	FLEXIO peripheral instance number

Returns

Error or success status returned by API

3.35.4.2 FLEXIO_DRV_InitDevice()

```
status_t FLEXIO_DRV_InitDevice (
            uint32_t instance,
            flexio_device_state_t * deviceState )
```

Initializes the FlexIO device.

This function resets the FlexIO device, enables interrupts in interrupt manager and enables the device.

Parameters

instance	FLEXIO peripheral instance number
deviceState	Pointer to the FLEXIO device context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the device is de-initialized using FLEXIO_DRV_DeinitDevice().

Returns

Error or success status returned by API

3.35.4.3 FLEXIO_DRV_Reset()

```
status_t FLEXIO_DRV_Reset (
            uint32_t instance )
```

Resets the FlexIO device.

This function resets the FlexIO device.

Parameters

instance	FLEXIO peripheral instance number

Returns

Error or success status returned by API

3.36 FlexIO HAL

3.36.1 Detailed Description

FlexIO Hardware Abstraction Layer (FLEXIO_HAL)

This module covers the functionality of the Flexible I/O (FlexIO) peripheral.

FlexIO HAL provides the API for reading and writing register bit-fields belonging to the FlexIO module. It also provides an initialization function for bringing the module to the reset state.

For higher-level functionality, use the FlexIO driver.

Data Structures

- struct flexio_version_info_t

  FlexIO module version number Implements : flexio_version_info_t_Class. More...

Enumerations

- enum flexio_timer_polarity_t { FLEXIO_TIMER_POLARITY_POSEDGE = 0x00U, FLEXIO_TIMER_POLA↩RITY_NEGEDGE = 0x01U }

  Shift clock polarity options Implements : flexio_timer_polarity_t_Class.
- enum flexio_pin_polarity_t { FLEXIO_PIN_POLARITY_HIGH = 0x00U, FLEXIO_PIN_POLARITY_LOW = 0x01U }

  Pin polarity options Implements : flexio_pin_polarity_t_Class.
- enum flexio_pin_config_t { FLEXIO_PIN_CONFIG_DISABLED = 0x00U, FLEXIO_PIN_CONFIG_OPEN↩_DRAIN = 0x01U, FLEXIO_PIN_CONFIG_BIDIR_OUTPUT = 0x02U, FLEXIO_PIN_CONFIG_OUTPUT = 0x03U }

  Pin configuration options Implements : flexio_pin_config_t_Class.
- enum flexio_shifter_mode_t {
  FLEXIO_SHIFTER_MODE_DISABLED = 0x00U, FLEXIO_SHIFTER_MODE_RECEIVE = 0x01U, FLEXIO↩_SHIFTER_MODE_TRANSMIT = 0x02U, FLEXIO_SHIFTER_MODE_MATCH_STORE = 0x04U,
  FLEXIO_SHIFTER_MODE_MATCH_CONTINUOUS = 0x05U }

  Shifter mode options Implements : flexio_shifter_mode_t_Class.
- enum flexio_shifter_source_t { FLEXIO_SHIFTER_SOURCE_PIN = 0x00U, FLEXIO_SHIFTER_SOURCE↩_SHIFTER = 0x01U }

  Shifter input source options Implements : flexio_shifter_source_t_Class.
- enum flexio_shifter_buffer_mode_t { FLEXIO_SHIFTER_RW_MODE_NORMAL = 0x00U, FLEXIO_SHIFT↩ER_RW_MODE_BIT_SWAP = 0x01U, FLEXIO_SHIFTER_RW_MODE_BYTE_SWAP = 0x02U, FLEXIO_↩SHIFTER_RW_MODE_BB_SWAP = 0x03U }

  Read/Write mode for shifter buffer Implements : flexio_shifter_buffer_mode_t_Class.
- enum flexio_trigger_polarity_t { FLEXIO_TRIGGER_POLARITY_HIGH = 0x00U, FLEXIO_TRIGGER_PO↩LARITY_LOW = 0x01U }

  Trigger polarity Implements : flexio_trigger_polarity_t_Class.
- enum flexio_trigger_source_t { FLEXIO_TRIGGER_SOURCE_EXTERNAL = 0x00U, FLEXIO_TRIGGER_↩SOURCE_INTERNAL = 0x01U }

  Trigger sources Implements : flexio_trigger_source_t_Class.
- enum flexio_timer_mode_t { FLEXIO_TIMER_MODE_DISABLED = 0x00U, FLEXIO_TIMER_MODE_8BI↩T_BAUD = 0x01U, FLEXIO_TIMER_MODE_8BIT_PWM = 0x02U, FLEXIO_TIMER_MODE_16BIT = 0x03U }

  Timer mode options Implements : flexio_timer_mode_t_Class.

- enum flexio_timer_output_t { FLEXIO_TIMER_INITOUT_ONE = 0x00U, FLEXIO_TIMER_INITOUT_ZERO = 0x01U, FLEXIO_TIMER_INITOUT_ONE_RESET = 0x02U, FLEXIO_TIMER_INITOUT_ZERO_RESET = 0x03U }

  Timer initial output options Implements : flexio_timer_output_t_Class.
- enum flexio_timer_decrement_t { FLEXIO_TIMER_DECREMENT_CLK_SHIFT_TMR = 0x00U, FLEXIO↵_TIMER_DECREMENT_TRG_SHIFT_TMR = 0x01U, FLEXIO_TIMER_DECREMENT_PIN_SHIFT_PIN = 0x02U, FLEXIO_TIMER_DECREMENT_TRG_SHIFT_TRG = 0x03U }

  Timer decrement options Implements : flexio_timer_decrement_t_Class.
- enum flexio_timer_reset_t {
  FLEXIO_TIMER_RESET_NEVER = 0x00U, FLEXIO_TIMER_RESET_PIN_OUT = 0x02U, FLEXIO_TIME↵R_RESET_TRG_OUT = 0x03U, FLEXIO_TIMER_RESET_PIN_RISING = 0x04U,
  FLEXIO_TIMER_RESET_TRG_RISING = 0x06U, FLEXIO_TIMER_RESET_TRG_BOTH = 0x07U }

  Timer reset options Implements : flexio_timer_reset_t_Class.
- enum flexio_timer_disable_t {
  FLEXIO_TIMER_DISABLE_NEVER = 0x00U, FLEXIO_TIMER_DISABLE_TIM_DISABLE = 0x01U, FLEX↵IO_TIMER_DISABLE_TIM_CMP = 0x02U, FLEXIO_TIMER_DISABLE_TIM_CMP_TRG_LOW = 0x03U,
  FLEXIO_TIMER_DISABLE_PIN = 0x04U, FLEXIO_TIMER_DISABLE_PIN_TRG_HIGH = 0x05U, FLEXIO↵_TIMER_DISABLE_TRG = 0x06U }

  Timer disable options Implements : flexio_timer_disable_t_Class.
- enum flexio_timer_enable_t {
  FLEXIO_TIMER_ENABLE_ALWAYS = 0x00U, FLEXIO_TIMER_ENABLE_TIM_ENABLE = 0x01U, FLEXI↵O_TIMER_ENABLE_TRG_HIGH = 0x02U, FLEXIO_TIMER_ENABLE_TRG_PIN_HIGH = 0x03U,
  FLEXIO_TIMER_ENABLE_PIN_POSEDGE = 0x04U, FLEXIO_TIMER_ENABLE_PIN_POSEDGE_TRG_↵HIGH = 0x05U, FLEXIO_TIMER_ENABLE_TRG_POSEDGE = 0x06U, FLEXIO_TIMER_ENABLE_TRG_↵EDGE = 0x07U }

  Timer disable options Implements : flexio_timer_enable_t_Class.
- enum flexio_timer_stop_t { FLEXIO_TIMER_STOP_BIT_DISABLED = 0x00U, FLEXIO_TIMER_STOP_BI↵T_TIM_CMP = 0x01U, FLEXIO_TIMER_STOP_BIT_TIM_DIS = 0x02U, FLEXIO_TIMER_STOP_BIT_TIM↵_CMP_DIS = 0x03U }

  Timer stop bit options Implements : flexio_timer_stop_t_Class.
- enum flexio_shifter_stop_t { FLEXIO_SHIFTER_STOP_BIT_DISABLED = 0x00U, FLEXIO_SHIFTER_ST↵OP_BIT_0 = 0x02U, FLEXIO_SHIFTER_STOP_BIT_1 = 0x03U }

  Timer stop bit options - for Transmit, Receive or Match Store modes only Implements : flexio_shifter_stop_t_Class.
- enum flexio_shifter_start_t { FLEXIO_SHIFTER_START_BIT_DISABLED = 0x00U, FLEXIO_SHIFTER_S↵TART_BIT_DISABLED_SH = 0x01U, FLEXIO_SHIFTER_START_BIT_0 = 0x02U, FLEXIO_SHIFTER_ST↵ART_BIT_1 = 0x03U }

  Timer start bit options - for Transmit, Receive or Match Store modes only Implements : flexio_shifter_start_t_Class.
- enum flexio_timer_start_t { FLEXIO_TIMER_START_BIT_DISABLED = 0x00U, FLEXIO_TIMER_START_↵BIT_ENABLED = 0x01U }

  Timer start bit options Implements : flexio_timer_start_t_Class.

Template Group

- static void FLEXIO_HAL_GetVersion (const FLEXIO_Type ∗baseAddr, flexio_version_info_t ∗versionInfo)

  Reads the version of the FlexIO module.
- static uint8_t FLEXIO_HAL_GetTriggerNum (const FLEXIO_Type ∗baseAddr)

  Returns the number of external triggers of the FlexIO module.
- static uint8_t FLEXIO_HAL_GetPinNum (const FLEXIO_Type ∗baseAddr)

  Returns the number of pins of the FlexIO module.
- static uint8_t FLEXIO_HAL_GetTimerNum (const FLEXIO_Type ∗baseAddr)

  Returns the number of timers of the FlexIO module.
- static uint8_t FLEXIO_HAL_GetShifterNum (const FLEXIO_Type ∗baseAddr)

  Returns the number of shifters of the FlexIO module.

- static bool FLEXIO_HAL_GetDozeMode (const FLEXIO_Type ∗baseAddr)

  Returns the current doze mode setting.
- static void FLEXIO_HAL_SetDozeMode (FLEXIO_Type ∗baseAddr, bool enable)

  Set the FlexIO module behavior in doze mode.
- static bool FLEXIO_HAL_GetDebugMode (const FLEXIO_Type ∗baseAddr)

  Returns the current debug mode setting.
- static void FLEXIO_HAL_SetDebugMode (FLEXIO_Type ∗baseAddr, bool enable)

  Set the FlexIO module behavior in debug mode.
- static bool FLEXIO_HAL_GetFastAccess (const FLEXIO_Type ∗baseAddr)

  Returns the current fast access setting.
- static void FLEXIO_HAL_SetFastAccess (FLEXIO_Type ∗baseAddr, bool enable)

  Configure the FlexIO fast access feature.
- static bool FLEXIO_HAL_GetSoftwareReset (const FLEXIO_Type ∗baseAddr)

  Returns the current software reset state.
- static void FLEXIO_HAL_SetSoftwareReset (FLEXIO_Type ∗baseAddr, bool enable)

  Set/clear the FlexIO reset command.
- static bool FLEXIO_HAL_GetEnable (const FLEXIO_Type ∗baseAddr)

  Returns the current enable/disable setting of the FlexIO.
- static void FLEXIO_HAL_SetEnable (FLEXIO_Type ∗baseAddr, bool enable)

  Enables of disables the FlexIO module.
- static uint8_t FLEXIO_HAL_GetPinData (const FLEXIO_Type ∗baseAddr)

  Returns the current input data read from the FlexIO pins.
- static bool FLEXIO_HAL_GetShifterStatus (const FLEXIO_Type ∗baseAddr, uint8_t shifter)

  Returns the current status of the specified shifter.
- static uint32_t FLEXIO_HAL_GetAllShifterStatus (const FLEXIO_Type ∗baseAddr)

  Returns the current status flags for all shifters.
- static void FLEXIO_HAL_ClearShifterStatus (FLEXIO_Type ∗baseAddr, uint8_t shifter)

  Clears the status of the specified shifter.
- static bool FLEXIO_HAL_GetShifterErrorStatus (const FLEXIO_Type ∗baseAddr, uint8_t shifter)

  Returns the current error status of the specified shifter.
- static uint32_t FLEXIO_HAL_GetAllShifterErrorStatus (const FLEXIO_Type ∗baseAddr)

  Returns the current error status for all shifters.
- static void FLEXIO_HAL_ClearShifterErrorStatus (FLEXIO_Type ∗baseAddr, uint8_t shifter)

  Clears the error status of the specified shifter.
- static bool FLEXIO_HAL_GetTimerStatus (const FLEXIO_Type ∗baseAddr, uint8_t timer)

  Returns the current status of the specified timer.
- static uint32_t FLEXIO_HAL_GetAllTimerStatus (const FLEXIO_Type ∗baseAddr)

  Returns the current status of all timers.
- static void FLEXIO_HAL_ClearTimerStatus (FLEXIO_Type ∗baseAddr, uint8_t timer)

  Clears the status of the specified timer.
- static bool FLEXIO_HAL_GetShifterInterrupt (const FLEXIO_Type ∗baseAddr, uint8_t interruptNo)

  Returns the current status of the shifter interrupts.
- static uint32_t FLEXIO_HAL_GetAllShifterInterrupt (const FLEXIO_Type ∗baseAddr)

  Returns the current status of all the shifter interrupts.
- static void FLEXIO_HAL_SetShifterInterrupt (FLEXIO_Type ∗baseAddr, uint8_t interruptMask, bool enable)

  Enables or disables the shifter interrupts.
- static bool FLEXIO_HAL_GetShifterErrorInterrupt (const FLEXIO_Type ∗baseAddr, uint8_t interruptNo)

  Returns the current status of the shifter error interrupts.
- static uint32_t FLEXIO_HAL_GetAllShifterErrorInterrupt (const FLEXIO_Type ∗baseAddr)

  Returns the current status of all the shifter error interrupts.

- static void FLEXIO_HAL_SetShifterErrorInterrupt (FLEXIO_Type ∗baseAddr, uint8_t interruptMask, bool enable)

    Enables or disables the shifter error interrupts.
- static bool FLEXIO_HAL_GetTimerInterrupt (const FLEXIO_Type ∗baseAddr, uint8_t interruptNo)

    Returns the current status of the timer interrupts.
- static uint32_t FLEXIO_HAL_GetAllTimerInterrupt (const FLEXIO_Type ∗baseAddr)

    Returns the current status of all the timer interrupts.
- static void FLEXIO_HAL_SetTimerInterrupt (FLEXIO_Type ∗baseAddr, uint8_t interruptMask, bool enable)

    Enables or disables the timer interrupts.
- static bool FLEXIO_HAL_GetShifterDMARequest (const FLEXIO_Type ∗baseAddr, uint8_t requestNo)

    Returns the current status of the shifter DMA requests.
- static void FLEXIO_HAL_SetShifterDMARequest (FLEXIO_Type ∗baseAddr, uint8_t requestMask, bool enable)

    Enables or disables the shifter DMA requests.
- static void FLEXIO_HAL_GetShifterTimer (const FLEXIO_Type ∗baseAddr, uint8_t shifter, uint8_t ∗timer, flexio_timer_polarity_t ∗polarity)

    Returns the timer currently assigned to control the specified shifter.
- static void FLEXIO_HAL_SetShifterTimer (FLEXIO_Type ∗baseAddr, uint8_t shifter, uint8_t timer, flexio_↩ timer_polarity_t polarity)

    Assigns the specified timer to control the specified shifter.
- static void FLEXIO_HAL_GetShifterPin (const FLEXIO_Type ∗baseAddr, uint8_t shifter, uint8_t ∗pin, flexio↩ _pin_polarity_t ∗polarity, flexio_pin_config_t ∗config)

    Returns the pin currently assigned to the specified shifter, and its configured settings.
- static void FLEXIO_HAL_SetShifterPin (FLEXIO_Type ∗baseAddr, uint8_t shifter, uint8_t pin, flexio_pin_↩ polarity_t polarity, flexio_pin_config_t config)

    Assigns the specified pin to the specified shifter.
- static void FLEXIO_HAL_SetShifterPinConfig (FLEXIO_Type ∗baseAddr, uint8_t shifter, flexio_pin_config_t config)

    Configures the pin assigned to the specified shifter.
- static flexio_shifter_mode_t FLEXIO_HAL_GetShifterMode (const FLEXIO_Type ∗baseAddr, uint8_t shifter)

    Returns the mode of the specified shifter.
- static void FLEXIO_HAL_SetShifterMode (FLEXIO_Type ∗baseAddr, uint8_t shifter, flexio_shifter_mode_↩ t mode)

    Sets the mode of the specified shifter.
- static void FLEXIO_HAL_SetShifterControl (FLEXIO_Type ∗baseAddr, uint8_t shifter, flexio_shifter_mode_t mode, uint8_t pin, flexio_pin_polarity_t pinPolarity, flexio_pin_config_t pinConfig, uint8_t timer, flexio_timer↩ _polarity_t timerPolarity)

    Sets all control settings for the specified shifter.
- static flexio_shifter_source_t FLEXIO_HAL_GetShifterInputSource (const FLEXIO_Type ∗baseAddr, uint8↩ _t shifter)

    Returns the input source of the specified shifter.
- static void FLEXIO_HAL_SetShifterInputSource (FLEXIO_Type ∗baseAddr, uint8_t shifter, flexio_shifter_↩ source_t source)

    Configures the input source of the specified shifter.
- static flexio_shifter_stop_t FLEXIO_HAL_GetShifterStopBit (const FLEXIO_Type ∗baseAddr, uint8_t shifter)

    Returns the stop bit configuration for the specified shifter.
- static void FLEXIO_HAL_SetShifterStopBit (FLEXIO_Type ∗baseAddr, uint8_t shifter, flexio_shifter_stop_t stop)

    Configures the stop bit of the specified shifter.
- static flexio_shifter_start_t FLEXIO_HAL_GetShifterStartBit (const FLEXIO_Type ∗baseAddr, uint8_t shifter)

    Returns the start bit configuration for the specified shifter.
- static void FLEXIO_HAL_SetShifterStartBit (FLEXIO_Type ∗baseAddr, uint8_t shifter, flexio_shifter_start_t start)

Configures the start bit of the specified shifter.
- static void FLEXIO_HAL_SetShifterConfig (FLEXIO_Type ∗baseAddr, uint8_t shifter, flexio_shifter_start_t start, flexio_shifter_stop_t stop, flexio_shifter_source_t source)

  Sets all configuration settings for specified shifter.
- static uint32_t FLEXIO_HAL_ReadShifterBuffer (const FLEXIO_Type ∗baseAddr, uint8_t shifter, flexio_shifter_buffer_mode_t mode)

  Reads the value from the specified shifter buffer.
- static void FLEXIO_HAL_WriteShifterBuffer (FLEXIO_Type ∗baseAddr, uint8_t shifter, uint32_t value, flexio_shifter_buffer_mode_t mode)

  Writes a value in the specified shifter buffer.
- static void FLEXIO_HAL_GetTimerTrigger (const FLEXIO_Type ∗baseAddr, uint8_t timer, uint8_t ∗trigger, flexio_trigger_polarity_t ∗polarity, flexio_trigger_source_t ∗source)

  Returns the currently configured trigger for the specified timer.
- static void FLEXIO_HAL_SetTimerTrigger (FLEXIO_Type ∗baseAddr, uint8_t timer, uint8_t trigger, flexio_trigger_polarity_t polarity, flexio_trigger_source_t source)

  Configures the trigger for the specified timer.
- static void FLEXIO_HAL_GetTimerPin (const FLEXIO_Type ∗baseAddr, uint8_t timer, uint8_t ∗pin, flexio_pin_polarity_t ∗polarity, flexio_pin_config_t ∗config)

  Returns the currently configured pin for the specified timer.
- static void FLEXIO_HAL_SetTimerPin (FLEXIO_Type ∗baseAddr, uint8_t timer, uint8_t pin, flexio_pin_polarity_t polarity, flexio_pin_config_t config)

  Configures the pin for the specified timer.
- static flexio_timer_mode_t FLEXIO_HAL_GetTimerMode (const FLEXIO_Type ∗baseAddr, uint8_t timer)

  Returns the mode of the specified timer.
- static void FLEXIO_HAL_SetTimerMode (FLEXIO_Type ∗baseAddr, uint8_t timer, flexio_timer_mode_t mode)

  Sets the mode of the specified timer.
- static void FLEXIO_HAL_SetTimerControl (FLEXIO_Type ∗baseAddr, uint8_t timer, uint8_t trigger, flexio_trigger_polarity_t triggerPolarity, flexio_trigger_source_t triggerSource, uint8_t pin, flexio_pin_polarity_t pinPolarity, flexio_pin_config_t pinConfig, flexio_timer_mode_t mode)

  Sets all control settings for specified timer.
- static flexio_timer_output_t FLEXIO_HAL_GetTimerInitialOutput (const FLEXIO_Type ∗baseAddr, uint8_t timer)

  Returns the initial output configuration of the specified timer.
- static void FLEXIO_HAL_SetTimerInitialOutput (FLEXIO_Type ∗baseAddr, uint8_t timer, flexio_timer_output_t output)

  Configures the initial output of the specified timer.
- static flexio_timer_decrement_t FLEXIO_HAL_GetTimerDecrement (const FLEXIO_Type ∗baseAddr, uint8_t timer)

  Returns the decrement configuration of the specified timer.
- static void FLEXIO_HAL_SetTimerDecrement (FLEXIO_Type ∗baseAddr, uint8_t timer, flexio_timer_decrement_t decrement)

  Configures the decrement condition for the specified timer.
- static flexio_timer_reset_t FLEXIO_HAL_GetTimerReset (const FLEXIO_Type ∗baseAddr, uint8_t timer)

  Returns the reset configuration of the specified timer.
- static void FLEXIO_HAL_SetTimerReset (FLEXIO_Type ∗baseAddr, uint8_t timer, flexio_timer_reset_t reset)

  Configures the reset condition for the specified timer.
- static flexio_timer_disable_t FLEXIO_HAL_GetTimerDisable (const FLEXIO_Type ∗baseAddr, uint8_t timer)

  Returns the disable configuration of the specified timer.
- static void FLEXIO_HAL_SetTimerDisable (FLEXIO_Type ∗baseAddr, uint8_t timer, flexio_timer_disable_t disable)

  Configures the disable condition for the specified timer.
- static flexio_timer_enable_t FLEXIO_HAL_GetTimerEnable (const FLEXIO_Type ∗baseAddr, uint8_t timer)

Returns the enable condition configuration of the specified timer.
- static void FLEXIO_HAL_SetTimerEnable (FLEXIO_Type ∗baseAddr, uint8_t timer, flexio_timer_enable_↩
t enable)

  Configures the enable condition for the specified timer.
- static flexio_timer_stop_t FLEXIO_HAL_GetTimerStop (const FLEXIO_Type ∗baseAddr, uint8_t timer)

  Returns the stop bit configuration of the specified timer.
- static void FLEXIO_HAL_SetTimerStop (FLEXIO_Type ∗baseAddr, uint8_t timer, flexio_timer_stop_t stop)

  Configures the stop bit for the specified timer.
- static flexio_timer_start_t FLEXIO_HAL_GetTimerStart (const FLEXIO_Type ∗baseAddr, uint8_t timer)

  Returns the start bit configuration of the specified timer.
- static void FLEXIO_HAL_SetTimerStart (FLEXIO_Type ∗baseAddr, uint8_t timer, flexio_timer_start_t start)

  Configures the start bit for the specified timer.
- static void FLEXIO_HAL_SetTimerConfig (FLEXIO_Type ∗baseAddr, uint8_t timer, flexio_timer_start_t start,
flexio_timer_stop_t stop, flexio_timer_enable_t enable, flexio_timer_disable_t disable, flexio_timer_reset_↩
t reset, flexio_timer_decrement_t decrement, flexio_timer_output_t output)

  Sets all configuration settings for specified timer.
- static uint16_t FLEXIO_HAL_GetTimerCompare (const FLEXIO_Type ∗baseAddr, uint8_t timer)

  Returns the compare value of the specified timer.
- static void FLEXIO_HAL_SetTimerCompare (FLEXIO_Type ∗baseAddr, uint8_t timer, uint16_t value)

  Configures the compare value for the specified timer.
- void FLEXIO_HAL_Init (FLEXIO_Type ∗baseAddr)

  Initializes the FlexIO module to a known state.

3.36.2 Data Structure Documentation

3.36.2.1 struct flexio_version_info_t

FlexIO module version number Implements : flexio_version_info_t_Class.

Data Fields

- uint8_t majorNumber
- uint8_t minorNumber
- uint16_t featureNumber

Field Documentation

3.36.2.1.1 featureNumber

```
uint16_t featureNumber
```

Feature Specification Number

3.36.2.1.2 majorNumber

```
uint8_t majorNumber
```

Major Version Number

3.36.2.1.3 minorNumber

```
uint8_t minorNumber
```

Minor Version Number

3.36.3 Enumeration Type Documentation

3.36.3.1 flexio_pin_config_t

```
enum flexio_pin_config_t
```

Pin configuration options Implements : flexio_pin_config_t_Class.

Enumerator

FLEXIO_PIN_CONFIG_DISABLED	Shifter pin output disabled
FLEXIO_PIN_CONFIG_OPEN_DRAIN	Shifter pin open drain or bidirectional output enable
FLEXIO_PIN_CONFIG_BIDIR_OUTPUT	Shifter pin bidirectional output data
FLEXIO_PIN_CONFIG_OUTPUT	Shifter pin output

3.36.3.2 flexio_pin_polarity_t

```
enum flexio_pin_polarity_t
```

Pin polarity options Implements : flexio_pin_polarity_t_Class.

Enumerator

FLEXIO_PIN_POLARITY_HIGH	Pin is active high
FLEXIO_PIN_POLARITY_LOW	Pin is active low

3.36.3.3 flexio_shifter_buffer_mode_t

```
enum flexio_shifter_buffer_mode_t
```

Read/Write mode for shifter buffer Implements : flexio_shifter_buffer_mode_t_Class.

Enumerator

FLEXIO_SHIFTER_RW_MODE_NORMAL	Normal shifter buffer read/write
FLEXIO_SHIFTER_RW_MODE_BIT_SWAP	Data is bit-swapped
FLEXIO_SHIFTER_RW_MODE_BYTE_SWAP	Data is byte-swapped
FLEXIO_SHIFTER_RW_MODE_BB_SWAP	Data in each byte is bit-swapped

3.36.3.4 flexio_shifter_mode_t

```
enum flexio_shifter_mode_t
```

Shifter mode options Implements : flexio_shifter_mode_t_Class.

Enumerator

FLEXIO_SHIFTER_MODE_DISABLED	Shifter disabled
FLEXIO_SHIFTER_MODE_RECEIVE	Receive mode
FLEXIO_SHIFTER_MODE_TRANSMIT	Transmit mode
FLEXIO_SHIFTER_MODE_MATCH_STORE	Match Store mode
FLEXIO_SHIFTER_MODE_MATCH_CONTINUOUS	Match Continuous mode

3.36.3.5 flexio_shifter_source_t

```
enum flexio_shifter_source_t
```

Shifter input source options Implements : flexio_shifter_source_t_Class.

Enumerator

FLEXIO_SHIFTER_SOURCE_PIN	Input source is selected pin
FLEXIO_SHIFTER_SOURCE_SHIFTER	Input source is shifter N+1 output

3.36.3.6 flexio_shifter_start_t

```
enum flexio_shifter_start_t
```

Timer start bit options - for Transmit, Receive or Match Store modes only Implements : flexio_shifter_start_t_Class.

Enumerator

FLEXIO_SHIFTER_START_BIT_DISABLED	Start bit disabled, transmitter loads data on enable
FLEXIO_SHIFTER_START_BIT_DISABLED_SH	Start bit disabled, transmitter loads data on first shift
FLEXIO_SHIFTER_START_BIT_0	Transmit/expect start bit value 0
FLEXIO_SHIFTER_START_BIT_1	Transmit/expect start bit value 1

3.36.3.7 flexio_shifter_stop_t

```
enum flexio_shifter_stop_t
```

Timer stop bit options - for Transmit, Receive or Match Store modes only Implements : flexio_shifter_stop_t_Class.

Enumerator

FLEXIO_SHIFTER_STOP_BIT_DISABLED	Stop bit disabled.
FLEXIO_SHIFTER_STOP_BIT_0	Transmit/expect stop bit value 0
FLEXIO_SHIFTER_STOP_BIT_1	Transmit/expect stop bit value 1

3.36.3.8 flexio_timer_decrement_t

```
enum flexio_timer_decrement_t
```

Timer decrement options Implements : flexio_timer_decrement_t_Class.

Enumerator

FLEXIO_TIMER_DECREMENT_CLK_SHIFT_TMR	Decrement counter on FlexIO clock, Shift clock equals Timer output.
FLEXIO_TIMER_DECREMENT_TRG_SHIFT_TMR	Decrement counter on Trigger input (both edges), Shift clock equals Timer output.
FLEXIO_TIMER_DECREMENT_PIN_SHIFT_PIN	Decrement counter on Pin input (both edges), Shift clock equals Pin input.
FLEXIO_TIMER_DECREMENT_TRG_SHIFT_TRG	Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

3.36.3.9 flexio_timer_disable_t

```
enum flexio_timer_disable_t
```

Timer disable options Implements : flexio_timer_disable_t_Class.

Enumerator

FLEXIO_TIMER_DISABLE_NEVER	Timer never disabled.
FLEXIO_TIMER_DISABLE_TIM_DISABLE	Timer disabled on Timer N-1 disable.
FLEXIO_TIMER_DISABLE_TIM_CMP	Timer disabled on Timer compare.
FLEXIO_TIMER_DISABLE_TIM_CMP_TRG_LOW	Timer disabled on Timer compare and Trigger Low.
FLEXIO_TIMER_DISABLE_PIN	Timer disabled on Pin rising or falling edge.
FLEXIO_TIMER_DISABLE_PIN_TRG_HIGH	Timer disabled on Pin rising or falling edge provided Trigger is high.
FLEXIO_TIMER_DISABLE_TRG	Timer disabled on Trigger falling edge.

3.36.3.10 flexio_timer_enable_t

```
enum flexio_timer_enable_t
```

Timer disable options Implements : flexio_timer_enable_t_Class.

Enumerator

FLEXIO_TIMER_ENABLE_ALWAYS	Timer always enabled.
FLEXIO_TIMER_ENABLE_TIM_ENABLE	Timer enabled on Timer N-1 enable.
FLEXIO_TIMER_ENABLE_TRG_HIGH	Timer enabled on Trigger high.
FLEXIO_TIMER_ENABLE_TRG_PIN_HIGH	Timer enabled on Trigger high and Pin high.
FLEXIO_TIMER_ENABLE_PIN_POSEDGE	Timer enabled on Pin rising edge.
FLEXIO_TIMER_ENABLE_PIN_POSEDGE_TRG_HIGH	Timer enabled on Pin rising edge and Trigger high.
FLEXIO_TIMER_ENABLE_TRG_POSEDGE	Timer enabled on Trigger rising edge.
FLEXIO_TIMER_ENABLE_TRG_EDGE	Timer enabled on Trigger rising or falling edge.

3.36.3.11 flexio_timer_mode_t

```
enum flexio_timer_mode_t
```

Timer mode options Implements : flexio_timer_mode_t_Class.

Enumerator

FLEXIO_TIMER_MODE_DISABLED	Timer Disabled.
FLEXIO_TIMER_MODE_8BIT_BAUD	Dual 8-bit counters baud/bit mode.
FLEXIO_TIMER_MODE_8BIT_PWM	Dual 8-bit counters PWM mode.
FLEXIO_TIMER_MODE_16BIT	Single 16-bit counter mode.

3.36.3.12 flexio_timer_output_t

`enum flexio_timer_output_t`

Timer initial output options Implements : flexio_timer_output_t_Class.

Enumerator

FLEXIO_TIMER_INITOUT_ONE	Timer output is logic one when enabled, unaffected by timer reset.
FLEXIO_TIMER_INITOUT_ZERO	Timer output is logic zero when enabled, unaffected by timer reset.
FLEXIO_TIMER_INITOUT_ONE_RESET	Timer output is logic one when enabled and on timer reset.
FLEXIO_TIMER_INITOUT_ZERO_RESET	Timer output is logic zero when enabled and on timer reset.

3.36.3.13 flexio_timer_polarity_t

`enum flexio_timer_polarity_t`

Shift clock polarity options Implements : flexio_timer_polarity_t_Class.

Enumerator

FLEXIO_TIMER_POLARITY_POSEDGE	Shift on positive edge of Shift clock
FLEXIO_TIMER_POLARITY_NEGEDGE	Shift on negative edge of Shift clock

3.36.3.14 flexio_timer_reset_t

`enum flexio_timer_reset_t`

Timer reset options Implements : flexio_timer_reset_t_Class.

Enumerator

FLEXIO_TIMER_RESET_NEVER	Timer never reset.
FLEXIO_TIMER_RESET_PIN_OUT	Timer reset on Timer Pin equal to Timer Output.
FLEXIO_TIMER_RESET_TRG_OUT	Timer reset on Timer Trigger equal to Timer Output.
FLEXIO_TIMER_RESET_PIN_RISING	Timer reset on Timer Pin rising edge.
FLEXIO_TIMER_RESET_TRG_RISING	Timer reset on Trigger rising edge.
FLEXIO_TIMER_RESET_TRG_BOTH	Timer reset on Trigger rising or falling edge.

3.36.3.15 flexio_timer_start_t

```
enum flexio_timer_start_t
```

Timer start bit options Implements : flexio_timer_start_t_Class.

Enumerator

FLEXIO_TIMER_START_BIT_DISABLED	Start bit disabled.
FLEXIO_TIMER_START_BIT_ENABLED	Start bit enabled.

3.36.3.16 flexio_timer_stop_t

```
enum flexio_timer_stop_t
```

Timer stop bit options Implements : flexio_timer_stop_t_Class.

Enumerator

FLEXIO_TIMER_STOP_BIT_DISABLED	Stop bit disabled.
FLEXIO_TIMER_STOP_BIT_TIM_CMP	Stop bit is enabled on timer compare.
FLEXIO_TIMER_STOP_BIT_TIM_DIS	Stop bit is enabled on timer disable.
FLEXIO_TIMER_STOP_BIT_TIM_CMP_DIS	Stop bit is enabled on timer compare and disable.

3.36.3.17 flexio_trigger_polarity_t

```
enum flexio_trigger_polarity_t
```

Trigger polarity Implements : flexio_trigger_polarity_t_Class.

Enumerator

FLEXIO_TRIGGER_POLARITY_HIGH	Trigger is active high
FLEXIO_TRIGGER_POLARITY_LOW	Trigger is active low

3.36.3.18 flexio_trigger_source_t

```
enum flexio_trigger_source_t
```

Trigger sources Implements : flexio_trigger_source_t_Class.

Enumerator

FLEXIO_TRIGGER_SOURCE_EXTERNAL	External trigger selected
FLEXIO_TRIGGER_SOURCE_INTERNAL	Internal trigger selected

3.36.4 Function Documentation

3.36.4.1 FLEXIO_HAL_ClearShifterErrorStatus()

```
static void FLEXIO_HAL_ClearShifterErrorStatus (
            FLEXIO_Type * baseAddr,
            uint8_t shifter )  [inline], [static]
```

Clears the error status of the specified shifter.

This function clears the error status flag for the specified shifter.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter. Implements : FLEXIO_HAL_ClearShifterErrorStatus_Activity

3.36.4.2 FLEXIO_HAL_ClearShifterStatus()

```
static void FLEXIO_HAL_ClearShifterStatus (
            FLEXIO_Type * baseAddr,
            uint8_t shifter )  [inline], [static]
```

Clears the status of the specified shifter.

This function clears the status flag for the specified shifter. This is possible in all modes except Match Continuous mode.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter. Implements : FLEXIO_HAL_ClearShifterStatus_Activity

3.36.4.3 FLEXIO_HAL_ClearTimerStatus()

```
static void FLEXIO_HAL_ClearTimerStatus (
            FLEXIO_Type * baseAddr,
            uint8_t timer )  [inline], [static]
```

Clears the status of the specified timer.

This function clears the status flag for the specified timer.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer. Implements : FLEXIO_HAL_ClearTimerStatus_Activity

3.36.4.4 FLEXIO_HAL_GetAllShifterErrorInterrupt()

```
static uint32_t FLEXIO_HAL_GetAllShifterErrorInterrupt (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current status of all the shifter error interrupts.

Returns the state of the error interrupt for all shifters. Each bit in the returned value specifies the interrupt state for one shifter, starting with shifter 0 from least significant bit.

Parameters

in	*baseAddr*	Base address of the FlexIO module.

Returns

Status of the interrupts. Implements : FLEXIO_HAL_GetAllShifterErrorInterrupt_Activity

3.36.4.5 FLEXIO_HAL_GetAllShifterErrorStatus()

```
static uint32_t FLEXIO_HAL_GetAllShifterErrorStatus (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current error status for all shifters.

This function returns the value of the error status flags for all shifters. Each bit in the returned value specifies the error status for one shifter, starting with shifter 0 from least significant bit. The meaning of the error status flag depends on the current mode.

- Transmit mode: shifter buffer was not written before it was transferred in the shifter (buffer overrun)
- Receive mode: shifter buffer was not read before new data was transferred from the shifter (buffer underrun)
- Match Store mode: match event occurred before the previous match data was read from shifter buffer (buffer overrun)
- Match Continuous mode: match occurred between shifter buffer and shifter

Parameters

in	*baseAddr*	Base address of the FlexIO module.

Returns

Error status of the shifters. Implements : FLEXIO_HAL_GetAllShifterErrorStatus_Activity

3.36.4.6 FLEXIO_HAL_GetAllShifterInterrupt()

```
static uint32_t FLEXIO_HAL_GetAllShifterInterrupt (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current status of all the shifter interrupts.

Returns the state of the interrupt for all shifters. Each bit in the returned value specifies the interrupt state for one shifter, starting with shifter 0 from least significant bit.

Parameters

in	*baseAddr*	Base address of the FlexIO module.

Returns

Status of the interrupts. Implements : FLEXIO_HAL_GetAllShifterInterrupt_Activity

3.36.4.7 FLEXIO_HAL_GetAllShifterStatus()

```
static uint32_t FLEXIO_HAL_GetAllShifterStatus (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current status flags for all shifters.

This function returns the value of the status flags for all shifters. Each bit in the returned value specifies the status for one shifter, starting with shifter 0 from least significant bit. The meaning of the status flag depends on the current mode.

- Transmit mode: shifter buffer is empty and ready to accept more data
- Receive mode: shifter buffer is full and received data can be read from it
- Match Store mode: match occurred between shifter buffer and shifter
- Match Continuous mode: current match result between shifter buffer and shifter

Parameters

in	*baseAddr*	Base address of the FlexIO module.

Returns

Status of the shifters. Implements : FLEXIO_HAL_GetAllShifterStatus_Activity

3.36.4.8 FLEXIO_HAL_GetAllTimerInterrupt()

```
static uint32_t FLEXIO_HAL_GetAllTimerInterrupt (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current status of all the timer interrupts.

Returns the state of the interrupt for all timers. Each bit in the returned value specifies the interrupt state for one timer, starting with timer 0 from least significant bit.

Parameters

in	*baseAddr*	Base address of the FlexIO module.

Returns

Status of the interrupts. Implements : FLEXIO_HAL_GetAllTimerInterrupt_Activity

3.36.4.9 FLEXIO_HAL_GetAllTimerStatus()

```
static uint32_t FLEXIO_HAL_GetAllTimerStatus (
             const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current status of all timers.

This function returns the value of the status flags for all timers. Each bit in the returned value specifies the status for one timer, starting with timer 0 from least significant bit. The meaning of the status flag depends on the current mode.

- 8-bit counter mode: the timer status flag is set when the upper 8-bit counter equals zero and decrements. This also causes the counter to reload with the value in the compare register.
- 8-bit PWM mode: the upper 8-bit counter equals zero and decrements. This also causes the counter to reload with the value in the compare register.
- 16-bit counter mode: the 16-bit counter equals zero and decrements. This also causes the counter to reload with the value in the compare register.

Parameters

in	*baseAddr*	Base address of the FlexIO module.

Returns

Status of the timers. Implements : FLEXIO_HAL_GetAllTimerStatus_Activity

3.36.4.10 FLEXIO_HAL_GetDebugMode()

```
static bool FLEXIO_HAL_GetDebugMode (
             const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current debug mode setting.

This function returns the current debug mode setting for the FlexIO module

Parameters

in	*baseAddr*	Base address of the FlexIO module

Returns

The current debug mode setting. Implements : FLEXIO_HAL_GetDebugMode_Activity

3.36.4.11 FLEXIO_HAL_GetDozeMode()

```
static bool FLEXIO_HAL_GetDozeMode (
             const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current doze mode setting.

This function returns the current doze mode setting for the FlexIO module

Parameters

in	*baseAddr*	Base address of the FlexIO module

Returns

The current doze mode setting. Implements : FLEXIO_HAL_GetDozeMode_Activity

3.36.4.12 FLEXIO_HAL_GetEnable()

```
static bool FLEXIO_HAL_GetEnable (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current enable/disable setting of the FlexIO.

This function checks whether or not the FlexIO module is enabled.

Parameters

in	*baseAddr*	Base address of the FlexIO module

Returns

The current enable/disable setting. Implements : FLEXIO_HAL_GetEnable_Activity

3.36.4.13 FLEXIO_HAL_GetFastAccess()

```
static bool FLEXIO_HAL_GetFastAccess (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current fast access setting.

This function returns the current fast access setting for the FlexIO module. Fast access allows faster accesses to FlexIO registers, but requires the FlexIO clock to be at least twice the frequency of the bus clock.

Parameters

in	*baseAddr*	Base address of the FlexIO module

Returns

The current fast access setting. Implements : FLEXIO_HAL_GetFastAccess_Activity

3.36.4.14 FLEXIO_HAL_GetPinData()

```
static uint8_t FLEXIO_HAL_GetPinData (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current input data read from the FlexIO pins.

This function returns the data read from all the FLEXIO pins. Only the lower n bits are valid, where n is the number of pins returned by FLEXIO_HAL_GetPinNum().

Parameters

in	*baseAddr*	Base address of the FlexIO module.

Returns

Data read from the FlexIO pins. Implements : FLEXIO_HAL_GetPinData_Activity

3.36.4.15 FLEXIO_HAL_GetPinNum()

```
static uint8_t FLEXIO_HAL_GetPinNum (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the number of pins of the FlexIO module.

This function returns the number of pins of the FlexIO module

Parameters

in	*baseAddr*	Base address of the FlexIO module

Returns

the number of pins of the FlexIO module. Implements : FLEXIO_HAL_GetPinNum_Activity

3.36.4.16 FLEXIO_HAL_GetShifterDMARequest()

```
static bool FLEXIO_HAL_GetShifterDMARequest (
            const FLEXIO_Type * baseAddr,
            uint8_t requestNo )  [inline], [static]
```

Returns the current status of the shifter DMA requests.

Returns the state of the DMA request for the specified shifter.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*requestNo*	Number of the DMA request.

Returns

Status of the specified DMA request. Implements : FLEXIO_HAL_GetShifterDMARequest_Activity

3.36.4.17 FLEXIO_HAL_GetShifterErrorInterrupt()

```
static bool FLEXIO_HAL_GetShifterErrorInterrupt (
            const FLEXIO_Type * baseAddr,
            uint8_t interruptNo )  [inline], [static]
```

Returns the current status of the shifter error interrupts.

Returns the state of the error interrupt for the specified shifter.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*interruptNo*	Number of the interrupt.

Returns

Status of the specified interrupt. Implements : FLEXIO_HAL_GetShifterErrorInterrupt_Activity

3.36.4.18 FLEXIO_HAL_GetShifterErrorStatus()

```
static bool FLEXIO_HAL_GetShifterErrorStatus (
            const FLEXIO_Type * baseAddr,
            uint8_t shifter )  [inline], [static]
```

Returns the current error status of the specified shifter.

This function returns the value of the error status flag for the specified shifter. The meaning of the error status flag depends on the current mode.

- Transmit mode: shifter buffer was not written before it was transferred in the shifter (buffer overrun)
- Receive mode: shifter buffer was not read before new data was transferred from the shifter (buffer underrun)
- Match Store mode: match event occurred before the previous match data was read from shifter buffer (buffer overrun)
- Match Continuous mode: match occurred between shifter buffer and shifter

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.

Returns

Error status of the specified shifter. Implements : FLEXIO_HAL_GetShifterErrorStatus_Activity

3.36.4.19 FLEXIO_HAL_GetShifterInputSource()

```
static flexio_shifter_source_t FLEXIO_HAL_GetShifterInputSource (
            const FLEXIO_Type * baseAddr,
            uint8_t shifter )  [inline], [static]
```

Returns the input source of the specified shifter.

This function returns the input source of the specified shifter.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.

Returns

Input source selected for the shifter. Implements : FLEXIO_HAL_GetShifterInputSource_Activity

3.36.4.20 FLEXIO_HAL_GetShifterInterrupt()

```
static bool FLEXIO_HAL_GetShifterInterrupt (
            const FLEXIO_Type * baseAddr,
            uint8_t interruptNo )  [inline], [static]
```

Returns the current status of the shifter interrupts.

Returns the state of the interrupt for the specified shifter.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*interruptNo*	Number of the interrupt (shifter).

Returns

Status of the specified interrupt. Implements : FLEXIO_HAL_GetShifterInterrupt_Activity

3.36.4.21 FLEXIO_HAL_GetShifterMode()

```
static flexio_shifter_mode_t FLEXIO_HAL_GetShifterMode (
            const FLEXIO_Type * baseAddr,
            uint8_t shifter )  [inline], [static]
```

Returns the mode of the specified shifter.

This function returns the currently configured mode for the specified shifter.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.

Returns

Mode assigned to the shifter. Implements : FLEXIO_HAL_GetShifterMode_Activity

3.36.4.22 FLEXIO_HAL_GetShifterNum()

```
static uint8_t FLEXIO_HAL_GetShifterNum (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the number of shifters of the FlexIO module.

This function returns the number of shifters of the FlexIO module

Parameters

in	*baseAddr*	Base address of the FlexIO module

Returns

the number of shifters of the FlexIO module. Implements : FLEXIO_HAL_GetShifterNum_Activity

3.36.4.23 FLEXIO_HAL_GetShifterPin()

```
static void FLEXIO_HAL_GetShifterPin (
            const FLEXIO_Type * baseAddr,
            uint8_t shifter,
            uint8_t * pin,
            flexio_pin_polarity_t * polarity,
            flexio_pin_config_t * config )  [inline], [static]
```

Returns the pin currently assigned to the specified shifter, and its configured settings.

This function returns the pin currently assigned to the specified shifter, and also its polarity and configuration.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
out	*pin*	Number of the pin.
out	*polarity*	Polarity of the pin.
out	*config*	Pin configuration. Implements : FLEXIO_HAL_GetShifterPin_Activity

3.36.4.24 FLEXIO_HAL_GetShifterStartBit()

```
static flexio_shifter_start_t FLEXIO_HAL_GetShifterStartBit (
            const FLEXIO_Type * baseAddr,
            uint8_t shifter )  [inline], [static]
```

Returns the start bit configuration for the specified shifter.

This function returns the current configuration for sending or receiving a start bit in Transmit, Receive or Match Store modes.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.

Returns

Start bit configuration for the shifter. Implements : FLEXIO_HAL_GetShifterStartBit_Activity

3.36.4.25 FLEXIO_HAL_GetShifterStatus()

```
static bool FLEXIO_HAL_GetShifterStatus (
            const FLEXIO_Type * baseAddr,
            uint8_t shifter )  [inline], [static]
```

Returns the current status of the specified shifter.

This function returns the value of the status flag for the specified shifter. The meaning of the status flag depends on the current mode.

- Transmit mode: shifter buffer is empty and ready to accept more data
- Receive mode: shifter buffer is full and received data can be read from it
- Match Store mode: match occurred between shifter buffer and shifter
- Match Continuous mode: current match result between shifter buffer and shifter

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.

Returns

Status of the specified shifter. Implements : FLEXIO_HAL_GetShifterStatus_Activity

3.36.4.26 FLEXIO_HAL_GetShifterStopBit()

```
static flexio_shifter_stop_t FLEXIO_HAL_GetShifterStopBit (
            const FLEXIO_Type * baseAddr,
            uint8_t shifter )  [inline], [static]
```

Returns the stop bit configuration for the specified shifter.

This function returns the current configuration for sending or receiving a stop bit in Transmit, Receive or Match Store modes.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.

Returns

Stop bit configuration for the shifter. Implements : FLEXIO_HAL_GetShifterStopBit_Activity

3.36.4.27 FLEXIO_HAL_GetShifterTimer()

```
static void FLEXIO_HAL_GetShifterTimer (
             const FLEXIO_Type * baseAddr,
             uint8_t shifter,
             uint8_t * timer,
             flexio_timer_polarity_t * polarity )  [inline], [static]
```

Returns the timer currently assigned to control the specified shifter.

This function returns the timer currently assigned to control the specified shifter, and also its polarity.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
out	*timer*	Number of the timer.
out	*polarity*	Polarity of the timer. Implements : FLEXIO_HAL_GetShifterTimer_Activity

3.36.4.28 FLEXIO_HAL_GetSoftwareReset()

```
static bool FLEXIO_HAL_GetSoftwareReset (
             const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the current software reset state.

This function returns the state of the FlexIO software reset bit.

Parameters

in	*baseAddr*	Base address of the FlexIO module

Returns

The current software reset setting. Implements : FLEXIO_HAL_GetSoftwareReset_Activity

3.36.4.29 FLEXIO_HAL_GetTimerCompare()

```
static uint16_t FLEXIO_HAL_GetTimerCompare (
             const FLEXIO_Type * baseAddr,
             uint8_t timer )  [inline], [static]
```

Returns the compare value of the specified timer.

This function returns the compare value currently set for the specified timer.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.

Returns

Compare value for the specified timer. Implements : FLEXIO_HAL_GetTimerCompare_Activity

3.36.4.30 FLEXIO_HAL_GetTimerDecrement()

```
static flexio_timer_decrement_t FLEXIO_HAL_GetTimerDecrement (
            const FLEXIO_Type * baseAddr,
            uint8_t timer )  [inline], [static]
```

Returns the decrement configuration of the specified timer.

This function returns the decrement configuration for the specified timer. See description of type flexio_timer_↩ decrement_t for a list of options.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.

Returns

Decrement configuration for the timer. Implements : FLEXIO_HAL_GetTimerDecrement_Activity

3.36.4.31 FLEXIO_HAL_GetTimerDisable()

```
static flexio_timer_disable_t FLEXIO_HAL_GetTimerDisable (
            const FLEXIO_Type * baseAddr,
            uint8_t timer )  [inline], [static]
```

Returns the disable configuration of the specified timer.

This function returns the condition that cause the specified timer to be disabled. See description of type flexio_↩ timer_disable_t for a list of options.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.

Returns

Disable configuration for the timer. Implements : FLEXIO_HAL_GetTimerDisable_Activity

3.36.4.32 FLEXIO_HAL_GetTimerEnable()

```
static flexio_timer_enable_t FLEXIO_HAL_GetTimerEnable (
            const FLEXIO_Type * baseAddr,
            uint8_t timer )  [inline], [static]
```

Returns the enable condition configuration of the specified timer.

This function returns the condition that cause the specified timer to be enabled and start decrementing. See description of type flexio_timer_disable_t for a list of options.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.

Returns

Enable condition configuration for the timer. Implements : FLEXIO_HAL_GetTimerEnable_Activity

3.36.4.33 FLEXIO_HAL_GetTimerInitialOutput()

```
static flexio_timer_output_t FLEXIO_HAL_GetTimerInitialOutput (
            const FLEXIO_Type * baseAddr,
            uint8_t timer )  [inline], [static]
```

Returns the initial output configuration of the specified timer.

This function returns the initial output configuration of the specified timer.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.

Returns

Output configuration for the timer. Implements : FLEXIO_HAL_GetTimerInitialOutput_Activity

3.36.4.34 FLEXIO_HAL_GetTimerInterrupt()

```
static bool FLEXIO_HAL_GetTimerInterrupt (
            const FLEXIO_Type * baseAddr,
            uint8_t interruptNo )  [inline], [static]
```

Returns the current status of the timer interrupts.

Returns the state of the interrupt for the specified timer.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*interruptNo*	Number of the interrupt.

Returns

Status of the specified interrupt. Implements : FLEXIO_HAL_GetTimerInterrupt_Activity

3.36.4.35 FLEXIO_HAL_GetTimerMode()

```
static flexio_timer_mode_t FLEXIO_HAL_GetTimerMode (
            const FLEXIO_Type * baseAddr,
            uint8_t timer )  [inline], [static]
```

Returns the mode of the specified timer.

This function returns the currently configured mode for the specified timer.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.

Returns

Mode assigned to the timer. Implements : FLEXIO_HAL_GetTimerMode_Activity

3.36.4.36 FLEXIO_HAL_GetTimerNum()

```
static uint8_t FLEXIO_HAL_GetTimerNum (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the number of timers of the FlexIO module.

This function returns the number of timers of the FlexIO module

Parameters

in	*baseAddr*	Base address of the FlexIO module

Returns

the number of timers of the FlexIO module. Implements : FLEXIO_HAL_GetTimerNum_Activity

3.36.4.37 FLEXIO_HAL_GetTimerPin()

```
static void FLEXIO_HAL_GetTimerPin (
            const FLEXIO_Type * baseAddr,
            uint8_t timer,
            uint8_t * pin,
            flexio_pin_polarity_t * polarity,
            flexio_pin_config_t * config )  [inline], [static]
```

Returns the currently configured pin for the specified timer.

This function returns the pin currently assigned to the specified timer, and also its polarity and configuration.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
out	*pin*	Number of the pin.
out	*polarity*	Polarity of the pin.
out	*config*	Pin configuration. Implements : FLEXIO_HAL_GetTimerPin_Activity

3.36.4.38 FLEXIO_HAL_GetTimerReset()

```
static flexio_timer_reset_t FLEXIO_HAL_GetTimerReset (
            const FLEXIO_Type * baseAddr,
            uint8_t timer )  [inline], [static]
```

Returns the reset configuration of the specified timer.

This function returns the reset configuration for the specified timer. See description of type flexio_timer_reset_t for a list of options.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.

Returns

Reset configuration for the timer. Implements : FLEXIO_HAL_GetTimerReset_Activity

3.36.4.39 FLEXIO_HAL_GetTimerStart()

```
static flexio_timer_start_t FLEXIO_HAL_GetTimerStart (
            const FLEXIO_Type * baseAddr,
            uint8_t timer )  [inline], [static]
```

Returns the start bit configuration of the specified timer.

This function returns the current start bit configuration for the specified timer. When start bit is enabled, configured shifters will output the contents of the start bit when the timer is enabled and the timer counter will reload from the compare register on the first rising edge of the shift clock.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.

Returns

Start bit configuration for the timer. Implements : FLEXIO_HAL_GetTimerStart_Activity

3.36.4.40 FLEXIO_HAL_GetTimerStatus()

```
static bool FLEXIO_HAL_GetTimerStatus (
            const FLEXIO_Type * baseAddr,
            uint8_t timer )  [inline], [static]
```

Returns the current status of the specified timer.

This function returns the value of the status flag for the specified timer. The meaning of the status flag depends on the current mode.

- 8-bit counter mode: the timer status flag is set when the upper 8-bit counter equals zero and decrements. This also causes the counter to reload with the value in the compare register.
- 8-bit PWM mode: the upper 8-bit counter equals zero and decrements. This also causes the counter to reload with the value in the compare register.
- 16-bit counter mode: the 16-bit counter equals zero and decrements. This also causes the counter to reload with the value in the compare register.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.

Returns

Status of the specified timer. Implements : FLEXIO_HAL_GetTimerStatus_Activity

3.36.4.41 FLEXIO_HAL_GetTimerStop()

```
static flexio_timer_stop_t FLEXIO_HAL_GetTimerStop (
            const FLEXIO_Type * baseAddr,
            uint8_t timer ) [inline], [static]
```

Returns the stop bit configuration of the specified timer.

This function returns the current stop bit configuration for the specified timer. The stop bit can be added on a timer compare (between each word) or on a timer disable. When stop bit is enabled, configured shifters will output the contents of the stop bit when the timer is disabled. When stop bit is enabled on timer disable, the timer remains disabled until the next rising edge of the shift clock. If configured for both timer compare and timer disable, only one stop bit is inserted on timer disable.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.

Returns

Stop bit configuration for the timer. Implements : FLEXIO_HAL_GetTimerStop_Activity

3.36.4.42 FLEXIO_HAL_GetTimerTrigger()

```
static void FLEXIO_HAL_GetTimerTrigger (
            const FLEXIO_Type * baseAddr,
            uint8_t timer,
            uint8_t * trigger,
            flexio_trigger_polarity_t * polarity,
            flexio_trigger_source_t * source ) [inline], [static]
```

Returns the currently configured trigger for the specified timer.

This function returns the currently configured trigger for the specified timer, and also its source (internal or external) and polarity settings.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
out	*trigger*	Number of the trigger.
out	*polarity*	Polarity of the trigger.
out	*source*	Trigger source. Implements : FLEXIO_HAL_GetTimerTrigger_Activity

3.36.4.43 FLEXIO_HAL_GetTriggerNum()

```
static uint8_t FLEXIO_HAL_GetTriggerNum (
            const FLEXIO_Type * baseAddr )  [inline], [static]
```

Returns the number of external triggers of the FlexIO module.

This function returns the number of external triggers of the FlexIO module

Parameters

in	*baseAddr*	Base address of the FlexIO module

Returns

the number of external triggers of the FlexIO module. Implements : FLEXIO_HAL_GetTriggerNum_Activity

3.36.4.44 FLEXIO_HAL_GetVersion()

```
static void FLEXIO_HAL_GetVersion (
            const FLEXIO_Type * baseAddr,
            flexio_version_info_t * versionInfo )  [inline], [static]
```

Reads the version of the FlexIO module.

This function reads the version number of the FLEXIO hardware module

Parameters

in	*baseAddr*	Base address of the FlexIO module
out	*versionInfo*	Device Version Number Implements : FLEXIO_HAL_GetVersion_Activity

3.36.4.45 FLEXIO_HAL_Init()

```
void FLEXIO_HAL_Init (
            FLEXIO_Type * baseAddr )
```

Initializes the FlexIO module to a known state.

This function initializes all the registers of the FlexIO module to their reset value.

Parameters

in	*baseAddr*	Base address of the FlexIO module.

3.36.4.46 FLEXIO_HAL_ReadShifterBuffer()

```
static uint32_t FLEXIO_HAL_ReadShifterBuffer (
             const FLEXIO_Type * baseAddr,
             uint8_t shifter,
             flexio_shifter_buffer_mode_t mode )  [inline], [static]
```

Reads the value from the specified shifter buffer.

This function reads data from the specified shifter buffer. The data can be read in any of the four ways allowed by the hardware - see description of type flexio_shifter_buffer_mode_t for details.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*mode*	Read mode.

Returns

Value read from the shifter buffer. Implements : FLEXIO_HAL_ReadShifterBuffer_Activity

3.36.4.47 FLEXIO_HAL_SetDebugMode()

```
static void FLEXIO_HAL_SetDebugMode (
             FLEXIO_Type * baseAddr,
             bool enable )  [inline], [static]
```

Set the FlexIO module behavior in debug mode.

This function enables of disables FlexIO functionality in debug mode.

Parameters

in	*baseAddr*	Base address of the FlexIO module
in	*enable*	Specifies whether to enable or disable FlexIO in debug mode Implements : FLEXIO_HAL_SetDebugMode_Activity

3.36.4.48 FLEXIO_HAL_SetDozeMode()

```
static void FLEXIO_HAL_SetDozeMode (
             FLEXIO_Type * baseAddr,
             bool enable )  [inline], [static]
```

Set the FlexIO module behavior in doze mode.

This function enables of disables FlexIO functionality in doze mode.

Parameters

in	*baseAddr*	Base address of the FlexIO module
in	*enable*	Specifies whether to enable or disable FlexIO in doze mode Implements : FLEXIO_HAL_SetDozeMode_Activity

3.36.4.49 FLEXIO_HAL_SetEnable()

```
static void FLEXIO_HAL_SetEnable (
            FLEXIO_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enables of disables the FlexIO module.

This function enables or disables the FlexIO module.

Parameters

in	*baseAddr*	Base address of the FlexIO module
in	*enable*	Specifies the enable/disable state of the FlexIO module Implements : FLEXIO_HAL_SetEnable_Activity

3.36.4.50 FLEXIO_HAL_SetFastAccess()

```
static void FLEXIO_HAL_SetFastAccess (
            FLEXIO_Type * baseAddr,
            bool enable )  [inline], [static]
```

Configure the FlexIO fast access feature.

This function enables of disables the fast access feature for the FlexIO module. Fast access allows faster accesses to FlexIO registers, but requires the FlexIO clock to be at least twice the frequency of the bus clock.

Parameters

in	*baseAddr*	Base address of the FlexIO module
in	*enable*	Enables fast register accesses to FlexIO registers Implements : FLEXIO_HAL_SetFastAccess_Activity

3.36.4.51 FLEXIO_HAL_SetShifterConfig()

```
static void FLEXIO_HAL_SetShifterConfig (
            FLEXIO_Type * baseAddr,
            uint8_t shifter,
            flexio_shifter_start_t start,
            flexio_shifter_stop_t stop,
            flexio_shifter_source_t source )  [inline], [static]
```

Sets all configuration settings for specified shifter.

This function sets the following configurations for the specified shifter: start bit, stop bit, input source

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*start*	Start bit configuration for the shifter.
in	*stop*	Stop bit configuration for the shifter.
in	*source*	Input source selected for the shifter. Implements : FLEXIO_HAL_SetShifterConfig_Activity

3.36.4.52 FLEXIO_HAL_SetShifterControl()

```
static void FLEXIO_HAL_SetShifterControl (
              FLEXIO_Type * baseAddr,
              uint8_t shifter,
              flexio_shifter_mode_t mode,
              uint8_t pin,
              flexio_pin_polarity_t pinPolarity,
              flexio_pin_config_t pinConfig,
              uint8_t timer,
              flexio_timer_polarity_t timerPolarity )  [inline], [static]
```

Sets all control settings for the specified shifter.

This function configures the control settings for the specified shifter: mode settings, pin settings and timer settings. It sums up all settings from FLEXIO_HAL_SetShifterMode(), FLEXIO_HAL_SetShifterPin() and FLEXIO_HAL_↩ SetShifterTimer()

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*mode*	Mode assigned to the shifter.
in	*pin*	Number of the pin.
in	*pinPolarity*	Polarity of the pin.
in	*pinConfig*	Pin configuration.
in	*timer*	Number of the timer.
in	*timerPolarity*	Polarity of the timer. Implements : FLEXIO_HAL_SetShifterControl_Activity

3.36.4.53 FLEXIO_HAL_SetShifterDMARequest()

```
static void FLEXIO_HAL_SetShifterDMARequest (
              FLEXIO_Type * baseAddr,
              uint8_t requestMask,
              bool enable )  [inline], [static]
```

Enables or disables the shifter DMA requests.

Enable or disable specified shifter DMA requests. The request mask must contain a bit of 1 for each shifter who's DMA requests must be enabled or disabled.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*requestMask*	Mask of the DMA requests to be enabled/disabled.
in	*enable*	Specifies whether to enable or disable specified DMA requests. Implements : FLEXIO_HAL_SetShifterDMARequest_Activity

3.36.4.54 FLEXIO_HAL_SetShifterErrorInterrupt()

```
static void FLEXIO_HAL_SetShifterErrorInterrupt (
            FLEXIO_Type * baseAddr,
            uint8_t interruptMask,
            bool enable )  [inline], [static]
```

Enables or disables the shifter error interrupts.

Enable or disable specified shifter error interrupts. The interrupt mask must contain a bit of 1 for each shifter who's error interrupt must be enabled or disabled.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*interruptMask*	Mask of the interrupts to be enabled/disabled.
in	*enable*	Specifies whether to enable or disable specified interrupts. Implements : FLEXIO_HAL_SetShifterErrorInterrupt_Activity

3.36.4.55 FLEXIO_HAL_SetShifterInputSource()

```
static void FLEXIO_HAL_SetShifterInputSource (
            FLEXIO_Type * baseAddr,
            uint8_t shifter,
            flexio_shifter_source_t source )  [inline], [static]
```

Configures the input source of the specified shifter.

This function configures the input source of the specified shifter.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*source*	Input source selected for the shifter. Implements : FLEXIO_HAL_SetShifterInputSource_Activity

3.36.4.56 FLEXIO_HAL_SetShifterInterrupt()

```
static void FLEXIO_HAL_SetShifterInterrupt (
            FLEXIO_Type * baseAddr,
            uint8_t interruptMask,
            bool enable )  [inline], [static]
```

Enables or disables the shifter interrupts.

Enable or disable specified shifter interrupts. The interrupt mask must contain a bit of 1 for each shifter who's interrupt must be enabled or disabled.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*interruptMask*	Mask of the interrupts to be enabled/disabled.
in	*enable*	Specifies whether to enable or disable specified interrupts. Implements : FLEXIO_HAL_SetShifterInterrupt_Activity

3.36.4.57 FLEXIO_HAL_SetShifterMode()

```
static void FLEXIO_HAL_SetShifterMode (
            FLEXIO_Type * baseAddr,
            uint8_t shifter,
            flexio_shifter_mode_t mode )  [inline], [static]
```

Sets the mode of the specified shifter.

This function configures the mode for the specified shifter.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*mode*	Mode assigned to the shifter. Implements : FLEXIO_HAL_SetShifterMode_Activity

3.36.4.58 FLEXIO_HAL_SetShifterPin()

```
static void FLEXIO_HAL_SetShifterPin (
            FLEXIO_Type * baseAddr,
            uint8_t shifter,
            uint8_t pin,
            flexio_pin_polarity_t polarity,
            flexio_pin_config_t config )  [inline], [static]
```

Assigns the specified pin to the specified shifter.

This function assigns the specified pin to the specified shifter, and also sets its polarity and configuration.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*pin*	Number of the pin.
in	*polarity*	Polarity of the pin.
in	*config*	Pin configuration. Implements : FLEXIO_HAL_SetShifterPin_Activity

3.36.4.59 FLEXIO_HAL_SetShifterPinConfig()

```
static void FLEXIO_HAL_SetShifterPinConfig (
            FLEXIO_Type * baseAddr,
            uint8_t shifter,
            flexio_pin_config_t config )  [inline], [static]
```

Configures the pin assigned to the specified shifter.

This function configures the pin assigned to the specified specified shifter. It does not change the other pin-related settings.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*config*	Pin configuration. Implements : FLEXIO_HAL_SetShifterPinConfig_Activity

3.36.4.60 FLEXIO_HAL_SetShifterStartBit()

```
static void FLEXIO_HAL_SetShifterStartBit (
             FLEXIO_Type * baseAddr,
             uint8_t shifter,
             flexio_shifter_start_t start )  [inline], [static]
```

Configures the start bit of the specified shifter.

This function configures the sending or receiving of a start bit in Transmit, Receive or Match Store modes.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*start*	Start bit configuration for the shifter. Implements : FLEXIO_HAL_SetShifterStartBit_Activity

3.36.4.61 FLEXIO_HAL_SetShifterStopBit()

```
static void FLEXIO_HAL_SetShifterStopBit (
             FLEXIO_Type * baseAddr,
             uint8_t shifter,
             flexio_shifter_stop_t stop )  [inline], [static]
```

Configures the stop bit of the specified shifter.

This function configures the sending or receiving of a stop bit in Transmit, Receive or Match Store modes.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*stop*	Stop bit configuration for the shifter. Implements : FLEXIO_HAL_SetShifterStopBit_Activity

3.36.4.62 FLEXIO_HAL_SetShifterTimer()

```
static void FLEXIO_HAL_SetShifterTimer (
             FLEXIO_Type * baseAddr,
             uint8_t shifter,
             uint8_t timer,
             flexio_timer_polarity_t polarity )  [inline], [static]
```

Assigns the specified timer to control the specified shifter.

This function assigns a timer to control the specified shifter, and also configures its polarity.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*timer*	Number of the timer.
in	*polarity*	Polarity of the timer. Implements : FLEXIO_HAL_SetShifterTimer_Activity

3.36.4.63 FLEXIO_HAL_SetSoftwareReset()

```
static void FLEXIO_HAL_SetSoftwareReset (
            FLEXIO_Type * baseAddr,
            bool enable )  [inline], [static]
```

Set/clear the FlexIO reset command.

Calling this function with enable parameter set to true resets all internal master logic and registers, except the FlexIO Control Register. The reset state persists until this function is called with enable parameter set to false.

Parameters

in	*baseAddr*	Base address of the FlexIO module
in	*enable*	Specifies the reset state of the FlexIO logic Implements : FLEXIO_HAL_SetSoftwareReset_Activity

3.36.4.64 FLEXIO_HAL_SetTimerCompare()

```
static void FLEXIO_HAL_SetTimerCompare (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            uint16_t value )  [inline], [static]
```

Configures the compare value for the specified timer.

This function configures the compare value for the specified timer. The timer compare value is loaded into the timer counter when the timer is first enabled, when the timer is reset and when the timer decrements down to zero. In dual 8-bit counters baud/bit mode, the lower 8-bits configure the baud rate divider and the upper 8-bits configure the number of bits in each word. In dual 8-bit counters PWM mode, the lower 8-bits configure the high period of the output and the upper 8-bits configure the low period. In 16-bit counter mode, the compare value can be used to generate the baud rate divider (if shift clock source is timer output) or the number of bits in each word (when the shift clock source is a pin or trigger input).

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*value*	Compare value for the specified timer. Implements : FLEXIO_HAL_SetTimerCompare_Activity

3.36.4.65 FLEXIO_HAL_SetTimerConfig()

```
static void FLEXIO_HAL_SetTimerConfig (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            flexio_timer_start_t start,
            flexio_timer_stop_t stop,
            flexio_timer_enable_t enable,
            flexio_timer_disable_t disable,
            flexio_timer_reset_t reset,
            flexio_timer_decrement_t decrement,
            flexio_timer_output_t output )  [inline], [static]
```

Sets all configuration settings for specified timer.

This function sets the following configurations for the specified timer: start bit, stop bit, enable condition, disable condition, reset condition, decrement condition, initial output

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*start*	Start bit configuration for the timer.
in	*stop*	Stop bit configuration for the timer.
in	*enable*	Enable condition configuration for the timer.
in	*disable*	Disable configuration for the timer.
in	*reset*	Reset configuration for the timer.
in	*decrement*	Decrement configuration for the timer.
in	*output*	Output configuration for the timer. Implements : FLEXIO_HAL_SetTimerConfig_Activity

3.36.4.66 FLEXIO_HAL_SetTimerControl()

```
static void FLEXIO_HAL_SetTimerControl (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            uint8_t trigger,
            flexio_trigger_polarity_t triggerPolarity,
            flexio_trigger_source_t triggerSource,
            uint8_t pin,
            flexio_pin_polarity_t pinPolarity,
            flexio_pin_config_t pinConfig,
            flexio_timer_mode_t mode )  [inline], [static]
```

Sets all control settings for specified timer.

This function configures the control settings for the specified timer: mode settings, pin settings and trigger settings. It sums up all settings from FLEXIO_HAL_SetTimerMode(), FLEXIO_HAL_SetTimerPin() and FLEXIO_HAL_Set←TimerTrigger()

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*trigger*	Number of the trigger.
in	*triggerPolarity*	Polarity of the trigger.
in	*triggerSource*	Trigger source.
in	*pin*	Number of the pin.
in	*pinPolarity*	Polarity of the pin.
in	*pinConfig*	Pin configuration.
in	*mode*	Mode assigned to the timer. Implements : FLEXIO_HAL_SetTimerControl_Activity

3.36.4.67 FLEXIO_HAL_SetTimerDecrement()

```
static void FLEXIO_HAL_SetTimerDecrement (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            flexio_timer_decrement_t decrement )  [inline], [static]
```

Configures the decrement condition for the specified timer.

This function configures the decrement condition for the specified timer and the source of the shift clock. See description of type flexio_timer_decrement_t for a list of options.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*decrement*	Decrement configuration for the timer. Implements : FLEXIO_HAL_SetTimerDecrement_Activity

3.36.4.68 FLEXIO_HAL_SetTimerDisable()

```
static void FLEXIO_HAL_SetTimerDisable (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            flexio_timer_disable_t disable )  [inline], [static]
```

Configures the disable condition for the specified timer.

This function configures the condition that cause the specified timer to be disabled. See description of type flexio↩
_timer_disable_t for a list of options.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*disable*	Disable configuration for the timer. Implements : FLEXIO_HAL_SetTimerDisable_Activity

3.36.4.69 FLEXIO_HAL_SetTimerEnable()

```
static void FLEXIO_HAL_SetTimerEnable (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            flexio_timer_enable_t enable )  [inline], [static]
```

Configures the enable condition for the specified timer.

This function configures the condition that cause the specified timer to be enabled and start decrementing. See description of type flexio_timer_disable_t for a list of options.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*enable*	Enable condition configuration for the timer. Implements : FLEXIO_HAL_SetTimerEnable_Activity

3.36.4.70 FLEXIO_HAL_SetTimerInitialOutput()

```
static void FLEXIO_HAL_SetTimerInitialOutput (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            flexio_timer_output_t output )  [inline], [static]
```

Configures the initial output of the specified timer.

This function configures the initial output of the specified timer and whether it is affected by the Timer reset.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*output*	Output configuration for the timer. Implements : FLEXIO_HAL_SetTimerInitialOutput_Activity

3.36.4.71 FLEXIO_HAL_SetTimerInterrupt()

```
static void FLEXIO_HAL_SetTimerInterrupt (
            FLEXIO_Type * baseAddr,
            uint8_t interruptMask,
            bool enable )  [inline], [static]
```

Enables or disables the timer interrupts.

Enable or disable specified timer interrupts. The interrupt mask must contain a bit of 1 for each timer who's interrupt must be enabled or disabled.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*interruptMask*	Mask of the interrupts to be enabled/disabled.
in	*enable*	Specifies whether to enable or disable specified interrupts. Implements : FLEXIO_HAL_SetTimerInterrupt_Activity

3.36.4.72 FLEXIO_HAL_SetTimerMode()

```
static void FLEXIO_HAL_SetTimerMode (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            flexio_timer_mode_t mode )  [inline], [static]
```

Sets the mode of the specified timer.

This function configures the mode for the specified timer. In 8-bit counter mode, the lower 8-bits of the counter and compare register are used to configure the baud rate of the timer shift clock and the upper 8-bits are used to configure the shifter bit count. In 8-bit PWM mode, the lower 8-bits of the counter and compare register are used to configure the high period of the timer shift clock and the upper 8-bits are used to configure the low period of the timer shift clock. The shifter bit count is configured using another timer or external signal. In 16-bit counter mode, the full 16-bits of the counter and compare register are used to configure either the baud rate of the shift clock or the shifter bit count.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*mode*	Mode assigned to the timer. Implements : FLEXIO_HAL_SetTimerMode_Activity

3.36.4.73 FLEXIO_HAL_SetTimerPin()

```
static void FLEXIO_HAL_SetTimerPin (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            uint8_t pin,
            flexio_pin_polarity_t polarity,
            flexio_pin_config_t config )  [inline], [static]
```

Configures the pin for the specified timer.

This function assigns the specified pin to the specified timer, and also sets its polarity and configuration.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*pin*	Number of the pin.
in	*polarity*	Polarity of the pin.
in	*config*	Pin configuration. Implements : FLEXIO_HAL_SetTimerPin_Activity

3.36.4.74 FLEXIO_HAL_SetTimerReset()

```
static void FLEXIO_HAL_SetTimerReset (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            flexio_timer_reset_t reset )  [inline], [static]
```

Configures the reset condition for the specified timer.

This function configures the conditions that cause the timer counter (and optionally output) to be reset. In 8-bit counter mode, the timer reset will only reset the lower 8-bits that configure the baud rate. In all other modes, the timer reset will reset the full 16-bits of the counter. See description of type flexio_timer_decrement_t for a list of options.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*reset*	Reset configuration for the timer. Implements : FLEXIO_HAL_SetTimerReset_Activity

3.36.4.75 FLEXIO_HAL_SetTimerStart()

```
static void FLEXIO_HAL_SetTimerStart (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            flexio_timer_start_t start )  [inline], [static]
```

Configures the start bit for the specified timer.

This function configures start bit insertion for the specified timer. When start bit is enabled, configured shifters will output the contents of the start bit when the timer is enabled and the timer counter will reload from the compare register on the first rising edge of the shift clock.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*start*	Start bit configuration for the timer. Implements : FLEXIO_HAL_SetTimerStart_Activity

3.36.4.76 FLEXIO_HAL_SetTimerStop()

```
static void FLEXIO_HAL_SetTimerStop (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            flexio_timer_stop_t stop )  [inline], [static]
```

Configures the stop bit for the specified timer.

This function configures stop bit insertion for the specified timer. The stop bit can be added on a timer compare (between each word) or on a timer disable. When stop bit is enabled, configured shifters will output the contents of the stop bit when the timer is disabled. When stop bit is enabled on timer disable, the timer remains disabled until the next rising edge of the shift clock. If configured for both timer compare and timer disable, only one stop bit is inserted on timer disable.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*stop*	Stop bit configuration for the timer. Implements : FLEXIO_HAL_SetTimerStop_Activity

3.36.4.77 FLEXIO_HAL_SetTimerTrigger()

```
static void FLEXIO_HAL_SetTimerTrigger (
            FLEXIO_Type * baseAddr,
            uint8_t timer,
            uint8_t trigger,
            flexio_trigger_polarity_t polarity,
            flexio_trigger_source_t source )  [inline], [static]
```

Configures the trigger for the specified timer.

This function configures the trigger for the specified timer, and also its source (internal or external) and polarity settings. For internal triggers, the selection is as follows:

- 4∗N - Pin 2∗N input
- 4∗N+1 - Shifter N status flag
- 4∗N+2 - Pin 2∗N+1 input
- 4∗N+3 - Timer N trigger output

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*timer*	Number of the timer.
in	*trigger*	Number of the trigger.
in	*polarity*	Polarity of the trigger.
in	*source*	Trigger source. Implements : FLEXIO_HAL_SetTimerTrigger_Activity

3.36.4.78 FLEXIO_HAL_WriteShifterBuffer()

```
static void FLEXIO_HAL_WriteShifterBuffer (
             FLEXIO_Type * baseAddr,
             uint8_t shifter,
             uint32_t value,
             flexio_shifter_buffer_mode_t mode )  [inline], [static]
```

Writes a value in the specified shifter buffer.

This function writes data in the specified shifter buffer. The data can be written in any of the four ways allowed by the hardware - see description of type flexio_shifter_buffer_mode_t for details.

Parameters

in	*baseAddr*	Base address of the FlexIO module.
in	*shifter*	Number of the shifter.
in	*value*	Value to write in the shifter buffer.
in	*mode*	Write mode. Implements : FLEXIO_HAL_WriteShifterBuffer_Activity

3.37 FlexIO I2C Driver

3.37.1 Detailed Description

I2C communication over FlexIO module (FLEXIO_I2C)

The FLEXIO_I2C Driver allows communication on an I2C bus using the FlexIO module in the S32144K processor.

Features

- Master operation only
- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transmit and receive functions
- 7-bit addressing
- Clock stretching
- Configurable baud rate

Functionality

Before using any Flexio driver the device must first be initialized using function FLEXIO_DRV_InitDevice. Then the FLEXIO_I2C Driver must be initialized using functions FLEXIO_I2C_DRV_MasterInit(). It is possible to use more driver instances on the same FlexIO device, as long as sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using FLEXIO_I2C_DRV_MasterDeinit(). This will release the hardware resources, allowing other driver instances to be initialized.

Master Mode

Master Mode provides functions for transmitting or receiving data to/from any I2C slave. Slave address and baud rate are provided at initialization time through the master configuration structure, but they can be changed at runtime by using FLEXIO_I2C_DRV_MasterSetBaudRate() or FLEXIO_I2C_DRV_MasterSetSlaveAddr(). Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call FLEXIO_I2C_DRV_MasterGetBaudRate() after FLEXIO_I2C_DRV_MasterSetBaudRate() to check what baud rate was actually set.

To send or receive data to/from the currently configured slave address, use functions FLEXIO_I2C_DRV_Master↵ SendData() or FLEXIO_I2C_DRV_MasterReceiveData() (or their blocking counterparts). Parameter `sendStop` can be used to chain multiple transfers with repeated START condition between them, for example when sending a command and then immediately receiving a response. The application should ensure that any send or receive transfer with `sendStop` set to `false` is followed by another transfer. The last transfer from a chain should always have `sendStop` set to `true`. This driver does not support continuous send/receive using a user callback function. The callback function is only used to signal the end of a transfer.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return STATUS_SUCCESS, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application will be notified through the user callback when the transfer completes, or it can check the status of the current transfer by calling FLEXIO_I2↵ C_DRV_MasterGetStatus(). If the transfer is still ongoing this function will return STATUS_BUSY. If the transfer is completed, the function will return either STATUS_SUCCESS or an error code, depending on the outcome of the last transfer.

The driver supports interrupt, DMA and polling mode. In polling mode the function FLEXIO_I2C_DRV_Master↵ GetStatus() ensures the progress of the transfer by checking and handling transmit and receive events reported by the FlexIO module. The application should ensure that this function is called often enough (at least once per transferred byte) to avoid Tx underflows or Rx overflows. In DMA mode the DMA channels that will be used by the driver are received through the configuration structure. The channels must be initialized by the application before the flexio_i2c driver is initialized. The flexio_i2c driver will only set the DMA request source.

Important Notes

- Before using the FLEXIO_I2C Driver the FlexIO clock must be configured. Refer to SCG HAL and PCC HAL for clock configuration.
- Before using the FLEXIO_I2C Driver the pins must be routed to the FlexIO module. Refer to PORT HAL for pin routing configuration. Note that any of the available FlexIO pins can be used for SDA and SCL (configurable at initialization time).
- The driver enables the interrupts for the corresponding FlexIO module, but any interrupt priority setting must be done by the application.
- Aborting a transfer with the function FLEXIO_I2C_DRV_MasterTransferAbort() can't generally be done safely due to device limitation; there is no way to know the exact stage of the transfer, and if we disable the module during the ACK bit (transmit) or during a 0 data bit (receive) the slave will hold the SDA line low forever and block the I2C bus. Therefore this function should only be used in extreme circumstances, and the application must have a way to reset the I2C slave. NACK reception is the only exception, as there is no slave to hold the line low, so in this case the driver will automatically abort the transfer.
- The module can handle clock stretching done by the slave, but will not do clock stretching when the application does not provide data fast enough, so Tx underflows and Rx overflows are possible. This can be an issue especially in polling mode if the function FLEXIO_I2C_DRV_MasterGetStatus() is not called often enough.
- Due to device limitations it is not always possible to tell the difference between NACK reception and receiver overflow. When in doubt, the driver will treat these events as overflow and continue the transfer, in order to avoid the risk of blocking the i2c bus.
- Due to device limitations there is a maximum limit of 13 bytes (FLEXIO_I2C_MAX_SIZE) on the size of any transfer.
- The driver does not support multi-master mode. It does not detect arbitration loss condition.
- Timeout feature for blocking transfers does not work in polling mode.
- This driver needs two shifters and two timers for its operation. Initialization will fail if there are not enough shifters and timers available on the FlexIO device.
- This driver needs two DMA channels for its operation when it is initialized in DMA mode. The DMA channels must be initialized by the application before initializing the driver. Refer to EDMA driver for DMA channels initialization.
- If the application uses an RTOS, this driver uses a semaphore for blocking transfers. Initialization will fail if the semaphore cannot be created. If the driver uses polling mode no semaphore is used.
- If the application uses an RTOS, the FlexIO drivers use a mutex for channel allocation. Only one mutex per device is needed, not per driver instance. Device initialization will fail if the mutex cannot be created.

Data Structures

- struct flexio_i2c_master_user_config_t

  Master configuration structure. More...
- struct flexio_i2c_master_state_t

  Master internal context structure. More...

Macros

- #define FLEXIO_I2C_MAX_SIZE (((uint32_t)((0xFFU - 1U) / 18U)) - 1U)

  Maximum size of a transfer. The restriction is that the total number of SCL edges must not exceed 8 bits, such that it can be programmed in the upper part of the timer compare register. There are 2 SCL edges per bit, 9 bits per byte (including ACK). The extra 1 is for the STOP condition.

FLEXIO_I2C Driver

- status_t FLEXIO_I2C_DRV_MasterInit (uint32_t instance, const flexio_i2c_master_user_config_t ∗user←ConfigPtr, flexio_i2c_master_state_t ∗master)

  Initialize the FLEXIO_I2C master mode driver.
- status_t FLEXIO_I2C_DRV_MasterDeinit (flexio_i2c_master_state_t ∗master)

  De-initialize the FLEXIO_I2C master mode driver.
- status_t FLEXIO_I2C_DRV_MasterSetBaudRate (flexio_i2c_master_state_t ∗master, uint32_t baudRate)

  Set the baud rate for any subsequent I2C communication.
- status_t FLEXIO_I2C_DRV_MasterGetBaudRate (flexio_i2c_master_state_t ∗master, uint32_t ∗baudRate)

  Get the currently configured baud rate.
- status_t FLEXIO_I2C_DRV_MasterSetSlaveAddr (flexio_i2c_master_state_t ∗master, const uint16_t address)

  Set the slave address for any subsequent I2C communication.
- status_t FLEXIO_I2C_DRV_MasterSendData (flexio_i2c_master_state_t ∗master, const uint8_t ∗txBuff, uint32_t txSize, bool sendStop)

  Perform a non-blocking send transaction on the I2C bus.
- status_t FLEXIO_I2C_DRV_MasterSendDataBlocking (flexio_i2c_master_state_t ∗master, const uint8_←t ∗txBuff, uint32_t txSize, bool sendStop, uint32_t timeout)

  Perform a blocking send transaction on the I2C bus.
- status_t FLEXIO_I2C_DRV_MasterReceiveData (flexio_i2c_master_state_t ∗master, uint8_t ∗rxBuff, uint32_t rxSize, bool sendStop)

  Perform a non-blocking receive transaction on the I2C bus.
- status_t FLEXIO_I2C_DRV_MasterReceiveDataBlocking (flexio_i2c_master_state_t ∗master, uint8_t ∗rx←Buff, uint32_t rxSize, bool sendStop, uint32_t timeout)

  Perform a blocking receive transaction on the I2C bus.
- status_t FLEXIO_I2C_DRV_MasterTransferAbort (flexio_i2c_master_state_t ∗master)

  Aborts a non-blocking I2C master transaction.
- status_t FLEXIO_I2C_DRV_MasterGetStatus (flexio_i2c_master_state_t ∗master, uint32_t ∗bytes←Remaining)

  Get the status of the current non-blocking I2C master transaction.

3.37.2 Data Structure Documentation

3.37.2.1 struct flexio_i2c_master_user_config_t

Master configuration structure.

This structure is used to provide configuration parameters for the flexio_i2c master at initialization time. Implements : flexio_i2c_master_user_config_t_Class

Data Fields

- uint16_t slaveAddress
- flexio_driver_type_t driverType
- uint32_t baudRate
- uint8_t sdaPin
- uint8_t sclPin
- flexio_callback_t callback
- void ∗ callbackParam
- uint8_t rxDMAChannel
- uint8_t txDMAChannel

Field Documentation

3.37.2.1.1 baudRate

```
uint32_t baudRate
```

Baud rate in hertz

3.37.2.1.2 callback

```
flexio_callback_t callback
```

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

3.37.2.1.3 callbackParam

```
void* callbackParam
```

Parameter for the callback function

3.37.2.1.4 driverType

```
flexio_driver_type_t driverType
```

Driver type: interrupts/polling/DMA

3.37.2.1.5 rxDMAChannel

```
uint8_t rxDMAChannel
```

Rx DMA channel number. Only used in DMA mode

3.37.2.1.6 sclPin

```
uint8_t sclPin
```

Flexio pin to use as I2C SCL pin

3.37.2.1.7 sdaPin

```
uint8_t sdaPin
```

Flexio pin to use as I2C SDA pin

3.37.2.1.8 slaveAddress

```
uint16_t slaveAddress
```

Slave address, 7-bit

3.37.2.1.9 txDMAChannel

```
uint8_t txDMAChannel
```

Tx DMA channel number. Only used in DMA mode

3.37.2.2 struct flexio_i2c_master_state_t

Master internal context structure.

This structure is used by the driver for its internal logic. It must be provided by the application through the FLEXIO_I2C_DRV_MasterInit() function, then it cannot be freed until the driver is de-initialized using FLEXIO_I2C_DRV_MasterDeinit(). The application should make no assumptions about the content of this structure.

3.37.3 Macro Definition Documentation

3.37.3.1 FLEXIO_I2C_MAX_SIZE

```
#define FLEXIO_I2C_MAX_SIZE (((uint32_t)((0xFFU - 1U) / 18U)) - 1U)
```

Maximum size of a transfer. The restriction is that the total number of SCL edges must not exceed 8 bits, such that it can be programmed in the upper part of the timer compare register. There are 2 SCL edges per bit, 9 bits per byte (including ACK). The extra 1 is for the STOP condition.

3.37.4 Function Documentation

3.37.4.1 FLEXIO_I2C_DRV_MasterDeinit()

```
status_t FLEXIO_I2C_DRV_MasterDeinit (
             flexio_i2c_master_state_t * master )
```

De-initialize the FLEXIO_I2C master mode driver.

This function de-initializes the FLEXIO_I2C driver in master mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

Parameters

master	Pointer to the FLEXIO_I2C master driver context structure.

Returns

Error or success status returned by API

3.37.4.2 FLEXIO_I2C_DRV_MasterGetBaudRate()

```
status_t FLEXIO_I2C_DRV_MasterGetBaudRate (
             flexio_i2c_master_state_t * master,
             uint32_t * baudRate )
```

Get the currently configured baud rate.

This function returns the currently configured I2C baud rate.

Parameters

master	Pointer to the FLEXIO_I2C master driver context structure.
baudRate	the current baud rate in hertz

Returns

Error or success status returned by API

3.37.4.3 FLEXIO_I2C_DRV_MasterGetStatus()

```
status_t FLEXIO_I2C_DRV_MasterGetStatus (
            flexio_i2c_master_state_t * master,
            uint32_t * bytesRemaining )
```

Get the status of the current non-blocking I2C master transaction.

This function returns the current status of a non-blocking I2C master transaction. A return code of STATUS_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

Parameters

master	Pointer to the FLEXIO_I2C master driver context structure.
bytesRemaining	The remaining number of bytes to be transferred

Returns

Error or success status returned by API

3.37.4.4 FLEXIO_I2C_DRV_MasterInit()

```
status_t FLEXIO_I2C_DRV_MasterInit (
            uint32_t instance,
            const flexio_i2c_master_user_config_t * userConfigPtr,
            flexio_i2c_master_state_t * master )
```

Initialize the FLEXIO_I2C master mode driver.

This function initializes the FLEXIO_I2C driver in master mode.

Parameters

instance	FLEXIO peripheral instance number
userConfigPtr	Pointer to the FLEXIO_I2C master user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
master	Pointer to the FLEXIO_I2C master driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using FLEXIO_I2C_DRV_MasterDeinit().

Returns

Error or success status returned by API

3.37.4.5 FLEXIO_I2C_DRV_MasterReceiveData()

```
status_t FLEXIO_I2C_DRV_MasterReceiveData (
              flexio_i2c_master_state_t * master,
              uint8_t * rxBuff,
              uint32_t rxSize,
              bool sendStop )
```

Perform a non-blocking receive transaction on the I2C bus.

This function starts the reception of a block of data from the currently configured slave address and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO_I2C_DRV_MasterGetStatus function (if the driver is initialized in polling mode). Use FLEXIO_I2C_DRV_MasterGetStatus() to check the progress of the reception.

Parameters

master	Pointer to the FLEXIO_I2C master driver context structure.
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred
sendStop	specifies whether or not to generate stop condition after the reception

Returns

Error or success status returned by API

3.37.4.6 FLEXIO_I2C_DRV_MasterReceiveDataBlocking()

```
status_t FLEXIO_I2C_DRV_MasterReceiveDataBlocking (
              flexio_i2c_master_state_t * master,
              uint8_t * rxBuff,
              uint32_t rxSize,
              bool sendStop,
              uint32_t timeout )
```

Perform a blocking receive transaction on the I2C bus.

This function receives a block of data from the currently configured slave address, and only returns when the transmission is complete.

Parameters

master	Pointer to the FLEXIO_I2C master driver context structure.
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred
sendStop	specifies whether or not to generate stop condition after the reception
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.37.4.7 FLEXIO_I2C_DRV_MasterSendData()

```
status_t FLEXIO_I2C_DRV_MasterSendData (
             flexio_i2c_master_state_t * master,
             const uint8_t * txBuff,
             uint32_t txSize,
             bool sendStop )
```

Perform a non-blocking send transaction on the I2C bus.

This function starts the transmission of a block of data to the currently configured slave address and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO_I2C_DRV_MasterGetStatus function (if the driver is initialized in polling mode). Use FLEXIO_I2C_DRV_MasterGetStatus() to check the progress of the transmission.

Parameters

master	Pointer to the FLEXIO_I2C master driver context structure.
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred
sendStop	specifies whether or not to generate stop condition after the transmission

Returns

Error or success status returned by API

3.37.4.8 FLEXIO_I2C_DRV_MasterSendDataBlocking()

```
status_t FLEXIO_I2C_DRV_MasterSendDataBlocking (
             flexio_i2c_master_state_t * master,
             const uint8_t * txBuff,
             uint32_t txSize,
             bool sendStop,
             uint32_t timeout )
```

Perform a blocking send transaction on the I2C bus.

This function sends a block of data to the currently configured slave address, and only returns when the transmission is complete.

Parameters

master	Pointer to the FLEXIO_I2C master driver context structure.
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred
sendStop	specifies whether or not to generate stop condition after the transmission
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.37.4.9 FLEXIO_I2C_DRV_MasterSetBaudRate()

```
status_t FLEXIO_I2C_DRV_MasterSetBaudRate (
            flexio_i2c_master_state_t * master,
            uint32_t baudRate )
```

Set the baud rate for any subsequent I2C communication.

This function sets the baud rate (SCL frequency) for the I2C master. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call FLEXIO_I2C_DRV_MasterGetBaudRate() after FLEXIO_I2C_DRV_MasterSet↩ BaudRate() to check what baud rate was actually set.

Parameters

master	Pointer to the FLEXIO_I2C master driver context structure.
baudRate	the desired baud rate in hertz

Returns

Error or success status returned by API

3.37.4.10 FLEXIO_I2C_DRV_MasterSetSlaveAddr()

```
status_t FLEXIO_I2C_DRV_MasterSetSlaveAddr (
            flexio_i2c_master_state_t * master,
            const uint16_t address )
```

Set the slave address for any subsequent I2C communication.

This function sets the slave address which will be used for any future transfer.

Parameters

master	Pointer to the FLEXIO_I2C master driver context structure.
address	slave address, 7-bit

Returns

Error or success status returned by API

3.37.4.11 FLEXIO_I2C_DRV_MasterTransferAbort()

```
status_t FLEXIO_I2C_DRV_MasterTransferAbort (
            flexio_i2c_master_state_t * master )
```

Aborts a non-blocking I2C master transaction.

This function aborts a non-blocking I2C transfer.

Parameters

master	Pointer to the FLEXIO_I2C master driver context structure.

Returns

Error or success status returned by API

3.38 FlexIO I2S Driver

3.38.1 Detailed Description

I2S communication over FlexIO module (FLEXIO_I2S)

The FLEXIO_I2S Driver allows communication on an I2S bus using the FlexIO module in the S32144K processor.

Features

- Master or slave operation
- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transmit and receive functions
- Configurable baud rate and bit count

Functionality

Before using any Flexio driver the device must first be initialized using function FLEXIO_DRV_InitDevice. Then the FLEXIO_I2S Driver must be initialized, using functions FLEXIO_I2S_DRV_MasterInit() or FLEXIO_I2S_DRV↩
_SlaveInit(). It is possible to use more driver instances on the same FlexIO device, as long as sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using FLEXIO_I2S_DRV_MasterDeinit() or FLEXIO_I2S_↩
DRV_SlaveDeinit(). This will release the hardware resources, allowing other driver instances to be initialized.

Master Mode

Master Mode provides functions for transmitting or receiving data to/from any I2S slave. The number of bits per word and the baud rate are provided at initialization time through the master configuration structure, but they can be changed at runtime by using FLEXIO_I2S_DRV_MasterSetConfig(). Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call FLEXIO_I2S_DRV_MasterGetBaudRate() to check what baud rate was actually set.

To send or receive data to/from the currently configured slave address, use functions FLEXIO_I2S_DRV_Master↩
SendData() or FLEXIO_I2S_DRV_MasterReceiveData() (or their blocking counterparts). The driver is not full-duplex, only one direction (send or receive) can be used at one time. It is possible to configure both Rx and Tx pin to use the same Flexio pin.

Continuous send/receive can be realized by registering a user callback function. When the driver completes the transmission or reception of the current buffer, it will invoke the user callback with an appropriate event. The callback function can the use FLEXIO_I2S_DRV_MasterSetTxBuffer() or FLEXIO_I2S_DRV_MasterSetRxBuffer() to provide a new buffer.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return STATUS_SUCCESS, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application will be notified through the user callback when the transfer completes, or it can check the status of the current transfer by calling FLEXIO_I2↩
S_DRV_MasterGetStatus(). If the transfer is still ongoing this function will return STATUS_BUSY. If the transfer is completed, the function will return either STATUS_SUCCESS or an error code, depending on the outcome of the last transfer.

The driver supports interrupt, DMA and polling mode. In polling mode the function FLEXIO_I2S_DRV_Master↩
GetStatus() ensures the progress of the transfer by checking and handling transmit and receive events reported by the FlexIO module. The application should ensure that this function is called often enough (at least once per transferred byte) to avoid Tx underflows or Rx overflows. In DMA mode the DMA channels that will be used by the driver are received through the configuration structure. The channels must be initialized by the application before the flexio_i2s driver is initialized. The flexio_i2s driver will only set the DMA request source.

Slave Mode

Slave Mode is very similar to master mode, the main difference being that the FLEXIO_I2S_DRV_SlaveInit() function initializes the FlexIO module to use the clock signal received from the master instead of generating it. Consequently, there is no baud rate setting in slave mode. Other than that, the slave mode offers a similar interface to the master mode. FLEXIO_I2S_DRV_MasterSendData() or FLEXIO_I2S_DRV_MasterReceiveData() (or their blocking counterparts) can be used to initiate transfers, and FLEXIO_I2S_DRV_SlaveGetStatus() is used to check the status of the transfer and advance the transfer in polling mode. All other specifications from the Master Mode description apply for Slave Mode too.

Important Notes

- Before using the FLEXIO_I2S Driver the FlexIO clock must be configured. Refer to SCG HAL and PCC HAL for clock configuration.
- Before using the FLEXIO_I2S Driver the pins must be routed to the FlexIO module. Refer to PORT HAL for pin routing configuration. Note that any of the available FlexIO pins can be used for any of the TX, RX, SCK and WS signals (configurable at initialization time). If more than one driver instance is used on the same Flexio module, it is the responsibility of the application to ensure there are no conflicts between pins.
- The driver enables the interrupts for the corresponding FlexIO module, but any interrupt priority setting must be done by the application.
- Timeout feature for blocking transfers does not work in polling mode.
- This driver needs two shifters and two timers for its operation. Initialization will fail if there are not enough shifters and timers available on the FlexIO device.
- This driver needs two DMA channels for its operation when it is initialized in DMA mode. The DMA channels must be initialized by the application before initializing the driver. Refer to EDMA driver for DMA channels initialization.
- If the application uses an RTOS, this driver uses a semaphore for blocking transfers. Initialization will fail if the semaphore cannot be created. If the driver uses polling mode no semaphore is used.
- If the application uses an RTOS, the FlexIO drivers use a mutex for channel allocation. Only one mutex per device is needed, not per driver instance. Device initialization will fail if the mutex cannot be created.

Data Structures

- struct flexio_i2s_master_user_config_t

  Master configuration structure. More...
- struct flexio_i2s_slave_user_config_t

  Slave configuration structure. More...
- struct flexio_i2s_master_state_t

  Master internal context structure. More...

Typedefs

- typedef flexio_i2s_master_state_t flexio_i2s_slave_state_t

  Slave internal context structure.

FLEXIO_I2S Driver

- status_t FLEXIO_I2S_DRV_MasterInit (uint32_t instance, const flexio_i2s_master_user_config_t ∗user↩ ConfigPtr, flexio_i2s_master_state_t ∗master)

  Initialize the FLEXIO_I2S master mode driver.
- status_t FLEXIO_I2S_DRV_MasterDeinit (flexio_i2s_master_state_t ∗master)

  De-initialize the FLEXIO_I2S master mode driver.
- status_t FLEXIO_I2S_DRV_MasterSetConfig (flexio_i2s_master_state_t ∗master, uint32_t baudRate, uint8_t bitsWidth)

  Set the baud rate and bit width for any subsequent I2S communication.
- status_t FLEXIO_I2S_DRV_MasterGetBaudRate (flexio_i2s_master_state_t ∗master, uint32_t ∗baudRate)

  Get the currently configured baud rate.
- status_t FLEXIO_I2S_DRV_MasterSendData (flexio_i2s_master_state_t ∗master, const uint8_t ∗txBuff, uint32_t txSize)

  Perform a non-blocking send transaction on the I2S bus.
- status_t FLEXIO_I2S_DRV_MasterSendDataBlocking (flexio_i2s_master_state_t ∗master, const uint8_↩ t ∗txBuff, uint32_t txSize, uint32_t timeout)

  Perform a blocking send transaction on the I2S bus.
- status_t FLEXIO_I2S_DRV_MasterReceiveData (flexio_i2s_master_state_t ∗master, uint8_t ∗rxBuff, uint32_t rxSize)

  Perform a non-blocking receive transaction on the I2S bus.
- status_t FLEXIO_I2S_DRV_MasterReceiveDataBlocking (flexio_i2s_master_state_t ∗master, uint8_t ∗rx↩ Buff, uint32_t rxSize, uint32_t timeout)

  Perform a blocking receive transaction on the I2S bus.
- status_t FLEXIO_I2S_DRV_MasterTransferAbort (flexio_i2s_master_state_t ∗master)

  Aborts a non-blocking I2S master transaction.
- status_t FLEXIO_I2S_DRV_MasterGetStatus (flexio_i2s_master_state_t ∗master, uint32_t ∗bytes↩ Remaining)

  Get the status of the current non-blocking I2S master transaction.
- status_t FLEXIO_I2S_DRV_MasterSetRxBuffer (flexio_i2s_master_state_t ∗master, uint8_t ∗rxBuff, uint32↩ _t rxSize)

  Provide a buffer for receiving data.
- status_t FLEXIO_I2S_DRV_MasterSetTxBuffer (flexio_i2s_master_state_t ∗master, const uint8_t ∗txBuff, uint32_t txSize)

  Provide a buffer for transmitting data.
- status_t FLEXIO_I2S_DRV_SlaveInit (uint32_t instance, const flexio_i2s_slave_user_config_t ∗userConfig↩ Ptr, flexio_i2s_slave_state_t ∗slave)

  Initialize the FLEXIO_I2S slave mode driver.
- static status_t FLEXIO_I2S_DRV_SlaveDeinit (flexio_i2s_slave_state_t ∗slave)

  De-initialize the FLEXIO_I2S slave mode driver.
- status_t FLEXIO_I2S_DRV_SlaveSetConfig (flexio_i2s_slave_state_t ∗slave, uint8_t bitsWidth)

  Set the bit width for any subsequent I2S communication.
- static status_t FLEXIO_I2S_DRV_SlaveSendData (flexio_i2s_slave_state_t ∗slave, const uint8_t ∗txBuff, uint32_t txSize)

  Perform a non-blocking send transaction on the I2S bus.
- static status_t FLEXIO_I2S_DRV_SlaveSendDataBlocking (flexio_i2s_slave_state_t ∗slave, const uint8_↩ t ∗txBuff, uint32_t txSize, uint32_t timeout)

  Perform a blocking send transaction on the I2S bus.
- static status_t FLEXIO_I2S_DRV_SlaveReceiveData (flexio_i2s_slave_state_t ∗slave, uint8_t ∗rxBuff, uint32_t rxSize)

  Perform a non-blocking receive transaction on the I2S bus.
- static status_t FLEXIO_I2S_DRV_SlaveReceiveDataBlocking (flexio_i2s_slave_state_t ∗slave, uint8_t ∗rx↩ Buff, uint32_t rxSize, uint32_t timeout)

Perform a blocking receive transaction on the I2S bus.
- static status_t FLEXIO_I2S_DRV_SlaveTransferAbort (flexio_i2s_slave_state_t ∗slave)

  Aborts a non-blocking I2S slave transaction.
- static status_t FLEXIO_I2S_DRV_SlaveGetStatus (flexio_i2s_slave_state_t ∗slave, uint32_t ∗bytes←Remaining)

  Get the status of the current non-blocking I2S slave transaction.
- static status_t FLEXIO_I2S_DRV_SlaveSetRxBuffer (flexio_i2s_slave_state_t ∗slave, uint8_t ∗rxBuff, uint32_t rxSize)

  Provide a buffer for receiving data.
- static status_t FLEXIO_I2S_DRV_SlaveSetTxBuffer (flexio_i2s_slave_state_t ∗slave, const uint8_t ∗txBuff, uint32_t txSize)

  Provide a buffer for transmitting data.

3.38.2 Data Structure Documentation

3.38.2.1 struct flexio_i2s_master_user_config_t

Master configuration structure.

This structure is used to provide configuration parameters for the flexio_i2s master at initialization time. Implements : flexio_i2s_master_user_config_t_Class

Data Fields

- flexio_driver_type_t driverType
- uint32_t baudRate
- uint8_t bitsWidth
- uint8_t txPin
- uint8_t rxPin
- uint8_t sckPin
- uint8_t wsPin
- flexio_callback_t callback
- void ∗ callbackParam
- uint8_t rxDMAChannel
- uint8_t txDMAChannel

Field Documentation

3.38.2.1.1 baudRate

```
uint32_t baudRate
```

Baud rate in hertz

3.38.2.1.2 bitsWidth

```
uint8_t bitsWidth
```

Number of bits in a word - multiple of 8

3.38.2.1.3 callback

```
flexio_callback_t callback
```

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

3.38.2.1.4 callbackParam

```
void* callbackParam
```

Parameter for the callback function

3.38.2.1.5 driverType

```
flexio_driver_type_t driverType
```

Driver type: interrupts/polling/DMA

3.38.2.1.6 rxDMAChannel

```
uint8_t rxDMAChannel
```

Rx DMA channel number. Only used in DMA mode

3.38.2.1.7 rxPin

```
uint8_t rxPin
```

Flexio pin to use for receive

3.38.2.1.8 sckPin

```
uint8_t sckPin
```

Flexio pin to use for serial clock

3.38.2.1.9 txDMAChannel

```
uint8_t txDMAChannel
```

Tx DMA channel number. Only used in DMA mode

3.38.2.1.10 txPin

```
uint8_t txPin
```

Flexio pin to use for transmit

3.38.2.1.11 wsPin

```
uint8_t wsPin
```

Flexio pin to use for word select

3.38.2.2 struct flexio_i2s_slave_user_config_t

Slave configuration structure.

This structure is used to provide configuration parameters for the flexio_i2s slave at initialization time. Implements : flexio_i2s_slave_user_config_t_Class

Data Fields

- flexio_driver_type_t driverType
- uint8_t bitsWidth
- uint8_t txPin
- uint8_t rxPin
- uint8_t sckPin
- uint8_t wsPin
- flexio_callback_t callback
- void ∗ callbackParam
- uint8_t rxDMAChannel
- uint8_t txDMAChannel

Field Documentation

3.38.2.2.1 bitsWidth

```
uint8_t bitsWidth
```

Number of bits in a word - multiple of 8

3.38.2.2.2 callback

```
flexio_callback_t callback
```

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

3.38.2.2.3 callbackParam

```
void* callbackParam
```

Parameter for the callback function

3.38.2.2.4 driverType

```
flexio_driver_type_t driverType
```

Driver type: interrupts/polling/DMA

3.38.2.2.5 rxDMAChannel

```
uint8_t rxDMAChannel
```

Rx DMA channel number. Only used in DMA mode

3.38.2.2.6 rxPin

```
uint8_t rxPin
```

Flexio pin to use for receive

3.38.2.2.7 sckPin

```
uint8_t sckPin
```

Flexio pin to use for serial clock

3.38.2.2.8 txDMAChannel

```
uint8_t txDMAChannel
```

Tx DMA channel number. Only used in DMA mode

3.38.2.2.9 txPin

```
uint8_t txPin
```

Flexio pin to use for transmit

3.38.2.2.10 wsPin

```
uint8_t wsPin
```

Flexio pin to use for word select

3.38.2.3 struct flexio_i2s_master_state_t

Master internal context structure.

This structure is used by the driver for its internal logic. It must be provided by the application through the FLEXIO_I2S_DRV_MasterInit() function, then it cannot be freed until the driver is de-initialized using FLEXIO_I2S_DRV_MasterDeinit(). The application should make no assumptions about the content of this structure.

3.38.3 Typedef Documentation

3.38.3.1 flexio_i2s_slave_state_t

```
typedef flexio_i2s_master_state_t flexio_i2s_slave_state_t
```

Slave internal context structure.

This structure is used by the driver for its internal logic. It must be provided by the application through the FLEXIO_I2S_DRV_SlaveInit() function, then it cannot be freed until the driver is de-initialized using FLEXIO_I2S_DRV_SlaveDeinit(). The application should make no assumptions about the content of this structure.

3.38.4 Function Documentation

3.38.4.1 FLEXIO_I2S_DRV_MasterDeinit()

```
status_t FLEXIO_I2S_DRV_MasterDeinit (
            flexio_i2s_master_state_t * master )
```

De-initialize the FLEXIO_I2S master mode driver.

This function de-initializes the FLEXIO_I2S driver in master mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.

Returns

Error or success status returned by API

3.38.4.2 FLEXIO_I2S_DRV_MasterGetBaudRate()

```
status_t FLEXIO_I2S_DRV_MasterGetBaudRate (
            flexio_i2s_master_state_t * master,
            uint32_t * baudRate )
```

Get the currently configured baud rate.

This function returns the currently configured I2S baud rate.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.
baudRate	the current baud rate in hertz

Returns

Error or success status returned by API

3.38.4.3 FLEXIO_I2S_DRV_MasterGetStatus()

```
status_t FLEXIO_I2S_DRV_MasterGetStatus (
            flexio_i2s_master_state_t * master,
            uint32_t * bytesRemaining )
```

Get the status of the current non-blocking I2S master transaction.

This function returns the current status of a non-blocking I2S master transaction. A return code of STATUS_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.
bytesRemaining	the remaining number of bytes to be transferred

Returns

Error or success status returned by API

3.38.4.4 FLEXIO_I2S_DRV_MasterInit()

```
status_t FLEXIO_I2S_DRV_MasterInit (
             uint32_t instance,
             const flexio_i2s_master_user_config_t * userConfigPtr,
             flexio_i2s_master_state_t * master )
```

Initialize the FLEXIO_I2S master mode driver.

This function initializes the FLEXIO_I2S driver in master mode.

Parameters

instance	FLEXIO peripheral instance number
userConfigPtr	Pointer to the FLEXIO_I2S master user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
master	Pointer to the FLEXIO_I2S master driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using FLEXIO_I2S_DRV_MasterDeinit().

Returns

Error or success status returned by API

3.38.4.5 FLEXIO_I2S_DRV_MasterReceiveData()

```
status_t FLEXIO_I2S_DRV_MasterReceiveData (
             flexio_i2s_master_state_t * master,
             uint8_t * rxBuff,
             uint32_t rxSize )
```

Perform a non-blocking receive transaction on the I2S bus.

This function starts the reception of a block of data and returns immediately. The rest of the reception is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO_I2S_DRV_MasterGet←Status function (if the driver is initialized in polling mode). Use FLEXIO_I2S_DRV_MasterGetStatus() to check the progress of the reception.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.38.4.6 FLEXIO_I2S_DRV_MasterReceiveDataBlocking()

```
status_t FLEXIO_I2S_DRV_MasterReceiveDataBlocking (
             flexio_i2s_master_state_t * master,
             uint8_t * rxBuff,
             uint32_t rxSize,
             uint32_t timeout )
```

Perform a blocking receive transaction on the I2S bus.

This function receives a block of data and only returns when the reception is complete.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.38.4.7 FLEXIO_I2S_DRV_MasterSendData()

```
status_t FLEXIO_I2S_DRV_MasterSendData (
            flexio_i2s_master_state_t * master,
            const uint8_t * txBuff,
            uint32_t txSize )
```

Perform a non-blocking send transaction on the I2S bus.

This function starts the transmission of a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO_I2S_DRV_↩ MasterGetStatus function (if the driver is initialized in polling mode). Use FLEXIO_I2S_DRV_MasterGetStatus() to check the progress of the transmission.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.38.4.8 FLEXIO_I2S_DRV_MasterSendDataBlocking()

```
status_t FLEXIO_I2S_DRV_MasterSendDataBlocking (
            flexio_i2s_master_state_t * master,
            const uint8_t * txBuff,
            uint32_t txSize,
            uint32_t timeout )
```

Perform a blocking send transaction on the I2S bus.

This function sends a block of data, and only returns when the transmission is complete.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.38.4.9 FLEXIO_I2S_DRV_MasterSetConfig()

```
status_t FLEXIO_I2S_DRV_MasterSetConfig (
             flexio_i2s_master_state_t * master,
             uint32_t baudRate,
             uint8_t bitsWidth )
```

Set the baud rate and bit width for any subsequent I2S communication.

This function sets the baud rate (SCK frequency) and bit width for the I2S master. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call FLEXIO_I2S_DRV_MasterGetBaudRate() after FLEXIO_I2S_DRV_↩
MasterSetConfig() to check what baud rate was actually set.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.
baudRate	the desired baud rate in hertz
bitsWidth	number of bits per word

Returns

Error or success status returned by API

3.38.4.10 FLEXIO_I2S_DRV_MasterSetRxBuffer()

```
status_t FLEXIO_I2S_DRV_MasterSetRxBuffer (
             flexio_i2s_master_state_t * master,
             uint8_t * rxBuff,
             uint32_t rxSize )
```

Provide a buffer for receiving data.

This function can be used to provide a new buffer for receiving data to the driver. It can be called from the user callback when event STATUS_I2S_RX_OVERRUN is reported. This way the reception will continue without interruption.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.38.4.11 FLEXIO_I2S_DRV_MasterSetTxBuffer()

```
status_t FLEXIO_I2S_DRV_MasterSetTxBuffer (
              flexio_i2s_master_state_t * master,
              const uint8_t * txBuff,
              uint32_t txSize )
```

Provide a buffer for transmitting data.

This function can be used to provide a new buffer for transmitting data to the driver. It can be called from the user callback when event STATUS_I2S_TX_UNDERRUN is reported. This way the transmission will continue without interruption.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.
txBuff	pointer to the buffer containing transmit data
txSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.38.4.12 FLEXIO_I2S_DRV_MasterTransferAbort()

```
status_t FLEXIO_I2S_DRV_MasterTransferAbort (
              flexio_i2s_master_state_t * master )
```

Aborts a non-blocking I2S master transaction.

This function aborts a non-blocking I2S transfer.

Parameters

master	Pointer to the FLEXIO_I2S master driver context structure.

Returns

Error or success status returned by API

3.38.4.13 FLEXIO_I2S_DRV_SlaveDeinit()

```
static status_t FLEXIO_I2S_DRV_SlaveDeinit (
              flexio_i2s_slave_state_t * slave ) [inline], [static]
```

De-initialize the FLEXIO_I2S slave mode driver.

This function de-initializes the FLEXIO_I2S driver in slave mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

Parameters

slave	Pointer to the FLEXIO_I2S slave driver context structure.

Returns

Error or success status returned by API Implements : FLEXIO_I2S_DRV_SlaveDeinit_Activity

3.38.4.14 FLEXIO_I2S_DRV_SlaveGetStatus()

```
static status_t FLEXIO_I2S_DRV_SlaveGetStatus (
            flexio_i2s_slave_state_t * slave,
            uint32_t * bytesRemaining )  [inline], [static]
```

Get the status of the current non-blocking I2S slave transaction.

This function returns the current status of a non-blocking I2S slave transaction. A return code of STATUS_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

Parameters

slave	Pointer to the FLEXIO_I2S slave driver context structure.
bytesRemaining	the remaining number of bytes to be transferred

Returns

Error or success status returned by API Implements : FLEXIO_I2S_DRV_SlaveGetStatus_Activity

3.38.4.15 FLEXIO_I2S_DRV_SlaveInit()

```
status_t FLEXIO_I2S_DRV_SlaveInit (
            uint32_t instance,
            const flexio_i2s_slave_user_config_t * userConfigPtr,
            flexio_i2s_slave_state_t * slave )
```

Initialize the FLEXIO_I2S slave mode driver.

This function initializes the FLEXIO_I2S driver in slave mode.

Parameters

instance	FLEXIO peripheral instance number
userConfigPtr	Pointer to the FLEXIO_I2S slave user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
slave	Pointer to the FLEXIO_I2S slave driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using FLEXIO_I2S_DRV_SlaveDeinit().

Returns

Error or success status returned by API

3.38.4.16 FLEXIO_I2S_DRV_SlaveReceiveData()

```
static status_t FLEXIO_I2S_DRV_SlaveReceiveData (
              flexio_i2s_slave_state_t * slave,
              uint8_t * rxBuff,
              uint32_t rxSize )  [inline], [static]
```

Perform a non-blocking receive transaction on the I2S bus.

This function starts the reception of a block of data and returns immediately. The rest of the reception is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO_I2S_DRV_SlaveGet←Status function (if the driver is initialized in polling mode). Use FLEXIO_I2S_DRV_SlaveGetStatus() to check the progress of the reception.

Parameters

slave	Pointer to the FLEXIO_I2S slave driver context structure.
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API Implements : FLEXIO_I2S_DRV_SlaveReceiveData_Activity

3.38.4.17 FLEXIO_I2S_DRV_SlaveReceiveDataBlocking()

```
static status_t FLEXIO_I2S_DRV_SlaveReceiveDataBlocking (
              flexio_i2s_slave_state_t * slave,
              uint8_t * rxBuff,
              uint32_t rxSize,
              uint32_t timeout )  [inline], [static]
```

Perform a blocking receive transaction on the I2S bus.

This function receives a block of data and only returns when the reception is complete.

Parameters

slave	Pointer to the FLEXIO_I2S slave driver context structure.
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API Implements : FLEXIO_I2S_DRV_SlaveReceiveDataBlocking_Activity

3.38.4.18 FLEXIO_I2S_DRV_SlaveSendData()

```
static status_t FLEXIO_I2S_DRV_SlaveSendData (
            flexio_i2s_slave_state_t * slave,
            const uint8_t * txBuff,
            uint32_t txSize )  [inline], [static]
```

Perform a non-blocking send transaction on the I2S bus.

This function starts the transmission of a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO_I2S_DRV↵
_SlaveGetStatus function (if the driver is initialized in polling mode). Use FLEXIO_I2S_DRV_SlaveGetStatus() to check the progress of the transmission.

Parameters

slave	Pointer to the FLEXIO_I2S slave driver context structure.
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API Implements : FLEXIO_I2S_DRV_SlaveSendData_Activity

3.38.4.19 FLEXIO_I2S_DRV_SlaveSendDataBlocking()

```
static status_t FLEXIO_I2S_DRV_SlaveSendDataBlocking (
            flexio_i2s_slave_state_t * slave,
            const uint8_t * txBuff,
            uint32_t txSize,
            uint32_t timeout )  [inline], [static]
```

Perform a blocking send transaction on the I2S bus.

This function sends a block of data, and only returns when the transmission is complete.

Parameters

slave	Pointer to the FLEXIO_I2S slave driver context structure.
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API Implements : FLEXIO_I2S_DRV_SlaveSendDataBlocking_Activity

3.38.4.20 FLEXIO_I2S_DRV_SlaveSetConfig()

```
status_t FLEXIO_I2S_DRV_SlaveSetConfig (
            flexio_i2s_slave_state_t * slave,
            uint8_t bitsWidth )
```

Set the bit width for any subsequent I2S communication.

This function sets the bit width for the I2S slave.

Parameters

slave	Pointer to the FLEXIO_I2S slave driver context structure.
bitsWidth	number of bits per word

Returns

Error or success status returned by API

3.38.4.21 FLEXIO_I2S_DRV_SlaveSetRxBuffer()

```
static status_t FLEXIO_I2S_DRV_SlaveSetRxBuffer (
            flexio_i2s_slave_state_t * slave,
            uint8_t * rxBuff,
            uint32_t rxSize )  [inline], [static]
```

Provide a buffer for receiving data.

This function can be used to provide a driver with a new buffer for receiving data. It can be called from the user callback when event STATUS_I2S_RX_OVERRUN is reported. This way the reception will continue without interruption.

Parameters

slave	Pointer to the FLEXIO_I2S slave driver context structure.
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API Implements : FLEXIO_I2S_DRV_SlaveSetRxBuffer_Activity

3.38.4.22 FLEXIO_I2S_DRV_SlaveSetTxBuffer()

```
static status_t FLEXIO_I2S_DRV_SlaveSetTxBuffer (
            flexio_i2s_slave_state_t * slave,
            const uint8_t * txBuff,
            uint32_t txSize )  [inline], [static]
```

Provide a buffer for transmitting data.

This function can be used to provide a driver with a new buffer for transmitting data. It can be called from the user callback when event STATUS_I2S_TX_UNDERRUN is reported. This way the transmission will continue without interruption.

Parameters

slave	Pointer to the FLEXIO_I2S slave driver context structure.
txBuff	pointer to the buffer containing transmit data
txSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API Implements : FLEXIO_I2S_DRV_SlaveSetTxBuffer_Activity

3.38.4.23 FLEXIO_I2S_DRV_SlaveTransferAbort()

```
static status_t FLEXIO_I2S_DRV_SlaveTransferAbort (
            flexio_i2s_slave_state_t * slave )  [inline], [static]
```

Aborts a non-blocking I2S slave transaction.

This function aborts a non-blocking I2S transfer.

Parameters

slave	Pointer to the FLEXIO_I2S slave driver context structure.

Returns

Error or success status returned by API Implements : FLEXIO_I2S_DRV_SlaveTransferAbort_Activity

3.39 FlexIO SPI Driver

3.39.1 Detailed Description

SPI communication over FlexIO module (FLEXIO_SPI)

The FLEXIO_SPI Driver allows communication on an SPI bus using the FlexIO module in the S32144K processor.

Features

- Master or slave operation
- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transfer functions
- Configurable baud rate
- Configurable clock polarity and phase
- Configurable bit order and data size

Functionality

Before using any Flexio driver the device must first be initialized using function FLEXIO_DRV_InitDevice. Then the FLEXIO_SPI Driver must be initialized, using functions FLEXIO_SPI_DRV_MasterInit() or FLEXIO_SPI_DRV_←SlaveInit(). It is possible to use more driver instances on the same FlexIO device, as long as sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using FLEXIO_SPI_DRV_MasterDeinit() or FLEXIO_S←PI_DRV_SlaveDeinit(). This will release the hardware resources, allowing other driver instances to be initialized other.

Master Mode

Master Mode provides functions for transmitting or receiving data to/from an SPI slave. Baud rate is provided at initialization time through the master configuration structure, but can be changed at runtime by using FLEXIO_SP←I_DRV_MasterSetBaudRate() function. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call FLEXIO_SPI_DRV_MasterGetBaudRate() after FLEXIO_SPI_DRV_MasterSetBaudRate() to check what baud rate was actually set.

To send or receive data, use function FLEXIO_SPI_DRV_MasterTransfer(). The transmit and receive buffers, together with parameters for the transfer are provided through the flexio_spi_transfer_t structure. If only transmit or receive is desired, any one of the Rx/Tx buffers can be set to NULL. This driver does not support continuous send/receive using a user callback function. The callback function is only used to signal the end of a transfer.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return STATUS_SUCCESS, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application will be notified through the user callback when the transfer completes, or it can check the status of the current transfer by calling FLEXIO_S←PI_DRV_MasterGetStatus(). If the transfer is still ongoing this function will return STATUS_BUSY. If the transfer is completed, the function will return either STATUS_SUCCESS or an error code, depending on the outcome of the last transfer.

The driver supports interrupt, DMA and polling mode. In polling mode the function FLEXIO_SPI_DRV_Master←GetStatus() ensures the progress of the transfer by checking and handling transmit and receive events reported by the FlexIO module. The application should ensure that this function is called often enough (at least once per transferred byte)to avoid Tx underflows or Rx overflows. In DMA mode the DMA channels that will be used by the driver are received through the configuration structure. The channels must be initialized by the application before the flexio_spi driver is initialized. The flexio_spi driver will only set the DMA request source.

Slave Mode

Slave Mode is very similar to master mode, the main difference being that the FLEXIO_SPI_DRV_SlaveInit() function initializes the FlexIO module to use the clock signal received from the master instead of generating it. Consequently, there is no `SetBaudRate` function in slave mode. Other than that, the slave mode offers a similar interface to the master mode. FLEXIO_SPI_DRV_MasterTransfer() can be used to initiate transfers, and FLEXIO↩ _SPI_DRV_SlaveGetStatus() is used to check the status of the transfer and advance the transfer in polling mode. All other specifications from the Master Mode description apply for Slave Mode too

Important Notes

- Before using the FLEXIO_SPI Driver the protocol clock of the module must be configured. Refer to SCG HAL and PCC HAL for clock configuration.
- Before using the FLEXIO_SPI Driver the pins must be routed to the FlexIO module. Refer to PORT HAL for pin routing configuration. Note that any of the available FlexIO pins can be used for MOSI, MISO, SCK and SS (configurable at initialization time).
- The driver enables the interrupts for the corresponding FlexIO module, but any interrupt priority setting must be done by the application.
- The driver does not support back-to-back transmission mode for CPHA = 1
- The driver does not support configurable polarity for SS signal (only active-low is supported)
- Timeout feature for blocking transfers does not work in polling mode.
- This driver needs two shifters and two timers for its operation. Initialization will fail if there are not enough shifters and timers available on the FlexIO device.
- This driver needs two DMA channels for its operation when it is initialized in DMA mode. The DMA channels must be initialized by the application before initializing the driver. Refer to EDMA driver for DMA channels initialization.
- If the application uses an RTOS, this driver uses a semaphore for blocking transfers. Initialization will fail if the semaphore cannot be created. If the driver uses polling mode no semaphore is used.
- If the application uses an RTOS, the FlexIO drivers use a mutex for channel allocation. Only one mutex per device is needed, not per driver instance. Device initialization will fail if the mutex cannot be created.

Data Structures

- struct flexio_spi_master_user_config_t

  Master configuration structure. More...
- struct flexio_spi_slave_user_config_t

  Slave configuration structure. More...
- struct flexio_spi_master_state_t

  Master internal context structure. More...

Typedefs

- typedef flexio_spi_master_state_t flexio_spi_slave_state_t

  Slave internal context structure.

Enumerations

- enum flexio_spi_transfer_bit_order_t { FLEXIO_SPI_TRANSFER_MSB_FIRST = 0U, FLEXIO_SPI_TRAN↵ SFER_LSB_FIRST = 1U }

  Order in which the data bits are transferred Implements : flexio_spi_transfer_bit_order_t_Class.
- enum flexio_spi_transfer_size_t { FLEXIO_SPI_TRANSFER_1BYTE = 1U, FLEXIO_SPI_TRANSFER_2B↵ YTE = 2U, FLEXIO_SPI_TRANSFER_4BYTE = 4U }

  Size of transferred data in bytes Implements : flexio_spi_transfer_size_t_Class.

FLEXIO_SPI Driver

- status_t FLEXIO_SPI_DRV_MasterInit (uint32_t instance, const flexio_spi_master_user_config_t ∗user↵ ConfigPtr, flexio_spi_master_state_t ∗master)

  Initialize the FLEXIO_SPI master mode driver.
- status_t FLEXIO_SPI_DRV_MasterDeinit (flexio_spi_master_state_t ∗master)

  De-initialize the FLEXIO_SPI master mode driver.
- status_t FLEXIO_SPI_DRV_MasterSetBaudRate (flexio_spi_master_state_t ∗master, uint32_t baudRate)

  Set the baud rate for any subsequent SPI communication.
- status_t FLEXIO_SPI_DRV_MasterGetBaudRate (flexio_spi_master_state_t ∗master, uint32_t ∗baudRate)

  Get the currently configured baud rate.
- status_t FLEXIO_SPI_DRV_MasterTransfer (flexio_spi_master_state_t ∗master, const uint8_t ∗txData, uint8_t ∗rxData, uint32_t dataSize)

  Perform a non-blocking SPI master transaction.
- status_t FLEXIO_SPI_DRV_MasterTransferBlocking (flexio_spi_master_state_t ∗master, const uint8_t ∗tx↵ Data, uint8_t ∗rxData, uint32_t dataSize, uint32_t timeout)

  Perform a blocking SPI master transaction.
- status_t FLEXIO_SPI_DRV_MasterTransferAbort (flexio_spi_master_state_t ∗master)

  Aborts a non-blocking SPI master transaction.
- status_t FLEXIO_SPI_DRV_MasterGetStatus (flexio_spi_master_state_t ∗master, uint32_t ∗bytes↵ Remaining)

  Get the status of the current non-blocking SPI master transaction.
- status_t FLEXIO_SPI_DRV_SlaveInit (uint32_t instance, const flexio_spi_slave_user_config_t ∗userConfig↵ Ptr, flexio_spi_slave_state_t ∗slave)

  Initialize the FLEXIO_SPI slave mode driver.
- static status_t FLEXIO_SPI_DRV_SlaveDeinit (flexio_spi_slave_state_t ∗slave)

  De-initialize the FLEXIO_SPI slave mode driver.
- static status_t FLEXIO_SPI_DRV_SlaveTransfer (flexio_spi_slave_state_t ∗slave, const uint8_t ∗txData, uint8_t ∗rxData, uint32_t dataSize)

  Perform a non-blocking SPI slave transaction.
- static status_t FLEXIO_SPI_DRV_SlaveTransferBlocking (flexio_spi_slave_state_t ∗slave, const uint8_t ∗tx↵ Data, uint8_t ∗rxData, uint32_t dataSize, uint32_t timeout)

  Perform a blocking SPI slave transaction.
- static status_t FLEXIO_SPI_DRV_SlaveTransferAbort (flexio_spi_slave_state_t ∗slave)

  Aborts a non-blocking SPI slave transaction.
- static status_t FLEXIO_SPI_DRV_SlaveGetStatus (flexio_spi_slave_state_t ∗slave, uint32_t ∗bytes↵ Remaining)

  Get the status of the current non-blocking SPI slave transaction.

3.39.2 Data Structure Documentation

3.39.2.1 struct flexio_spi_master_user_config_t

Master configuration structure.

This structure is used to provide configuration parameters for the flexio_spi master at initialization time. Implements : flexio_spi_master_user_config_t_Class

Data Fields

- uint32_t baudRate
- flexio_driver_type_t driverType
- flexio_spi_transfer_bit_order_t bitOrder
- flexio_spi_transfer_size_t transferSize
- uint8_t clockPolarity
- uint8_t clockPhase
- uint8_t mosiPin
- uint8_t misoPin
- uint8_t sckPin
- uint8_t ssPin
- flexio_callback_t callback
- void ∗ callbackParam
- uint8_t rxDMAChannel
- uint8_t txDMAChannel

Field Documentation

3.39.2.1.1 baudRate

```
uint32_t baudRate
```

Baud rate in hertz

3.39.2.1.2 bitOrder

```
flexio_spi_transfer_bit_order_t bitOrder
```

Bit order: LSB-first / MSB-first

3.39.2.1.3 callback

```
flexio_callback_t callback
```

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

3.39.2.1.4 callbackParam

```
void* callbackParam
```

Parameter for the callback function

3.39.2.1.5 clockPhase

```
uint8_t clockPhase
```

Clock Phase (CPHA) 0 = sample on leading clock edge; 1 = sample on trailing clock edge

3.39.2.1.6 clockPolarity

```
uint8_t clockPolarity
```

Clock Polarity (CPOL) 0 = active-high clock; 1 = active-low clock

3.39.2.1.7 driverType

```
flexio_driver_type_t driverType
```

Driver type: interrupts/polling/DMA

3.39.2.1.8 misoPin

```
uint8_t misoPin
```

Flexio pin to use as MISO pin

3.39.2.1.9 mosiPin

```
uint8_t mosiPin
```

Flexio pin to use as MOSI pin

3.39.2.1.10 rxDMAChannel

```
uint8_t rxDMAChannel
```

Rx DMA channel number. Only used in DMA mode

3.39.2.1.11 sckPin

```
uint8_t sckPin
```

Flexio pin to use as SCK pin

3.39.2.1.12 ssPin

```
uint8_t ssPin
```

Flexio pin to use as SS pin

3.39.2.1.13 transferSize

```
flexio_spi_transfer_size_t transferSize
```

Transfer size in bytes: 1/2/4

3.39.2.1.14 txDMAChannel

```
uint8_t txDMAChannel
```

Tx DMA channel number. Only used in DMA mode

3.39.2.2 struct flexio_spi_slave_user_config_t

Slave configuration structure.

This structure is used to provide configuration parameters for the flexio_spi slave at initialization time. Implements : flexio_spi_slave_user_config_t_Class

Data Fields

- flexio_driver_type_t driverType
- flexio_spi_transfer_bit_order_t bitOrder
- flexio_spi_transfer_size_t transferSize
- uint8_t clockPolarity
- uint8_t clockPhase
- uint8_t mosiPin
- uint8_t misoPin
- uint8_t sckPin
- uint8_t ssPin
- flexio_callback_t callback
- void ∗ callbackParam
- uint8_t rxDMAChannel
- uint8_t txDMAChannel

Field Documentation

3.39.2.2.1 bitOrder

```
flexio_spi_transfer_bit_order_t bitOrder
```

Bit order: LSB-first / MSB-first

3.39.2.2.2 callback

```
flexio_callback_t callback
```

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

3.39.2.2.3 callbackParam

```
void* callbackParam
```

Parameter for the callback function

3.39.2.2.4 clockPhase

```
uint8_t clockPhase
```

Clock Phase (CPHA) 0 = sample on leading clock edge; 1 = sample on trailing clock edge

3.39.2.2.5 clockPolarity

```
uint8_t clockPolarity
```

Clock Polarity (CPOL) 0 = active-low clock; 1 = active-high clock

3.39.2.2.6 driverType

```
flexio_driver_type_t driverType
```

Driver type: interrupts/polling/DMA

3.39.2.2.7 misoPin

```
uint8_t misoPin
```

Flexio pin to use as MISO pin

3.39.2.2.8 mosiPin

```
uint8_t mosiPin
```

Flexio pin to use as MOSI pin

3.39.2.2.9 rxDMAChannel

```
uint8_t rxDMAChannel
```

Rx DMA channel number. Only used in DMA mode

3.39.2.2.10 sckPin

```
uint8_t sckPin
```

Flexio pin to use as SCK pin

3.39.2.2.11 ssPin

```
uint8_t ssPin
```

Flexio pin to use as SS pin

3.39.2.2.12 transferSize

```
flexio_spi_transfer_size_t transferSize
```

Transfer size in bytes: 1/2/4

3.39.2.2.13 txDMAChannel

```
uint8_t txDMAChannel
```

Tx DMA channel number. Only used in DMA mode

3.39.2.3 struct flexio_spi_master_state_t

Master internal context structure.

This structure is used by the master-mode driver for its internal logic. It must be provided by the application through the FLEXIO_SPI_DRV_MasterInit() function, then it cannot be freed until the driver is de-initialized using FLEXIO_SPI_DRV_MasterDeinit(). The application should make no assumptions about the content of this structure.

3.39.3 Typedef Documentation

3.39.3.1 flexio_spi_slave_state_t

```
typedef flexio_spi_master_state_t flexio_spi_slave_state_t
```

Slave internal context structure.

This structure is used by the slave-mode driver for its internal logic. It must be provided by the application through the FLEXIO_SPI_DRV_SlaveInit() function, then it cannot be freed until the driver is de-initialized using FLEXIO_SPI_DRV_SlaveDeinit(). The application should make no assumptions about the content of this structure.

3.39.4 Enumeration Type Documentation

3.39.4.1 flexio_spi_transfer_bit_order_t

```
enum flexio_spi_transfer_bit_order_t
```

Order in which the data bits are transferred Implements : flexio_spi_transfer_bit_order_t_Class.

Enumerator

FLEXIO_SPI_TRANSFER_MSB_FIRST	Transmit data starting with most significant bit
FLEXIO_SPI_TRANSFER_LSB_FIRST	Transmit data starting with least significant bit

3.39.4.2 flexio_spi_transfer_size_t

```
enum flexio_spi_transfer_size_t
```

Size of transferred data in bytes Implements : flexio_spi_transfer_size_t_Class.

Enumerator

FLEXIO_SPI_TRANSFER_1BYTE	Data size is 1-byte
FLEXIO_SPI_TRANSFER_2BYTE	Data size is 2-bytes
FLEXIO_SPI_TRANSFER_4BYTE	Data size is 4-bytes

3.39.5 Function Documentation

3.39.5.1 FLEXIO_SPI_DRV_MasterDeinit()

```
status_t FLEXIO_SPI_DRV_MasterDeinit (
             flexio_spi_master_state_t * master )
```

De-initialize the FLEXIO_SPI master mode driver.

This function de-initializes the FLEXIO_SPI driver in master mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

Parameters

master	Pointer to the FLEXIO_SPI master driver context structure.

Returns

Error or success status returned by API

3.39.5.2 FLEXIO_SPI_DRV_MasterGetBaudRate()

```
status_t FLEXIO_SPI_DRV_MasterGetBaudRate (
             flexio_spi_master_state_t * master,
             uint32_t * baudRate )
```

Get the currently configured baud rate.

This function returns the currently configured SPI baud rate.

Parameters

master	Pointer to the FLEXIO_SPI master driver context structure.
baudRate	the current baud rate in hertz

Returns

Error or success status returned by API

3.39.5.3 FLEXIO_SPI_DRV_MasterGetStatus()

```
status_t FLEXIO_SPI_DRV_MasterGetStatus (
             flexio_spi_master_state_t * master,
             uint32_t * bytesRemaining )
```

Get the status of the current non-blocking SPI master transaction.

This function returns the current status of a non-blocking SPI master transaction. A return code of STATUS_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

Parameters

master	Pointer to the FLEXIO_SPI master driver context structure.
bytesRemaining	the remaining number of bytes to be transferred

Returns

Error or success status returned by API

3.39.5.4 FLEXIO_SPI_DRV_MasterInit()

```
status_t FLEXIO_SPI_DRV_MasterInit (
            uint32_t instance,
            const flexio_spi_master_user_config_t * userConfigPtr,
            flexio_spi_master_state_t * master )
```

Initialize the FLEXIO_SPI master mode driver.

This function initializes the FLEXIO_SPI driver in master mode.

Parameters

instance	FLEXIO peripheral instance number
userConfigPtr	Pointer to the FLEXIO_SPI master user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
master	Pointer to the FLEXIO_SPI master driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using FLEXIO_SPI_DRV_MasterDeinit().

Returns

Error or success status returned by API

3.39.5.5 FLEXIO_SPI_DRV_MasterSetBaudRate()

```
status_t FLEXIO_SPI_DRV_MasterSetBaudRate (
            flexio_spi_master_state_t * master,
            uint32_t baudRate )
```

Set the baud rate for any subsequent SPI communication.

This function sets the baud rate for the SPI master. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call FLEXIO_SPI_DRV_MasterGetBaudRate() after FLEXIO_SPI_DRV_MasterSetBaudRate() to check what baud rate was actually set.

Parameters

master	Pointer to the FLEXIO_SPI master driver context structure.
baudRate	the desired baud rate in hertz

Returns

Error or success status returned by API

3.39.5.6 FLEXIO_SPI_DRV_MasterTransfer()

```
status_t FLEXIO_SPI_DRV_MasterTransfer (
            flexio_spi_master_state_t * master,
            const uint8_t * txData,
            uint8_t * rxData,
            uint32_t dataSize )
```

Perform a non-blocking SPI master transaction.

This function performs an SPI full-duplex transaction, transmit and receive in parallel. If only transmit or receive are required, it is possible to provide NULL pointers for txData or rxData. The transfer is non-blocking, the function only initiates the transfer and then returns, leaving the transfer to complete asynchronously). FLEXIO_SPI_DRV↩_MasterGetStatus() can be called to check the status of the transfer.

Parameters

master	Pointer to the FLEXIO_SPI master driver context structure.
txData	pointer to the data to be transmitted
rxData	pointer to the buffer where to store received data
dataSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.39.5.7 FLEXIO_SPI_DRV_MasterTransferAbort()

```
status_t FLEXIO_SPI_DRV_MasterTransferAbort (
            flexio_spi_master_state_t * master )
```

Aborts a non-blocking SPI master transaction.

This function aborts a non-blocking SPI transfer.

Parameters

master	Pointer to the FLEXIO_SPI master driver context structure.

Returns

Error or success status returned by API

3.39.5.8 FLEXIO_SPI_DRV_MasterTransferBlocking()

```
status_t FLEXIO_SPI_DRV_MasterTransferBlocking (
            flexio_spi_master_state_t * master,
            const uint8_t * txData,
            uint8_t * rxData,
            uint32_t dataSize,
            uint32_t timeout )
```

Perform a blocking SPI master transaction.

This function performs an SPI full-duplex transaction, transmit and receive in parallel. If only transmit or receive are required, it is possible to provide NULL pointers for txData or rxData. The transfer is blocking, the function only returns when the transfer is complete.

Parameters

master	Pointer to the FLEXIO_SPI master driver context structure.
txData	pointer to the data to be transmitted
rxData	pointer to the buffer where to store received data
dataSize	length in bytes of the data to be transferred
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.39.5.9 FLEXIO_SPI_DRV_SlaveDeinit()

```
static status_t FLEXIO_SPI_DRV_SlaveDeinit (
            flexio_spi_slave_state_t * slave )  [inline], [static]
```

De-initialize the FLEXIO_SPI slave mode driver.

This function de-initializes the FLEXIO_SPI driver in slave mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

Parameters

slave	Pointer to the FLEXIO_SPI slave driver context structure.

Returns

Error or success status returned by API Implements : FLEXIO_SPI_DRV_SlaveDeinit_Activity

3.39.5.10 FLEXIO_SPI_DRV_SlaveGetStatus()

```
static status_t FLEXIO_SPI_DRV_SlaveGetStatus (
            flexio_spi_slave_state_t * slave,
            uint32_t * bytesRemaining )  [inline], [static]
```

Get the status of the current non-blocking SPI slave transaction.

This function returns the current status of a non-blocking SPI slave transaction. A return code of STATUS_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

Parameters

slave	Pointer to the FLEXIO_SPI slave driver context structure.
bytesRemaining	the remaining number of bytes to be transferred

Returns

Error or success status returned by API Implements : FLEXIO_SPI_DRV_SlaveGetStatus_Activity

3.39.5.11 FLEXIO_SPI_DRV_SlaveInit()

```
status_t FLEXIO_SPI_DRV_SlaveInit (
            uint32_t instance,
            const flexio_spi_slave_user_config_t * userConfigPtr,
            flexio_spi_slave_state_t * slave )
```

Initialize the FLEXIO_SPI slave mode driver.

This function initializes the FLEXIO_SPI driver in slave mode.

Parameters

instance	FLEXIO peripheral instance number
userConfigPtr	Pointer to the FLEXIO_SPI slave user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
slave	Pointer to the FLEXIO_SPI slave driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using FLEXIO_SPI_DRV_SlaveDeinit().

Returns

Error or success status returned by API

3.39.5.12 FLEXIO_SPI_DRV_SlaveTransfer()

```
static status_t FLEXIO_SPI_DRV_SlaveTransfer (
            flexio_spi_slave_state_t * slave,
            const uint8_t * txData,
            uint8_t * rxData,
            uint32_t dataSize ) [inline], [static]
```

Perform a non-blocking SPI slave transaction.

This function performs an SPI full-duplex transaction, transmit and receive in parallel. If only transmit or receive are required, it is possible to provide NULL pointers for txData or rxData. The transfer is non-blocking, the function only initiates the transfer and then returns, leaving the transfer to complete asynchronously). FLEXIO_SPI_DRV←_SlaveGetStatus() can be called to check the status of the transfer.

Parameters

slave	Pointer to the FLEXIO_SPI slave driver context structure.
txData	pointer to the data to be transmitted
rxData	pointer to the buffer where to store received data
dataSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API Implements : FLEXIO_SPI_DRV_SlaveTransfer_Activity

3.39.5.13 FLEXIO_SPI_DRV_SlaveTransferAbort()

```
static status_t FLEXIO_SPI_DRV_SlaveTransferAbort (
            flexio_spi_slave_state_t * slave )  [inline], [static]
```

Aborts a non-blocking SPI slave transaction.

This function aborts a non-blocking SPI transfer.

Parameters

slave	Pointer to the FLEXIO_SPI slave driver context structure.

Returns

Error or success status returned by API Implements : FLEXIO_SPI_DRV_SlaveTransferAbort_Activity

3.39.5.14 FLEXIO_SPI_DRV_SlaveTransferBlocking()

```
static status_t FLEXIO_SPI_DRV_SlaveTransferBlocking (
            flexio_spi_slave_state_t * slave,
            const uint8_t * txData,
            uint8_t * rxData,
            uint32_t dataSize,
            uint32_t timeout )  [inline], [static]
```

Perform a blocking SPI slave transaction.

This function performs an SPI full-duplex transaction, transmit and receive in parallel. If only transmit or receive are required, it is possible to provide NULL pointers for txData or rxData. The transfer is blocking, the function only returns when the transfer is complete.

Parameters

slave	Pointer to the FLEXIO_SPI slave driver context structure.
txData	pointer to the data to be transmitted
rxData	pointer to the buffer where to store received data
dataSize	length in bytes of the data to be transferred
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API Implements : FLEXIO_SPI_DRV_SlaveTransferBlocking_Activity

3.40 FlexIO UART Driver

3.40.1 Detailed Description

UART communication over FlexIO module (FLEXIO_UART)

The FLEXIO_UART Driver allows UART communication using the FlexIO module in the S32144K processor.

Features

- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transmit and receive functions
- Configurable baud rate and number of bits
- Single stop bit only
- Parity bit not supported

Functionality

Initialization

Before using any Flexio driver the device must first be initialized using function FLEXIO_DRV_InitDevice. Then the FLEXIO_UART Driver must be initialized, using function FLEXIO_UART_DRV_Init(). It is possible to use more driver instances on the same FlexIO device, as long as sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using FLEXIO_UART_DRV_Deinit(). This will release the hardware resources, allowing other driver instances to be initialized.

Choosing transmit/receive mode

To initialize the UART driver in transmit / receive mode the `direction` field of the configuration structure must be set to `FLEXIO_UART_DIRECTION_TX` / `FLEXIO_UART_DIRECTION_RX` when calling FLEXIO_UART_DRV_Init(). Once configured for one direction the driver must be used only for the chosen direction until it is de-initialized. One driver instance can only work in one direction at a time, but more driver instances can be created on the same device, up to the number of shifters present on the device (for example on S32K144 up to 4 driver instances can run in parallel on one device).

Setting the baud rate and bit count

The baud rate and bit count are provided at initialization time through the master configuration structure, but they can be changed at runtime by using function FLEXIO_UART_DRV_SetConfig(). Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call FLEXIO_UART_DRV_GetBaudRate() to check what baud rate was actually set.

Transmitting / Receiving

To send or receive data to/from the currently configured slave address, use functions FLEXIO_UART_DRV_Send←Data() or FLEXIO_UART_DRV_ReceiveData() (or their blocking counterparts). Continuous send/receive can be realized by registering a user callback function. When the driver completes the transmission or reception of the current buffer, it will invoke the user callback with an appropriate event. The callback function can the use FLEXI←O_UART_DRV_SetTxBuffer() orFLEXIO_UART_DRV_SetRxBuffer() to provide a new buffer.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return STATUS_SUCCESS, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application will be notified through the user callback when the transfer completes, or it can check the status of the current transfer by calling FLEXIO←_UART_DRV_GetStatus(). If the transfer is still ongoing this function will return STATUS_BUSY. If the transfer is completed, the function will return either STATUS_SUCCESS or an error code, depending on the outcome of the last transfer.

The driver supports interrupt, DMA and polling mode. In polling mode the function FLEXIO_UART_DRV_Get←Status() ensures the progress of the transfer by checking and handling transmit and receive events reported by the FlexIO module. The application should ensure that this function is called often enough (at least once per transferred byte) to avoid Tx underflows or Rx overflows. In DMA mode the DMA channel that will be used by the driver is received through the configuration structure. The channel must be initialized by the application before the flexio_←uart driver is initialized. The flexio_uart driver will only set the DMA request source.

Important Notes

- Before using the FLEXIO_UART Driver the FlexIO clock must be configured. Refer to SCG HAL and PCC HAL for clock configuration.
- Before using the FLEXIO_UART Driver the pins must be routed to the FlexIO module. Refer to PORT HAL for pin routing configuration. Note that any of the available FlexIO pins can be used for the UART TX / RX line (configurable at initialization time). If more than one driver instance is used on the same Flexio module, it is the responsibility of the application to ensure there are no conflicts between pins.
- The driver enables the interrupts for the corresponding FlexIO module, but any interrupt priority setting must be done by the application.
- Timeout feature for blocking transfers does not work in polling mode.
- This driver needs one shifter and one timer for its operation. Initialization will fail if there are not enough shifters and timers available on the FlexIO device.
- This driver needs one DMA channel for its operation when it is initialized in DMA mode. The DMA channels must be initialized by the application before initializing the driver. Refer to EDMA driver for DMA channels initialization.
- If the application uses an RTOS, this driver uses a semaphore for blocking transfers. Initialization will fail if the semaphore cannot be created. If the driver uses polling mode no semaphore is used.
- If the application uses an RTOS, the FlexIO drivers use a mutex for channel allocation. Only one mutex per device is needed, not per driver instance. Device initialization will fail if the mutex cannot be created.

Data Structures

- struct flexio_uart_user_config_t

    Driver configuration structure. More...
- struct flexio_uart_state_t

    Driver internal context structure. More...

Enumerations

- enum flexio_uart_driver_direction_t { FLEXIO_UART_DIRECTION_TX = 0x01U, FLEXIO_UART_DIRECT←ION_RX = 0x00U }

  flexio_uart driver direction (tx or rx)

FLEXIO_UART Driver

- status_t FLEXIO_UART_DRV_Init (uint32_t instance, const flexio_uart_user_config_t ∗userConfigPtr, flexio_uart_state_t ∗state)

  Initialize the FLEXIO_UART driver.
- status_t FLEXIO_UART_DRV_Deinit (flexio_uart_state_t ∗state)

  De-initialize the FLEXIO_UART driver.
- status_t FLEXIO_UART_DRV_SetConfig (flexio_uart_state_t ∗state, uint32_t baudRate, uint8_t bitCount)

  Set the baud rate and bit width for any subsequent UART communication.
- status_t FLEXIO_UART_DRV_GetBaudRate (flexio_uart_state_t ∗state, uint32_t ∗baudRate)

  Get the currently configured baud rate.
- status_t FLEXIO_UART_DRV_SendDataBlocking (flexio_uart_state_t ∗state, const uint8_t ∗txBuff, uint32←_t txSize, uint32_t timeout)

  Perform a blocking UART transmission.
- status_t FLEXIO_UART_DRV_SendData (flexio_uart_state_t ∗state, const uint8_t ∗txBuff, uint32_t txSize)

  Perform a non-blocking UART transmission.
- status_t FLEXIO_UART_DRV_ReceiveDataBlocking (flexio_uart_state_t ∗state, uint8_t ∗rxBuff, uint32_←t rxSize, uint32_t timeout)

  Perform a blocking UART reception.
- status_t FLEXIO_UART_DRV_ReceiveData (flexio_uart_state_t ∗state, uint8_t ∗rxBuff, uint32_t rxSize)

  Perform a non-blocking UART reception.
- status_t FLEXIO_UART_DRV_GetStatus (flexio_uart_state_t ∗state, uint32_t ∗bytesRemaining)

  Get the status of the current non-blocking UART transfer.
- status_t FLEXIO_UART_DRV_TransferAbort (flexio_uart_state_t ∗state)

  Aborts a non-blocking UART transfer.
- status_t FLEXIO_UART_DRV_SetRxBuffer (flexio_uart_state_t ∗state, uint8_t ∗rxBuff, uint32_t rxSize)

  Provide a buffer for receiving data.
- status_t FLEXIO_UART_DRV_SetTxBuffer (flexio_uart_state_t ∗state, const uint8_t ∗txBuff, uint32_t txSize)

  Provide a buffer for transmitting data.

3.40.2 Data Structure Documentation

3.40.2.1 struct flexio_uart_user_config_t

Driver configuration structure.

This structure is used to provide configuration parameters for the flexio_uart driver at initialization time. Implements : flexio_uart_user_config_t_Class

Data Fields

- flexio_driver_type_t driverType
- uint32_t baudRate
- uint8_t bitCount
- flexio_uart_driver_direction_t direction
- uint8_t dataPin
- flexio_callback_t callback
- void ∗ callbackParam
- uint8_t dmaChannel

Field Documentation

3.40.2.1.1 baudRate

`uint32_t baudRate`

Baud rate in hertz

3.40.2.1.2 bitCount

`uint8_t bitCount`

Number of bits per word

3.40.2.1.3 callback

`flexio_callback_t callback`

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

3.40.2.1.4 callbackParam

`void* callbackParam`

Parameter for the callback function

3.40.2.1.5 dataPin

`uint8_t dataPin`

Flexio pin to use as Tx or Rx pin

3.40.2.1.6 direction

`flexio_uart_driver_direction_t direction`

Driver direction: Tx or Rx

3.40.2.1.7 dmaChannel

`uint8_t dmaChannel`

DMA channel number. Only used in DMA mode

3.40.2.1.8 driverType

`flexio_driver_type_t driverType`

Driver type: interrupts/polling/DMA

3.40.2.2 struct flexio_uart_state_t

Driver internal context structure.

This structure is used by the flexio_uart driver for its internal logic. It must be provided by the application through the FLEXIO_UART_DRV_Init() function, then it cannot be freed until the driver is de-initialized using FLEXIO_U↩ART_DRV_DeInit(). The application should make no assumptions about the content of this structure.

3.40.3 Enumeration Type Documentation

3.40.3.1 flexio_uart_driver_direction_t

`enum flexio_uart_driver_direction_t`

flexio_uart driver direction (tx or rx)

This structure describes the direction configuration options for the flexio_uart driver. Implements : flexio_uart_↩driver_direction_t_Class

Enumerator

FLEXIO_UART_DIRECTION_TX	Tx UART driver
FLEXIO_UART_DIRECTION_RX	Rx UART driver

3.40.4 Function Documentation

3.40.4.1 FLEXIO_UART_DRV_Deinit()

```
status_t FLEXIO_UART_DRV_Deinit (
            flexio_uart_state_t * state )
```

De-initialize the FLEXIO_UART driver.

This function de-initializes the FLEXIO_UART driver. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

Parameters

state	Pointer to the FLEXIO_UART driver context structure.

Returns

Error or success status returned by API

3.40.4.2 FLEXIO_UART_DRV_GetBaudRate()

```
status_t FLEXIO_UART_DRV_GetBaudRate (
            flexio_uart_state_t * state,
            uint32_t * baudRate )
```

Get the currently configured baud rate.

This function returns the currently configured UART baud rate.

Parameters

state	Pointer to the FLEXIO_UART driver context structure.
baudRate	the current baud rate in hertz

Returns

Error or success status returned by API

3.40.4.3 FLEXIO_UART_DRV_GetStatus()

```
status_t FLEXIO_UART_DRV_GetStatus (
            flexio_uart_state_t * state,
            uint32_t * bytesRemaining )
```

Get the status of the current non-blocking UART transfer.

This function returns the current status of a non-blocking UART transfer. A return code of STATUS_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

Parameters

state	Pointer to the FLEXIO_UART driver context structure.
bytesRemaining	the remaining number of bytes to be transferred

Returns

Error or success status returned by API

3.40.4.4 FLEXIO_UART_DRV_Init()

```
status_t FLEXIO_UART_DRV_Init (
            uint32_t instance,
            const flexio_uart_user_config_t * userConfigPtr,
            flexio_uart_state_t * state )
```

Initialize the FLEXIO_UART driver.

This function initializes the FLEXIO_UART driver.

Parameters

instance	FLEXIO peripheral instance number
userConfigPtr	Pointer to the FLEXIO_UART user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
state	Pointer to the FLEXIO_UART driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using FLEXIO_UART_DRV_Deinit().

Returns

Error or success status returned by API

3.40.4.5 FLEXIO_UART_DRV_ReceiveData()

```
status_t FLEXIO_UART_DRV_ReceiveData (
            flexio_uart_state_t * state,
            uint8_t * rxBuff,
            uint32_t rxSize )
```

Perform a non-blocking UART reception.

This function receives a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO_UART_DRV_GetReceive↩ Status() function (if the driver is initialized in polling mode).

Parameters

state	Pointer to the FLEXIO_UART driver context structure.
rxBuff	pointer to the receive buffer
rxSize	length in bytes of the data to be received

Returns

Error or success status returned by API

3.40.4.6 FLEXIO_UART_DRV_ReceiveDataBlocking()

```
status_t FLEXIO_UART_DRV_ReceiveDataBlocking (
             flexio_uart_state_t * state,
             uint8_t * rxBuff,
             uint32_t rxSize,
             uint32_t timeout )
```

Perform a blocking UART reception.

This function receives a block of data and only returns when the transmission is complete.

Parameters

state	Pointer to the FLEXIO_UART driver context structure.
rxBuff	pointer to the receive buffer
rxSize	length in bytes of the data to be received
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.40.4.7 FLEXIO_UART_DRV_SendData()

```
status_t FLEXIO_UART_DRV_SendData (
             flexio_uart_state_t * state,
             const uint8_t * txBuff,
             uint32_t txSize )
```

Perform a non-blocking UART transmission.

This function sends a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO_UART_DRV_GetTransmitStatus() function (if the driver is initialized in polling mode).

Parameters

state	Pointer to the FLEXIO_UART driver context structure.
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.40.4.8 FLEXIO_UART_DRV_SendDataBlocking()

```
status_t FLEXIO_UART_DRV_SendDataBlocking (
             flexio_uart_state_t * state,
             const uint8_t * txBuff,
             uint32_t txSize,
             uint32_t timeout )
```

Perform a blocking UART transmission.

This function sends a block of data and only returns when the transmission is complete.

Parameters

state	Pointer to the FLEXIO_UART driver context structure.
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.40.4.9 FLEXIO_UART_DRV_SetConfig()

```
status_t FLEXIO_UART_DRV_SetConfig (
             flexio_uart_state_t * state,
             uint32_t baudRate,
             uint8_t bitCount )
```

Set the baud rate and bit width for any subsequent UART communication.

This function sets the baud rate and bit width for the UART driver. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call FLEXIO_UART_DRV_GetBaudRate() after FLEXIO_UART_DRV_SetConfig() to check what baud rate was actually set.

Parameters

state	Pointer to the FLEXIO_UART driver context structure.
baudRate	the desired baud rate in hertz
bitCount	number of bits per word

Returns

Error or success status returned by API

3.40.4.10 FLEXIO_UART_DRV_SetRxBuffer()

```
status_t FLEXIO_UART_DRV_SetRxBuffer (
            flexio_uart_state_t * state,
            uint8_t * rxBuff,
            uint32_t rxSize )
```

Provide a buffer for receiving data.

This function can be used to provide a new buffer for receiving data to the driver. It can be called from the user callback when event STATUS_UART_RX_OVERRUN is reported. This way the reception will continue without interruption.

Parameters

state	Pointer to the FLEXIO_UART driver context structure.
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.40.4.11 FLEXIO_UART_DRV_SetTxBuffer()

```
status_t FLEXIO_UART_DRV_SetTxBuffer (
            flexio_uart_state_t * state,
            const uint8_t * txBuff,
            uint32_t txSize )
```

Provide a buffer for transmitting data.

This function can be used to provide a new buffer for transmitting data to the driver. It can be called from the user callback when event STATUS_UART_TX_UNDERRUN is reported. This way the transmission will continue without interruption.

Parameters

state	Pointer to the FLEXIO_UART driver context structure.
txBuff	pointer to the buffer containing transmit data
txSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.40.4.12 FLEXIO_UART_DRV_TransferAbort()

```
status_t FLEXIO_UART_DRV_TransferAbort (
            flexio_uart_state_t * state )
```

Aborts a non-blocking UART transfer.

This function aborts a non-blocking UART transfer.

Parameters

state	Pointer to the FLEXIO_UART driver context structure.

Returns

Error or success status returned by API

3.41 FlexTimer (FTM)

3.41.1 Detailed Description

FlexTimer Peripheral Driver.

Hardware background

The FTM of the S32K144 is based on a 16 bits counter and supports: input capture, output compare, PWM and some instances include quadrature decoder. The main features are:

•FTM source clock is selectable (Source clock can be the system clock, the fixed frequency clock, or an external clock)

•Prescaler: 1, 2, 4, 8, 16, 32, 64, 128

•16 bit counter (up and up-down counting)

•Each channel can be configured for input capture, output compare, or edge-aligned PWM mode.

•Input Capture mode (single edge, dual edge)

•Output Compare mode (set, cleared or toggle on match)

•All channels can be configured for center-aligned PWM mode.

•Each pair of channels can be combined to generate a PWM signal with independent control of both edges of PWM signal and with dead-time insertion.

•Up to 4 fault inputs for global fault control

•Dual edge capture for pulse and period width measurement

•Quadrature decoder with input filters, relative position counting, and interrupt on position count or capture of position count on external event.

How to use FTM driver in your application

For all operation modes (without Quadrature Decoder mode) the user need to configure ftm_user_config_t. This structure will be used for initialization (FTM_DRV_Init). The next functions used are specific for each operation mode.

Output compare mode

For this mode the user needs to configure maximum counter value, number of channels used and output mode for each channel (toggle/clear/set on match). This information is stored in ftm_output_cmp_param_t data type and are used in FTM_DRV_InitOutputCompare. Next step is to set a value for comparison with FTM_DRV_UpdateOutput↩CompareChannel.

Example:

```
/* The state structure of instance in the output compare mode */
ftm_state_t stateOutputCompare;
#define FTM_OUTPUT_COMPARE_INSTANCE        2UL
/* Channels configuration structure for PWM output compare */
ftm_output_cmp_ch_param_t PWM_OutputCompareChannelConfig[2] =
{
    {
        0,                          /* Channel id */
        FTM_TOGGLE_ON_MATCH,        /* Output mode */
        10000U,                     /* Compared value */
        false,                      /* External Trigger */
    },
    {
        1,                          /* Channel id */
        FTM_TOGGLE_ON_MATCH,        /* Output mode */
        20000U,                     /* Compared value */
        false,                      /* External Trigger */
    }
};

/* Output compare configuration for PWM */
ftm_output_cmp_param_t PWM_OutputCompareConfig =
{
    2,                              /* Number of channels */
    FTM_MODE_OUTPUT_COMPARE,        /* FTM mode */
    40000U,                         /* Maximum count value */
    PWM_OutputCompareChannelConfig  /* Channels configuration */
};
/* Timer mode configuration for PWM */
/* Global configuration of PWM */
ftm_user_config_t  PWM_InitConfig =
{
    {
        true,                       /* Software trigger state */
        false,                      /* Hardware trigger 1 state */
        false,                      /* Hardware trigger 2 state */
        false,                      /* Hardware trigger 3 state */
        true,                       /* Maximum loading point state */
        true,                       /* Min loading point state */
        FTM_SYSTEM_CLOCK,           /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,           /* Update mode for CNTIN register */
        false,                      /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,             /* select synchronization method */
    },
    FTM_MODE_OUTPUT_COMPARE,        /* Mode of operation for FTM */
    FTM_CLOCK_DIVID_BY_4,           /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK,     /* FTM clock source */
    FTM_BDM_MODE_11,                /* FTM debug mode */
    false,                          /* Interrupt state */
    false                           /* Initialization trigger */
};
FTM_DRV_Init(FTM_OUTPUT_COMPARE_INSTANCE, &PWM_InitConfig, &stateOutputCompare);
FTM_DRV_InitOutputCompare(FTM_OUTPUT_COMPARE_INSTANCE, &PWM_OutputCompareConfig);
/* If you want to change compared value */
FTM_DRV_UpdateOutputCompareChannel(FTM_OUTPUT_COMPARE_INSTANCE, 0UL, 1500
      0U );
```

PWM mode

For this mode the user needs to configure parameters such: number of PWM channels, frequency, dead time, fault channels and duty cycle, alignment (edge or center). All this information is included in ftm_pwm_param_t data type.

FTM_DRV_UpdatePwmChannel can be used to update duty cycles at run time. If the type of update in the duty cycle when the duty cycle can have value between 0x0 (0%) and 0x8000 (100%). If the type of update in ticks when the firstEdge and secondEdge variables can have value between 0 and ftmPeriod which is stored in the state structure.

Example:

```
/* The state structure of instance in the PWM mode */
ftm_state_t statePwm;
#define FTM_PWM_INSTANCE        1UL
/* Fault configuration structure */
ftm_pwm_fault_param_t PWM_FaultConfig =
{
    false,
    true,
    5U,                             /* Fault filter value */
    FTM_FAULT_CONTROL_MAN_EVEN,
    {
        {
            true,                   /* Fault channel state (Enabled/Disabled) */
            false,                  /* Fault channel filter state (Enabled/Disabled) */
            FTM_POLARITY_HIGH,      /* Channel output state on fault */
        },
        {
            false,                  /* Fault Channel state (Enabled/Disabled) */
            false,                  /* Fault channel filter state (Enabled/Disabled) */
            FTM_POLARITY_LOW        /* Channel output state on fault */
        },
        {
            false,                  /* Fault Channel state (Enabled/Disabled) */
            false,                  /* Fault channel filter state (Enabled/Disabled) */
            FTM_POLARITY_LOW        /* Channel output state on fault */
        },
        {
            false,                  /* Fault Channel state (Enabled/Disabled) */
            false,                  /* Fault channel filter state (Enabled/Disabled) */
            FTM_POLARITY_LOW        /* Channel output state on fault */
        }
    }
};
/* Independent channels configuration structure for PWM */
ftm_independent_ch_param_t PWM_IndependentChannelsConfig[1] =
{
    {
        0U,                         /* hwChannelId */
        FTM_POLARITY_HIGH,          /* edgeMode */
        10922,                      /* uDutyCyclePercent (0-0x8000) */
        false,                      /* External Trigger */
    }
};

/* PWM configuration for PWM */
ftm_pwm_param_t PWM_PwmConfig =
{
    1U,                             /* Number of independent PWM channels */
    0U,                             /* Number of combined PWM channels */
    FTM_MODE_EDGE_ALIGNED_PWM,      /* PWM mode */
    0U,                             /* DeadTime Value */
    FTM_DEADTIME_DIVID_BY_4,        /* DeadTime clock divider */
    7481U,                          /* PWM frequency */
    PWM_IndependentChannelsConfig,  /* Independent PWM channels configuration structure */
    NULL,                           /* Combined PWM channels configuration structure */
    &PWM_FaultConfig                /* PWM fault configuration structure */
};
/* Timer mode configuration for PWM */
/* Global configuration of PWM */
ftm_user_config_t  PWM_InitConfig =
{
    {
        true,                       /* Software trigger state */
        false,                      /* Hardware trigger 1 state */
        false,                      /* Hardware trigger 2 state */
        false,                      /* Hardware trigger 3 state */
        true,                       /* Maximum loading point state */
        true,                       /* Min loading point state */
        FTM_SYSTEM_CLOCK,           /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,           /* Update mode for CNTIN register */
        false,                      /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,             /* Select synchronization method */
    },
    FTM_MODE_EDGE_ALIGNED_PWM,      /* PWM mode */
    FTM_CLOCK_DIVID_BY_4,           /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK,     /* FTM clock source */
    FTM_BDM_MODE_11,                /* FTM debug mode */
    false,                          /* Interrupt state */
    false                           /* Initialization trigger */
};
FTM_DRV_Init(FTM_PWM_INSTANCE, &PWM_InitConfig, &statePwm);
FTM_DRV_InitPwm(FTM_PWM_INSTANCE, &PWM_PwmConfig);
/* It's recommended to use softwareTrigger = true */
/* SECOND_EDGE value is used only when PWM is used in combined mode */
FTM_DRV_UpdatePwmChannel(FTM_PWM_INSTANCE, 0UL,
      FTM_PWM_UPDATE_IN_DUTY_CYCLE, 0x800, 0x2000, true);
```

PWM in Modified Combine mode

For this mode the user needs to configure parameters such: number of PWM channels, frequency, dead time, fault channels and duty cycle, alignment (edge or center). All this information is included in ftm_pwm_param_t data type. The Modified Combine PWM mode is intended to support the generation of PWM signals where the period is not modified while the signal is being generated, but the duty cycle will be varied. FTM_DRV_UpdatePwmChannel can be used to update duty cycles at run time. If the type of update in the duty cycle when the duty cycle can have value between 0x0 (0%) and 0x8000 (100%). If the type of update in ticks when the firstEdge and secondEdge variables can have value between 0 and ftmPeriod which is stored in the state structure.In this mode, an even channel (n) and adjacent odd channel (n+1) are combined to generate a PWM signal in the channel (n) output. Thus, the channel (n) match edge is fixed and the channel (n+1) match edge can be varied.

Example:

```
/* The state structure of instance in the PWM mode */
ftm_state_t statePwm;
#define FTM_PWM_INSTANCE        0UL
/* Fault configuration structure */
ftm_pwm_fault_param_t PWM_FaultConfig =
{
    false,
    true,
    5U,                             /* Fault filter value */
    FTM_FAULT_CONTROL_MAN_EVEN,
    {
        {
            true,                   /* Fault channel state (Enabled/Disabled )*/
            false,                  /* Fault channel filter state (Enabled/Disabled) */
            FTM_POLARITY_HIGH,      /* Channel output state on fault */
        },
        {
            false,                  /* Fault Channel state (Enabled/Disabled) */
            false,                  /* Fault channel filter state (Enabled/Disabled) */
            FTM_POLARITY_LOW        /* Channel output state on fault */
        },
        {
            false,                  /* Fault Channel state (Enabled/Disabled) */
            false,                  /* Fault channel filter state (Enabled/Disabled) */
            FTM_POLARITY_LOW        /* Channel output state on fault */
        },
        {
            false,                  /* Fault Channel state (Enabled/Disabled) */
            false,                  /* Fault channel filter state (Enabled/Disabled) */
            FTM_POLARITY_LOW        /* Channel output state on fault */
        }
    }
};
/* Combine channels configuration structure for PWM */
 ftm_combined_ch_param_t flexTimer1_CombinedChannelsConfig[1] =
{
    {
        0U,                         /* Hardware channel for channel (n) */
        512U,                       /* First edge time */
        16384U,                     /* Second edge time */
        false,                      /* Dead time enabled/disabled */
        true,                       /* The modified combine mode enabled/disabled */
        FTM_POLARITY_HIGH,          /* Channel polarity */
        true,                       /* Output enabled/disabled for channel (n+1) */
        FTM_MAIN_DUPLICATED,        /* Polarity for channel (n+1) */
        false,                      /* External Trigger on the channel (n) */
        false,                      /* External Trigger on the channel (n+1) */
    }
};
/* PWM configuration for PWM */
ftm_pwm_param_t PWM_PwmConfig =
{
    0U,                             /* Number of independent PWM channels */
    1U,                             /* Number of combined PWM channels */
    FTM_MODE_EDGE_ALIGNED_PWM,      /* PWM mode */
    0U,                             /* DeadTime Value */
    FTM_DEADTIME_DIVID_BY_4,        /* DeadTime clock divider */
    7481U,                          /* PWM frequency */
    NULL,                           /* Independent PWM channels configuration structure */
    flexTimer1_CombinedChannelsConfig, /* Combined PWM channels configuration structure */
    &PWM_FaultConfig                /* PWM fault configuration structure */
};
/* Timer mode configuration for PWM */
/* Global configuration of PWM */
ftm_user_config_t  PWM_InitConfig =
{
    {
        true,                       /* Software trigger state */
        false,                      /* Hardware trigger 1 state */
        false,                      /* Hardware trigger 2 state */
        false,                      /* Hardware trigger 3 state */
        true,                       /* Maximum loading point state */
        true,                       /* Min loading point state */
        FTM_SYSTEM_CLOCK,           /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,           /* Update mode for CNTIN register */
        false,                      /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,             /* Select synchronization method */
    },
    FTM_MODE_EDGE_ALIGNED_PWM,      /* PWM mode */
    FTM_CLOCK_DIVID_BY_4,           /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK,     /* FTM clock source */
    FTM_BDM_MODE_11,                /* FTM debug mode */
    false,                          /* Interrupt state */
    false                           /* Initialization trigger */
};
FTM_DRV_Init(FTM_PWM_INSTANCE, &PWM_InitConfig, &statePwm);
FTM_DRV_InitPwm(FTM_PWM_INSTANCE, &PWM_PwmConfig);
/* It's recommended to use softwareTrigger = true */
/* SECOND_EDGE value is used only when PWM is used in combined mode */
FTM_DRV_UpdatePwmChannel(FTM_PWM_INSTANCE, 0UL,
      FTM_PWM_UPDATE_IN_DUTY_CYCLE, 0x0, 0x2000, true);
```

Single edge input capture mode

For this mode the user needs to configure parameters such: maximum counter value, number of channels, input capture operation mode (for single edge input are used edge detect mode) and edge alignment. All this information is included in ftm_input_param_t.

Example:

```
/* The state structure of instance in the input capture mode */
ftm_state_t stateInputCapture;
#define FTM_IC_INSTANCE        0UL
/* Channels configuration structure for inputCapture input capture */
ftm_input_ch_param_t inputCapture_InputCaptureChannelConfig[1] =
{
    {
    0U,                             /* Channel id */
    FTM_EDGE_DETECT,                /* Input capture operation Mode */
    FTM_RISING_EDGE,                /* Edge alignment Mode */
    FTM_NO_MEASUREMENT,             /* Signal measurement operation type */
    0U,                             /* Filter value */
    false,                          /* Filter disabled */
    true                            /* Continuous mode measurement */
    NULL,                           /* Vector of callbacks  parameters for channels events */
    NULL                            /* Vector of callbacks for channels events */
    }
};
/* Input capture configuration for inputCapture */
ftm_input_param_t inputCapture_InputCaptureConfig =
{
    1U,                                     /* Number of channels */
    65535U,                                 /* Maximum count value */
    inputCapture_InputCaptureChannelConfig  /* Channels configuration */
};
/* Timer mode configuration for inputCapture */
/* Global configuration of inputCapture */
ftm_user_config_t  inputCapture_InitConfig =
{
    {
        false,                      /* Software trigger state */
        false,                      /* Hardware trigger 1 state */
        false,                      /* Hardware trigger 2 state */
        false,                      /* Hardware trigger 3 state */
        false,                      /* Maximum loading point state */
        false,                      /* Min loading point state */
        FTM_SYSTEM_CLOCK,           /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,           /* Update mode for CNTIN register */
        false,                      /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,             /* Select synchronization method */
    },
    FTM_MODE_INPUT_CAPTURE,         /* Mode of operation for FTM */
    FTM_CLOCK_DIVID_BY_4,           /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK,     /* FTM clock source */
    FTM_BDM_MODE_00,                /* FTM debug mode */
    false,                          /* Interrupt state */
    false                           /* Initialization trigger */
};
FTM_DRV_Init(FTM_IC_INSTANCE, &inputCapture_InitConfig, &stateInputCapture);
FTM_DRV_InitInputCapture(FTM_IC_INSTANCE, &inputCapture_InputCaptureConfig);
counter = FTM_DRV_GetInputCaptureMeasurement(FTM_IC_INSTANCE, 0UL);
```

FTM_DRV_GetInputCaptureMeasurement is now used in interrupt mode and this function is used to save time stamps in internal buffers.

Counter mode

For this mode the user needs to configure parameters like: counter mode (up-counting or up-down counting), maximum counter value, initial counter value. All this information is included in ftm_timer_param_t.

Example:

```
/* The state structure of instance in the input capture mode */
ftm_state_t stateTimer;
#define FTM_TIMER_INSTANCE        3UL
/* Timer mode configuration for Timer */
ftm_timer_param_t Timer_TimerConfig =
{
    FTM_MODE_UP_TIMER,              /* Counter mode */
    0U,                             /* Initial counter value */
    0x8000U                         /* Final counter value */
};

/* Global configuration of Timer*/
ftm_user_config_t  Timer_InitConfig =
{
    {
        false,                      /* Software trigger state */
        false,                      /* Hardware trigger 1 state */
        false,                      /* Hardware trigger 2 state */
        false,                      /* Hardware trigger 3 state */
        false,                      /* Maximum loading point state */
        false,                      /* Min loading point state */
        FTM_SYSTEM_CLOCK,           /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,           /* Update mode for CNTIN register */
        false,                      /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,             /* Select synchronization method */
    },
    FTM_MODE_UP_TIMER,              /* Mode of operation for FTM */
    FTM_CLOCK_DIVID_BY_2,           /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK,     /* FTM clock source */
    FTM_BDM_MODE_11,                /* FTM debug mode */
    false,                          /* Interrupt state */
    false                           /* Initialization trigger */
};
FTM_DRV_Init(FTM_TIMER_INSTANCE,&Timer_InitConfig, &stateTimer);
FTM_DRV_InitCounter(FTM_TIMER_INSTANCE, &Timer_TimerConfig);
FTM_DRV_CounterStart(FTM_TIMER_INSTANCE);
```

Quadrature decoder mode

For this mode the user needs to configure parameters like: maximum counter value, initial counter value, mode (Count and Direction Encoding mode, Count and Direction Encoding mode), and for both input phases polarity and filtering. All this information is included in ftm_quad_decode_config_t. In this mode the counter is clocked by the phase A and phase B. The current state of the decoder can be obtained using FTM_DRV_QuadGetState.

Hardware limitation:

In count and direction mode if initial value of the PHASE_A is HIGH the counter will be incremented.

Example:

```
/* The state structure of instance in the quadrature mode */
ftm_state_t stateQuad;
#define FTM_QUADRATURE_INSTANCE        0UL
ftm_quad_decoder_state_t quadra_state;
ftm_quad_decode_config_t quadrature_decoder_configuration =
{
    FTM_QUAD_COUNT_AND_DIR,         /* Quadrature decoder mode */
    0U,                             /* Initial counter value */
    32500U,                         /* Maximum counter value */
    {
        false,                      /* Filter state */
        0U,                         /* Filter value */
        FTM_QUAD_PHASE_NORMAL       /* Phase polarity */
    },
    {
        false,                      /* Filter state */
        0U,                         /* Filter value */
        FTM_QUAD_PHASE_NORMAL       /* Phase polarity */
    }
};
/* Timer mode configuration for Quadrature */
/* Global configuration of Quadrature */
ftm_user_config_t  Quadrature_InitConfig =
{
    {
        false,                      /* Software trigger state */
        false,                      /* Hardware trigger 1 state */
        false,                      /* Hardware trigger 2 state */
        false,                      /* Hardware trigger 3 state */
        false,                      /* Maximum loading point state */
        false,                      /* Min loading point state */
        FTM_SYSTEM_CLOCK,           /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,           /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,           /* Update mode for CNTIN register */
        false,                      /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,             /* Select synchronization method */
    },
    FTM_MODE_QUADRATURE_DECODER,    /* Mode of operation for FTM */
    FTM_CLOCK_DIVID_BY_2,           /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK,     /* FTM clock source */
    FTM_BDM_MODE_11,                /* FTM debug mode */
    false,                          /* Interrupt state */
    false                           /* Initialization trigger */
};
FTM_DRV_Init(FTM_QUADRATURE_INSTANCE, &Quadrature_InitConfig, &stateQuad);
FTM_DRV_QuadDecodeStart(FTM_QUADRATURE_INSTANCE, &quadrature_decoder_configuration);
quadra_state = FTM_DRV_QuadGetState(FTM_QUADRATURE_INSTANCE);
```

Modules

- FTM Driver

  FlexTimer Peripheral Driver.
- FTM HAL

  FlexTimer Module Hardware Abstraction Level. FTM HAL provides low level APIs for reading and writing to all hardware features of the FlexTimer module.

3.42 Flexible I/O (FlexIO)

3.42.1 Detailed Description

The FlexIO is a highly configurable module providing a wide range of functionality including:

- Emulation of a variety of serial communication protocols, such as SPI, I2C, I2S or UART, while requiring low CPU overhead and being more efficient that having multiple dedicated peripherals for each protocol.
- Flexible 16-bit timers with support for a variety of trigger, reset, enable and disable conditions
- PWM/Waveform generation

Several drivers are provided for this device, implementing a variety of communication protocols. There is also a common layer on which all the drivers are based, allowing more driver instances, either of the same type or different types, to function in parallel on the same FlexIO device. Each driver instance needs a certain number of Flex←IO resources (shifters and timers) and as long as there are enough free resources new driver instances can be initialized. The table below shows the driver types and the number of resources needed by each one:

Drivers	Timers	Shifters	Pins
SPI	2	2	4
I2C	2	2	2
I2S	2	2	4
UART	1	1	1

The number of timers and shifters available on a specific device can be found in the reference manual.

Modules

- FlexIO Common Driver

  Common services for FlexIO drivers.
- FlexIO HAL

  FlexIO Hardware Abstraction Layer (FLEXIO_HAL)
- FlexIO I2C Driver

  I2C communication over FlexIO module (FLEXIO_I2C)
- FlexIO I2S Driver

  I2S communication over FlexIO module (FLEXIO_I2S)
- FlexIO SPI Driver

  SPI communication over FlexIO module (FLEXIO_SPI)
- FlexIO UART Driver

  UART communication over FlexIO module (FLEXIO_UART)

3.43 GPIO HAL

3.43.1 Detailed Description

General Purpose Input/Output Hardware Abstraction Layer.

This HAL provides low-level access to all hardware features of the GPIO module.

This module provides methods for basic operations on the pins of the GPIO modules: configure data direction, read data, write, set and toggle pins.

The methods operate on the entire port (all 32 pins) so bit masks must be used to access individual bits. Some methods only act on the bits that have value '1' (leaving bits with value '0' unchanged), but others will affect all the pins, in this case a Read-Modify-Write operation is needed to keep the previous configuration.

This module assumes that the clocking and PORT configuration have been set up correctly. To see how to configure these modules, please refer to the Clock Manager (PCC HAL) and PORT modules.

In the following documentation the terms 'Set' and 'Clear' are used to refer to values of pins:

- The 'Clear' value is logic value of '0', also referred to as LOW
- The 'Set' value is logic value of '1', also referred to as HIGH

Writing a port means configuring each of its pins with the bit value corresponding to the pin's position.

The Set, Clear and Toggle methods operate only on pins with corresponding bits set to '1' in the bit mask. These methods do not require a Read-Modify-Write operation, as they will leave pins corresponding to bits with value '0' unchanged.

To use a pin as a GPIO pin, it needs to be configured as GPIO and, optionally, have pull up/down, drive strength etc. configured. This needs to be done using the PCC and PORT modules. The example below shows how to configure and use Pin 7 of Port C.

```
const uint8_t pin_num = 7U;
PCC_Type *pccBase = PCC_BASE_PTRS;
PCC_HAL_SetClockMode(pccBase, PCC_PORTC_INDEX, true);

PORT_HAL_SetMuxModeSel(PORTC, pin_num, PORT_MUX_AS_GPIO);
PORT_HAL_SetDriveStrengthMode(PORTC, pin_num,
      PORT_HIGH_DRIVE_STRENGTH);

GPIO_HAL_SetPinsDirection(PTC, 1U<<pin_num);

GPIO_HAL_SetPins(PTC, 1U<<pin_num);
```

GPIO

General GPIO functions.

- static void GPIO_HAL_WritePin (GPIO_Type ∗const baseAddr, const uint32_t pinNumber, const uint32_t pinValue)

    Write a pin of a port with a given value.
- static void GPIO_HAL_WritePins (GPIO_Type ∗const baseAddr, const uint32_t pins)

    Write all pins of a port.
- static uint32_t GPIO_HAL_GetPinsOutput (const GPIO_Type ∗const baseAddr)

    Get the current output from a port.

- static void GPIO_HAL_SetPins (GPIO_Type ∗const baseAddr, const uint32_t pins)

    Write pins with 'Set' value.
- static void GPIO_HAL_ClearPins (GPIO_Type ∗const baseAddr, const uint32_t pins)

    Write pins to 'Clear' value.
- static void GPIO_HAL_TogglePins (GPIO_Type ∗const baseAddr, const uint32_t pins)

    Toggle pins value.
- static uint32_t GPIO_HAL_ReadPins (const GPIO_Type ∗const baseAddr)

    Read input pins.
- static uint32_t GPIO_HAL_GetPinsDirection (const GPIO_Type ∗const baseAddr)

    Get the pins directions configuration for a port.
- static void GPIO_HAL_SetPinDirection (GPIO_Type ∗const baseAddr, const uint32_t pinNumber, const uint32_t pinDirection)

    Configure the direction for a certain pin from a port.
- static void GPIO_HAL_SetPinsDirection (GPIO_Type ∗const baseAddr, const uint32_t pins)

    Set the pins directions configuration for a port.
- static void GPIO_HAL_SetPortInputDisable (GPIO_Type ∗const baseAddr, const uint32_t pins)

    Set the pins input disable state for a port.
- static uint32_t GPIO_HAL_GetPortInputDisable (const GPIO_Type ∗const baseAddr)

    Get the pins input disable state for a port.

3.43.2 Function Documentation

3.43.2.1 GPIO_HAL_ClearPins()

```
static void GPIO_HAL_ClearPins (
            GPIO_Type *const baseAddr,
            const uint32_t pins )  [inline], [static]
```

Write pins to 'Clear' value.

This function configures output pins listed in parameter pins (bits that are '1') to have a 'cleared' value (LOW). Pins corresponding to '0' will be unaffected.

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)
pins	pin mask of bits to be cleared. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit: • 0: corresponding pin is unaffected • 1: corresponding pin is cleared (set to LOW) Implements : GPIO_HAL_ClearPins_Activity

3.43.2.2 GPIO_HAL_GetPinsDirection()

```
static uint32_t GPIO_HAL_GetPinsDirection (
            const GPIO_Type *const baseAddr )  [inline], [static]
```

Get the pins directions configuration for a port.

This function returns the current pins directions for a port. Pins corresponding to bits with value of '1' are configured as output and pins corresponding to bits with value of '0' are configured as input.

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)

Returns

GPIO directions. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit:

- 0: corresponding pin is set to input
- 1: corresponding pin is set to output Implements : GPIO_HAL_GetPinsDirection_Activity

3.43.2.3 GPIO_HAL_GetPinsOutput()

```
static uint32_t GPIO_HAL_GetPinsOutput (
            const GPIO_Type *const baseAddr )  [inline], [static]
```

Get the current output from a port.

This function returns the current output that is written to a port. Only pins that are configured as output will have meaningful values.

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)

Returns

GPIO outputs. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit:

- 0: corresponding pin is set to LOW
- 1: corresponding pin is set to HIGH Implements : GPIO_HAL_GetPinsOutput_Activity

3.43.2.4 GPIO_HAL_GetPortInputDisable()

```
static uint32_t GPIO_HAL_GetPortInputDisable (
            const GPIO_Type *const baseAddr )  [inline], [static]
```

Get the pins input disable state for a port.

This function returns the current pins input state for a port. Pins corresponding to bits with value of '1' are not configured as input and pins corresponding to bits with value of '0' are configured as input.

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)

Returns

GPIO input state. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit:

- 0: corresponding pin is set to input
- 1: corresponding pin is not set to input Implements : GPIO_HAL_GetPortInputDisable_Activity

3.43.2.5 GPIO_HAL_ReadPins()

```
static uint32_t GPIO_HAL_ReadPins (
            const GPIO_Type *const baseAddr )  [inline], [static]
```

Read input pins.

This function returns the current input values from a port. Only pins configured as input will have meaningful values.

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)

Returns

GPIO inputs. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit:

- 0: corresponding pin is read as LOW
- 1: corresponding pin is read as HIGH Implements : GPIO_HAL_ReadPins_Activity

3.43.2.6 GPIO_HAL_SetPinDirection()

```
static void GPIO_HAL_SetPinDirection (
            GPIO_Type *const baseAddr,
            const uint32_t pinNumber,
            const uint32_t pinDirection )  [inline], [static]
```

Configure the direction for a certain pin from a port.

This function configures the direction for the given pin, with the given value('1' for pin to be configured as output and '0' for pin to be configured as input)

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)
pinNumber	the pin number for which to configure the direction
pinDirection	the pin direction: • 0: corresponding pin is set to input • 1: corresponding pin is set to output Implements : GPIO_HAL_SetPinDirection_Activity

3.43.2.7 GPIO_HAL_SetPins()

```
static void GPIO_HAL_SetPins (
            GPIO_Type *const baseAddr,
            const uint32_t pins )  [inline], [static]
```

Write pins with 'Set' value.

This function configures output pins listed in parameter pins (bits that are '1') to have a value of 'set' (HIGH). Pins corresponding to '0' will be unaffected.

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)
pins	pin mask of bits to be set. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit: • 0: corresponding pin is unaffected • 1: corresponding pin is set to HIGH Implements : GPIO_HAL_SetPins_Activity

3.43.2.8 GPIO_HAL_SetPinsDirection()

```
static void GPIO_HAL_SetPinsDirection (
             GPIO_Type *const baseAddr,
             const uint32_t pins )  [inline], [static]
```

Set the pins directions configuration for a port.

This function sets the direction configuration for all pins in a port. Pins corresponding to bits with value of '1' will be configured as output and pins corresponding to bits with value of '0' will be configured as input.

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)
pins	pin mask where each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit: • 0: corresponding pin is set to input • 1: corresponding pin is set to output Implements : GPIO_HAL_SetPinsDirection_Activity

3.43.2.9 GPIO_HAL_SetPortInputDisable()

```
static void GPIO_HAL_SetPortInputDisable (
             GPIO_Type *const baseAddr,
             const uint32_t pins )  [inline], [static]
```

Set the pins input disable state for a port.

This function sets the pins input state for a port. Pins corresponding to bits with value of '1' will not be configured as input and pins corresponding to bits with value of '0' will be configured as input.

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)
pins	pin mask where each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit: • 0: corresponding pin is set to input • 1: corresponding pin is not set to input Implements : GPIO_HAL_SetPortInputDisable_Activity

3.43.2.10 GPIO_HAL_TogglePins()

```
static void GPIO_HAL_TogglePins (
            GPIO_Type *const baseAddr,
            const uint32_t pins )  [inline], [static]
```

Toggle pins value.

This function toggles output pins listed in parameter pins (bits that are '1'). Pins corresponding to '0' will be unaffected.

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)
pins	pin mask of bits to be toggled. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit: • 0: corresponding pin is unaffected • 1: corresponding pin is toggled Implements : GPIO_HAL_TogglePins_Activity

3.43.2.11 GPIO_HAL_WritePin()

```
static void GPIO_HAL_WritePin (
            GPIO_Type *const baseAddr,
            const uint32_t pinNumber,
            const uint32_t pinValue )  [inline], [static]
```

Write a pin of a port with a given value.

This function writes the given pin from a port, with the given value ('0' represents LOW, '1' represents HIGH).

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)
pinNumber	pin number to be written
pinValue	pin value to be written • 0: corresponding pin is set to LOW • 1: corresponding pin is set to HIGH Implements : GPIO_HAL_WritePin_Activity

3.43.2.12 GPIO_HAL_WritePins()

```
static void GPIO_HAL_WritePins (
            GPIO_Type *const baseAddr,
            const uint32_t pins )  [inline], [static]
```

Write all pins of a port.

This function writes all pins configured as output with the values given in the parameter pins. '0' represents LOW, '1' represents HIGH.

Parameters

baseAddr	GPIO base pointer (PTA, PTB, PTC, etc.)

Parameters

pins	pin mask to be written • 0: corresponding pin is set to LOW • 1: corresponding pin is set to HIGH Implements : GPIO_HAL_WritePins_Activity

3.44 General Purpose Input-Output (GPIO)

3.44.1 Detailed Description

The S32 SDK provides HAL for the General Purpose Input-Output (GPIO) module of S32 SDK devices.

The GPIO allows setting a pin either as an input or output and has dedicated registers for the following operations:

- configuring the pin direction (input-output)
- reading inputs and writing outputs
- setting/clearing/toggling pins without a read-modify-write operation

Modules

- GPIO HAL
  General Purpose Input/Output Hardware Abstraction Layer.

3.45 Interrupt Manager (Interrupt)

3.45.1 Detailed Description

The S32 SDK Interrupt Manager provides a set of API/services to configure the Interrupt Controller (NVIC).

The Nested-Vectored Interrupt Controller (NVIC) module implements a relocatable vector table supporting many external interrupts, a single non-maskable interrupt (NMI), and priority levels. The NVIC contains the address of the function to execute for a particular handler. The address is fetched via the instruction port allowing parallel register stacking and look-up. The first sixteen entries are allocated to internal sources with the others mapping to MCU-defined interrupts.

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 priority, and other features. Additionally, 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 (see IRQn_Type).

This is an example to enable/disable an interrupt for the ADC0_IRQn:

```
#include "interrupt_manager.h"
INT_SYS_EnableIRQ(ADC0_IRQn);
INT_SYS_DisableIRQ(ADC0_IRQn);
```

Typedefs

- typedef void(∗ isr_t) (void)

  Interrupt handler type.

Functions

- void DefaultISR (void)

  Default ISR.

Interrupt manager APIs

- void INT_SYS_InstallHandler (IRQn_Type irqNumber, const isr_t newHandler, isr_t ∗const oldHandler)

  Installs an interrupt handler routine for a given IRQ number.
- void INT_SYS_EnableIRQ (IRQn_Type irqNumber)

  Enables an interrupt for a given IRQ number.
- void INT_SYS_DisableIRQ (IRQn_Type irqNumber)

  Disables an interrupt for a given IRQ number.
- void INT_SYS_EnableIRQGlobal (void)

  Enables system interrupt.
- void INT_SYS_DisableIRQGlobal (void)

  Disable system interrupt.
- static void INT_SYS_SetPriority (IRQn_Type irqNumber, uint8_t priority)

  Set Interrupt Priority.
- static uint8_t INT_SYS_GetPriority (IRQn_Type irqNumber)

  Get Interrupt Priority.
- static void INT_SYS_ClearPending (IRQn_Type irqNumber)

  Clear Pending Interrupt.
- static void INT_SYS_SetPending (IRQn_Type irqNumber)

  Set Pending Interrupt.
- static uint32_t INT_SYS_GetPending (IRQn_Type irqNumber)

  Get Pending Interrupt.
- static uint32_t INT_SYS_GetActive (IRQn_Type irqNumber)

  Get Active Interrupt.

3.45.2 Typedef Documentation

3.45.2.1 isr_t

```
typedef void(∗ isr_t) (void)
```

Interrupt handler type.

3.45.3 Function Documentation

3.45.3.1 DefaultISR()

```
void DefaultISR (
             void )
```

Default ISR.

3.45.3.2 INT_SYS_ClearPending()

```
static void INT_SYS_ClearPending (
             IRQn_Type irqNumber )  [inline], [static]
```

Clear Pending Interrupt.

The function clears the pending bit of a peripheral interrupt or a directed interrupt to this CPU (if available). Implements INT_SYS_ClearPending_Activity

Parameters

irqNumber	IRQ number

3.45.3.3 INT_SYS_DisableIRQ()

```
void INT_SYS_DisableIRQ (
            IRQn_Type irqNumber )
```

Disables an interrupt for a given IRQ number.

This function disables the individual interrupt for a specified IRQ number. It calls the system NVIC API to access the interrupt control register and MSCM (if available) API for interrupt routing.

Parameters

irqNumber	IRQ number

3.45.3.4 INT_SYS_DisableIRQGlobal()

```
void INT_SYS_DisableIRQGlobal (
            void  )
```

Disable system interrupt.

This function disables the global interrupt by calling the core API.

3.45.3.5 INT_SYS_EnableIRQ()

```
void INT_SYS_EnableIRQ (
            IRQn_Type irqNumber )
```

Enables an interrupt for a given IRQ number.

This function enables the individual interrupt for a specified IRQ number. It calls the system NVIC API to access the interrupt control register and MSCM (if available) API for interrupt routing. The input IRQ number does not include the core interrupts, only the peripheral interrupts and directed CPU interrupts (if available), from 0 to a maximum supported IRQ.

Parameters

irqNumber	IRQ number

3.45.3.6 INT_SYS_EnableIRQGlobal()

```
void INT_SYS_EnableIRQGlobal (
            void  )
```

Enables system interrupt.

This function enables the global interrupt by calling the core API.

3.45.3.7 INT_SYS_GetActive()

```
static uint32_t INT_SYS_GetActive (
             IRQn_Type irqNumber )  [inline], [static]
```

Get Active Interrupt.

The function gets the active state of a peripheral interrupt. Implements INT_SYS_GetActive_Activity

Parameters

irqNumber	IRQ number

Returns

active Active status 0/1

3.45.3.8 INT_SYS_GetPending()

```
static uint32_t INT_SYS_GetPending (
             IRQn_Type irqNumber )  [inline], [static]
```

Get Pending Interrupt.

The function gets the pending bit of a peripheral interrupt or a directed interrupt to this CPU (if available). Implements INT_SYS_GetPending_Activity

Parameters

irqNumber	IRQ number

Returns

pending Pending status 0/1

3.45.3.9 INT_SYS_GetPriority()

```
static uint8_t INT_SYS_GetPriority (
             IRQn_Type irqNumber )  [inline], [static]
```

Get Interrupt Priority.

The function gets the priority of an interrupt.

Note: The priority cannot be obtained for every core interrupt. Implements INT_SYS_GetPriority_Activity

Parameters

irqNumber	Interrupt number.

Returns

priority Priority of the interrupt.

3.45.3.10 INT_SYS_InstallHandler()

```
void INT_SYS_InstallHandler (
             IRQn_Type irqNumber,
             const isr_t newHandler,
             isr_t *const oldHandler )
```

Installs an interrupt handler routine for a given IRQ number.

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_<SoC>.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.

Note

This method is applicable only if interrupt vector is copied in RAM, **flash_vector_table** symbol is used to control this from linker options.

Parameters

irqNumber	IRQ number
newHandler	New interrupt handler routine address pointer
oldHandler	Pointer to a location to store current interrupt handler

3.45.3.11 INT_SYS_SetPending()

```
static void INT_SYS_SetPending (
             IRQn_Type irqNumber ) [inline], [static]
```

Set Pending Interrupt.

The function configures the pending bit of a peripheral interrupt. Implements INT_SYS_SetPending_Activity

Parameters

irqNumber	IRQ number

3.45.3.12 INT_SYS_SetPriority()

```
static void INT_SYS_SetPriority (
             IRQn_Type irqNumber,
             uint8_t priority ) [inline], [static]
```

Set Interrupt Priority.

The function sets the priority of an interrupt.

Note: The priority cannot be set for every core interrupt. Implements INT_SYS_SetPriority_Activity

Parameters

irqNumber	Interrupt number.
priority	Priority to set.

3.46 LIN Driver

3.46.1 Detailed Description

This section describes the programming interface of the Peripheral driver for LIN.

3.46.2 LIN Driver Overview

The LIN (Local Interconnect Network) Driver is an use-case driven High Level Peripheral Driver. The driver is built on HAL drivers and provides users important key features. NXP provides LIN Stack as a middleware software package that is developed on LIN driver. Users also can create their own LIN applications and LIN stack that are compatible with LIN Specification.
In this release package, LIN Driver is built on LPUART interface.

3.46.3 LIN Driver Device structures

The driver uses instantiations of the lin_state_t to maintain the current state of a particular LIN Hardware instance module driver.
The user is required to provide memory for the driver state structures during the initialization. The driver itself does not statically allocate memory.

3.46.4 LIN Driver Initialization

1. To initialize the LIN driver, call the LIN_DRV_Init() function and pass the instance number of the relevant LIN hardware interface instance which is LPUART instance in this release.
   For example: to use LPUART0 pass value 0 to the initialization function.
2. Pass a user configuration structure lin_user_config_t as shown here:

```
/* LIN Driver configuration structure */
typedef struct {
    uint32_t baudRate;
    bool nodeFunction;
    bool autobaudEnable;
    lin_timer_get_time_interval_t timerGetTimeIntervalCallback;
} lin_user_config_t;
```

3. For LIN, typically the user configures the lin_user_config_t instantiation with a baudrate from 1000bps to 20000bps.
   -E.g. 19200 bps linUserConfig.baudRate = 19200U.
4. Node function can be MASTER or SLAVE.
   -E.g. linUserConfig.nodeFunction = MASTER
5. If users do not want to use Autobaud feature, then just configure linUserConfig.autobaudEnable = FALSE.
6. Users shall assign measurement callback function pointer that is *timerGetTimeIntervalCallback*. This function must return time period between two consecutive calls in nano seconds with accuracy at least 0.1 microsecond and if this function is called for the first time, it will start the timer to measure time. When an event (such as detecting a falling edge of a dominant signal while node is in sleep mode) occurs, LIN driver will call *timerGetTimeIntervalCallback* to start time measurement. Then on rising edge of that signal, LIN driver will call *timerGetTimeIntervalCallback* function to get time interval of that dominant signal in nano seconds. If Autobaud feature is enabled, LIN driver uses *timerGetTimeIntervalCallback* to measure two bit time length between two consecutive falling edges of the sync byte in order to evaluate Master's baudrate. Users can implement this function in their applications. -E.g. linUserConfig.timerGetTimeIntervalCallback = timerGet↩ TimeIntervalCallback0; This is a code example to set up a FTM0 for LIN Driver:

```
/* Global variables */
uint16_t timerCounterValue[2] = {0u};
uint16_t timerOverflowInterruptCount = 0u;

/* Callback function to get time interval in nano seconds */
uint32_t timerGetTimeIntervalCallback0(uint32_t *ns)
{
    timerCounterValue[1] = (uint16_t)(ftmBase->CNT);
    *ns = ((uint32_t)(timerCounterValue[1] + timerOverflowInterruptCount*65536u - timerCounterValue[0]))*10
      00 / TIMER_1US;
    timerOverflowInterruptCount = 0U;
    timerCounterValue[0] = timerCounterValue[1];
    return 0U;
}
```

7. This is a code example to set up a user LIN Driver configuration instantiation:

```
/* Device instance number as LPUART instance*/
#define LI0 (0U)

lin_state_t linState;
lin_user_config_t linUserConfig;
/* Set baudrate 19200 bps */
linUserConfig.baudRate = 19200U;
/* Node is MASTER */
linUserConfig.nodeFunction = MASTER;
/* Disable autobaud feature */
linUserConfig.autobaudEnable = FALSE;
/* Callback function to get time interval in nano seconds */
linUserConfig.timerGetTimeIntervalCallback = (lin_timer_get_time_t)
      timerGetTimeIntervalCallback0;

/* Initialize LIN Hardware interface */
LIN_DRV_Init(LI0, (lin_user_config_t *) &linUserConfig, (
      lin_state_t *) &linState);
```

8. The users are required to initialize a timer for LIN.
E.g. a Flex Timer (FTM). FTM instance should be initialized in Output Compare mode with an interrupt(E.g. FTM0_Ch0_Ch1_IRQHandler) period of about 500 us. Users can choose a different interrupt period that is appropriate to their applications. In timer interrupt handler, users shall call LIN_DRV_TimeoutService to handle linCurrentState->timeoutCounter while sending or receiving data.

3.46.5 LIN Data Transfers

The driver implements transmit and receive functions to transfer buffers of data by blocking and non-blocking modes.

The blocking transmit and receive functions include LIN_DRV_SendFrameDataBlocking() and the LIN_DRV_↵ ReceiveFrameDataBlocking() functions.

The non-blocking (async) transmit and receive functions include the LIN_DRV_SendFrameData() and the LIN_D↵ RV_ReceiveFrameData() functions.

Master nodes can transmit frame headers in non-blocking mode using LIN_DRV_MasterSendHeader().

In all these cases, the functions are interrupt-driven.

3.46.6 Autobaud feature

AUTOBAUD is an extensive feature in LIN Driver which allows a slave node to automatically detect baudrate of LIN bus and adapt its original baudrate to bus value. Auto Baud is applied when the baudrate of the incoming data is unknown. Currently autobaud feature is supported to detect LIN bus baudrates 2400, 4800, 9600, 14400, 19200 bps.

1. If autobaud feature is enabled, at LIN driver initialization slave's baudrate is set to 19200bps. The application should use a timer interrupt in input capture mode of both rising and falling edges(E.g FTM), call LIN_DR↩ V_AutoBaudCapture(uint32_t instance) function to calculate and set Slave's baudrate like Master's baudrate. When receiving a frame header, the slave detect LIN bus's baudrate based on the synchronization byte and adapts its baudrate accordingly. On changing baudrate, the slave set current event ID to LIN_BAUDRATE_↩ ADJUSTED and call the callback function. In that callback function users might change the frame data count timeout. Users can look at CallbackHandler() in lin.c of lin middleware for a reference.

   Note: Lin driver should be initiated before initiating a timer interrupt(E.g FTM).
2. Baudrate evaluation process is executed until autobaud successfully. During run-time if LIN bus's baudrate is changed suddenly to a value other than the slave's current baudrate, users shall reset MCU to execute baudrate evaluation process.

Data Structures

- struct lin_user_config_t

  LIN hardware configuration structure Implements : lin_user_config_t_Class. More...
- struct lin_state_t

  Runtime state of the LIN driver. More...

Macros

- #define SLAVE 0U
- #define MASTER 1U
- #define MAKE_PARITY 0U
- #define CHECK_PARITY 1U

Typedefs

- typedef uint32_t(∗ lin_timer_get_time_interval_t) (uint32_t ∗nanoSeconds)

  Callback function to get time interval in nanoseconds Implements : lin_timer_get_time_interval_t_Class.
- typedef void(∗ lin_callback_t) (uint32_t instance, void ∗linState)

  LIN Driver callback function type Implements : lin_callback_t_Class.

Enumerations

- enum lin_event_id_t {
  LIN_NO_EVENT = 0x00U, LIN_WAKEUP_SIGNAL = 0x01U, LIN_BAUDRATE_ADJUSTED = 0x02U, LIN↩ _RECV_BREAK_FIELD_OK = 0x03U,
  LIN_SYNC_OK = 0x04U, LIN_SYNC_ERROR = 0x05U, LIN_PID_OK = 0x06U, LIN_PID_ERROR = 0x07U,
  LIN_FRAME_ERROR = 0x08U, LIN_READBACK_ERROR = 0x09U, LIN_CHECKSUM_ERROR = 0x0AU,
  LIN_TX_COMPLETED = 0x0BU,
  LIN_RX_COMPLETED = 0x0CU }

  Defines types for an enumerating event related to an Identifier. Implements : lin_event_id_t_Class.
- enum lin_node_state_t {
  LIN_NODE_STATE_UNINIT = 0x00U, LIN_NODE_STATE_SLEEP_MODE = 0x01U, LIN_NODE_STATE↩ _IDLE = 0x02U, LIN_NODE_STATE_SEND_BREAK_FIELD = 0x03U,
  LIN_NODE_STATE_RECV_SYNC = 0x04U, LIN_NODE_STATE_SEND_PID = 0x05U, LIN_NODE_STA↩ TE_RECV_PID = 0x06U, LIN_NODE_STATE_RECV_DATA = 0x07U,
  LIN_NODE_STATE_RECV_DATA_COMPLETED = 0x08U, LIN_NODE_STATE_SEND_DATA = 0x09U, L↩ IN_NODE_STATE_SEND_DATA_COMPLETED = 0x0AU }

  Define type for an enumerating LIN Node state. Implements : lin_node_state_t_Class.

LIN DRIVER

- status_t LIN_DRV_Init (uint32_t instance, lin_user_config_t ∗linUserConfig, lin_state_t ∗linCurrentState)

  Initializes an instance LIN Hardware Interface for LIN Network.
- status_t LIN_DRV_Deinit (uint32_t instance)

  Shuts down the LIN Hardware Interface by disabling interrupts and transmitter/receiver.
- lin_callback_t LIN_DRV_InstallCallback (uint32_t instance, lin_callback_t function)

  Installs callback function that is used for LIN_DRV_IRQHandler.
- status_t LIN_DRV_SendFrameDataBlocking (uint32_t instance, const uint8_t ∗txBuff, uint8_t txSize, uint32←_t timeoutMSec)

  Sends Frame data out through the LIN Hardware Interface using blocking method. This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete. This function checks if txSize is in range from 1 to 8. If not, it will return STATUS_ERROR. This function also returns STATUS_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN_BUS_BUSY. The function does not return until the transmission is complete. If the transmission is successful, it will return STATUS_SUCCESS. If not, it will return STATUS_TIMEOUT.
- status_t LIN_DRV_SendFrameData (uint32_t instance, const uint8_t ∗txBuff, uint8_t txSize)

  Sends frame data out through the LIN Hardware Interface using non-blocking method. This enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data. The function will return immediately after calling this function. If txSize is equal to 0 or greater than 8 or node's current state is in SLEEP mode then the function will return STATUS_ERROR. If isBusBusy is currently true then the function will return LIN_BUS_BUSY.
- status_t LIN_DRV_GetTransmitStatus (uint32_t instance, uint8_t ∗bytesRemaining)

  Get status of an on-going non-blocking transmission While sending frame data using non-blocking method, users can use this function to get status of that transmission. The bytesRemaining shows number of bytes that still needed to transmit.
- status_t LIN_DRV_ReceiveFrameDataBlocking (uint32_t instance, uint8_t ∗rxBuff, uint8_t rxSize, uint32_t timeoutMSec)

  Receives frame data through the LIN Hardware Interface using blocking method. This function receives data from LPUART module using blocking method, the function does not return until the receive is complete. The interrupt handler LIN_LPUART_DRV_IRQHandler will check the checksum byte. If the checksum is correct, it will receive the frame data. If the checksum is incorrect, this function will return STATUS_TIMEOUT and data in rxBuff might be wrong. This function also check if rxSize is in range from 1 to 8. If not, it will return STATUS_ERROR. This function also returns STATUS_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN_BUS_BUSY.
- status_t LIN_DRV_ReceiveFrameData (uint32_t instance, uint8_t ∗rxBuff, uint8_t rxSize)

  Receives frame data through the LIN Hardware Interface using non-blocking method. This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete. The interrupt handler LIN_LPUART_DRV_IRQHandler will check the checksum byte. If the checksum is correct, it will receive the frame data. If the checksum is incorrect, this function will return STATUS_TIMEOUT and data in rxBuff might be wrong. This function also check if rxSize is in range from 1 to 8. If not, it will return STATUS_ERROR. This function also returns STATUS_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN_BUS_BUSY.
- status_t LIN_DRV_AbortTransferData (uint32_t instance)

  Aborts an on-going non-blocking transmission/reception. While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.
- status_t LIN_DRV_GetReceiveStatus (uint32_t instance, uint8_t ∗bytesRemaining)

  Get status of an on-going non-blocking reception. This function returns whether the data reception is complete. When performing non-blocking transmit, the user can call this function to ascertain the state of the current receive progress: in progress (STATUS_BUSY) or timeout (STATUS_TIMEOUT) or complete (STATUS_SUCCESS). In addition, if the reception is still in progress, the user can obtain the number of bytes that still needed to receive.
- status_t LIN_DRV_GoToSleepMode (uint32_t instance)

  Puts current LIN node to sleep mode This function changes current node state to LIN_NODE_STATE_SLEEP_M←ODE.
- status_t LIN_DRV_GotoIdleState (uint32_t instance)

Puts current LIN node to Idle state This function changes current node state to LIN_NODE_STATE_IDLE.
- status_t LIN_DRV_SendWakeupSignal (uint32_t instance)

  Sends a wakeup signal through the LIN Hardware Interface.
- lin_node_state_t LIN_DRV_GetCurrentNodeState (uint32_t instance)

  Get the current LIN node state.
- void LIN_DRV_TimeoutService (uint32_t instance)

  Callback function for Timer Interrupt Handler Users may use (optional, not required) LIN_DRV_TimeoutService to check if timeout has occurred during non-blocking frame data transmission and reception. User may initialize a timer (for example FTM) in Output Compare Mode with period of 500 micro seconds (recommended). In timer IRQ handler, call this function.
- void LIN_DRV_SetTimeoutCounter (uint32_t instance, uint32_t timeoutValue)

  Set Value for Timeout Counter that is used in LIN_DRV_TimeoutService.
- status_t LIN_DRV_MasterSendHeader (uint32_t instance, uint8_t id)

  Sends frame header out through the LIN Hardware Interface using a non-blocking method. This function sends LIN Break field, sync field then the ID with correct parity. This function checks if the interface is Master, if not, it will return STATUS_ERROR.This function checks if id is in range from 0 to 0x3F, if not it will return STATUS_ERROR.
- status_t LIN_DRV_EnableIRQ (uint32_t instance)

  Enables LIN hardware interrupts.
- status_t LIN_DRV_DisableIRQ (uint32_t instance)

  Disables LIN hardware interrupts.
- void LIN_DRV_IRQHandler (uint32_t instance)

  Interrupt handler for LIN Hardware Interface.
- uint8_t LIN_DRV_ProcessParity (uint8_t PID, uint8_t typeAction)

  Makes or checks parity bits. If action is checking parity, the function returns ID value if parity bits are correct or 0xFF if parity bits are incorrect. If action is making parity bits, then from input value of ID, the function returns PID. This is not a public API as it is called by other API functions.
- uint8_t LIN_DRV_MakeChecksumByte (const uint8_t ∗buffer, uint8_t sizeBuffer, uint8_t PID)

  Makes the checksum byte for a frame.
- status_t LIN_DRV_AutoBaudCapture (uint32_t instance)

  Captures time interval to capture baudrate automatically when enable autobaud feature. This function should only be used in Slave. The timer should be in input capture mode of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

3.46.7 Data Structure Documentation

3.46.7.1 struct lin_user_config_t

LIN hardware configuration structure Implements : lin_user_config_t_Class.

Data Fields

- uint32_t baudRate
- bool nodeFunction
- bool autobaudEnable
- lin_timer_get_time_interval_t timerGetTimeIntervalCallback

Field Documentation

3.46.7.1.1 autobaudEnable

```
bool autobaudEnable
```

Enable Autobaud feature

3.46.7.1.2 baudRate

```
uint32_t baudRate
```

baudrate of LIN Hardware Interface to configure

3.46.7.1.3 nodeFunction

```
bool nodeFunction
```

Node function as Master or Slave

3.46.7.1.4 timerGetTimeIntervalCallback

```
lin_timer_get_time_interval_t timerGetTimeIntervalCallback
```

Callback function to get time interval in nanoseconds

3.46.7.2 struct lin_state_t

Runtime state of the LIN driver.

Note that the caller provides memory for the driver state structures during initialization because the driver does not statically allocate memory. Implements : lin_state_t_Class

Data Fields

- const uint8_t ∗ txBuff
- uint8_t ∗ rxBuff
- uint8_t cntByte
- volatile uint8_t txSize
- volatile uint8_t rxSize
- uint8_t checkSum
- volatile bool isTxBusy
- volatile bool isRxBusy
- volatile bool isBusBusy
- volatile bool isTxBlocking
- volatile bool isRxBlocking
- lin_callback_t Callback
- uint8_t currentId
- uint8_t currentPid
- volatile lin_event_id_t currentEventId
- volatile lin_node_state_t currentNodeState
- volatile uint32_t timeoutCounter
- volatile bool timeoutCounterFlag
- volatile bool baudrateEvalEnable
- volatile uint8_t fallingEdgeInterruptCount
- uint32_t linSourceClockFreq
- semaphore_t txCompleted
- semaphore_t rxCompleted

Field Documentation

3.46.7.2.1 baudrateEvalEnable

```
volatile bool baudrateEvalEnable
```

Baudrate Evaluation Process Enable

3.46.7.2.2 Callback

```
lin_callback_t Callback
```

Callback function to invoke after receiving a byte or transmitting a byte.

3.46.7.2.3 checkSum

```
uint8_t checkSum
```

Checksum byte.

3.46.7.2.4 cntByte

```
uint8_t cntByte
```

To count number of bytes already transmitted or received.

3.46.7.2.5 currentEventId

```
volatile lin_event_id_t currentEventId
```

Current ID Event

3.46.7.2.6 currentId

```
uint8_t currentId
```

Current ID

3.46.7.2.7 currentNodeState

```
volatile lin_node_state_t currentNodeState
```

Current Node state

3.46.7.2.8 currentPid

```
uint8_t currentPid
```

Current PID

3.46.7.2.9 fallingEdgeInterruptCount

```
volatile uint8_t fallingEdgeInterruptCount
```

Falling Edge count of a sync byte

3.46.7.2.10 isBusBusy

```
volatile bool isBusBusy
```

True if there are data, frame headers being transferred on bus

3.46.7.2.11 isRxBlocking

```
volatile bool isRxBlocking
```

True if receive is blocking transaction.

3.46.7.2.12 isRxBusy

```
volatile bool isRxBusy
```

True if the LIN interface is receiving frame data.

3.46.7.2.13 isTxBlocking

```
volatile bool isTxBlocking
```

True if transmit is blocking transaction.

3.46.7.2.14 isTxBusy

```
volatile bool isTxBusy
```

True if the LIN interface is transmitting frame data.

3.46.7.2.15 linSourceClockFreq

```
uint32_t linSourceClockFreq
```

Frequency of the source clock for LIN

3.46.7.2.16 rxBuff

```
uint8_t* rxBuff
```

The buffer of received data.

3.46.7.2.17 rxCompleted

```
semaphore_t rxCompleted
```

Used to wait for LIN interface ISR to complete reception

3.46.7.2.18 rxSize

```
volatile uint8_t rxSize
```

The remaining number of bytes to be received.

3.46.7.2.19 timeoutCounter

```
volatile uint32_t timeoutCounter
```

Value of the timeout counter

3.46.7.2.20 timeoutCounterFlag

```
volatile bool timeoutCounterFlag
```

Timeout counter flag

3.46.7.2.21 txBuff

```
const uint8_t* txBuff
```

The buffer of data being sent.

3.46.7.2.22 txCompleted

```
semaphore_t txCompleted
```

Used to wait for LIN interface ISR to complete transmission.

3.46.7.2.23 txSize

```
volatile uint8_t txSize
```

The remaining number of bytes to be transmitted.

3.46.8 Macro Definition Documentation

3.46.8.1 CHECK_PARITY

```
#define CHECK_PARITY 1U
```

3.46.8.2 MAKE_PARITY

```
#define MAKE_PARITY 0U
```

3.46.8.3 MASTER

```
#define MASTER 1U
```

3.46.8.4 SLAVE

```
#define SLAVE 0U
```

3.46.9 Typedef Documentation

3.46.9.1 lin_callback_t

```
typedef void(* lin_callback_t) (uint32_t instance, void *linState)
```

LIN Driver callback function type Implements : lin_callback_t_Class.

3.46.9.2 lin_timer_get_time_interval_t

```
typedef uint32_t(* lin_timer_get_time_interval_t) (uint32_t *nanoSeconds)
```

Callback function to get time interval in nanoseconds Implements : lin_timer_get_time_interval_t_Class.

3.46.10 Enumeration Type Documentation

3.46.10.1 lin_event_id_t

```
enum lin_event_id_t
```

Defines types for an enumerating event related to an Identifier. Implements : lin_event_id_t_Class.

Enumerator

LIN_NO_EVENT	No event yet
LIN_WAKEUP_SIGNAL	Received a wakeup signal
LIN_BAUDRATE_ADJUSTED	Indicate that baudrate was adjusted to Master's baudrate
LIN_RECV_BREAK_FIELD_OK	Indicate that correct Break Field was received
LIN_SYNC_OK	Sync byte is correct
LIN_SYNC_ERROR	Sync byte is incorrect
LIN_PID_OK	PID correct
LIN_PID_ERROR	PID incorrect
LIN_FRAME_ERROR	Framing Error
LIN_READBACK_ERROR	Readback data is incorrect
LIN_CHECKSUM_ERROR	Checksum byte is incorrect
LIN_TX_COMPLETED	Sending data completed
LIN_RX_COMPLETED	Receiving data completed

3.46.10.2 lin_node_state_t

```
enum lin_node_state_t
```

Define type for an enumerating LIN Node state. Implements : lin_node_state_t_Class.

Enumerator

LIN_NODE_STATE_UNINIT	Uninitialized state
LIN_NODE_STATE_SLEEP_MODE	Sleep mode state
LIN_NODE_STATE_IDLE	Idle state
LIN_NODE_STATE_SEND_BREAK_FIELD	Send break field state
LIN_NODE_STATE_RECV_SYNC	Receive the synchronization byte state
LIN_NODE_STATE_SEND_PID	Send PID state
LIN_NODE_STATE_RECV_PID	Receive PID state
LIN_NODE_STATE_RECV_DATA	Receive data state
LIN_NODE_STATE_RECV_DATA_COMPLETED	Receive data completed state
LIN_NODE_STATE_SEND_DATA	Send data state
LIN_NODE_STATE_SEND_DATA_COMPLETED	Send data completed state

3.46.11 Function Documentation

3.46.11.1 LIN_DRV_AbortTransferData()

```
status_t LIN_DRV_AbortTransferData (
            uint32_t instance )
```

Aborts an on-going non-blocking transmission/reception. While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.

Parameters

instance	LIN Hardware Interface instance number

Returns

An error code or status_t

3.46.11.2 LIN_DRV_AutoBaudCapture()

```
status_t LIN_DRV_AutoBaudCapture (
            uint32_t instance )
```

Captures time interval to capture baudrate automatically when enable autobaud feature. This function should only be used in Slave. The timer should be in input capture mode of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

Parameters

instance	LIN Hardware Interface instance number

Returns

operation status

- STATUS_SUCCESS: Operation was successful.

- STATUS_BUSY: Operation is running.
- STATUS_ERROR: Operation failed due to break char incorrect, wakeup signal incorrect or calculate baudrate failed.

3.46.11.3 LIN_DRV_Deinit()

```
status_t LIN_DRV_Deinit (
            uint32_t instance )
```

Shuts down the LIN Hardware Interface by disabling interrupts and transmitter/receiver.

Parameters

instance	LIN Hardware Interface instance number

Returns

An error code or status_t

3.46.11.4 LIN_DRV_DisableIRQ()

```
status_t LIN_DRV_DisableIRQ (
            uint32_t instance )
```

Disables LIN hardware interrupts.

Parameters

instance	LIN Hardware Interface instance number

Returns

An error code or status_t

3.46.11.5 LIN_DRV_EnableIRQ()

```
status_t LIN_DRV_EnableIRQ (
            uint32_t instance )
```

Enables LIN hardware interrupts.

Parameters

instance	LIN Hardware Interface instance number

Returns

An error code or status_t

3.46.11.6 LIN_DRV_GetCurrentNodeState()

```
lin_node_state_t LIN_DRV_GetCurrentNodeState (
             uint32_t instance )
```

Get the current LIN node state.

Parameters

instance	LIN Hardware Interface instance number

Returns

current LIN node state

3.46.11.7 LIN_DRV_GetReceiveStatus()

```
status_t LIN_DRV_GetReceiveStatus (
             uint32_t instance,
             uint8_t * bytesRemaining )
```

Get status of an on-going non-blocking reception. This function returns whether the data reception is complete. When performing non-blocking transmit, the user can call this function to ascertain the state of the current receive progress: in progress (STATUS_BUSY) or timeout (STATUS_TIMEOUT) or complete (STATUS_SUCCESS). In addition, if the reception is still in progress, the user can obtain the number of bytes that still needed to receive.

Parameters

instance	LIN Hardware Interface instance number
bytesRemaining	Number of bytes still needed to receive

Returns

status_t STATUS_BUSY, STATUS_TIMEOUT or STATUS_SUCCESS

3.46.11.8 LIN_DRV_GetTransmitStatus()

```
status_t LIN_DRV_GetTransmitStatus (
             uint32_t instance,
             uint8_t * bytesRemaining )
```

Get status of an on-going non-blocking transmission While sending frame data using non-blocking method, users can use this function to get status of that transmission. The bytesRemaining shows number of bytes that still needed to transmit.

Parameters

instance	LIN Hardware Interface instance number
bytesRemaining	Number of bytes still needed to transmit

Returns

status_t STATUS_BUSY if the transmission is still in progress. STATUS_TIMEOUT if timeout occurred and transmission was not completed. STATUS_SUCCESS if the transmission was successful.

3.46.11.9 LIN_DRV_GotoIdleState()

```
status_t LIN_DRV_GotoIdleState (
             uint32_t instance )
```

Puts current LIN node to Idle state This function changes current node state to LIN_NODE_STATE_IDLE.

Parameters

instance	LIN Hardware Interface instance number

Returns

An error code or status_t

3.46.11.10 LIN_DRV_GoToSleepMode()

```
status_t LIN_DRV_GoToSleepMode (
             uint32_t instance )
```

Puts current LIN node to sleep mode This function changes current node state to LIN_NODE_STATE_SLEEP_↩ MODE.

Parameters

instance	LIN Hardware Interface instance number

Returns

An error code or status_t

3.46.11.11 LIN_DRV_Init()

```
status_t LIN_DRV_Init (
             uint32_t instance,
             lin_user_config_t * linUserConfig,
             lin_state_t * linCurrentState )
```

Initializes an instance LIN Hardware Interface for LIN Network.

The caller provides memory for the driver state structures during initialization. The user must select the LIN Hardware Interface clock source in the application to initialize the LIN Hardware Interface.

Parameters

instance	LIN Hardware Interface instance number
linUserConfig	user configuration of type lin_user_config_t
linCurrentState	pointer to the LIN Hardware Interface driver state structure

Returns

An error code or status_t

3.46.11.12 LIN_DRV_InstallCallback()

```
lin_callback_t LIN_DRV_InstallCallback (
            uint32_t instance,
            lin_callback_t function )
```

Installs callback function that is used for LIN_DRV_IRQHandler.

Note

After a callback is installed, it bypasses part of the LIN Hardware Interface IRQHandler logic. Therefore, the callback needs to handle the indexes of txBuff and txSize.

Parameters

instance	LIN Hardware Interface instance number.
function	the LIN receive callback function.

Returns

Former LIN callback function pointer.

3.46.11.13 LIN_DRV_IRQHandler()

```
void LIN_DRV_IRQHandler (
            uint32_t instance )
```

Interrupt handler for LIN Hardware Interface.

Parameters

instance	LIN Hardware Interface instance number

Returns

void

3.46.11.14 LIN_DRV_MakeChecksumByte()

```
uint8_t LIN_DRV_MakeChecksumByte (
            const uint8_t * buffer,
            uint8_t sizeBuffer,
            uint8_t PID )
```

Makes the checksum byte for a frame.

Parameters

buffer	Pointer to Tx buffer
sizeBuffer	Number of bytes that are contained in the buffer.
PID	Protected Identifier byte.

Returns

the checksum byte.

3.46.11.15 LIN_DRV_MasterSendHeader()

```
status_t LIN_DRV_MasterSendHeader (
             uint32_t instance,
             uint8_t id )
```

Sends frame header out through the LIN Hardware Interface using a non-blocking method. This function sends LIN Break field, sync field then the ID with correct parity. This function checks if the interface is Master, if not, it will return STATUS_ERROR.This function checks if id is in range from 0 to 0x3F, if not it will return STATUS_ERROR.

Parameters

instance	LIN Hardware Interface instance number
id	Frame Identifier

Returns

An error code or status_t

3.46.11.16 LIN_DRV_ProcessParity()

```
uint8_t LIN_DRV_ProcessParity (
             uint8_t PID,
             uint8_t typeAction )
```

Makes or checks parity bits. If action is checking parity, the function returns ID value if parity bits are correct or 0xFF if parity bits are incorrect. If action is making parity bits, then from input value of ID, the function returns PID. This is not a public API as it is called by other API functions.

Parameters

PID	PID byte in case of checking parity bits or ID byte in case of making parity bits.
typeAction	1 for Checking parity bits, 0 for making parity bits

Returns

0xFF if parity bits are incorrect, ID in case of checking parity bits and they are correct. Function returns PID in case of making parity bits.

3.46.11.17 LIN_DRV_ReceiveFrameData()

```
status_t LIN_DRV_ReceiveFrameData (
            uint32_t instance,
            uint8_t * rxBuff,
            uint8_t rxSize )
```

Receives frame data through the LIN Hardware Interface using non-blocking method. This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete. The interrupt handler LIN_LPUART_DRV_IRQHandler will check the checksum byte. If the checksum is correct, it will receive the frame data. If the checksum is incorrect, this function will return STATUS_TIMEOUT and data in rxBuff might be wrong. This function also check if rxSize is in range from 1 to 8. If not, it will return STATUS_ERROR. This function also returns STATUS_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN_BUS_BUSY.

Note

If users use LIN_DRV_TimeoutService in a timer interrupt handler, then before using this function, users have to set timeout counter to an appropriate value by using LIN_DRV_SetTimeoutCounter(instance, timeout↩ Value). The timeout value should be big enough to complete the reception. Timeout in real time is (timeout↩ Value) * (time period that LIN_DRV_TimeoutService is called). For example, if LIN_DRV_TimeoutService is called in an timer interrupt with period of 500 micro seconds, then timeout in real time is timeoutValue * 500 micro seconds.

Parameters

instance	LIN Hardware Interface instance number
rxBuff	buffer containing 8-bit received data
rxSize	the number of bytes to receive

Returns

An error code or status_t

3.46.11.18 LIN_DRV_ReceiveFrameDataBlocking()

```
status_t LIN_DRV_ReceiveFrameDataBlocking (
            uint32_t instance,
            uint8_t * rxBuff,
            uint8_t rxSize,
            uint32_t timeoutMSec )
```

Receives frame data through the LIN Hardware Interface using blocking method. This function receives data from LPUART module using blocking method, the function does not return until the receive is complete. The interrupt handler LIN_LPUART_DRV_IRQHandler will check the checksum byte. If the checksum is correct, it will receive the frame data. If the checksum is incorrect, this function will return STATUS_TIMEOUT and data in rxBuff might be wrong. This function also check if rxSize is in range from 1 to 8. If not, it will return STATUS_ERROR. This function also returns STATUS_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN_BUS_BUSY.

Parameters

instance	LIN Hardware Interface instance number
rxBuff	buffer containing 8-bit received data
rxSize	the number of bytes to receive
timeoutMSec	timeout value in milliseconds

Returns

An error code or status_t

3.46.11.19 LIN_DRV_SendFrameData()

```
status_t LIN_DRV_SendFrameData (
            uint32_t instance,
            const uint8_t * txBuff,
            uint8_t txSize )
```

Sends frame data out through the LIN Hardware Interface using non-blocking method. This enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data. The function will return immediately after calling this function. If txSize is equal to 0 or greater than 8 or node's current state is in SLEEP mode then the function will return STATUS_ERROR. If isBusBusy is currently true then the function will return LIN_BUS_BUSY.

Note

If users use LIN_DRV_TimeoutService in a timer interrupt handler, then before using this function, users have to set timeout counter to an appropriate value by using LIN_DRV_SetTimeoutCounter(instance, timeout↩ Value). The timeout value should be big enough to complete the transmission. Timeout in real time is (timeoutValue) $*$ (time period that LIN_DRV_TimeoutService is called). For example, if LIN_DRV_Timeout↩ Service is called in an timer interrupt with period of 500 micro seconds, then timeout in real time is timeout↩ Value $*$ 500 micro seconds.

Parameters

instance	LIN Hardware Interface instance number
txBuff	source buffer containing 8-bit data chars to send
txSize	the number of bytes to send

Returns

An error code or status_t STATUS_BUSY if the bus is currently busy, transmission cannot be started. STA↩ TUS_SUCCESS if the transmission was completed.

3.46.11.20 LIN_DRV_SendFrameDataBlocking()

```
status_t LIN_DRV_SendFrameDataBlocking (
            uint32_t instance,
            const uint8_t * txBuff,
            uint8_t txSize,
            uint32_t timeoutMSec )
```

Sends Frame data out through the LIN Hardware Interface using blocking method. This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete. This function checks if txSize is in range from 1 to 8. If not, it will return STATUS_ER↩ ROR. This function also returns STATUS_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN_BUS_BUSY. The function does not return until the transmission is complete. If the transmission is successful, it will return STATUS_SUCCESS. If not, it will return STATUS_TIME↩ OUT.

Parameters

instance	LIN Hardware Interface instance number
txBuff	source buffer containing 8-bit data chars to send
txSize	the number of bytes to send
timeoutMSec	timeout value in milliseconds

Returns

An error code or status_t STATUS_BUSY if the bus is currently busy, transmission cannot be started. ST↩ ATUS_TIMEOUT if timeout occurred and transmission was not completed. STATUS_SUCCESS if the transmission was completed.

3.46.11.21 LIN_DRV_SendWakeupSignal()

```
status_t LIN_DRV_SendWakeupSignal (
            uint32_t instance )
```

Sends a wakeup signal through the LIN Hardware Interface.

Parameters

instance	LIN Hardware Interface instance number

Returns

An error code or status_t

3.46.11.22 LIN_DRV_SetTimeoutCounter()

```
void LIN_DRV_SetTimeoutCounter (
            uint32_t instance,
            uint32_t timeoutValue )
```

Set Value for Timeout Counter that is used in LIN_DRV_TimeoutService.

Parameters

instance	LIN Hardware Interface instance number
timeoutValue	Timeout Value to be set

Returns

void

3.46.11.23 LIN_DRV_TimeoutService()

```
void LIN_DRV_TimeoutService (
            uint32_t instance )
```

Callback function for Timer Interrupt Handler Users may use (optional, not required) LIN_DRV_TimeoutService to check if timeout has occurred during non-blocking frame data transmission and reception. User may initialize a timer (for example FTM) in Output Compare Mode with period of 500 micro seconds (recommended). In timer IRQ handler, call this function.

Parameters

instance	LIN Hardware Interface instance number

Returns

void

3.47 LPI2C Driver

3.47.1 Detailed Description

Low Power Inter-Integrated Circuit (LPI2C) Peripheral Driver.

Low Power Inter-Integrated Circuit Driver.

The LPI2C driver allows communication on an I2C bus using the LPI2C module in the S32144K processor.

Features

- Interrupt based
- Master or slave operation
- Provides blocking and non-blocking transmit and receive functions
- 7-bit or 10-bit addressing
- Configurable baud rate
- Provides support for all operating modes supported by the hardware
  - Standard-mode (Sm): bidirectional data transfers up to 100 kbit/s
  - Fast-mode (Fm): bidirectional data transfers up to 400 kbit/s

Functionality

In order to use the LPI2C driver it must be first initialized in either master of slave mode, using functions LPI2↩ C_DRV_MasterInit() or LPI2C_DRV_SlaveInit(). Once initialized, it cannot be initialized again for the same LPI2C module instance until it is de-initialized, using LPI2C_DRV_MasterDeinit() or LPI2C_DRV_SlaveDeinit(). Different LPI2C module instances can function independently of each other.

Master Mode

Master Mode provides functions for transmitting or receiving data to/from any I2C slave. Slave address and baud rate are provided at initialization time through the master configuration structure, but they can be changed at run-time by using LPI2C_DRV_MasterSetBaudRate() or LPI2C_DRV_MasterSetSlaveAddr(). Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency protocol clock for the LPI2C module. The application should call LPI2C_DRV_MasterGetBaudRate() after LPI2C_DRV_MasterSetBaudRate() to check what baud rate was actually set.

To send or receive data to/from the currently configured slave address, use functions LPI2C_DRV_MasterSend↩ Data() or LPI2C_DRV_MasterReceiveData() (or their blocking counterparts). Parameter `sendStop` can be used to chain multiple transfers with repeated START condition between them, for example when sending a command and then immediately receiving a response. The application should ensure that any send or receive transfer with `sendStop` set to `false` is followed by another transfer, otherwise the LPI2C master will hold the SCL line low indefinitely and block the I2C bus. The last transfer from a chain should always have `sendStop` set to `true`.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return STATUS_SUCCESS, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application can check the status of the current transfer by calling LPI2C_DRV_MasterGetTransferStatus(). If the transfer is completed, the functions will return either STATUS_SUCCESS or an error code, depending on the outcome of the last transfer.

The driver supports any operating mode supported by the module. The operating mode is set together with the baud rate, by LPI2C_DRV_MasterSetBaudRate(). For High-Speed mode a second baud rate is required, for high-speed communication. Note that due to module limitation (common prescaler setting for normal and fast baud rate) there is a limit on the maximum difference between the two baud rates. LPI2C_DRV_MasterGetBaudRate() can be used to check the baud rate setting for both modes.

Slave Mode

Slave Mode provides functions for transmitting or receiving data to/from any I2C master. There are two slave operating modes, selected by the field `slaveListening` in the slave configuration structure:

- Slave always listening: the slave interrupt is enabled at initialization time and the slave always listens to the line for a master addressing it. Any events are reported to the application through the callback function provided at initialization time. The callback can use LPI2C_DRV_SlaveSetRxBuffer() or LPI2C_DRV_Slave↩ SetTxBuffer() to provide the appropriate buffers for transmit or receive, as needed.
- On-demand operation: the slave is commanded to transmit or receive data through the call of LPI2C_↩ DRV_SlaveSendData() and LPI2C_DRV_SlaveReceiveData() (or their blocking counterparts). The actual moment of the transfer depends on the I2C master. The use of callbacks optional in this case, for example to treat events like LPI2C_SLAVE_EVENT_TX_EMPTY or LPI2C_SLAVE_EVENT_RX_FULL. Outside the commanded receive / transmit operations the LPI2C interrupts are disabled and the module will not react to master transfer requests.

Important Notes

- Before using the LPI2C driver in master mode the protocol clock of the module must be configured. Refer to SCG HAL and PCC HAL for clock configuration.
- Before using the LPI2C driver the pins must be routed to the LPI2C module. Refer to PORT HAL for pin routing configuration.
- The driver enables the interrupts for the corresponding LPI2C module, but any interrupt priority setting must be done by the application.
- Fast+, high-speed and ultra-fast mode aren't supported.
- Aborting a master reception is not currently supported due to hardware behavior (the module will continue a started reception even if the FIFO is reset).
- In listening mode, the init function must be called before the master starts the transfer. In non-listening mode, the init function and the appropriate send/receive function must be called before the master starts the transfer.

Data Structures

- struct lpi2c_master_user_config_t

  Master configuration structure. More...
- struct lpi2c_slave_user_config_t

  Slave configuration structure. More...
- struct lpi2c_baud_rate_params_t

  Baud rate structure. More...
- struct lpi2c_master_state_t

  Master internal context structure. More...
- struct lpi2c_slave_state_t

  Slave internal context structure. More...

Typedefs

- typedef void(∗ lpi2c_master_callback_t) (uint8_t instance, lpi2c_master_event_t masterEvent, void ∗user↩ Data)

  Defines the example structure.
- typedef void(∗ lpi2c_slave_callback_t) (uint8_t instance, lpi2c_slave_event_t slaveEvent, void ∗userData)

  LPI2C slave callback function.

Enumerations

- enum lpi2c_mode_t { LPI2C_STANDARD_MODE = 0x0U, LPI2C_FAST_MODE = 0x1U }

  I2C operating modes Implements : lpi2c_mode_t_Class.
- enum lpi2c_master_event_t {
  LPI2C_MASTER_EVENT_TX = 0x0U, LPI2C_MASTER_EVENT_RX = 0x1U, LPI2C_MASTER_EVENT_←
  FIFO_ERROR = 0x2U, LPI2C_MASTER_EVENT_ARBITRATION_LOST = 0x3U,
  LPI2C_MASTER_EVENT_NACK = 0x4U }

  LPI2C master events Implements : lpi2c_master_event_t_Class.
- enum lpi2c_slave_event_t {
  LPI2C_SLAVE_EVENT_TX_REQ = 0x02U, LPI2C_SLAVE_EVENT_RX_REQ = 0x04U, LPI2C_SLAVE_E←
  VENT_TX_EMPTY = 0x10U, LPI2C_SLAVE_EVENT_RX_FULL = 0x20U,
  LPI2C_SLAVE_EVENT_STOP = 0x80U }

  LPI2C slave events Implements : lpi2c_slave_event_t_Class.
- enum lpi2c_transfer_type_t { LPI2C_USING_DMA = 0, LPI2C_USING_INTERRUPTS = 1 }

  Type of LPI2C transfer (based on interrupts or DMA). Implements : lpi2c_transfer_type_t_Class.

LPI2C Driver

- status_t LPI2C_DRV_MasterInit (uint32_t instance, const lpi2c_master_user_config_t ∗userConfigPtr, lpi2c_master_state_t ∗master)

  Initialize the LPI2C master mode driver.
- status_t LPI2C_DRV_MasterDeinit (uint32_t instance)

  De-initialize the LPI2C master mode driver.
- void LPI2C_DRV_MasterGetBaudRate (uint32_t instance, lpi2c_baud_rate_params_t ∗baudRate)

  Get the currently configured baud rate.
- void LPI2C_DRV_MasterSetBaudRate (uint32_t instance, const lpi2c_mode_t operatingMode, const lpi2c←
  _baud_rate_params_t baudRate)

  Set the baud rate for any subsequent I2C communication.
- void LPI2C_DRV_MasterSetSlaveAddr (uint32_t instance, const uint16_t address, const bool is10bitAddr)

  Set the slave address for any subsequent I2C communication.
- status_t LPI2C_DRV_MasterSendData (uint32_t instance, const uint8_t ∗txBuff, uint32_t txSize, bool send←
  Stop)

  Perform a non-blocking send transaction on the I2C bus.
- status_t LPI2C_DRV_MasterSendDataBlocking (uint32_t instance, const uint8_t ∗txBuff, uint32_t txSize, bool sendStop, uint32_t timeout)

  Perform a blocking send transaction on the I2C bus.
- status_t LPI2C_DRV_MasterAbortTransferData (uint32_t instance)

  Abort a non-blocking I2C Master transmission or reception.
- status_t LPI2C_DRV_MasterReceiveData (uint32_t instance, uint8_t ∗rxBuff, uint32_t rxSize, bool sendStop)

  Perform a non-blocking receive transaction on the I2C bus.
- status_t LPI2C_DRV_MasterReceiveDataBlocking (uint32_t instance, uint8_t ∗rxBuff, uint32_t rxSize, bool sendStop, uint32_t timeout)

  Perform a blocking receive transaction on the I2C bus.
- status_t LPI2C_DRV_MasterGetTransferStatus (uint32_t instance, uint32_t ∗bytesRemaining)

  Return the current status of the I2C master transfer.
- void LPI2C_DRV_MasterIRQHandler (uint32_t instance)

  Handle master operation when I2C interrupt occurs.
- status_t LPI2C_DRV_SlaveInit (uint32_t instance, const lpi2c_slave_user_config_t ∗userConfigPtr, lpi2c_←
  slave_state_t ∗slave)

  Initialize the I2C slave mode driver.
- status_t LPI2C_DRV_SlaveDeinit (uint32_t instance)

De-initialize the I2C slave mode driver.
- status_t LPI2C_DRV_SlaveSetTxBuffer (uint32_t instance, const uint8_t ∗txBuff, uint32_t txSize)

  Provide a buffer for transmitting data.
- status_t LPI2C_DRV_SlaveSetRxBuffer (uint32_t instance, uint8_t ∗rxBuff, uint32_t rxSize)

  Provide a buffer for receiving data.
- status_t LPI2C_DRV_SlaveSendData (uint32_t instance, const uint8_t ∗txBuff, uint32_t txSize)

  Perform a non-blocking send transaction on the I2C bus.
- status_t LPI2C_DRV_SlaveSendDataBlocking (uint32_t instance, const uint8_t ∗txBuff, uint32_t txSize, uint32_t timeout)

  Perform a blocking send transaction on the I2C bus.
- status_t LPI2C_DRV_SlaveReceiveData (uint32_t instance, uint8_t ∗rxBuff, uint32_t rxSize)

  Perform a non-blocking receive transaction on the I2C bus.
- status_t LPI2C_DRV_SlaveReceiveDataBlocking (uint32_t instance, uint8_t ∗rxBuff, uint32_t rxSize, uint32↩_t timeout)

  Perform a blocking receive transaction on the I2C bus.
- status_t LPI2C_DRV_SlaveGetTransferStatus (uint32_t instance, uint32_t ∗bytesRemaining)

  Return the current status of the I2C slave transfer.
- status_t LPI2C_DRV_SlaveAbortTransferData (uint32_t instance)

  Abort a non-blocking I2C Master transmission or reception.
- void LPI2C_DRV_SlaveIRQHandler (uint32_t instance)

  Handle slave operation when I2C interrupt occurs.

3.47.2 Data Structure Documentation

3.47.2.1 struct lpi2c_master_user_config_t

Master configuration structure.

This structure is used to provide configuration parameters for the LPI2C master at initialization time. Implements : lpi2c_master_user_config_t_Class

Data Fields

- uint16_t slaveAddress
- bool is10bitAddr
- lpi2c_mode_t operatingMode
- uint32_t baudRate
- lpi2c_transfer_type_t transferType
- uint8_t dmaChannel
- lpi2c_master_callback_t masterCallback
- void ∗ callbackParam

Field Documentation

3.47.2.1.1 baudRate

```
uint32_t baudRate
```

The baud rate in hertz to use with current slave device

3.47.2.1.2 callbackParam

```
void* callbackParam
```

Parameter for the master callback function

3.47.2.1.3 dmaChannel

```
uint8_t dmaChannel
```

Channel number for DMA channel. If DMA mode isn't used this field will be ignored.

3.47.2.1.4 is10bitAddr

```
bool is10bitAddr
```

Selects 7-bit or 10-bit slave address

3.47.2.1.5 masterCallback

```
lpi2c_master_callback_t masterCallback
```

Master callback function. Note that this function will be called from the interrupt service routine at the end of a transfer, so its execution time should be as small as possible. It can be NULL if you want to check manually the status of the transfer.

3.47.2.1.6 operatingMode

```
lpi2c_mode_t operatingMode
```

I2C Operating mode

3.47.2.1.7 slaveAddress

```
uint16_t slaveAddress
```

Slave address, 7-bit or 10-bit

3.47.2.1.8 transferType

```
lpi2c_transfer_type_t transferType
```

Type of LPI2C transfer

3.47.2.2 struct lpi2c_slave_user_config_t

Slave configuration structure.

This structure is used to provide configuration parameters for the LPI2C slave at initialization time. Implements : lpi2c_slave_user_config_t_Class

Data Fields

- uint16_t slaveAddress
- bool is10bitAddr
- lpi2c_mode_t operatingMode
- bool slaveListening
- lpi2c_transfer_type_t transferType
- uint8_t dmaChannel
- lpi2c_slave_callback_t slaveCallback
- void ∗ callbackParam

Field Documentation

3.47.2.2.1 callbackParam

```
void* callbackParam
```

Parameter for the slave callback function

3.47.2.2.2 dmaChannel

```
uint8_t dmaChannel
```

Channel number for DMA rx channel. If DMA mode isn't used this field will be ignored.

3.47.2.2.3 is10bitAddr

```
bool is10bitAddr
```

Selects 7-bit or 10-bit slave address

3.47.2.2.4 operatingMode

```
lpi2c_mode_t operatingMode
```

I2C Operating mode

3.47.2.2.5 slaveAddress

```
uint16_t slaveAddress
```

Slave address, 7-bit or 10-bit

3.47.2.2.6 slaveCallback

```
lpi2c_slave_callback_t slaveCallback
```

Slave callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if the slave is not in listening mode (slaveListening = false)

3.47.2.2.7 slaveListening

```
bool slaveListening
```

Slave mode (always listening or on demand only)

3.47.2.2.8 transferType

```
lpi2c_transfer_type_t transferType
```

Type of LPI2C transfer

3.47.2.3 struct lpi2c_baud_rate_params_t

Baud rate structure.

This structure is used for setting or getting the baud rate. Implements : lpi2c_baud_rate_params_t_Class

Data Fields

- uint32_t baudRate

Field Documentation

3.47.2.3.1 baudRate

```
uint32_t baudRate
```

3.47.2.4 struct lpi2c_master_state_t

Master internal context structure.

This structure is used by the master-mode driver for its internal logic. It must be provided by the application through the LPI2C_DRV_MasterInit() function, then it cannot be freed until the driver is de-initialized using LPI2C_DRV_↩ MasterDeinit(). The application should make no assumptions about the content of this structure.

3.47.2.5 struct lpi2c_slave_state_t

Slave internal context structure.

This structure is used by the slave-mode driver for its internal logic. It must be provided by the application through the LPI2C_DRV_SlaveInit() function, then it cannot be freed until the driver is de-initialized using LPI2C_DRV_↩ SlaveDeinit(). The application should make no assumptions about the content of this structure.

3.47.3 Typedef Documentation

3.47.3.1 lpi2c_master_callback_t

```
typedef void(* lpi2c_master_callback_t) (uint8_t instance, lpi2c_master_event_t masterEvent,
void *userData)
```

Defines the example structure.

This structure is used as an example.

LPI2C master callback function

Callback functions are called by the LPI2C master at the end of a transfer on the I2C bus. After a transfer is completed the function is called and you can make further computation or in case of error you can tread that error.

3.47.3.2 lpi2c_slave_callback_t

```
typedef void(* lpi2c_slave_callback_t) (uint8_t instance, lpi2c_slave_event_t slaveEvent, void
*userData)
```

LPI2C slave callback function.

Callback functions are called by the LPI2C slave when relevant events are detected on the I2C bus. See type lpi2c_slave_event_t for a list of events. The callback can then react to the event, for example providing the buffers for transmission or reception.

3.47.4 Enumeration Type Documentation

3.47.4.1 lpi2c_master_event_t

```
enum lpi2c_master_event_t
```

LPI2C master events Implements : lpi2c_master_event_t_Class.

Enumerator

LPI2C_MASTER_EVENT_TX	The I2C master has ended transmitting the data
LPI2C_MASTER_EVENT_RX	The I2C master has ended receiving the data
LPI2C_MASTER_EVENT_FIFO_ERROR	The I2C master has received a FIFO error
LPI2C_MASTER_EVENT_ARBITRATION_LOST	The I2C master is experiencing the loss of arbitration
LPI2C_MASTER_EVENT_NACK	The I2C master has received a NACK

3.47.4.2 lpi2c_mode_t

```
enum lpi2c_mode_t
```

I2C operating modes Implements : lpi2c_mode_t_Class.

Enumerator

LPI2C_STANDARD_MODE	Standard-mode (Sm), bidirectional data transfers up to 100 kbit/s
LPI2C_FAST_MODE	Fast-mode (Fm), bidirectional data transfers up to 400 kbit/s

3.47.4.3 lpi2c_slave_event_t

```
enum lpi2c_slave_event_t
```

LPI2C slave events Implements : lpi2c_slave_event_t_Class.

Enumerator

LPI2C_SLAVE_EVENT_TX_REQ	The I2C slave received a request to transmit data
LPI2C_SLAVE_EVENT_RX_REQ	The I2C slave received a request to receive data
LPI2C_SLAVE_EVENT_TX_EMPTY	The I2C slave transmit buffer empty
LPI2C_SLAVE_EVENT_RX_FULL	The I2C slave receive buffer full
LPI2C_SLAVE_EVENT_STOP	The I2C slave STOP signal detected

3.47.4.4 lpi2c_transfer_type_t

```
enum lpi2c_transfer_type_t
```

Type of LPI2C transfer (based on interrupts or DMA). Implements : lpi2c_transfer_type_t_Class.

Enumerator

LPI2C_USING_DMA	The driver will use DMA to perform I2C transfer
LPI2C_USING_INTERRUPTS	The driver will use interrupts to perform I2C transfer

3.47.5 Function Documentation

3.47.5.1 LPI2C_DRV_MasterAbortTransferData()

```
status_t LPI2C_DRV_MasterAbortTransferData (
             uint32_t instance )
```

Abort a non-blocking I2C Master transmission or reception.

Parameters

instance	LPI2C peripheral instance number

Returns

Error or success status returned by API

3.47.5.2 LPI2C_DRV_MasterDeinit()

```
status_t LPI2C_DRV_MasterDeinit (
             uint32_t instance )
```

De-initialize the LPI2C master mode driver.

This function de-initializes the LPI2C driver in master mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

Parameters

instance	LPI2C peripheral instance number

Returns

Error or success status returned by API

3.47.5.3 LPI2C_DRV_MasterGetBaudRate()

```
void LPI2C_DRV_MasterGetBaudRate (
            uint32_t instance,
            lpi2c_baud_rate_params_t * baudRate )
```

Get the currently configured baud rate.

This function returns the currently configured baud rate.

Parameters

instance	LPI2C peripheral instance number
baudRate	structure that contains the current baud rate in hertz and the baud rate in hertz for High-speed mode (unused in other modes, can be NULL)

3.47.5.4 LPI2C_DRV_MasterGetTransferStatus()

```
status_t LPI2C_DRV_MasterGetTransferStatus (
            uint32_t instance,
            uint32_t * bytesRemaining )
```

Return the current status of the I2C master transfer.

This function can be called during a non-blocking transmission to check the status of the transfer.

Parameters

instance	LPI2C peripheral instance number
bytesRemaining	the number of remaining bytes in the active I2C transfer

Returns

Error or success status returned by API

3.47.5.5 LPI2C_DRV_MasterInit()

```
status_t LPI2C_DRV_MasterInit (
            uint32_t instance,
            const lpi2c_master_user_config_t * userConfigPtr,
            lpi2c_master_state_t * master )
```

Initialize the LPI2C master mode driver.

This function initializes the LPI2C driver in master mode.

Parameters

instance	LPI2C peripheral instance number
userConfigPtr	Pointer to the LPI2C master user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
master	Pointer to the LPI2C master driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using LPI2C_DRV_MasterDeinit().

Returns

Error or success status returned by API

3.47.5.6 LPI2C_DRV_MasterIRQHandler()

```
void LPI2C_DRV_MasterIRQHandler (
            uint32_t instance )
```

Handle master operation when I2C interrupt occurs.

This is the interrupt service routine for the LPI2C master mode driver. It handles the rest of the transfer started by one of the send/receive functions.

Parameters

instance	LPI2C peripheral instance number

3.47.5.7 LPI2C_DRV_MasterReceiveData()

```
status_t LPI2C_DRV_MasterReceiveData (
            uint32_t instance,
            uint8_t * rxBuff,
            uint32_t rxSize,
            bool sendStop )
```

Perform a non-blocking receive transaction on the I2C bus.

This function starts the reception of a block of data from the currently configured slave address and returns immediately. The rest of the reception is handled by the interrupt service routine. Use LPI2C_DRV_MasterGetReceive←Status() to check the progress of the reception.

Parameters

instance	LPI2C peripheral instance number
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred
sendStop	specifies whether or not to generate stop condition after the reception

Returns

Error or success status returned by API

3.47.5.8 LPI2C_DRV_MasterReceiveDataBlocking()

```
status_t LPI2C_DRV_MasterReceiveDataBlocking (
            uint32_t instance,
            uint8_t * rxBuff,
            uint32_t rxSize,
            bool sendStop,
            uint32_t timeout )
```

Perform a blocking receive transaction on the I2C bus.

This function receives a block of data from the currently configured slave address, and only returns when the transmission is complete.

Parameters

instance	LPI2C peripheral instance number
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred
sendStop	specifies whether or not to generate stop condition after the reception
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.47.5.9 LPI2C_DRV_MasterSendData()

```
status_t LPI2C_DRV_MasterSendData (
            uint32_t instance,
            const uint8_t * txBuff,
            uint32_t txSize,
            bool sendStop )
```

Perform a non-blocking send transaction on the I2C bus.

This function starts the transmission of a block of data to the currently configured slave address and returns immediately. The rest of the transmission is handled by the interrupt service routine. Use LPI2C_DRV_MasterGetSend←Status() to check the progress of the transmission.

Parameters

instance	LPI2C peripheral instance number
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred
sendStop	specifies whether or not to generate stop condition after the transmission

Returns

Error or success status returned by API

3.47.5.10 LPI2C_DRV_MasterSendDataBlocking()

```
status_t LPI2C_DRV_MasterSendDataBlocking (
            uint32_t instance,
            const uint8_t * txBuff,
            uint32_t txSize,
            bool sendStop,
            uint32_t timeout )
```

Perform a blocking send transaction on the I2C bus.

This function sends a block of data to the currently configured slave address, and only returns when the transmission is complete.

Parameters

instance	LPI2C peripheral instance number
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred
sendStop	specifies whether or not to generate stop condition after the transmission
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.47.5.11 LPI2C_DRV_MasterSetBaudRate()

```
void LPI2C_DRV_MasterSetBaudRate (
            uint32_t instance,
            const lpi2c_mode_t operatingMode,
            const lpi2c_baud_rate_params_t baudRate )
```

Set the baud rate for any subsequent I2C communication.

This function sets the baud rate (SCL frequency) for the I2C master. It can also change the operating mode. If the operating mode is High-Speed, a second baud rate must be provided for high-speed communication. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency protocol clock for the LPI2C module. The application should call LPI2C_DRV_MasterGet↩ BaudRate() after LPI2C_DRV_MasterSetBaudRate() to check what baud rate was actually set.

Parameters

instance	LPI2C peripheral instance number
operatingMode	I2C operating mode
baudRate	structure that contains the baud rate in hertz to use by current slave device and also the baud rate in hertz for High-speed mode (unused in other modes)

3.47.5.12 LPI2C_DRV_MasterSetSlaveAddr()

```
void LPI2C_DRV_MasterSetSlaveAddr (
            uint32_t instance,
            const uint16_t address,
            const bool is10bitAddr )
```

Set the slave address for any subsequent I2C communication.

This function sets the slave address which will be used for any future transfer initiated by the LPI2C master.

Parameters

instance	LPI2C peripheral instance number
address	slave address, 7-bit or 10-bit
is10bitAddr	specifies if provided address is 10-bit

3.47.5.13 LPI2C_DRV_SlaveAbortTransferData()

```
status_t LPI2C_DRV_SlaveAbortTransferData (
            uint32_t instance )
```

Abort a non-blocking I2C Master transmission or reception.

Parameters

instance	LPI2C peripheral instance number

Returns

Error or success status returned by API

3.47.5.14 LPI2C_DRV_SlaveDeinit()

```
status_t LPI2C_DRV_SlaveDeinit (
            uint32_t instance )
```

De-initialize the I2C slave mode driver.

This function de-initializes the LPI2C driver in slave mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

Parameters

instance	LPI2C peripheral instance number

Returns

Error or success status returned by API

3.47.5.15 LPI2C_DRV_SlaveGetTransferStatus()

```
status_t LPI2C_DRV_SlaveGetTransferStatus (
            uint32_t instance,
            uint32_t * bytesRemaining )
```

Return the current status of the I2C slave transfer.

This function can be called during a non-blocking transmission to check the status of the transfer.

Parameters

instance	LPI2C peripheral instance number
bytesRemaining	the number of remaining bytes in the active I2C transfer

Returns

Error or success status returned by API

3.47.5.16 LPI2C_DRV_SlaveInit()

```
status_t LPI2C_DRV_SlaveInit (
            uint32_t instance,
            const lpi2c_slave_user_config_t * userConfigPtr,
            lpi2c_slave_state_t * slave )
```

Initialize the I2C slave mode driver.

Parameters

instance	LPI2C peripheral instance number
userConfigPtr	Pointer to the LPI2C slave user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
slave	Pointer to the LPI2C slave driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using LPI2C_DRV_SlaveDeinit().

Returns

Error or success status returned by API

3.47.5.17 LPI2C_DRV_SlaveIRQHandler()

```
void LPI2C_DRV_SlaveIRQHandler (
            uint32_t instance )
```

Handle slave operation when I2C interrupt occurs.

This is the interrupt service routine for the LPI2C slave mode driver. It handles any transfer initiated by an I2C master and notifies the application via the provided callback when relevant events occur.

Parameters

instance	LPI2C peripheral instance number

3.47.5.18 LPI2C_DRV_SlaveReceiveData()

```
status_t LPI2C_DRV_SlaveReceiveData (
            uint32_t instance,
            uint8_t * rxBuff,
            uint32_t rxSize )
```

Perform a non-blocking receive transaction on the I2C bus.

Performs a non-blocking receive transaction on the I2C bus when the slave is not in listening mode (initialized with slaveListening = false). It starts the reception and returns immediately. The rest of the reception is handled by the interrupt service routine. Use LPI2C_DRV_SlaveGetReceiveStatus() to check the progress of the reception.

Parameters

instance	LPI2C peripheral instance number
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.47.5.19 LPI2C_DRV_SlaveReceiveDataBlocking()

```
status_t LPI2C_DRV_SlaveReceiveDataBlocking (
            uint32_t instance,
            uint8_t * rxBuff,
            uint32_t rxSize,
            uint32_t timeout )
```

Perform a blocking receive transaction on the I2C bus.

Performs a blocking receive transaction on the I2C bus when the slave is not in listening mode (initialized with slaveListening = false). It sets up the reception and then waits for the transfer to complete before returning.

Parameters

instance	LPI2C peripheral instance number
rxBuff	pointer to the buffer where to store received data
rxSize	length in bytes of the data to be transferred
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.47.5.20 LPI2C_DRV_SlaveSendData()

```
status_t LPI2C_DRV_SlaveSendData (
            uint32_t instance,
            const uint8_t * txBuff,
            uint32_t txSize )
```

Perform a non-blocking send transaction on the I2C bus.

Performs a non-blocking send transaction on the I2C bus when the slave is not in listening mode (initialized with slaveListening = false). It starts the transmission and returns immediately. The rest of the transmission is handled by the interrupt service routine. Use LPI2C_DRV_SlaveGetTransmitStatus() to check the progress of the transmission.

Parameters

instance	LPI2C peripheral instance number
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.47.5.21 LPI2C_DRV_SlaveSendDataBlocking()

```
status_t LPI2C_DRV_SlaveSendDataBlocking (
             uint32_t instance,
             const uint8_t * txBuff,
             uint32_t txSize,
             uint32_t timeout )
```

Perform a blocking send transaction on the I2C bus.

Performs a blocking send transaction on the I2C bus when the slave is not in listening mode (initialized with slave←Listening = false). It sets up the transmission and then waits for the transfer to complete before returning.

Parameters

instance	LPI2C peripheral instance number
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred
timeout	timeout for the transfer in milliseconds

Returns

Error or success status returned by API

3.47.5.22 LPI2C_DRV_SlaveSetRxBuffer()

```
status_t LPI2C_DRV_SlaveSetRxBuffer (
             uint32_t instance,
             uint8_t * rxBuff,
             uint32_t rxSize )
```

Provide a buffer for receiving data.

This function provides a buffer in which the LPI2C slave-mode driver can store received data. It can be called for example from the user callback provided at initialization time, when the driver reports events LPI2C_SLAVE_EV←ENT_RX_REQ or LPI2C_SLAVE_EVENT_RX_FULL.

Parameters

instance	LPI2C peripheral instance number
rxBuff	pointer to the data to be transferred
rxSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.47.5.23 LPI2C_DRV_SlaveSetTxBuffer()

```
status_t LPI2C_DRV_SlaveSetTxBuffer (
             uint32_t instance,
             const uint8_t * txBuff,
             uint32_t txSize )
```

Provide a buffer for transmitting data.

This function provides a buffer from which the LPI2C slave-mode driver can transmit data. It can be called for example from the user callback provided at initialization time, when the driver reports events LPI2C_SLAVE_EV←ENT_TX_REQ or LPI2C_SLAVE_EVENT_TX_EMPTY.

Parameters

instance	LPI2C peripheral instance number
txBuff	pointer to the data to be transferred
txSize	length in bytes of the data to be transferred

Returns

Error or success status returned by API

3.48 LPI2C HAL

3.48.1 Detailed Description

Low Power Inter-Integrated Circuit (LPI2C) Hardware Abstraction Layer.

Low Power Inter-Integrated Circuit Hardware Abstraction Layer.

LPI2C HAL provides the API for reading and writing register bit-fields belonging to the LPI2C module. It also provides an initialization function for bringing the module to the reset state.

For higher-level functionality, use the LPI2C driver.

Data Structures

- struct lpi2c_version_info_t

  LPI2C module version number Implements : lpi2c_version_info_t_Class. More...

Macros

- #define LPI2C_HAL_MASTER_DATA_MATCH_INT 0x4000UL
- #define LPI2C_HAL_MASTER_PIN_LOW_TIMEOUT_INT 0x2000UL
- #define LPI2C_HAL_MASTER_FIFO_ERROR_INT 0x1000UL
- #define LPI2C_HAL_MASTER_ARBITRATION_LOST_INT 0x800UL
- #define LPI2C_HAL_MASTER_NACK_DETECT_INT 0x400UL
- #define LPI2C_HAL_MASTER_STOP_DETECT_INT 0x200UL
- #define LPI2C_HAL_MASTER_END_PACKET_INT 0x100UL
- #define LPI2C_HAL_MASTER_RECEIVE_DATA_INT 0x2UL
- #define LPI2C_HAL_MASTER_TRANSMIT_DATA_INT 0x1UL
- #define LPI2C_HAL_SLAVE_SMBUS_ALERT_RESPONSE_INT 0x8000UL
- #define LPI2C_HAL_SLAVE_GENERAL_CALL_INT 0x4000UL
- #define LPI2C_HAL_SLAVE_ADDRESS_MATCH_1_INT 0x2000UL
- #define LPI2C_HAL_SLAVE_ADDRESS_MATCH_0_INT 0x1000UL
- #define LPI2C_HAL_SLAVE_FIFO_ERROR_INT 0x800UL
- #define LPI2C_HAL_SLAVE_BIT_ERROR_INT 0x400UL
- #define LPI2C_HAL_SLAVE_STOP_DETECT_INT 0x200UL
- #define LPI2C_HAL_SLAVE_REPEATED_START_INT 0x100UL
- #define LPI2C_HAL_SLAVE_TRANSMIT_ACK_INT 0x8UL
- #define LPI2C_HAL_SLAVE_ADDRESS_VALID_INT 0x4UL
- #define LPI2C_HAL_SLAVE_RECEIVE_DATA_INT 0x2UL
- #define LPI2C_HAL_SLAVE_TRANSMIT_DATA_INT 0x1UL

Enumerations

- enum lpi2c_rx_data_match_t { LPI2C_RX_DATA_KEEP_ALL = 0U, LPI2C_RX_DATA_DROP_NON_MATCHING = 1U }

    Non_matching data discard options Implements : lpi2c_rx_data_match_t_Class.
- enum lpi2c_hreq_source_t { LPI2C_HREQ_EXTERNAL_PIN = 0U, LPI2C_HREQ_INTERNAL_TRIGGER = 1U }

    Host request input source selection Implements : lpi2c_hreq_source_t_Class.
- enum lpi2c_hreq_polarity_t { LPI2C_HREQ_POL_ACTIVE_HIGH = 0U, LPI2C_HREQ_POL_ACTIVE_LOW = 1U }

    Host request input polarity selection Implements : lpi2c_hreq_polarity_t_Class.
- enum lpi2c_pin_config_t {
  LPI2C_CFG_2PIN_OPEN_DRAIN = 0U, LPI2C_CFG_2PIN_OUTPUT_ONLY = 1U, LPI2C_CFG_2PIN_PUSH_PULL = 2U, LPI2C_CFG_4PIN_PUSH_PULL = 3U,
  LPI2C_CFG_2PIN_OPEN_DRAIN_SLAVE = 4U, LPI2C_CFG_2PIN_OUTPUT_ONLY_SLAVE = 5U, LPI2C_CFG_2PIN_PUSH_PULL_SLAVE = 6U, LPI2C_CFG_4PIN_PUSH_PULL_INVERTED = 7U }

    Pin configuration selection Implements : lpi2c_pin_config_t_Class.
- enum lpi2c_match_config_t {
  LPI2C_MATCH_DISABLED = 0U, LPI2C_MATCH_OR_FIRST = 2U, LPI2C_MATCH_OR_ANY = 3U, LPI2C_MATCH_AND_FIRST = 4U,
  LPI2C_MATCH_AND_ANY = 5U, LPI2C_MATCH_MASK_FIRST = 6U, LPI2C_MATCH_MASK_ANY = 7U }

    Data match selection Implements : lpi2c_match_config_t_Class.
- enum lpi2c_timeout_config_t { LPI2C_TIMEOUT_SCL = 0U, LPI2C_TIMEOUT_SCL_OR_SDA = 1U }

    SCL/SDA low time-out configuration Implements : lpi2c_timeout_config_t_Class.
- enum lpi2c_nack_config_t { LPI2C_NACK_RECEIVE = 0U, LPI2C_NACK_IGNORE = 1U }

    Master NACK reaction configuration Implements : lpi2c_nack_config_t_Class.
- enum lpi2c_master_prescaler_t {
  LPI2C_MASTER_PRESC_DIV_1 = 0U, LPI2C_MASTER_PRESC_DIV_2 = 1U, LPI2C_MASTER_PRESC_DIV_4 = 2U, LPI2C_MASTER_PRESC_DIV_8 = 3U,
  LPI2C_MASTER_PRESC_DIV_16 = 4U, LPI2C_MASTER_PRESC_DIV_32 = 5U, LPI2C_MASTER_PRESC_DIV_64 = 6U, LPI2C_MASTER_PRESC_DIV_128 = 7U }

    LPI2C master prescaler options Implements : lpi2c_master_prescaler_t_Class.
- enum lpi2c_master_command_t {
  LPI2C_MASTER_COMMAND_TRANSMIT = 0U, LPI2C_MASTER_COMMAND_RECEIVE = 1U, LPI2C_MASTER_COMMAND_STOP = 2U, LPI2C_MASTER_COMMAND_RECEIVE_DISCARD = 3U,
  LPI2C_MASTER_COMMAND_START = 4U, LPI2C_MASTER_COMMAND_START_NACK = 5U, LPI2C_MASTER_COMMAND_START_HS = 6U, LPI2C_MASTER_COMMAND_START_NACK_HS = 7U }

    LPI2C master commands Implements : lpi2c_master_command_t_Class.
- enum lpi2c_slave_addr_config_t {
  LPI2C_SLAVE_ADDR_MATCH_0_7BIT = 0U, LPI2C_SLAVE_ADDR_MATCH_0_10BIT = 1U, LPI2C_SLAVE_ADDR_MATCH_0_7BIT_OR_1_7BIT = 2U, LPI2C_SLAVE_ADDR_MATCH_0_10BIT_OR_1_10BIT = 3U,
  LPI2C_SLAVE_ADDR_MATCH_0_7BIT_OR_1_10BIT = 4U, LPI2C_SLAVE_ADDR_MATCH_0_10BIT_OR_1_7BIT = 5U, LPI2C_SLAVE_ADDR_MATCH_RANGE_7BIT = 6U, LPI2C_SLAVE_ADDR_MATCH_RANGE_10BIT = 7U }

    Slave address configuration Implements : lpi2c_slave_addr_config_t_Class.
- enum lpi2c_slave_nack_config_t { LPI2C_SLAVE_NACK_END_TRANSFER = 0U, LPI2C_SLAVE_NACK_CONTINUE_TRANSFER = 1U }

    Slave NACK reaction configuration Implements : lpi2c_slave_nack_config_t_Class.
- enum lpi2c_slave_rxdata_config_t { LPI2C_SLAVE_RXDATA_DATA_ONLY = 0U, LPI2C_SLAVE_RXDATA_DATA_OR_ADDR = 1U }

    Slave receive data register function configuration Implements : lpi2c_slave_rxdata_config_t_Class.
- enum lpi2c_slave_txflag_config_t { LPI2C_SLAVE_TXFLAG_TRANSFER_ONLY = 0U, LPI2C_SLAVE_TXFLAG_ALWAYS = 1U }

Slave Transmit Data Flag function control Implements : lpi2c_slave_txflag_config_t_Class.

- enum lpi2c_slave_addr_valid_t { LPI2C_SLAVE_ADDR_VALID = 0U, LPI2C_SLAVE_ADDR_NOT_VALID = 1U }

  Slave received address validity Implements : lpi2c_slave_addr_valid_t_Class.
- enum lpi2c_slave_nack_transmit_t { LPI2C_SLAVE_TRANSMIT_ACK = 0U, LPI2C_SLAVE_TRANSMIT_↩ NACK = 1U }

  Slave ACK transmission options Implements : lpi2c_slave_nack_transmit_t_Class.

Configuration

- static void LPI2C_HAL_GetVersion (const LPI2C_Type ∗baseAddr, lpi2c_version_info_t ∗versionInfo)

  Get the version of the LPI2C module.
- static uint16_t LPI2C_HAL_MasterGetRxFIFOSize (const LPI2C_Type ∗baseAddr)

  Get the size of the Master Receive FIFO.
- static uint16_t LPI2C_HAL_MasterGetTxFIFOSize (const LPI2C_Type ∗baseAddr)

  Get the size of the Master Transmit FIFO.
- static void LPI2C_HAL_MasterRxFIFOResetCmd (LPI2C_Type ∗baseAddr)

  Reset the master receive FIFO.
- static void LPI2C_HAL_MasterTxFIFOResetCmd (LPI2C_Type ∗baseAddr)

  Reset the master transmit FIFO.
- static void LPI2C_HAL_MasterSetDebugMode (LPI2C_Type ∗baseAddr, bool enable)

  Set the master behaviour in Debug mode.
- static void LPI2C_HAL_MasterSetDozeMode (LPI2C_Type ∗baseAddr, bool enable)

  Set the master behaviour in Doze mode.
- static void LPI2C_HAL_MasterSetSoftwareReset (LPI2C_Type ∗baseAddr, bool enable)

  Set/clear the master reset command.
- static void LPI2C_HAL_MasterSetEnable (LPI2C_Type ∗baseAddr, bool enable)

  Enable or disable the LPI2C master.
- static bool LPI2C_HAL_MasterGetDebugMode (const LPI2C_Type ∗baseAddr)

  Return the debug mode setting for the LPI2C master.
- static bool LPI2C_HAL_MasterGetDozeMode (const LPI2C_Type ∗baseAddr)

  Return the doze mode setting for the LPI2C master.
- static bool LPI2C_HAL_MasterGetSoftwareReset (const LPI2C_Type ∗baseAddr)

  Return the reset setting for the LPI2C master.
- static bool LPI2C_HAL_MasterGetEnable (const LPI2C_Type ∗baseAddr)

  Return the enable/disable setting for the LPI2C master.
- static bool LPI2C_HAL_MasterGetBusBusyEvent (const LPI2C_Type ∗baseAddr)

  Return the idle/busy state of the I2C bus.
- static bool LPI2C_HAL_MasterGetMasterBusyEvent (const LPI2C_Type ∗baseAddr)

  Return the idle/busy state of the LPI2C master.
- static bool LPI2C_HAL_MasterGetReceiveDataReadyEvent (const LPI2C_Type ∗baseAddr)

  Indicate the availability of receive data.
- static bool LPI2C_HAL_MasterGetTransmitDataRequestEvent (const LPI2C_Type ∗baseAddr)

  Indicate if the LPI2C master requests more data.
- static bool LPI2C_HAL_MasterGetDataMatchEvent (const LPI2C_Type ∗baseAddr)

  Check the occurrence of a data match event.
- static bool LPI2C_HAL_MasterGetPinLowTimeoutEvent (const LPI2C_Type ∗baseAddr)

  Check the occurrence of a pin low timeout event.
- static bool LPI2C_HAL_MasterGetFIFOErrorEvent (const LPI2C_Type ∗baseAddr)

  Check the occurrence of a FIFO error event.

- static bool LPI2C_HAL_MasterGetArbitrationLostEvent (const LPI2C_Type ∗baseAddr)

    Check the occurrence of an arbitration lost event.
- static bool LPI2C_HAL_MasterGetNACKDetectEvent (const LPI2C_Type ∗baseAddr)

    Check the occurrence of an unexpected NACK event.
- static bool LPI2C_HAL_MasterGetSTOPDetectEvent (const LPI2C_Type ∗baseAddr)

    Check the occurrence of a STOP detect event.
- static bool LPI2C_HAL_MasterGetEndPacketEvent (const LPI2C_Type ∗baseAddr)

    Check the occurrence of an end packet event.
- static void LPI2C_HAL_MasterClearDataMatchEvent (LPI2C_Type ∗baseAddr)

    Clear the data match event flag.
- static void LPI2C_HAL_MasterClearPinLowTimeoutEvent (LPI2C_Type ∗baseAddr)

    Clear the pin low timeout event flag.
- static void LPI2C_HAL_MasterClearFIFOErrorEvent (LPI2C_Type ∗baseAddr)

    Clear the FIFO error event flag.
- static void LPI2C_HAL_MasterClearArbitrationLostEvent (LPI2C_Type ∗baseAddr)

    Clear the arbitration lost event flag.
- static void LPI2C_HAL_MasterClearNACKDetectEvent (LPI2C_Type ∗baseAddr)

    Clear the unexpected NACK event flag.
- static void LPI2C_HAL_MasterClearSTOPDetectEvent (LPI2C_Type ∗baseAddr)

    Clear the STOP detect event flag.
- static void LPI2C_HAL_MasterClearEndPacketEvent (LPI2C_Type ∗baseAddr)

    Clear the end packet event flag.
- static void LPI2C_HAL_MasterSetRxDMA (LPI2C_Type ∗baseAddr, bool enable)

    Enable/disable receive data DMA requests.
- static void LPI2C_HAL_MasterSetTxDMA (LPI2C_Type ∗baseAddr, bool enable)

    Enable/disable transmit data DMA requests.
- static bool LPI2C_HAL_MasterGetRxDMA (const LPI2C_Type ∗baseAddr)

    Check if receive data DMA requests are enabled.
- static bool LPI2C_HAL_MasterGetTxDMA (const LPI2C_Type ∗baseAddr)

    Check if transmit data DMA requests are enabled.
- static void LPI2C_HAL_MasterSetInt (LPI2C_Type ∗baseAddr, uint32_t interrupts, bool enable)

    Enable or disable specified LPI2C master interrupts.
- static bool LPI2C_HAL_MasterGetInt (const LPI2C_Type ∗baseAddr, uint32_t interrupts)

    Return the state of the specified LPI2C master interrupt.
- static void LPI2C_HAL_MasterSetRxDataMatch (LPI2C_Type ∗baseAddr, lpi2c_rx_data_match_t rxData↩ Match)

    Control the discarding of data that does not match the configured criteria.
- static lpi2c_rx_data_match_t LPI2C_HAL_MasterGetRxDataMatch (const LPI2C_Type ∗baseAddr)

    Return the current data match discarding policy.
- static void LPI2C_HAL_MasterSetCircularFIFO (LPI2C_Type ∗baseAddr, bool enable)

    Set the circular FIFO mode for the transmit FIFO.
- static bool LPI2C_HAL_MasterGetCircularFIFO (const LPI2C_Type ∗baseAddr)

    Return the circular FIFO mode for the transmit FIFO.
- static void LPI2C_HAL_MasterSetHreqSelect (LPI2C_Type ∗baseAddr, lpi2c_hreq_source_t source)

    Set the source of the host request input.
- static void LPI2C_HAL_MasterSetHreqPolarity (LPI2C_Type ∗baseAddr, lpi2c_hreq_polarity_t polarity)

    Set the polarity of the host request input pin.
- static void LPI2C_HAL_MasterSetHreqEnable (LPI2C_Type ∗baseAddr, bool enable)

    Enable/disable the host request feature.
- static lpi2c_hreq_source_t LPI2C_HAL_MasterGetHreqSelect (const LPI2C_Type ∗baseAddr)

    Return the source of the host request input.

- static lpi2c_hreq_polarity_t LPI2C_HAL_MasterGetHreqPolarity (const LPI2C_Type ∗baseAddr)

  Return the polarity of the host request input pin.
- static bool LPI2C_HAL_MasterGetHreqEnable (const LPI2C_Type ∗baseAddr)

  Return the enable/disable state the host request feature.
- static void LPI2C_HAL_MasterSetPinConfig (LPI2C_Type ∗baseAddr, lpi2c_pin_config_t configuration)

  Set the pin mode of the module.
- static lpi2c_pin_config_t LPI2C_HAL_MasterGetPinConfig (const LPI2C_Type ∗baseAddr)

  Return the pin mode of the module.
- static void LPI2C_HAL_MasterSetMatchConfig (LPI2C_Type ∗baseAddr, lpi2c_match_config_t configuration)

  Set the match mode of the module.
- static lpi2c_match_config_t LPI2C_HAL_MasterGetMatchConfig (const LPI2C_Type ∗baseAddr)

  Return the match mode of the module.
- static void LPI2C_HAL_MasterSetTimeoutConfig (LPI2C_Type ∗baseAddr, lpi2c_timeout_config_t configuration)

  Set the timeout configuration of the module.
- static lpi2c_timeout_config_t LPI2C_HAL_MasterGetTimeoutConfig (const LPI2C_Type ∗baseAddr)

  Return the timeout configuration of the module.
- static void LPI2C_HAL_MasterSetNACKConfig (LPI2C_Type ∗baseAddr, lpi2c_nack_config_t configuration)

  Configure the reaction of the module on NACK reception.
- static lpi2c_nack_config_t LPI2C_HAL_MasterGetNACKConfig (const LPI2C_Type ∗baseAddr)

  Return the reaction of the module on NACK reception.
- static void LPI2C_HAL_MasterSetAutoStopConfig (LPI2C_Type ∗baseAddr, bool enable)

  Configure the automatic generation of STOP condition.
- static bool LPI2C_HAL_MasterGetAutoStopConfig (const LPI2C_Type ∗baseAddr)

  Return the current setting for automatic generation of STOP condition.
- static void LPI2C_HAL_MasterSetPrescaler (LPI2C_Type ∗baseAddr, lpi2c_master_prescaler_t prescaler)

  Configure the LPI2C master prescaler.
- static lpi2c_master_prescaler_t LPI2C_HAL_MasterGetPrescaler (const LPI2C_Type ∗baseAddr)

  Return the LPI2C master prescaler.
- static void LPI2C_HAL_MasterSetSDAGlitchFilter (LPI2C_Type ∗baseAddr, uint8_t cycles)

  Configure the LPI2C SDA glitch filter.
- static uint8_t LPI2C_HAL_MasterGetSDAGlitchFilter (const LPI2C_Type ∗baseAddr)

  Return the LPI2C SDA glitch filter configuration.
- static void LPI2C_HAL_MasterSetSCLGlitchFilter (LPI2C_Type ∗baseAddr, uint8_t cycles)

  Configure the LPI2C SCL glitch filter.
- static uint8_t LPI2C_HAL_MasterGetSCLGlitchFilter (const LPI2C_Type ∗baseAddr)

  Return the LPI2C SCL glitch filter configuration.
- static void LPI2C_HAL_MasterSetBusIdleTimeout (LPI2C_Type ∗baseAddr, uint16_t cycles)

  Configure the bus idle timeout.
- static uint16_t LPI2C_HAL_MasterGetBusIdleTimeout (const LPI2C_Type ∗baseAddr)

  Return the bus idle timeout configuration.
- static void LPI2C_HAL_MasterSetPinLowTimeout (LPI2C_Type ∗baseAddr, uint32_t cycles)

  Configure the pin low timeout.
- static uint32_t LPI2C_HAL_MasterGetPinLowTimeout (const LPI2C_Type ∗baseAddr)

  Return the pin low timeout configuration.
- static void LPI2C_HAL_MasterSetMatch0 (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the MATCH0 value for the data match feature.
- static uint8_t LPI2C_HAL_MasterGetMatch0 (const LPI2C_Type ∗baseAddr)

  Return the MATCH0 value for the data match feature.
- static void LPI2C_HAL_MasterSetMatch1 (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the MATCH1 value for the data match feature.

- static uint8_t LPI2C_HAL_MasterGetMatch1 (const LPI2C_Type ∗baseAddr)

  Return the MATCH1 value for the data match feature.
- static void LPI2C_HAL_MasterSetDataValidDelay (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the data hold time for SDA.
- static uint8_t LPI2C_HAL_MasterGetDataValidDelay (const LPI2C_Type ∗baseAddr)

  Return the configured data hold time for SDA.
- static void LPI2C_HAL_MasterSetSetupHoldDelay (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the setup and hold delay for a START / STOP condition.
- static uint8_t LPI2C_HAL_MasterGetSetupHoldDelay (const LPI2C_Type ∗baseAddr)

  Return the configured setup and hold time.
- static void LPI2C_HAL_MasterSetClockHighPeriod (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the minimum clock high period.
- static uint8_t LPI2C_HAL_MasterGetClockHighPeriod (const LPI2C_Type ∗baseAddr)

  Return the configured minimum clock high period.
- static void LPI2C_HAL_MasterSetClockLowPeriod (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the minimum clock low period.
- static uint8_t LPI2C_HAL_MasterGetClockLowPeriod (const LPI2C_Type ∗baseAddr)

  Return the configured minimum clock low period.
- static void LPI2C_HAL_MasterSetDataValidDelayHS (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the data hold time for SDA in high-speed mode.
- static uint8_t LPI2C_HAL_MasterGetDataValidDelayHS (const LPI2C_Type ∗baseAddr)

  Return the configured data hold time for SDA in high-speed mode.
- static void LPI2C_HAL_MasterSetSetupHoldDelayHS (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the setup and hold time for a START / STOP condition in high-speed mode.
- static uint8_t LPI2C_HAL_MasterGetSetupHoldDelayHS (const LPI2C_Type ∗baseAddr)

  Return the configured setup and hold time in high-speed mode.
- static void LPI2C_HAL_MasterSetClockHighPeriodHS (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the minimum clock high period in high-speed mode.
- static uint8_t LPI2C_HAL_MasterGetClockHighPeriodHS (const LPI2C_Type ∗baseAddr)

  Return the configured minimum clock high period in high-speed mode.
- static void LPI2C_HAL_MasterSetClockLowPeriodHS (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the minimum clock low period in high-speed mode.
- static uint8_t LPI2C_HAL_MasterGetClockLowPeriodHS (const LPI2C_Type ∗baseAddr)

  Return the configured minimum clock low period in high-speed mode.
- static void LPI2C_HAL_MasterSetRxFIFOWatermark (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the receive FIFO watermark.
- static uint8_t LPI2C_HAL_MasterGetRxFIFOWatermark (const LPI2C_Type ∗baseAddr)

  Return the configured receive FIFO watermark.
- static void LPI2C_HAL_MasterSetTxFIFOWatermark (LPI2C_Type ∗baseAddr, uint8_t value)

  Set the transmit FIFO watermark.
- static uint8_t LPI2C_HAL_MasterGetTxFIFOWatermark (const LPI2C_Type ∗baseAddr)

  Return the configured transmit FIFO watermark.
- static uint8_t LPI2C_HAL_MasterGetRxFIFOCount (const LPI2C_Type ∗baseAddr)

  Return the number of words in the receive FIFO.
- static uint8_t LPI2C_HAL_MasterGetTxFIFOCount (const LPI2C_Type ∗baseAddr)

  Return the number of words in the transmit FIFO.
- static void LPI2C_HAL_MasterTransmitCmd (LPI2C_Type ∗baseAddr, lpi2c_master_command_t cmd, uint8_t data)

  Provide commands and data for the LPI2C master.
- static uint8_t LPI2C_HAL_MasterGetRxData (const LPI2C_Type ∗baseAddr)

  Return the received data.

- static bool LPI2C_HAL_MasterGetRxEmpty (const LPI2C_Type ∗baseAddr)

  Check if the master receive FIFO is empty.
- static void LPI2C_HAL_SlaveSetFilterDoze (LPI2C_Type ∗baseAddr, bool enable)

  Configure the slave filter in doze mode.
- static bool LPI2C_HAL_SlaveGetFilterDoze (const LPI2C_Type ∗baseAddr)

  Return the slave filter configuration in doze mode.
- static void LPI2C_HAL_SlaveSetFilterEnable (LPI2C_Type ∗baseAddr, bool enable)

  Enable/disable the slave filter.
- static bool LPI2C_HAL_SlaveGetFilterEnable (const LPI2C_Type ∗baseAddr)

  Check if the slave filter is enabled or disabled.
- static void LPI2C_HAL_SlaveSetSoftwareReset (LPI2C_Type ∗baseAddr, bool enable)

  Set/clear the slave reset command.
- static bool LPI2C_HAL_SlaveGetSoftwareReset (const LPI2C_Type ∗baseAddr)

  Return the reset setting for the LPI2C slave.
- static void LPI2C_HAL_SlaveSetEnable (LPI2C_Type ∗baseAddr, bool enable)

  Enable or disable the LPI2C slave.
- static bool LPI2C_HAL_SlaveGetEnable (const LPI2C_Type ∗baseAddr)

  Return the enable/disable setting for the LPI2C slave.
- static bool LPI2C_HAL_SlaveGetBusBusyEvent (const LPI2C_Type ∗baseAddr)

  Check the busy state of the bus.
- static bool LPI2C_HAL_SlaveGetSlaveBusyEvent (const LPI2C_Type ∗baseAddr)

  Check the busy state of the slave.
- static bool LPI2C_HAL_SlaveGetSMBusAlertResponseEvent (const LPI2C_Type ∗baseAddr)

  Check the occurrence of an SMBus alert response.
- static bool LPI2C_HAL_SlaveGetGeneralCallEvent (const LPI2C_Type ∗baseAddr)

  Check the detection of the general call address.
- static bool LPI2C_HAL_SlaveGetAddressMatch1Event (const LPI2C_Type ∗baseAddr)

  Check the detection of an ADDR1 address match.
- static bool LPI2C_HAL_SlaveGetAddressMatch0Event (const LPI2C_Type ∗baseAddr)

  Check the detection of an ADDR0 address match.
- static bool LPI2C_HAL_SlaveGetFIFOErrorEvent (const LPI2C_Type ∗baseAddr)

  Check the detection of a FIFO overflow or underflow.
- static bool LPI2C_HAL_SlaveGetBitErrorEvent (const LPI2C_Type ∗baseAddr)

  Check the detection of a bit error.
- static bool LPI2C_HAL_SlaveGetSTOPDetectEvent (const LPI2C_Type ∗baseAddr)

  Check the detection of a STOP condition.
- static bool LPI2C_HAL_SlaveGetRepeatedStartEvent (const LPI2C_Type ∗baseAddr)

  Check the detection of a repeated START condition.
- static bool LPI2C_HAL_SlaveGetTransmitACKEvent (const LPI2C_Type ∗baseAddr)

  Check if transmitting ACK response is required.
- static bool LPI2C_HAL_SlaveGetAddressValidEvent (const LPI2C_Type ∗baseAddr)

  Check the validity of the Address Status Register.
- static bool LPI2C_HAL_SlaveGetReceiveDataEvent (const LPI2C_Type ∗baseAddr)

  Check the availability of receive data.
- static bool LPI2C_HAL_SlaveGetTransmitDataEvent (const LPI2C_Type ∗baseAddr)

  Check if transmit data is requested.
- static void LPI2C_HAL_SlaveClearFIFOErrorEvent (LPI2C_Type ∗baseAddr)

  Clear the FIFO overflow or underflow flag.
- static void LPI2C_HAL_SlaveClearBitErrorEvent (LPI2C_Type ∗baseAddr)

  Clear bit error flag.
- static void LPI2C_HAL_SlaveClearSTOPDetectEvent (LPI2C_Type ∗baseAddr)

Clear the STOP detect flag.

- static void LPI2C_HAL_SlaveClearRepeatedStartEvent (LPI2C_Type ∗baseAddr)

  Clear the repeated START detect flag.
- static void LPI2C_HAL_SlaveSetInt (LPI2C_Type ∗baseAddr, uint32_t interrupts, bool enable)

  Enable or disable specified LPI2C slave interrupts.
- static bool LPI2C_HAL_SlaveGetInt (const LPI2C_Type ∗baseAddr, uint32_t interrupts)

  Return the state of the specified LPI2C slave interrupt.
- static void LPI2C_HAL_SlaveSetAddrDMA (LPI2C_Type ∗baseAddr, bool enable)

  Enable/disable slave address valid DMA requests.
- static void LPI2C_HAL_SlaveSetRxDMA (LPI2C_Type ∗baseAddr, bool enable)

  Enable/disable slave receive data DMA requests.
- static void LPI2C_HAL_SlaveSetTxDMA (LPI2C_Type ∗baseAddr, bool enable)

  Enable/disable slave transmit data DMA requests.
- static bool LPI2C_HAL_SlaveGetAddrDMA (const LPI2C_Type ∗baseAddr)

  Check if slave address valid DMA requests are enabled.
- static bool LPI2C_HAL_SlaveGetRxDMA (const LPI2C_Type ∗baseAddr)

  Check if slave receive data DMA requests are enabled.
- static bool LPI2C_HAL_SlaveGetTxDMA (const LPI2C_Type ∗baseAddr)

  Check if slave transmit data DMA requests are enabled.
- static void LPI2C_HAL_SlaveSetAddrConfig (LPI2C_Type ∗baseAddr, lpi2c_slave_addr_config_t configuration)

  Control address match configuration.
- static lpi2c_slave_addr_config_t LPI2C_HAL_SlaveGetAddrConfig (const LPI2C_Type ∗baseAddr)

  Return the address match configuration.
- static void LPI2C_HAL_SlaveSetHighSpeedModeDetect (LPI2C_Type ∗baseAddr, bool enable)

  Control detection of the High-speed Mode master code.
- static bool LPI2C_HAL_SlaveGetHighSpeedModeDetect (const LPI2C_Type ∗baseAddr)

  Return the state of the High-speed Mode master code detection.
- static void LPI2C_HAL_SlaveSetIgnoreNACK (LPI2C_Type ∗baseAddr, lpi2c_slave_nack_config_t nack_↩ config)

  Control slave behaviour when NACK is detected.
- static lpi2c_slave_nack_config_t LPI2C_HAL_SlaveGetIgnoreNACK (const LPI2C_Type ∗baseAddr)

  Return the configured slave behaviour when NACK is detected.
- static void LPI2C_HAL_SlaveSetRxDataConfig (LPI2C_Type ∗baseAddr, lpi2c_slave_rxdata_config_t configuration)

  Control the functionality of the receive data register.
- static lpi2c_slave_rxdata_config_t LPI2C_HAL_SlaveGetRxDataConfig (const LPI2C_Type ∗baseAddr)

  Return the configured functionality of the receive data register.
- static void LPI2C_HAL_SlaveSetTxFlagConfig (LPI2C_Type ∗baseAddr, lpi2c_slave_txflag_config_t configuration)

  Control the conditions for setting the transmit data flag.
- static lpi2c_slave_txflag_config_t LPI2C_HAL_SlaveGetTxFlagConfig (const LPI2C_Type ∗baseAddr)

  Return the configured settings for the transmit data flag.
- static void LPI2C_HAL_SlaveSetSMBusAlert (LPI2C_Type ∗baseAddr, bool enable)

  Enable or disable match on SMBus Alert.
- static bool LPI2C_HAL_SlaveGetSMBusAlert (const LPI2C_Type ∗baseAddr)

  Return the configured state for the SMBus Alert match.
- static void LPI2C_HAL_SlaveSetGeneralCall (LPI2C_Type ∗baseAddr, bool enable)

  Enable or disable general call address.
- static bool LPI2C_HAL_SlaveGetGeneralCall (const LPI2C_Type ∗baseAddr)

  Return the configured state for the general call address.

- static void LPI2C_HAL_SlaveSetACKStall (LPI2C_Type ∗baseAddr, bool enable)

  Enable or disable clock stretching for the sending of the ACK bit.
- static bool LPI2C_HAL_SlaveGetACKStall (const LPI2C_Type ∗baseAddr)

  Return the configured state for clock stretching for the sending of the ACK bit.
- static void LPI2C_HAL_SlaveSetTXDStall (LPI2C_Type ∗baseAddr, bool enable)

  Enable or disable clock stretching for data transmission.
- static bool LPI2C_HAL_SlaveGetTXDStall (const LPI2C_Type ∗baseAddr)

  Return the configured state for clock stretching for data transmission.
- static void LPI2C_HAL_SlaveSetRXStall (LPI2C_Type ∗baseAddr, bool enable)

  Enable or disable clock stretching for data reception.
- static bool LPI2C_HAL_SlaveGetRXStall (const LPI2C_Type ∗baseAddr)

  Return the configured state for clock stretching for data reception.
- static void LPI2C_HAL_SlaveSetAddrStall (LPI2C_Type ∗baseAddr, bool enable)

  Enable or disable clock stretching for valid address reception.
- static bool LPI2C_HAL_SlaveGetAddrStall (const LPI2C_Type ∗baseAddr)

  Return the configured state for clock stretching for valid address reception.
- static void LPI2C_HAL_SlaveSetSDAGlitchFilter (LPI2C_Type ∗baseAddr, uint8_t cycles)

  Configure the LPI2C slave SDA glitch filter.
- static uint8_t LPI2C_HAL_SlaveGetSDAGlitchFilter (const LPI2C_Type ∗baseAddr)

  Return the LPI2C slave SDA glitch filter configuration.
- static void LPI2C_HAL_SlaveSetSCLGlitchFilter (LPI2C_Type ∗baseAddr, uint8_t cycles)

  Configure the LPI2C slave SCL glitch filter.
- static uint8_t LPI2C_HAL_SlaveGetSCLGlitchFilter (const LPI2C_Type ∗baseAddr)

  Return the LPI2C slave SCL glitch filter configuration.
- static void LPI2C_HAL_SlaveSetDataValidDelay (LPI2C_Type ∗baseAddr, uint8_t cycles)

  Configure the SDA data valid delay time for the I2C slave.
- static uint8_t LPI2C_HAL_SlaveGetDataValidDelay (const LPI2C_Type ∗baseAddr)

  Return the SDA data valid delay time configuration.
- static void LPI2C_HAL_SlaveSetClockHoldTime (LPI2C_Type ∗baseAddr, uint8_t cycles)

  Configure the minimum clock hold time for the I2C slave.
- static uint8_t LPI2C_HAL_SlaveGetClockHoldTime (const LPI2C_Type ∗baseAddr)

  Return the minimum clock hold time configuration.
- static void LPI2C_HAL_SlaveSetAddr1 (LPI2C_Type ∗baseAddr, uint16_t addr)

  Configure the ADDR1 address for slave address match.
- static uint16_t LPI2C_HAL_SlaveGetAddr1 (const LPI2C_Type ∗baseAddr)

  Return the ADDR1 address for slave address match.
- static void LPI2C_HAL_SlaveSetAddr0 (LPI2C_Type ∗baseAddr, uint16_t addr)

  Configure the ADDR0 address for slave address match.
- static uint16_t LPI2C_HAL_SlaveGetAddr0 (const LPI2C_Type ∗baseAddr)

  Return the ADDR0 address for slave address match.
- static lpi2c_slave_addr_valid_t LPI2C_HAL_SlaveGetAddrValid (const LPI2C_Type ∗baseAddr)

  Check the validity of received address.
- static uint16_t LPI2C_HAL_SlaveGetReceivedAddr (const LPI2C_Type ∗baseAddr)

  Return the received slave address.
- static void LPI2C_HAL_SlaveSetTransmitNACK (LPI2C_Type ∗baseAddr, lpi2c_slave_nack_transmit_t nack)

  Configure the ACK/NACK transmission after a received byte.
- static lpi2c_slave_nack_transmit_t LPI2C_HAL_SlaveGetTransmitNACK (const LPI2C_Type ∗baseAddr)

  Return the configured ACK/NACK transmission setting.
- static void LPI2C_HAL_SlaveTransmitData (LPI2C_Type ∗baseAddr, uint8_t data)

  Provide data for the LPI2C slave transmitter.
- static bool LPI2C_HAL_SlaveGetStartOfFrame (const LPI2C_Type ∗baseAddr)

Check if the current received data is the first in the current frame.

- static bool LPI2C_HAL_SlaveGetRXEmpty (const LPI2C_Type ∗baseAddr)

    Check if the receive data register is empty.
- static uint8_t LPI2C_HAL_SlaveGetData (const LPI2C_Type ∗baseAddr)

    Return the data received by the LPI2C slave receiver.
- void LPI2C_HAL_Init (LPI2C_Type ∗baseAddr)

    Initializes the LPI2C module to a known state.

3.48.2 Data Structure Documentation

3.48.2.1 struct lpi2c_version_info_t

LPI2C module version number Implements : lpi2c_version_info_t_Class.

Data Fields

- uint8_t majorNumber
- uint8_t minorNumber
- uint16_t featureNumber

Field Documentation

3.48.2.1.1 featureNumber

```
uint16_t featureNumber
```

Feature Specification Number

3.48.2.1.2 majorNumber

```
uint8_t majorNumber
```

Major Version Number

3.48.2.1.3 minorNumber

```
uint8_t minorNumber
```

Minor Version Number

3.48.3 Macro Definition Documentation

3.48.3.1 LPI2C_HAL_MASTER_ARBITRATION_LOST_INT

```
#define LPI2C_HAL_MASTER_ARBITRATION_LOST_INT 0x800UL
```

Arbitration Lost Interrupt

3.48.3.2 LPI2C_HAL_MASTER_DATA_MATCH_INT

```
#define LPI2C_HAL_MASTER_DATA_MATCH_INT 0x4000UL
```

LPI2C master interruptsData Match Interrupt

3.48.3.3 LPI2C_HAL_MASTER_END_PACKET_INT

```
#define LPI2C_HAL_MASTER_END_PACKET_INT 0x100UL
```

End Packet Interrupt

3.48.3.4 LPI2C_HAL_MASTER_FIFO_ERROR_INT

```
#define LPI2C_HAL_MASTER_FIFO_ERROR_INT 0x1000UL
```

FIFO Error Interrupt

3.48.3.5 LPI2C_HAL_MASTER_NACK_DETECT_INT

```
#define LPI2C_HAL_MASTER_NACK_DETECT_INT 0x400UL
```

NACK Detect Interrupt

3.48.3.6 LPI2C_HAL_MASTER_PIN_LOW_TIMEOUT_INT

```
#define LPI2C_HAL_MASTER_PIN_LOW_TIMEOUT_INT 0x2000UL
```

Pin Low Timeout Interrupt

3.48.3.7 LPI2C_HAL_MASTER_RECEIVE_DATA_INT

```
#define LPI2C_HAL_MASTER_RECEIVE_DATA_INT 0x2UL
```

Receive Data Interrupt

3.48.3.8 LPI2C_HAL_MASTER_STOP_DETECT_INT

```
#define LPI2C_HAL_MASTER_STOP_DETECT_INT 0x200UL
```

STOP Detect Interrupt

3.48.3.9 LPI2C_HAL_MASTER_TRANSMIT_DATA_INT

```
#define LPI2C_HAL_MASTER_TRANSMIT_DATA_INT 0x1UL
```

Transmit Data Interrupt

3.48.3.10 LPI2C_HAL_SLAVE_ADDRESS_MATCH_0_INT

```
#define LPI2C_HAL_SLAVE_ADDRESS_MATCH_0_INT 0x1000UL
```

Address Match 0 Interrupt

3.48.3.11 LPI2C_HAL_SLAVE_ADDRESS_MATCH_1_INT

```
#define LPI2C_HAL_SLAVE_ADDRESS_MATCH_1_INT 0x2000UL
```

Address Match 1 Interrupt

3.48.3.12 LPI2C_HAL_SLAVE_ADDRESS_VALID_INT

```
#define LPI2C_HAL_SLAVE_ADDRESS_VALID_INT 0x4UL
```

Address Valid Interrupt

3.48.3.13 LPI2C_HAL_SLAVE_BIT_ERROR_INT

```
#define LPI2C_HAL_SLAVE_BIT_ERROR_INT 0x400UL
```

Bit Error Interrupt

3.48.3.14 LPI2C_HAL_SLAVE_FIFO_ERROR_INT

```
#define LPI2C_HAL_SLAVE_FIFO_ERROR_INT 0x800UL
```

FIFO Error Interrupt

3.48.3.15 LPI2C_HAL_SLAVE_GENERAL_CALL_INT

```
#define LPI2C_HAL_SLAVE_GENERAL_CALL_INT 0x4000UL
```

General Call Interrupt

3.48.3.16 LPI2C_HAL_SLAVE_RECEIVE_DATA_INT

```
#define LPI2C_HAL_SLAVE_RECEIVE_DATA_INT 0x2UL
```

Receive Data Interrupt

3.48.3.17 LPI2C_HAL_SLAVE_REPEATED_START_INT

```
#define LPI2C_HAL_SLAVE_REPEATED_START_INT 0x100UL
```

Repeated Start Interrupt

3.48.3.18 LPI2C_HAL_SLAVE_SMBUS_ALERT_RESPONSE_INT

```
#define LPI2C_HAL_SLAVE_SMBUS_ALERT_RESPONSE_INT 0x8000UL
```

LPI2C slave interruptsSMBus Alert Response Interrupt

3.48.3.19 LPI2C_HAL_SLAVE_STOP_DETECT_INT

```
#define LPI2C_HAL_SLAVE_STOP_DETECT_INT 0x200UL
```

STOP Detect Interrupt

3.48.3.20 LPI2C_HAL_SLAVE_TRANSMIT_ACK_INT

```
#define LPI2C_HAL_SLAVE_TRANSMIT_ACK_INT 0x8UL
```

Transmit ACK Interrupt

3.48.3.21 LPI2C_HAL_SLAVE_TRANSMIT_DATA_INT

```
#define LPI2C_HAL_SLAVE_TRANSMIT_DATA_INT 0x1UL
```

Transmit Data Interrupt

3.48.4 Enumeration Type Documentation

3.48.4.1 lpi2c_hreq_polarity_t

```
enum lpi2c_hreq_polarity_t
```

Host request input polarity selection Implements : lpi2c_hreq_polarity_t_Class.

Enumerator

LPI2C_HREQ_POL_ACTIVE_HIGH	Host request active low
LPI2C_HREQ_POL_ACTIVE_LOW	Host request active high

3.48.4.2 lpi2c_hreq_source_t

```
enum lpi2c_hreq_source_t
```

Host request input source selection Implements : lpi2c_hreq_source_t_Class.

Enumerator

LPI2C_HREQ_EXTERNAL_PIN	Host request input is external pin
LPI2C_HREQ_INTERNAL_TRIGGER	Host request input is internal trigger

3.48.4.3 lpi2c_master_command_t

```
enum lpi2c_master_command_t
```

LPI2C master commands Implements : lpi2c_master_command_t_Class.

Enumerator

LPI2C_MASTER_COMMAND_TRANSMIT	Transmit DATA[7:0]
LPI2C_MASTER_COMMAND_RECEIVE	Receive (DATA[7:0] + 1) bytes
LPI2C_MASTER_COMMAND_STOP	Generate STOP condition
LPI2C_MASTER_COMMAND_RECEIVE_DISCARD	Receive and discard (DATA[7:0] + 1) bytes
LPI2C_MASTER_COMMAND_START	Generate START and transmit address in DATA[7:0]

Enumerator

LPI2C_MASTER_COMMAND_START_NACK	Generate START and transmit address in DATA[7:0], expect a NACK to be returned
LPI2C_MASTER_COMMAND_START_HS	Generate START and transmit address in DATA[7:0] in high speed mode
LPI2C_MASTER_COMMAND_START_NACK_HS	Generate START and transmit address in DATA[7:0] in high speed mode, expect a NACK to be returned

3.48.4.4 lpi2c_master_prescaler_t

```
enum lpi2c_master_prescaler_t
```

LPI2C master prescaler options Implements : lpi2c_master_prescaler_t_Class.

Enumerator

LPI2C_MASTER_PRESC_DIV_1	Divide by 1
LPI2C_MASTER_PRESC_DIV_2	Divide by 2
LPI2C_MASTER_PRESC_DIV_4	Divide by 4
LPI2C_MASTER_PRESC_DIV_8	Divide by 8
LPI2C_MASTER_PRESC_DIV_16	Divide by 16
LPI2C_MASTER_PRESC_DIV_32	Divide by 32
LPI2C_MASTER_PRESC_DIV_64	Divide by 64
LPI2C_MASTER_PRESC_DIV_128	Divide by 128

3.48.4.5 lpi2c_match_config_t

```
enum lpi2c_match_config_t
```

Data match selection Implements : lpi2c_match_config_t_Class.

Enumerator

LPI2C_MATCH_DISABLED	Match disabled
LPI2C_MATCH_OR_FIRST	Match enabled (1st data word equals MATCH0 OR MATCH1)
LPI2C_MATCH_OR_ANY	Match enabled (any data word equals MATCH0 OR MATCH1)
LPI2C_MATCH_AND_FIRST	Match enabled (1st data word equals MATCH0 AND 2nd data word equals MATCH1)
LPI2C_MATCH_AND_ANY	Match enabled (any data word equals MATCH0 AND next data word equals MATCH1)
LPI2C_MATCH_MASK_FIRST	Match enabled (1st data word AND MATCH1 equals MATCH0 AND MATCH1)
LPI2C_MATCH_MASK_ANY	Match enabled (any data word AND MATCH1 equals MATCH0 AND MATCH1)

3.48.4.6 lpi2c_nack_config_t

```
enum lpi2c_nack_config_t
```

Master NACK reaction configuration Implements : lpi2c_nack_config_t_Class.

Enumerator

LPI2C_NACK_RECEIVE	Receive ACK and NACK normally
LPI2C_NACK_IGNORE	Treat a received NACK as if it was an ACK

3.48.4.7 lpi2c_pin_config_t

```
enum lpi2c_pin_config_t
```

Pin configuration selection Implements : lpi2c_pin_config_t_Class.

Enumerator

LPI2C_CFG_2PIN_OPEN_DRAIN	2-pin open drain mode
LPI2C_CFG_2PIN_OUTPUT_ONLY	2-pin output only mode (ultra-fast mode)
LPI2C_CFG_2PIN_PUSH_PULL	2-pin push-pull mode
LPI2C_CFG_4PIN_PUSH_PULL	4-pin push-pull mode
LPI2C_CFG_2PIN_OPEN_DRAIN_SLAVE	2-pin open drain mode with separate LPI2C slave
LPI2C_CFG_2PIN_OUTPUT_ONLY_SLAVE	2-pin output only mode (ultra-fast mode) with separate LPI2C slave
LPI2C_CFG_2PIN_PUSH_PULL_SLAVE	2-pin push-pull mode with separate LPI2C slave
LPI2C_CFG_4PIN_PUSH_PULL_INVERTED	4-pin push-pull mode (inverted outputs)

3.48.4.8 lpi2c_rx_data_match_t

```
enum lpi2c_rx_data_match_t
```

Non_matching data discard options Implements : lpi2c_rx_data_match_t_Class.

Enumerator

LPI2C_RX_DATA_KEEP_ALL	Received data is stored in the receive FIFO as normal
LPI2C_RX_DATA_DROP_NON_MATCHING	Received data is discarded unless the RMF is set

3.48.4.9 lpi2c_slave_addr_config_t

```
enum lpi2c_slave_addr_config_t
```

Slave address configuration Implements : lpi2c_slave_addr_config_t_Class.

Enumerator

LPI2C_SLAVE_ADDR_MATCH_0_7BIT	Address match 0 (7-bit)
LPI2C_SLAVE_ADDR_MATCH_0_10BIT	Address match 0 (10-bit)
LPI2C_SLAVE_ADDR_MATCH_0_7BIT_OR_1_7BIT	Address match 0 (7-bit) or Address match 1 (7-bit)
LPI2C_SLAVE_ADDR_MATCH_0_10BIT_OR_1_↩ 10BIT	Address match 0 (10-bit) or Address match 1 (10-bit)

Enumerator

LPI2C_SLAVE_ADDR_MATCH_0_7BIT_OR_1_↩ 10BIT	Address match 0 (7-bit) or Address match 1 (10-bit)
LPI2C_SLAVE_ADDR_MATCH_0_10BIT_OR_1_↩ 7BIT	Address match 0 (10-bit) or Address match 1 (7-bit)
LPI2C_SLAVE_ADDR_MATCH_RANGE_7BIT	From Address match 0 (7-bit) to Address match 1 (7-bit)
LPI2C_SLAVE_ADDR_MATCH_RANGE_10BIT	From Address match 0 (10-bit) to Address match 1 (10-bit)

3.48.4.10 lpi2c_slave_addr_valid_t

```
enum lpi2c_slave_addr_valid_t
```

Slave received address validity Implements : lpi2c_slave_addr_valid_t_Class.

Enumerator

LPI2C_SLAVE_ADDR_VALID	RADDR is valid
LPI2C_SLAVE_ADDR_NOT_VALID	RADDR is not valid

3.48.4.11 lpi2c_slave_nack_config_t

```
enum lpi2c_slave_nack_config_t
```

Slave NACK reaction configuration Implements : lpi2c_slave_nack_config_t_Class.

Enumerator

LPI2C_SLAVE_NACK_END_TRANSFER	Slave will end transfer when NACK detected
LPI2C_SLAVE_NACK_CONTINUE_TRANSFER	Slave will not end transfer when NACK detected

3.48.4.12 lpi2c_slave_nack_transmit_t

```
enum lpi2c_slave_nack_transmit_t
```

Slave ACK transmission options Implements : lpi2c_slave_nack_transmit_t_Class.

Enumerator

LPI2C_SLAVE_TRANSMIT_ACK	Transmit ACK for received word
LPI2C_SLAVE_TRANSMIT_NACK	Transmit NACK for received word

3.48.4.13 lpi2c_slave_rxdata_config_t

```
enum lpi2c_slave_rxdata_config_t
```

Slave receive data register function configuration Implements : lpi2c_slave_rxdata_config_t_Class.

Enumerator

LPI2C_SLAVE_RXDATA_DATA_ONLY	Reading the receive data register will return only data
LPI2C_SLAVE_RXDATA_DATA_OR_ADDR	Reading the receive data register can return data or address

3.48.4.14 lpi2c_slave_txflag_config_t

```
enum lpi2c_slave_txflag_config_t
```

Slave Transmit Data Flag function control Implements : lpi2c_slave_txflag_config_t_Class.

Enumerator

LPI2C_SLAVE_TXFLAG_TRANSFER_ONLY	only assert during a slave-transmit transfer
LPI2C_SLAVE_TXFLAG_ALWAYS	assert whenever the transmit data register is empty

3.48.4.15 lpi2c_timeout_config_t

```
enum lpi2c_timeout_config_t
```

SCL/SDA low time-out configuration Implements : lpi2c_timeout_config_t_Class.

Enumerator

LPI2C_TIMEOUT_SCL	Pin Low Timeout Flag will set if SCL is low for longer than the configured timeout
LPI2C_TIMEOUT_SCL_OR_SDA	Pin Low Timeout Flag will set if either SCL or SDA is low for longer than the configured timeout

3.48.5 Function Documentation

3.48.5.1 LPI2C_HAL_GetVersion()

```
static void LPI2C_HAL_GetVersion (
             const LPI2C_Type * baseAddr,
             lpi2c_version_info_t * versionInfo )  [inline], [static]
```

Get the version of the LPI2C module.

This function reads the version number of the LPI2C hardware module

Parameters

baseAddr	base address of the LPI2C module
versionInfo	Device Version Number Implements : LPI2C_HAL_GetVersion_Activity

3.48.5.2 LPI2C_HAL_Init()

```
void LPI2C_HAL_Init (
            LPI2C_Type * baseAddr )
```

Initializes the LPI2C module to a known state.

This function initializes all the registers of the LPI2C module to their reset value.

Parameters

baseAddr	base address of the LPI2C module

3.48.5.3 LPI2C_HAL_MasterClearArbitrationLostEvent()

```
static void LPI2C_HAL_MasterClearArbitrationLostEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear the arbitration lost event flag.

This function clears the arbitration lost event. This event must be cleared before the LPI2C master can initiate a START condition.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_MasterClearArbitrationLostEvent_Activity

3.48.5.4 LPI2C_HAL_MasterClearDataMatchEvent()

```
static void LPI2C_HAL_MasterClearDataMatchEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear the data match event flag.

This function clears the data match event.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_MasterClearDataMatchEvent_Activity

3.48.5.5 LPI2C_HAL_MasterClearEndPacketEvent()

```
static void LPI2C_HAL_MasterClearEndPacketEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear the end packet event flag.

This function clears the end packet event.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_MasterClearEndPacketEvent_Activity

3.48.5.6 LPI2C_HAL_MasterClearFIFOErrorEvent()

```
static void LPI2C_HAL_MasterClearFIFOErrorEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear the FIFO error event flag.

This function clears the FIFO error event. This event must be cleared before the LPI2C master can initiate a START condition.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_MasterClearFIFOErrorEvent_Activity

3.48.5.7 LPI2C_HAL_MasterClearNACKDetectEvent()

```
static void LPI2C_HAL_MasterClearNACKDetectEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear the unexpected NACK event flag.

This function clears the unexpected NACK event. This event must be cleared before the LPI2C master can initiate a START condition.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_MasterClearNACKDetectEvent_Activity

3.48.5.8 LPI2C_HAL_MasterClearPinLowTimeoutEvent()

```
static void LPI2C_HAL_MasterClearPinLowTimeoutEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear the pin low timeout event flag.

This function clears the pin low timeout event. This event cannot be cleared for as long as the pin low timeout condition continues, and must be cleared before the LPI2C master can initiate a START condition.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_MasterClearPinLowTimeoutEvent_Activity

3.48.5.9 LPI2C_HAL_MasterClearSTOPDetectEvent()

```
static void LPI2C_HAL_MasterClearSTOPDetectEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear the STOP detect event flag.

This function clears the STOP detect event.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_MasterClearSTOPDetectEvent_Activity

3.48.5.10 LPI2C_HAL_MasterGetArbitrationLostEvent()

```
static bool LPI2C_HAL_MasterGetArbitrationLostEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the occurrence of an arbitration lost event.

This function returns true if the LPI2C master detects an arbitration lost condition, as defined by the I2C standard. When this flag sets, the LPI2C master will release the bus (go idle) and will not initiate a new START condition until this flag has been cleared.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of an arbitration lost event Implements : LPI2C_HAL_MasterGetArbitrationLostEvent_Activity

3.48.5.11 LPI2C_HAL_MasterGetAutoStopConfig()

```
static bool LPI2C_HAL_MasterGetAutoStopConfig (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the current setting for automatic generation of STOP condition.

This function returns the currently configured setting for automatic generation of STOP condition.

Parameters

baseAddr	base address of the LPI2C module

Returns

current setting for automatic generation of STOP condition Implements : LPI2C_HAL_MasterGetAutoStop↩
Config_Activity

3.48.5.12 LPI2C_HAL_MasterGetBusBusyEvent()

```
static bool LPI2C_HAL_MasterGetBusBusyEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the idle/busy state of the I2C bus.

This function returns true if the I2C bus is busy and false if the bus is idle.

Parameters

baseAddr	base address of the LPI2C module

Returns

the idle/busy state of the I2C bus Implements : LPI2C_HAL_MasterGetBusBusyEvent_Activity

3.48.5.13 LPI2C_HAL_MasterGetBusIdleTimeout()

```
static uint16_t LPI2C_HAL_MasterGetBusIdleTimeout (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the bus idle timeout configuration.

This function returns the currently configured bus idle timeout.

Parameters

baseAddr	base address of the LPI2C module

Returns

number of cycles for the bus idle timeout Implements : LPI2C_HAL_MasterGetBusIdleTimeout_Activity

3.48.5.14 LPI2C_HAL_MasterGetCircularFIFO()

```
static bool LPI2C_HAL_MasterGetCircularFIFO (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the circular FIFO mode for the transmit FIFO.

This function returns the current setting for the circular FIFO feature of the module.

Parameters

baseAddr	base address of the LPI2C module

Returns

circular FIFO mode for the transmit FIFO Implements : LPI2C_HAL_MasterGetCircularFIFO_Activity

3.48.5.15 LPI2C_HAL_MasterGetClockHighPeriod()

```
static uint8_t LPI2C_HAL_MasterGetClockHighPeriod (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured minimum clock high period.

This function returns the currently configured value for clock high period.

Parameters

baseAddr	base address of the LPI2C module

Returns

minimum clock high period Implements : LPI2C_HAL_MasterGetClockHighPeriod_Activity

3.48.5.16 LPI2C_HAL_MasterGetClockHighPeriodHS()

```
static uint8_t LPI2C_HAL_MasterGetClockHighPeriodHS (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured minimum clock high period in high-speed mode.

This function returns the currently configured value for clock high period in high-speed mode.

Parameters

baseAddr	base address of the LPI2C module

Returns

minimum clock high period Implements : LPI2C_HAL_MasterGetClockHighPeriodHS_Activity

3.48.5.17 LPI2C_HAL_MasterGetClockLowPeriod()

```
static uint8_t LPI2C_HAL_MasterGetClockLowPeriod (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured minimum clock low period.

This function returns the currently configured value for clock low period.

Parameters

baseAddr	base address of the LPI2C module

Returns

minimum clock low period Implements : LPI2C_HAL_MasterGetClockLowPeriod_Activity

3.48.5.18 LPI2C_HAL_MasterGetClockLowPeriodHS()

```
static uint8_t LPI2C_HAL_MasterGetClockLowPeriodHS (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured minimum clock low period in high-speed mode.

This function returns the currently configured value for clock low period in high-speed mode.

Parameters

baseAddr	base address of the LPI2C module

Returns

minimum clock low period Implements : LPI2C_HAL_MasterGetClockLowPeriodHS_Activity

3.48.5.19 LPI2C_HAL_MasterGetDataMatchEvent()

```
static bool LPI2C_HAL_MasterGetDataMatchEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the occurrence of a data match event.

This function returns true if there was a match on the received data according to the current data match option. Received data that is discarded due to commands does not cause this flag to set. See function LPI2C_HAL_↩ MasterSetMatchConfig() for more information on data match configuration.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of a data match event Implements : LPI2C_HAL_MasterGetDataMatchEvent_Activity

3.48.5.20 LPI2C_HAL_MasterGetDataValidDelay()

```
static uint8_t LPI2C_HAL_MasterGetDataValidDelay (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured data hold time for SDA.

This function returns the currently configured value for SDA data hold time.

Parameters

baseAddr	base address of the LPI2C module

Returns

value of the data hold time for SDA Implements : LPI2C_HAL_MasterGetDataValidDelay_Activity

3.48.5.21 LPI2C_HAL_MasterGetDataValidDelayHS()

```
static uint8_t LPI2C_HAL_MasterGetDataValidDelayHS (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured data hold time for SDA in high-speed mode.

This function returns the currently configured value for SDA data hold time in high-speed mode.

Parameters

baseAddr	base address of the LPI2C module

Returns

value of the data hold time for SDA Implements : LPI2C_HAL_MasterGetDataValidDelayHS_Activity

3.48.5.22 LPI2C_HAL_MasterGetDebugMode()

```
static bool LPI2C_HAL_MasterGetDebugMode (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the debug mode setting for the LPI2C master.

This function returns the current debug mode setting for the LPI2C master.

Parameters

baseAddr	base address of the LPI2C module

Returns

the debug mode setting for the LPI2C master Implements : LPI2C_HAL_MasterGetDebugMode_Activity

3.48.5.23 LPI2C_HAL_MasterGetDozeMode()

```
static bool LPI2C_HAL_MasterGetDozeMode (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the doze mode setting for the LPI2C master.

This function returns the current doze mode setting for the LPI2C master

Parameters

baseAddr	base address of the LPI2C module

Returns

the doze mode setting for the LPI2C master Implements : LPI2C_HAL_MasterGetDozeMode_Activity

3.48.5.24 LPI2C_HAL_MasterGetEnable()

```
static bool LPI2C_HAL_MasterGetEnable (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the enable/disable setting for the LPI2C master.

This function checks whether or not the LPI2C master is enabled.

Parameters

baseAddr	base address of the LPI2C module

Returns

the enable/disable setting for the LPI2C master Implements : LPI2C_HAL_MasterGetEnable_Activity

3.48.5.25 LPI2C_HAL_MasterGetEndPacketEvent()

```
static bool LPI2C_HAL_MasterGetEndPacketEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the occurrence of an end packet event.

This function returns true if the LPI2C master has generated a repeated START or a STOP condition. It does not return true when the master first generates a START condition.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of an end packet event Implements : LPI2C_HAL_MasterGetEndPacketEvent_Activity

3.48.5.26 LPI2C_HAL_MasterGetFIFOErrorEvent()

```
static bool LPI2C_HAL_MasterGetFIFOErrorEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the occurrence of a FIFO error event.

This function returns true if the LPI2C master detects an attempt to send or receive data without first generating a (repeated) START condition. This can occur if the transmit FIFO underflows when the AUTOSTOP bit is set. When this flag is set, the LPI2C master will send a STOP condition (if busy) and will not initiate a new START condition until this flag has been cleared.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of a FIFO error event Implements : LPI2C_HAL_MasterGetFIFOErrorEvent_Activity

3.48.5.27 LPI2C_HAL_MasterGetHreqEnable()

```
static bool LPI2C_HAL_MasterGetHreqEnable (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the enable/disable state the host request feature.

This function returns true if the host request signal is enabled, and false if it is disabled.

Parameters

baseAddr	base address of the LPI2C module

Returns

enable/disable state the host request feature Implements : LPI2C_HAL_MasterGetHreqEnable_Activity

3.48.5.28 LPI2C_HAL_MasterGetHreqPolarity()

```
static lpi2c_hreq_polarity_t LPI2C_HAL_MasterGetHreqPolarity (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the polarity of the host request input pin.

This function returns the currently configured polarity for the host request input.

Parameters

baseAddr	base address of the LPI2C module

Returns

polarity of the host request input Implements : LPI2C_HAL_MasterGetHreqPolarity_Activity

3.48.5.29 LPI2C_HAL_MasterGetHreqSelect()

```
static lpi2c_hreq_source_t LPI2C_HAL_MasterGetHreqSelect (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the source of the host request input.

This function returns the currently configured source for the host request input.

Parameters

baseAddr	base address of the LPI2C module

Returns

source of the host request input Implements : LPI2C_HAL_MasterGetHreqSelect_Activity

3.48.5.30 LPI2C_HAL_MasterGetInt()

```
static bool LPI2C_HAL_MasterGetInt (
             const LPI2C_Type * baseAddr,
             uint32_t interrupts )  [inline], [static]
```

Return the state of the specified LPI2C master interrupt.

This function returns the enabled/disabled state of the master interrupt source specified by the interrupt parameter.

Parameters

baseAddr	base address of the LPI2C module
interrupts	interrupt for which the check is made; must be one of the following constants: • LPI2C_HAL_MASTER_DATA_MATCH_INT - Data Match Interrupt • LPI2C_HAL_MASTER_PIN_LOW_TIMEOUT_INT - Pin Low Timeout Interrupt • LPI2C_HAL_MASTER_FIFO_ERROR_INT - FIFO Error Interrupt • LPI2C_HAL_MASTER_ARBITRATION_LOST_INT - Arbitration Lost Interrupt • LPI2C_HAL_MASTER_NACK_DETECT_INT - NACK Detect Interrupt • LPI2C_HAL_MASTER_STOP_DETECT_INT - STOP Detect Interrupt • LPI2C_HAL_MASTER_END_PACKET_INT - End Packet Interrupt • LPI2C_HAL_MASTER_RECEIVE_DATA_INT - Receive Data Interrupt • LPI2C_HAL_MASTER_TRANSMIT_DATA_INT - Transmit Data Interrupt

Returns

enable/disable state of specified interrupt Implements : LPI2C_HAL_MasterGetInt_Activity

3.48.5.31 LPI2C_HAL_MasterGetMasterBusyEvent()

```
static bool LPI2C_HAL_MasterGetMasterBusyEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the idle/busy state of the LPI2C master.

This function returns true if the LPI2C master is busy and false if the LPI2C master is idle.

Parameters

baseAddr	base address of the LPI2C module

Returns

the idle/busy state of the LPI2C master Implements : LPI2C_HAL_MasterGetMasterBusyEvent_Activity

3.48.5.32 LPI2C_HAL_MasterGetMatch0()

```
static uint8_t LPI2C_HAL_MasterGetMatch0 (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the MATCH0 value for the data match feature.

This function returns the currently configured value for MATCH0.

Parameters

baseAddr	base address of the LPI2C module

Returns

MATCH0 value Implements : LPI2C_HAL_MasterGetMatch0_Activity

3.48.5.33 LPI2C_HAL_MasterGetMatch1()

```
static uint8_t LPI2C_HAL_MasterGetMatch1 (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the MATCH1 value for the data match feature.

This function returns the currently configured value for MATCH1.

Parameters

baseAddr	base address of the LPI2C module

Returns

MATCH1 value Implements : LPI2C_HAL_MasterGetMatch1_Activity

3.48.5.34 LPI2C_HAL_MasterGetMatchConfig()

```
static lpi2c_match_config_t LPI2C_HAL_MasterGetMatchConfig (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the match mode of the module.

This function returns the currently configured match mode for the module.

Parameters

baseAddr	base address of the LPI2C module

Returns

match mode of the module Implements : LPI2C_HAL_MasterGetMatchConfig_Activity

3.48.5.35 LPI2C_HAL_MasterGetNACKConfig()

```
static lpi2c_nack_config_t LPI2C_HAL_MasterGetNACKConfig (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the reaction of the module on NACK reception.

This function returns the currently configured setting for handling NACK reception.

Parameters

baseAddr	base address of the LPI2C module

Returns

configured reaction of the module on NACK reception Implements : LPI2C_HAL_MasterGetNACKConfig_↩ Activity

3.48.5.36 LPI2C_HAL_MasterGetNACKDetectEvent()

```
static bool LPI2C_HAL_MasterGetNACKDetectEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the occurrence of an unexpected NACK event.

This function returns true if the LPI2C master detects a NACK when transmitting an address or data. If a NACK is expected for a given address (as configured by the command word) then the flag will set if a NACK is not generated. When set, the master will transmit a STOP condition and will not initiate a new START condition until this flag has been cleared.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of an unexpected NACK event Implements : LPI2C_HAL_MasterGetNACKDetectEvent_Activity

3.48.5.37 LPI2C_HAL_MasterGetPinConfig()

```
static lpi2c_pin_config_t LPI2C_HAL_MasterGetPinConfig (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the pin mode of the module.

This function returns the currently configured pin mode for the module.

Parameters

baseAddr	base address of the LPI2C module

Returns

pin mode of the module Implements : LPI2C_HAL_MasterGetPinConfig_Activity

3.48.5.38 LPI2C_HAL_MasterGetPinLowTimeout()

```
static uint32_t LPI2C_HAL_MasterGetPinLowTimeout (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the pin low timeout configuration.

This function returns the currently configured pin low timeout.

Parameters

baseAddr	base address of the LPI2C module

Returns

number of cycles for the pin low timeout Implements : LPI2C_HAL_MasterGetPinLowTimeout_Activity

3.48.5.39 LPI2C_HAL_MasterGetPinLowTimeoutEvent()

```
static bool LPI2C_HAL_MasterGetPinLowTimeoutEvent (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the occurrence of a pin low timeout event.

This function returns true if SCL and/or SDA input is low for more than PINLOW cycles This event can occur even when the LPI2C master is idle.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of a pin low timeout event Implements : LPI2C_HAL_MasterGetPinLowTimeoutEvent_Activity

3.48.5.40 LPI2C_HAL_MasterGetPrescaler()

```
static lpi2c_master_prescaler_t LPI2C_HAL_MasterGetPrescaler (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the LPI2C master prescaler.

This function returns the currently configured clock prescaler.

Parameters

baseAddr	base address of the LPI2C module

Returns

LPI2C master prescaler Implements : LPI2C_HAL_MasterGetPrescaler_Activity

3.48.5.41 LPI2C_HAL_MasterGetReceiveDataReadyEvent()

```
static bool LPI2C_HAL_MasterGetReceiveDataReadyEvent (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Indicate the availability of receive data.

This function returns true when the number of words in the receive FIFO is greater than the receive FIFO watermark. See function LPI2C_HAL_MasterSetRxFIFOWatermark() for configuring the receive FIFO watermark.

Parameters

baseAddr	base address of the LPI2C module

Returns

receive data ready/not ready Implements : LPI2C_HAL_MasterGetReceiveDataReadyEvent_Activity

3.48.5.42 LPI2C_HAL_MasterGetRxData()

```
static uint8_t LPI2C_HAL_MasterGetRxData (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the received data.

This function returns data received by the I2C master that has not been discarded due to data match settings or active command, and increments the FIFO read pointer.

Parameters

baseAddr	base address of the LPI2C module

Returns

data received by the LPI2C master Implements : LPI2C_HAL_MasterGetRxData_Activity

3.48.5.43 LPI2C_HAL_MasterGetRxDataMatch()

```
static lpi2c_rx_data_match_t LPI2C_HAL_MasterGetRxDataMatch (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the current data match discarding policy.

This function returns the currently configured policy for data that does not match the configured criteria

Parameters

baseAddr	base address of the LPI2C module

Returns

the current policy for non-matching data Implements : LPI2C_HAL_MasterGetRxDataMatch_Activity

3.48.5.44 LPI2C_HAL_MasterGetRxDMA()

```
static bool LPI2C_HAL_MasterGetRxDMA (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if receive data DMA requests are enabled.

This function returns true if Rx DMA requests are enabled.

Parameters

baseAddr	base address of the LPI2C module

Returns

enabled/disabled status of receive data DMA requests Implements : LPI2C_HAL_MasterGetRxDMA_Activity

3.48.5.45 LPI2C_HAL_MasterGetRxEmpty()

```
static bool LPI2C_HAL_MasterGetRxEmpty (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if the master receive FIFO is empty.

This function checks if the master receive FIFO is empty.

Parameters

baseAddr	base address of the LPI2C module

Returns

status of the master receive FIFO Implements : LPI2C_HAL_MasterGetRxEmpty_Activity

3.48.5.46 LPI2C_HAL_MasterGetRxFIFOCount()

```
static uint8_t LPI2C_HAL_MasterGetRxFIFOCount (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the number of words in the receive FIFO.

This function returns the number of words currently available in the receive FIFO.

Parameters

baseAddr	base address of the LPI2C module

Returns

the number of words in the receive FIFO Implements : LPI2C_HAL_MasterGetRxFIFOCount_Activity

3.48.5.47 LPI2C_HAL_MasterGetRxFIFOSize()

```
static uint16_t LPI2C_HAL_MasterGetRxFIFOSize (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Get the size of the Master Receive FIFO.

This function returns the size of the Master Receive FIFO, always a power of 2.

Parameters

baseAddr	base address of the LPI2C module

Returns

Master Receive FIFO Size Implements : LPI2C_HAL_MasterGetRxFIFOSize_Activity

3.48.5.48 LPI2C_HAL_MasterGetRxFIFOWatermark()

```
static uint8_t LPI2C_HAL_MasterGetRxFIFOWatermark (
            const LPI2C_Type * baseAddr ) [inline], [static]
```

Return the configured receive FIFO watermark.

This function returns the currently configured value for receive FIFO watermark

Parameters

baseAddr	base address of the LPI2C module

Returns

number of words in the receive FIFO that will cause the receive data flag to be set Implements : LPI2C_HA↵
L_MasterGetRxFIFOWatermark_Activity

3.48.5.49 LPI2C_HAL_MasterGetSCLGlitchFilter()

```
static uint8_t LPI2C_HAL_MasterGetSCLGlitchFilter (
            const LPI2C_Type * baseAddr ) [inline], [static]
```

Return the LPI2C SCL glitch filter configuration.

This function returns the currently configured SCL glitch filter.

Parameters

baseAddr	base address of the LPI2C module

Returns

number of cycles for the LPI2C SCL glitch filter Implements : LPI2C_HAL_MasterGetSCLGlitchFilter_Activity

3.48.5.50 LPI2C_HAL_MasterGetSDAGlitchFilter()

```
static uint8_t LPI2C_HAL_MasterGetSDAGlitchFilter (
            const LPI2C_Type * baseAddr ) [inline], [static]
```

Return the LPI2C SDA glitch filter configuration.

This function returns the currently configured master SDA glitch filter.

Parameters

baseAddr	base address of the LPI2C module

Returns

number of cycles for the LPI2C SDA glitch filter Implements : LPI2C_HAL_MasterGetSDAGlitchFilter_Activity

3.48.5.51 LPI2C_HAL_MasterGetSetupHoldDelay()

```
static uint8_t LPI2C_HAL_MasterGetSetupHoldDelay (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured setup and hold time.

This function returns the currently configured value for setup and hold delay for a START / STOP condition.

Parameters

baseAddr	base address of the LPI2C module

Returns

setup and hold time for a START / STOP condition Implements : LPI2C_HAL_MasterGetSetupHoldDelay_↩ Activity

3.48.5.52 LPI2C_HAL_MasterGetSetupHoldDelayHS()

```
static uint8_t LPI2C_HAL_MasterGetSetupHoldDelayHS (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured setup and hold time in high-speed mode.

This function returns the currently configured value for setup and hold delay for a START / STOP condition in high-speed mode.

Parameters

baseAddr	base address of the LPI2C module

Returns

setup and hold time for a START / STOP condition Implements : LPI2C_HAL_MasterGetSetupHoldDelayH↩ S_Activity

3.48.5.53 LPI2C_HAL_MasterGetSoftwareReset()

```
static bool LPI2C_HAL_MasterGetSoftwareReset (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the reset setting for the LPI2C master.

This function returns the state of the LPI2C master software reset bit.

Parameters

baseAddr	base address of the LPI2C module

Returns

the reset setting for the LPI2C master Implements : LPI2C_HAL_MasterGetSoftwareReset_Activity

3.48.5.54 LPI2C_HAL_MasterGetSTOPDetectEvent()

```
static bool LPI2C_HAL_MasterGetSTOPDetectEvent (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the occurrence of a STOP detect event.

This function returns true if the LPI2C master has generated a STOP condition.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of a STOP detect event Implements : LPI2C_HAL_MasterGetSTOPDetectEvent_Activity

3.48.5.55 LPI2C_HAL_MasterGetTimeoutConfig()

```
static lpi2c_timeout_config_t LPI2C_HAL_MasterGetTimeoutConfig (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the timeout configuration of the module.

This function returns the current pin low timeout configuration for the module.

Parameters

baseAddr	base address of the LPI2C module

Returns

timeout configuration of the module Implements : LPI2C_HAL_MasterGetTimeoutConfig_Activity

3.48.5.56 LPI2C_HAL_MasterGetTransmitDataRequestEvent()

```
static bool LPI2C_HAL_MasterGetTransmitDataRequestEvent (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Indicate if the LPI2C master requests more data.

This function returns true when the number of words in the transmit FIFO is equal or less than the transmit FIFO watermark. See function LPI2C_HAL_MasterSetTxFIFOWatermark() for configuring the transmit FIFO watermark.

Parameters

baseAddr	base address of the LPI2C module

Returns

transmit data requested/not requested Implements : LPI2C_HAL_MasterGetTransmitDataRequestEvent_↩
Activity

3.48.5.57 LPI2C_HAL_MasterGetTxDMA()

```
static bool LPI2C_HAL_MasterGetTxDMA (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if transmit data DMA requests are enabled.

This function returns true if Tx DMA requests are enabled.

Parameters

baseAddr	base address of the LPI2C module

Returns

enabled/disabled status of transmit data DMA requests Implements : LPI2C_HAL_MasterGetTxDMA_Activity

3.48.5.58 LPI2C_HAL_MasterGetTxFIFOCount()

```
static uint8_t LPI2C_HAL_MasterGetTxFIFOCount (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the number of words in the transmit FIFO.

This function returns the number of words currently available in the transmit FIFO.

Parameters

baseAddr	base address of the LPI2C module

Returns

the number of words in the transmit FIFO Implements : LPI2C_HAL_MasterGetTxFIFOCount_Activity

3.48.5.59 LPI2C_HAL_MasterGetTxFIFOSize()

```
static uint16_t LPI2C_HAL_MasterGetTxFIFOSize (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Get the size of the Master Transmit FIFO.

This function returns the size of the Master Transmit FIFO, always a power of 2.

Parameters

baseAddr	base address of the LPI2C module

Returns

Master Transmit FIFO Size Implements : LPI2C_HAL_MasterGetTxFIFOSize_Activity

3.48.5.60 LPI2C_HAL_MasterGetTxFIFOWatermark()

```
static uint8_t LPI2C_HAL_MasterGetTxFIFOWatermark (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured transmit FIFO watermark.

This function returns the currently configured value for transmit FIFO watermark

Parameters

baseAddr	base address of the LPI2C module

Returns

number of words in the transmit FIFO that will cause the transmit data flag to be set Implements : LPI2C_H↩
AL_MasterGetTxFIFOWatermark_Activity

3.48.5.61 LPI2C_HAL_MasterRxFIFOResetCmd()

```
static void LPI2C_HAL_MasterRxFIFOResetCmd (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Reset the master receive FIFO.

This function empties the receive FIFO of the LPI2C master.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_MasterRxFIFOResetCmd_Activity

3.48.5.62 LPI2C_HAL_MasterSetAutoStopConfig()

```
static void LPI2C_HAL_MasterSetAutoStopConfig (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Configure the automatic generation of STOP condition.

This function can enable or disable the automatic generation of STOP condition. When enabled, a STOP condition is generated whenever the LPI2C master is busy and the transmit FIFO is empty. The STOP condition can also be generated using a transmit FIFO command.

Parameters

baseAddr	base address of the LPI2C module
enable	enable/disable automatic generation of STOP condition Implements : LPI2C_HAL_MasterSetAutoStopConfig_Activity

3.48.5.63 LPI2C_HAL_MasterSetBusIdleTimeout()

```
static void LPI2C_HAL_MasterSetBusIdleTimeout (
            LPI2C_Type * baseAddr,
            uint16_t cycles )  [inline], [static]
```

Configure the bus idle timeout.

This function configures the bus idle timeout period in clock cycles. If both SCL and SDA are high for longer than the configured timeout in cycles, then the I2C bus is assumed to be idle and the master can generate a START condition. When set to zero, this feature is disabled.

Parameters

baseAddr	base address of the LPI2C module
cycles	number of cycles for the bus idle timeout Implements : LPI2C_HAL_MasterSetBusIdleTimeout_Activity

3.48.5.64 LPI2C_HAL_MasterSetCircularFIFO()

```
static void LPI2C_HAL_MasterSetCircularFIFO (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Set the circular FIFO mode for the transmit FIFO.

This function enables or disables the circular FIFO feature of the module. When enabled, the transmit FIFO read pointer is saved to a temporary register. The transmit FIFO will be emptied as normal, but once the LPI2C master is idle and the transmit FIFO is empty, then the read pointer value will be restored from the temporary register. This will cause the contents of the transmit FIFO to be cycled through repeatedly. If AUTOSTOP is set, a STOP condition will be sent whenever the transmit FIFO is empty and the read pointer is restored.

Parameters

baseAddr	base address of the LPI2C module
enable	enable/disable circular FIFO mode for the transmit FIFO Implements : LPI2C_HAL_MasterSetCircularFIFO_Activity

3.48.5.65 LPI2C_HAL_MasterSetClockHighPeriod()

```
static void LPI2C_HAL_MasterSetClockHighPeriod (
            LPI2C_Type * baseAddr,
            uint8_t value )  [inline], [static]
```

Set the minimum clock high period.

This function configures the minimum number of cycles (minus one) that the SCL clock is driven high by the master. The SCL high time is extended by the time it takes to detect a rising edge on the external SCL pin. Ignoring any additional board delay due to external loading, this is equal to (2 + FILTSCL) / 2^PRESCALE cycles.

Parameters

baseAddr	base address of the LPI2C module
value	minimum clock high period Implements : LPI2C_HAL_MasterSetClockHighPeriod_Activity

3.48.5.66 LPI2C_HAL_MasterSetClockHighPeriodHS()

```
static void LPI2C_HAL_MasterSetClockHighPeriodHS (
            LPI2C_Type * baseAddr,
            uint8_t value )  [inline], [static]
```

Set the minimum clock high period in high-speed mode.

This function configures the minimum number of cycles (minus one) that the SCL clock is driven high by the master. The SCL high time is extended by the time it takes to detect a rising edge on the external SCL pin. Ignoring any additional board delay due to external loading, this is equal to (2 + FILTSCL) / 2^PRESCALE cycles. This setting only has effect during High-Speed mode transfers.

Parameters

baseAddr	base address of the LPI2C module
value	minimum clock high period Implements : LPI2C_HAL_MasterSetClockHighPeriodHS_Activity

3.48.5.67 LPI2C_HAL_MasterSetClockLowPeriod()

```
static void LPI2C_HAL_MasterSetClockLowPeriod (
            LPI2C_Type * baseAddr,
            uint8_t value )  [inline], [static]
```

Set the minimum clock low period.

This function configures the minimum number of cycles (minus one) that the SCL clock is driven low by the master. This value is also used for the minimum bus free time between a STOP and a START condition.

Parameters

baseAddr	base address of the LPI2C module
value	minimum clock low period Implements : LPI2C_HAL_MasterSetClockLowPeriod_Activity

3.48.5.68 LPI2C_HAL_MasterSetClockLowPeriodHS()

```
static void LPI2C_HAL_MasterSetClockLowPeriodHS (
            LPI2C_Type * baseAddr,
            uint8_t value ) [inline], [static]
```

Set the minimum clock low period in high-speed mode.

This function configures the minimum number of cycles (minus one) that the SCL clock is driven low by the master. This value is also used for the minimum bus free time between a STOP and a START condition. This setting only has effect during High-Speed mode transfers.

Parameters

baseAddr	base address of the LPI2C module
value	minimum clock low period Implements : LPI2C_HAL_MasterSetClockLowPeriodHS_Activity

3.48.5.69 LPI2C_HAL_MasterSetDataValidDelay()

```
static void LPI2C_HAL_MasterSetDataValidDelay (
            LPI2C_Type * baseAddr,
            uint8_t value ) [inline], [static]
```

Set the data hold time for SDA.

This function sets the minimum number of cycles (minus one) that is used as the data hold time for SDA. Must be configured less than the minimum SCL low period.

Parameters

baseAddr	base address of the LPI2C module
value	value of the data hold time for SDA Implements : LPI2C_HAL_MasterSetDataValidDelay_Activity

3.48.5.70 LPI2C_HAL_MasterSetDataValidDelayHS()

```
static void LPI2C_HAL_MasterSetDataValidDelayHS (
            LPI2C_Type * baseAddr,
            uint8_t value ) [inline], [static]
```

Set the data hold time for SDA in high-speed mode.

This function sets the minimum number of cycles (minus one) that is used as the data hold time for SDA in High-↩ Speed mode. Must be configured less than the minimum SCL low period. This setting only has effect during High-Speed mode transfers.

Parameters

baseAddr	base address of the LPI2C module
value	value of the data hold time for SDA Implements : LPI2C_HAL_MasterSetDataValidDelayHS_Activity

3.48.5.71 LPI2C_HAL_MasterSetDebugMode()

```
static void LPI2C_HAL_MasterSetDebugMode (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Set the master behaviour in Debug mode.

This function enables or disables master functionality when the CPU is in debug mode.

Parameters

baseAddr	base address of the LPI2C module
enable	specifies whether to enable or disable the LPI2C master in debug mode Implements : LPI2C_HAL_MasterSetDebugMode_Activity

3.48.5.72 LPI2C_HAL_MasterSetDozeMode()

```
static void LPI2C_HAL_MasterSetDozeMode (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Set the master behaviour in Doze mode.

This function enables or disables master functionality in doze mode.

Parameters

baseAddr	base address of the LPI2C module
enable	specifies whether to enable or disable the LPI2C master in doze mode Implements : LPI2C_HAL_MasterSetDozeMode_Activity

3.48.5.73 LPI2C_HAL_MasterSetEnable()

```
static void LPI2C_HAL_MasterSetEnable (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable or disable the LPI2C master.

This function enables or disables the LPI2C module in master mode. If the module is enabled, the transmit FIFO is not empty and the I2C bus is idle, then the LPI2C master will immediately initiate a transfer on the I2C bus.

Parameters

baseAddr	base address of the LPI2C module
enable	specifies whether to enable or disable the LPI2C master Implements : LPI2C_HAL_MasterSetEnable_Activity

3.48.5.74 LPI2C_HAL_MasterSetHreqEnable()

```
static void LPI2C_HAL_MasterSetHreqEnable (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable/disable the host request feature.

This function enables or disables the host request signal. When enabled, the LPI2C master will only initiate a S↩ TART condition if the host request input is asserted and the bus is idle. A repeated START is not affected by the host request.

Parameters

baseAddr	base address of the LPI2C module
enable	enable/disable the host request feature Implements : LPI2C_HAL_MasterSetHreqEnable_Activity

3.48.5.75 LPI2C_HAL_MasterSetHreqPolarity()

```
static void LPI2C_HAL_MasterSetHreqPolarity (
            LPI2C_Type * baseAddr,
            lpi2c_hreq_polarity_t polarity )  [inline], [static]
```

Set the polarity of the host request input pin.

This function configures the polarity of the host request input pin.

Parameters

baseAddr	base address of the LPI2C module
polarity	selected polarity of the host request input Implements : LPI2C_HAL_MasterSetHreqPolarity_Activity

3.48.5.76 LPI2C_HAL_MasterSetHreqSelect()

```
static void LPI2C_HAL_MasterSetHreqSelect (
            LPI2C_Type * baseAddr,
            lpi2c_hreq_source_t source )  [inline], [static]
```

Set the source of the host request input.

This function selects the source of the host request input.

Parameters

baseAddr	base address of the LPI2C module
source	selected source of the host request input Implements : LPI2C_HAL_MasterSetHreqSelect_Activity

3.48.5.77 LPI2C_HAL_MasterSetInt()

```
static void LPI2C_HAL_MasterSetInt (
            LPI2C_Type * baseAddr,
            uint32_t interrupts,
            bool enable )  [inline], [static]
```

Enable or disable specified LPI2C master interrupts.

This function can enable or disable one or more master interrupt sources specified by the interrupts parameter.

Parameters

baseAddr	base address of the LPI2C module
interrupts	interrupts to be enabled or disabled; must be a bitwise or between one or more of the following constants: • LPI2C_HAL_MASTER_DATA_MATCH_INT - Data Match Interrupt • LPI2C_HAL_MASTER_PIN_LOW_TIMEOUT_INT - Pin Low Timeout Interrupt • LPI2C_HAL_MASTER_FIFO_ERROR_INT - FIFO Error Interrupt • LPI2C_HAL_MASTER_ARBITRATION_LOST_INT - Arbitration Lost Interrupt • LPI2C_HAL_MASTER_NACK_DETECT_INT - NACK Detect Interrupt • LPI2C_HAL_MASTER_STOP_DETECT_INT - STOP Detect Interrupt • LPI2C_HAL_MASTER_END_PACKET_INT - End Packet Interrupt • LPI2C_HAL_MASTER_RECEIVE_DATA_INT - Receive Data Interrupt • LPI2C_HAL_MASTER_TRANSMIT_DATA_INT - Transmit Data Interrupt
enable	specifies whether to enable or disable specified interrupts Implements : LPI2C_HAL_MasterSetInt_Activity

3.48.5.78 LPI2C_HAL_MasterSetMatch0()

```
static void LPI2C_HAL_MasterSetMatch0 (
            LPI2C_Type * baseAddr,
            uint8_t value )  [inline], [static]
```

Set the MATCH0 value for the data match feature.

This function sets the MATCH0 value for comparing against the received data when receive data match is enabled. See function LPI2C_HAL_MasterSetMatchConfig() for details on how this value is used.

Parameters

baseAddr	base address of the LPI2C module
value	MATCH0 value Implements : LPI2C_HAL_MasterSetMatch0_Activity

3.48.5.79 LPI2C_HAL_MasterSetMatch1()

```
static void LPI2C_HAL_MasterSetMatch1 (
            LPI2C_Type * baseAddr,
            uint8_t value )  [inline], [static]
```

Set the MATCH1 value for the data match feature.

This function sets the MATCH1 value for comparing against the received data when receive data match is enabled. See function LPI2C_HAL_MasterSetMatchConfig() for details on how this value is used.

Parameters

baseAddr	base address of the LPI2C module
value	MATCH1 value Implements : LPI2C_HAL_MasterSetMatch1_Activity

3.48.5.80 LPI2C_HAL_MasterSetMatchConfig()

```
static void LPI2C_HAL_MasterSetMatchConfig (
            LPI2C_Type * baseAddr,
            lpi2c_match_config_t configuration )  [inline], [static]
```

Set the match mode of the module.

This function configures the rules for matching incoming data against the two match values, MATCH0 and MATCH1. A successful match will trigger a data match event. See type lpi2c_match_config_t for a description of all available match options.

Parameters

baseAddr	base address of the LPI2C module
configuration	match mode of the module Implements : LPI2C_HAL_MasterSetMatchConfig_Activity

3.48.5.81 LPI2C_HAL_MasterSetNACKConfig()

```
static void LPI2C_HAL_MasterSetNACKConfig (
            LPI2C_Type * baseAddr,
            lpi2c_nack_config_t configuration )  [inline], [static]
```

Configure the reaction of the module on NACK reception.

This function configures how the LPI2C master reacts when receiving a NACK. NACK responses can be treated normally or ignored. In Ultra-Fast mode it is necessary to configure the module to ignore NACK responses.

Parameters

baseAddr	base address of the LPI2C module
configuration	set reaction of the module on NACK reception Implements : LPI2C_HAL_MasterSetNACKConfig_Activity

3.48.5.82 LPI2C_HAL_MasterSetPinConfig()

```
static void LPI2C_HAL_MasterSetPinConfig (
            LPI2C_Type * baseAddr,
            lpi2c_pin_config_t configuration )  [inline], [static]
```

Set the pin mode of the module.

This function sets the pin mode of the module. See type lpi2c_pin_config_t for a description of available modes.

Parameters

baseAddr	base address of the LPI2C module
configuration	pin mode of the module Implements : LPI2C_HAL_MasterSetPinConfig_Activity

3.48.5.83 LPI2C_HAL_MasterSetPinLowTimeout()

```
static void LPI2C_HAL_MasterSetPinLowTimeout (
            LPI2C_Type * baseAddr,
            uint32_t cycles )  [inline], [static]
```

Configure the pin low timeout.

This function configures the pin low timeout period in clock cycles. If SCL and/or SDA is low for longer than the pin low timeout cycles then PLTF is set. Timeout value will be truncated to a multiple of 256. When set to zero, this feature is disabled.

Parameters

baseAddr	base address of the LPI2C module
cycles	number of cycles for the pin low timeout; will be truncated to a multiple of 256 Implements : LPI2C_HAL_MasterSetPinLowTimeout_Activity

3.48.5.84 LPI2C_HAL_MasterSetPrescaler()

```
static void LPI2C_HAL_MasterSetPrescaler (
            LPI2C_Type * baseAddr,
            lpi2c_master_prescaler_t prescaler )  [inline], [static]
```

Configure the LPI2C master prescaler.

This function configures the clock prescaler used for all LPI2C master logic, except the digital glitch filters.

Parameters

baseAddr	base address of the LPI2C module
prescaler	LPI2C master prescaler Implements : LPI2C_HAL_MasterSetPrescaler_Activity

3.48.5.85 LPI2C_HAL_MasterSetRxDataMatch()

```
static void LPI2C_HAL_MasterSetRxDataMatch (
            LPI2C_Type * baseAddr,
            lpi2c_rx_data_match_t rxDataMatch )  [inline], [static]
```

Control the discarding of data that does not match the configured criteria.

This function configures the policy for handling data that does not match the configuration criteria. Received data can be stored in the receive FIFO either unconditionally or based on the data match event. See function LPI2C_↩ HAL_MasterGetDataMatchEvent() for more information on the data match event.

Parameters

baseAddr	base address of the LPI2C module
rxDataMatch	specifies whether or not to drop non-matching data Implements : LPI2C_HAL_MasterSetRxDataMatch_Activity

3.48.5.86 LPI2C_HAL_MasterSetRxDMA()

```
static void LPI2C_HAL_MasterSetRxDMA (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable/disable receive data DMA requests.

This function enables or disables generation of Rx DMA requests when data can be read from the receive FIFO, as configured by the receive FIFO watermark.

Parameters

baseAddr	base address of the LPI2C module
enable	specifies whether to enable or disable DMA requests Implements : LPI2C_HAL_MasterSetRxDMA_Activity

3.48.5.87 LPI2C_HAL_MasterSetRxFIFOWatermark()

```
static void LPI2C_HAL_MasterSetRxFIFOWatermark (
            LPI2C_Type * baseAddr,
            uint8_t value )  [inline], [static]
```

Set the receive FIFO watermark.

This function configures the receive FIFO watermark. Whenever the number of words in the receive FIFO is greater than the receive FIFO watermark, a receive data ready event is generated. Writing a value equal or greater than the FIFO size will be truncated.

Parameters

baseAddr	base address of the LPI2C module
value	number of words in the receive FIFO that will cause the receive data flag to be set Implements : LPI2C_HAL_MasterSetRxFIFOWatermark_Activity

3.48.5.88 LPI2C_HAL_MasterSetSCLGlitchFilter()

```
static void LPI2C_HAL_MasterSetSCLGlitchFilter (
            LPI2C_Type * baseAddr,
            uint8_t cycles )  [inline], [static]
```

Configure the LPI2C SCL glitch filter.

This function configures the I2C master digital glitch filters for SCL input, a configuration of 0 will disable the glitch filter. Glitches equal to or less than FILTSCL cycles long will be filtered out and ignored. The latency through the glitch filter is equal to FILTSCL cycles and must be configured less than the minimum SCL low or high period.

Parameters

baseAddr	base address of the LPI2C module
cycles	number of cycles for the LPI2C SCL glitch filter Implements : LPI2C_HAL_MasterSetSCLGlitchFilter_Activity

3.48.5.89 LPI2C_HAL_MasterSetSDAGlitchFilter()

```
static void LPI2C_HAL_MasterSetSDAGlitchFilter (
            LPI2C_Type * baseAddr,
            uint8_t cycles )  [inline], [static]
```

Configure the LPI2C SDA glitch filter.

This function configures the I2C master digital glitch filters for SDA input. A configuration of 0 will disable the glitch filter. Glitches equal to or less than FILTSDA cycles long will be filtered out and ignored. The latency through the glitch filter is equal to FILTSDA cycles and must be configured less than the minimum SCL low or high period.

Parameters

baseAddr	base address of the LPI2C module
cycles	number of cycles for the LPI2C SDA glitch filter Implements : LPI2C_HAL_MasterSetSDAGlitchFilter_Activity

3.48.5.90 LPI2C_HAL_MasterSetSetupHoldDelay()

```
static void LPI2C_HAL_MasterSetSetupHoldDelay (
            LPI2C_Type * baseAddr,
            uint8_t value )  [inline], [static]
```

Set the setup and hold delay for a START / STOP condition.

This function configures the Minimum number of cycles (minus one) that is used by the master as the setup and hold time for a (repeated) START condition and setup time for a STOP condition. The setup time is extended by the time it takes to detect a rising edge on the external SCL pin. Ignoring any additional board delay due to external loading, this is equal to (2 + FILTSCL) / 2^PRESCALE cycles.

Parameters

baseAddr	base address of the LPI2C module
value	setup and hold time for a START / STOP condition Implements : LPI2C_HAL_MasterSetSetupHoldDelay_Activity

3.48.5.91 LPI2C_HAL_MasterSetSetupHoldDelayHS()

```
static void LPI2C_HAL_MasterSetSetupHoldDelayHS (
            LPI2C_Type * baseAddr,
            uint8_t value )  [inline], [static]
```

Set the setup and hold time for a START / STOP condition in high-speed mode.

This function configures the Minimum number of cycles (minus one) that is used by the master as the setup and hold time for a (repeated) START condition and setup time for a STOP condition. The setup time is extended by the time it takes to detect a rising edge on the external SCL pin. Ignoring any additional board delay due to external loading, this is equal to (2 + FILTSCL) / 2^PRESCALE cycles. This setting only has effect during High-Speed mode transfers.

Parameters

baseAddr	base address of the LPI2C module
value	setup and hold time for a START / STOP condition Implements : LPI2C_HAL_MasterSetSetupHoldDelayHS_Activity

3.48.5.92 LPI2C_HAL_MasterSetSoftwareReset()

```
static void LPI2C_HAL_MasterSetSoftwareReset (
             LPI2C_Type * baseAddr,
             bool enable )  [inline], [static]
```

Set/clear the master reset command.

Calling this function with enable parameter set to true resets all internal master logic and registers, except the Master Control Register. The reset state persists until this function is called with enable parameter set to false.

Parameters

baseAddr	base address of the LPI2C module
enable	specifies the reset state of the LPI2C master logic Implements : LPI2C_HAL_MasterSetSoftwareReset_Activity

3.48.5.93 LPI2C_HAL_MasterSetTimeoutConfig()

```
static void LPI2C_HAL_MasterSetTimeoutConfig (
             LPI2C_Type * baseAddr,
             lpi2c_timeout_config_t configuration )  [inline], [static]
```

Set the timeout configuration of the module.

This function configures the condition for triggering a pin low timeout event. Selects between monitoring only SCL for timeout, or both SCL and SDA.

Parameters

baseAddr	base address of the LPI2C module
configuration	timeout configuration of the module Implements : LPI2C_HAL_MasterSetTimeoutConfig_Activity

3.48.5.94 LPI2C_HAL_MasterSetTxDMA()

```
static void LPI2C_HAL_MasterSetTxDMA (
             LPI2C_Type * baseAddr,
             bool enable )  [inline], [static]
```

Enable/disable transmit data DMA requests.

This function enables or disables generation of Tx DMA requests when data can be written to the transmit FIFO, as configured by the transmit FIFO watermark.

Parameters

baseAddr	base address of the LPI2C module
enable	specifies whether to enable or disable DMA requests Implements : LPI2C_HAL_MasterSetTxDMA_Activity

3.48.5.95 LPI2C_HAL_MasterSetTxFIFOWatermark()

```
static void LPI2C_HAL_MasterSetTxFIFOWatermark (
            LPI2C_Type * baseAddr,
            uint8_t value )  [inline], [static]
```

Set the transmit FIFO watermark.

This function configures the transmit FIFO watermark. Whenever the number of words in the transmit FIFO is greater than the transmit FIFO watermark, a transmit data request event is generated. Writing a value equal or greater than the FIFO size will be truncated.

Parameters

baseAddr	base address of the LPI2C module
value	number of words in the transmit FIFO that will cause the transmit data flag to be set Implements : LPI2C_HAL_MasterSetTxFIFOWatermark_Activity

3.48.5.96 LPI2C_HAL_MasterTransmitCmd()

```
static void LPI2C_HAL_MasterTransmitCmd (
            LPI2C_Type * baseAddr,
            lpi2c_master_command_t cmd,
            uint8_t data )  [inline], [static]
```

Provide commands and data for the LPI2C master.

This function stores commands and data in the transmit FIFO and increments the FIFO write pointer.

Parameters

baseAddr	base address of the LPI2C module
cmd	command for the LPI2C master
data	data for the LPI2C master Implements : LPI2C_HAL_MasterTransmitCmd_Activity

3.48.5.97 LPI2C_HAL_MasterTxFIFOResetCmd()

```
static void LPI2C_HAL_MasterTxFIFOResetCmd (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Reset the master transmit FIFO.

This function empties the transmit FIFO of the LPI2C master.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_MasterTxFIFOResetCmd_Activity

3.48.5.98 LPI2C_HAL_SlaveClearBitErrorEvent()

```
static void LPI2C_HAL_SlaveClearBitErrorEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear bit error flag.

This function clears the bit error event.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_SlaveClearBitErrorEvent_Activity

3.48.5.99 LPI2C_HAL_SlaveClearFIFOErrorEvent()

```
static void LPI2C_HAL_SlaveClearFIFOErrorEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear the FIFO overflow or underflow flag.

This function clears the FIFO overflow or underflow event.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_SlaveClearFIFOErrorEvent_Activity

3.48.5.100 LPI2C_HAL_SlaveClearRepeatedStartEvent()

```
static void LPI2C_HAL_SlaveClearRepeatedStartEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear the repeated START detect flag.

This function clears the repeated START detect event.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_SlaveClearRepeatedStartEvent_Activity

3.48.5.101 LPI2C_HAL_SlaveClearSTOPDetectEvent()

```
static void LPI2C_HAL_SlaveClearSTOPDetectEvent (
            LPI2C_Type * baseAddr )  [inline], [static]
```

Clear the STOP detect flag.

This function clears the STOP detect event.

Parameters

baseAddr	base address of the LPI2C module Implements : LPI2C_HAL_SlaveClearSTOPDetectEvent_Activity

3.48.5.102 LPI2C_HAL_SlaveGetACKStall()

```
static bool LPI2C_HAL_SlaveGetACKStall (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured state for clock stretching for the sending of the ACK bit.

This function returns the currently configured setting for clock stretching before the ACK bit.

Parameters

baseAddr	base address of the LPI2C module

Returns

indicates if clock stretching is enabled or disabled Implements : LPI2C_HAL_SlaveGetACKStall_Activity

3.48.5.103 LPI2C_HAL_SlaveGetAddr0()

```
static uint16_t LPI2C_HAL_SlaveGetAddr0 (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the ADDR0 address for slave address match.

This function returns the currently configured value for ADDR0.

Parameters

baseAddr	base address of the LPI2C module

Returns

ADDR0 address for slave address match Implements : LPI2C_HAL_SlaveGetAddr0_Activity

3.48.5.104 LPI2C_HAL_SlaveGetAddr1()

```
static uint16_t LPI2C_HAL_SlaveGetAddr1 (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the ADDR1 address for slave address match.

This function returns the currently configured value for ADDR1.

Parameters

baseAddr	base address of the LPI2C module

Returns

ADDR1 address for slave address match Implements : LPI2C_HAL_SlaveGetAddr1_Activity

3.48.5.105 LPI2C_HAL_SlaveGetAddrConfig()

```
static lpi2c_slave_addr_config_t LPI2C_HAL_SlaveGetAddrConfig (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the address match configuration.

This function returns the currently configured option for address match.

Parameters

baseAddr	base address of the LPI2C module

Returns

address match configuration Implements : LPI2C_HAL_SlaveGetAddrConfig_Activity

3.48.5.106 LPI2C_HAL_SlaveGetAddrDMA()

```
static bool LPI2C_HAL_SlaveGetAddrDMA (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if slave address valid DMA requests are enabled.

This function returns true if address valid DMA requests are enabled.

Parameters

baseAddr	base address of the LPI2C module

Returns

enabled/disabled status of address valid DMA requests Implements : LPI2C_HAL_SlaveGetAddrDMA_Activity

3.48.5.107 LPI2C_HAL_SlaveGetAddressMatch0Event()

```
static bool LPI2C_HAL_SlaveGetAddressMatch0Event (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the detection of an ADDR0 address match.

This function checks for a match of the ADDR0 address, as configured by function LPI2C_HAL_SlaveSetAddr↩Config() This event is cleared by reading the received address - see function LPI2C_HAL_SlaveGetReceivedAddr().

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of an ADDR0 address match Implements : LPI2C_HAL_SlaveGetAddressMatch0Event_Activity

3.48.5.108 LPI2C_HAL_SlaveGetAddressMatch1Event()

```
static bool LPI2C_HAL_SlaveGetAddressMatch1Event (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the detection of an ADDR1 address match.

This function checks for a match of the ADDR1 address, or ADDR0 to ADDR1 range as configured by function LPI2C_HAL_SlaveSetAddrConfig() This event is cleared by reading the received address - see function LPI2C_↩ HAL_SlaveGetReceivedAddr().

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of an ADDR1 address match Implements : LPI2C_HAL_SlaveGetAddressMatch1Event_Activity

3.48.5.109 LPI2C_HAL_SlaveGetAddressValidEvent()

```
static bool LPI2C_HAL_SlaveGetAddressValidEvent (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the validity of the Address Status Register.

This function checks for the detection of a valid address. The event is cleared by reading the address - see function LPI2C_HAL_SlaveGetReceivedAddr(). It can also be cleared by reading the data register, when data register has been configured to allow address reads - see functions LPI2C_HAL_SlaveSetRxDataConfig() and LPI2C_HAL_↩ SlaveGetData()

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of the validity of the Address Status Register Implements : LPI2C_HAL_SlaveGetAddressValid↩ Event_Activity

3.48.5.110 LPI2C_HAL_SlaveGetAddrStall()

```
static bool LPI2C_HAL_SlaveGetAddrStall (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured state for clock stretching for valid address reception.

This function returns the currently configured setting for address valid clock stretching.

Parameters

baseAddr	base address of the LPI2C module

Returns

indicates if clock stretching is enabled or disabled Implements : LPI2C_HAL_SlaveGetAddrStall_Activity

3.48.5.111 LPI2C_HAL_SlaveGetAddrValid()

```
static lpi2c_slave_addr_valid_t LPI2C_HAL_SlaveGetAddrValid (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the validity of received address.

This function checks whether the received address register contains a valid address.

Parameters

baseAddr	base address of the LPI2C module

Returns

validity of received address Implements : LPI2C_HAL_SlaveGetAddrValid_Activity

3.48.5.112 LPI2C_HAL_SlaveGetBitErrorEvent()

```
static bool LPI2C_HAL_SlaveGetBitErrorEvent (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the detection of a bit error.

This function checks for the occurrence of a bit error event. This event occurs if the LPI2C slave transmits a logic one and detects a logic zero on the I2C bus. The slave will ignore the rest of the transfer until the next (repeated) START condition.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of a bit error Implements : LPI2C_HAL_SlaveGetBitErrorEvent_Activity

3.48.5.113 LPI2C_HAL_SlaveGetBusBusyEvent()

```
static bool LPI2C_HAL_SlaveGetBusBusyEvent (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the busy state of the bus.

This function returns true if the I2C bus is busy and false if the bus is idle.

Parameters

baseAddr	base address of the LPI2C module

Returns

busy state of the bus Implements : LPI2C_HAL_SlaveGetBusBusyEvent_Activity

3.48.5.114 LPI2C_HAL_SlaveGetClockHoldTime()

```
static uint8_t LPI2C_HAL_SlaveGetClockHoldTime (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the minimum clock hold time configuration.

This function returns the currently configured slave clock hold time.

Parameters

baseAddr	base address of the LPI2C module

Returns

number of cycles for the minimum clock hold time for the I2C slave Implements : LPI2C_HAL_SlaveGet↩ ClockHoldTime_Activity

3.48.5.115 LPI2C_HAL_SlaveGetData()

```
static uint8_t LPI2C_HAL_SlaveGetData (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the data received by the LPI2C slave receiver.

This function returns the data received by the I2C slave. Calling this function clears the receive data event.

Parameters

baseAddr	base address of the LPI2C module

Returns

data received by the LPI2C slave receiver Implements : LPI2C_HAL_SlaveGetData_Activity

3.48.5.116 LPI2C_HAL_SlaveGetDataValidDelay()

```
static uint8_t LPI2C_HAL_SlaveGetDataValidDelay (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the SDA data valid delay time configuration.

This function returns the currently configured slave data valid delay.

Parameters

baseAddr	base address of the LPI2C module

Returns

number of cycles for the SDA data valid delay time for the I2C slave Implements : LPI2C_HAL_SlaveGet↩ DataValidDelay_Activity

3.48.5.117 LPI2C_HAL_SlaveGetEnable()

```
static bool LPI2C_HAL_SlaveGetEnable (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the enable/disable setting for the LPI2C slave.

This function checks whether or not the LPI2C slave is enabled.

Parameters

baseAddr	base address of the LPI2C module

Returns

the enable/disable setting for the LPI2C slave Implements : LPI2C_HAL_SlaveGetEnable_Activity

3.48.5.118 LPI2C_HAL_SlaveGetFIFOErrorEvent()

```
static bool LPI2C_HAL_SlaveGetFIFOErrorEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the detection of a FIFO overflow or underflow.

This function checks for the occurrence of a slave FIFO overflow or underflow. This event can only occur if clock stretching is disabled.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of a FIFO overflow or underflow Implements : LPI2C_HAL_SlaveGetFIFOErrorEvent_Activity

3.48.5.119 LPI2C_HAL_SlaveGetFilterDoze()

```
static bool LPI2C_HAL_SlaveGetFilterDoze (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the slave filter configuration in doze mode.

This function returns the currently configured settings for the digital filter in Doze mode.

Parameters

baseAddr	base address of the LPI2C module

Returns

slave filter configuration in doze mode Implements : LPI2C_HAL_SlaveGetFilterDoze_Activity

3.48.5.120 LPI2C_HAL_SlaveGetFilterEnable()

```
static bool LPI2C_HAL_SlaveGetFilterEnable (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if the slave filter is enabled or disabled.

This function returns the currently configured settings for the digital filter.

Parameters

baseAddr	base address of the LPI2C module

Returns

slave filter configuration Implements : LPI2C_HAL_SlaveGetFilterEnable_Activity

3.48.5.121 LPI2C_HAL_SlaveGetGeneralCall()

```
static bool LPI2C_HAL_SlaveGetGeneralCall (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured state for the general call address.

This function returns the currently configured setting for general call address matching.

Parameters

baseAddr	base address of the LPI2C module

Returns

specifies if general call address is enabled or disabled Implements : LPI2C_HAL_SlaveGetGeneralCall_↩ Activity

3.48.5.122 LPI2C_HAL_SlaveGetGeneralCallEvent()

```
static bool LPI2C_HAL_SlaveGetGeneralCallEvent (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the detection of the general call address.

This function checks for the detection of a General Call Address. This event is cleared by reading the received address - see function LPI2C_HAL_SlaveGetReceivedAddr().

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of a general call address detection Implements : LPI2C_HAL_SlaveGetGeneralCallEvent_Activity

3.48.5.123 LPI2C_HAL_SlaveGetHighSpeedModeDetect()

```
static bool LPI2C_HAL_SlaveGetHighSpeedModeDetect (
             const LPI2C_Type * baseAddr ) [inline], [static]
```

Return the state of the High-speed Mode master code detection.

This function returns the currently configured option for master code detection.

Parameters

baseAddr	base address of the LPI2C module

Returns

enabled/disabled state of the High-speed Mode master code detection Implements : LPI2C_HAL_SlaveGet↩ HighSpeedModeDetect_Activity

3.48.5.124 LPI2C_HAL_SlaveGetIgnoreNACK()

```
static lpi2c_slave_nack_config_t LPI2C_HAL_SlaveGetIgnoreNACK (
             const LPI2C_Type * baseAddr ) [inline], [static]
```

Return the configured slave behaviour when NACK is detected.

This function returns the currently configured option for handling NACKs.

Parameters

baseAddr	base address of the LPI2C module

Returns

configured slave behaviour when NACK is detected Implements : LPI2C_HAL_SlaveGetIgnoreNACK_Activity

3.48.5.125 LPI2C_HAL_SlaveGetInt()

```
static bool LPI2C_HAL_SlaveGetInt (
             const LPI2C_Type * baseAddr,
             uint32_t interrupts ) [inline], [static]
```

Return the state of the specified LPI2C slave interrupt.

This function returns the enabled/disabled state of the slave interrupt source specified by the interrupt parameter.

Parameters

baseAddr	base address of the LPI2C module

Parameters

interrupts	interrupt for which the check is made; must be one of the following constants: • LPI2C_HAL_SLAVE_SMBUS_ALERT_RESPONSE - SMBus Alert Response Interrupt • LPI2C_HAL_SLAVE_GENERAL_CALL - General Call Interrupt • LPI2C_HAL_SLAVE_ADDRESS_MATCH_1 - Address Match 1 Interrupt • LPI2C_HAL_SLAVE_ADDRESS_MATCH_0 - Address Match 0 Interrupt • LPI2C_HAL_SLAVE_FIFO_ERROR - FIFO Error Interrupt • LPI2C_HAL_SLAVE_BIT_ERROR - Bit Error Interrupt • LPI2C_HAL_SLAVE_STOP_DETECT - STOP Detect Interrupt • LPI2C_HAL_SLAVE_REPEATED_START - Repeated Start Interrupt • LPI2C_HAL_SLAVE_TRANSMIT_ACK - Transmit ACK Interrupt • LPI2C_HAL_SLAVE_ADDRESS_VALID - Address Valid Interrupt • LPI2C_HAL_SLAVE_RECEIVE_DATA - Receive Data Interrupt • LPI2C_HAL_SLAVE_TRANSMIT_DATA - Transmit Data Interrupt

Returns

enable/disable state of specified interrupt Implements : LPI2C_HAL_SlaveGetInt_Activity

3.48.5.126 LPI2C_HAL_SlaveGetReceivedAddr()

```
static uint16_t LPI2C_HAL_SlaveGetReceivedAddr (
            const LPI2C_Type * baseAddr ) [inline], [static]
```

Return the received slave address.

This function returns the received slave address. Reading the address clears the address valid event. The address can be 7-bit or 10-bit (10-bit addresses are prefixed by 11110) and includes the R/W bit in the least significant position. Use function LPI2C_HAL_SlaveGetAddrValid() before calling this function to ensure a valid value will be read.

Parameters

baseAddr	base address of the LPI2C module

Returns

received address Implements : LPI2C_HAL_SlaveGetReceivedAddr_Activity

3.48.5.127 LPI2C_HAL_SlaveGetReceiveDataEvent()

```
static bool LPI2C_HAL_SlaveGetReceiveDataEvent (
            const LPI2C_Type * baseAddr ) [inline], [static]
```

Check the availability of receive data.

This function checks for the availability of data received by the I2C slave. The event is cleared by reading the received data - see function LPI2C_HAL_SlaveGetData(). The event is not cleared by calling LPI2C_HAL_Slave←GetData() if the data register is configured to allow address reads and an address valid event is active.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of receive data availability Implements : LPI2C_HAL_SlaveGetReceiveDataEvent_Activity

3.48.5.128 LPI2C_HAL_SlaveGetRepeatedStartEvent()

```
static bool LPI2C_HAL_SlaveGetRepeatedStartEvent (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the detection of a repeated START condition.

This function checks for the detection of a repeated START condition, after the LPI2C slave matched the last address byte. This event does not occur when the slave first detects a START condition.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of a repeated START condition Implements : LPI2C_HAL_SlaveGetRepeatedStartEvent_Activity

3.48.5.129 LPI2C_HAL_SlaveGetRxDataConfig()

```
static lpi2c_slave_rxdata_config_t LPI2C_HAL_SlaveGetRxDataConfig (
             const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured functionality of the receive data register.

This function returns the current configuration for the receive data register.

Parameters

baseAddr	base address of the LPI2C module

Returns

specifies if address can be read from the receive data register Implements : LPI2C_HAL_SlaveGetRxData←Config_Activity

3.48.5.130 LPI2C_HAL_SlaveGetRxDMA()

```
static bool LPI2C_HAL_SlaveGetRxDMA (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if slave receive data DMA requests are enabled.

This function returns true if receive data DMA requests are enabled.

Parameters

baseAddr	base address of the LPI2C module

Returns

enabled/disabled status of receive data DMA requests Implements : LPI2C_HAL_SlaveGetRxDMA_Activity

3.48.5.131 LPI2C_HAL_SlaveGetRXEmpty()

```
static bool LPI2C_HAL_SlaveGetRXEmpty (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if the receive data register is empty.

This function checks if the received data register is empty.

Parameters

baseAddr	base address of the LPI2C module

Returns

specifies if the receive data register is empty Implements : LPI2C_HAL_SlaveGetRXEmpty_Activity

3.48.5.132 LPI2C_HAL_SlaveGetRXStall()

```
static bool LPI2C_HAL_SlaveGetRXStall (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured state for clock stretching for data reception.

This function returns the currently configured setting for data reception clock stretching.

Parameters

baseAddr	base address of the LPI2C module

Returns

indicates if clock stretching is enabled or disabled Implements : LPI2C_HAL_SlaveGetRXStall_Activity

3.48.5.133 LPI2C_HAL_SlaveGetSCLGlitchFilter()

```
static uint8_t LPI2C_HAL_SlaveGetSCLGlitchFilter (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the LPI2C slave SCL glitch filter configuration.

This function returns the currently configured slave SDA glitch filter.

Parameters

baseAddr	base address of the LPI2C module

Returns

number of cycles for the LPI2C slave SCL glitch filter Implements : LPI2C_HAL_SlaveGetSCLGlitchFilter_↩ Activity

3.48.5.134 LPI2C_HAL_SlaveGetSDAGlitchFilter()

```
static uint8_t LPI2C_HAL_SlaveGetSDAGlitchFilter (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the LPI2C slave SDA glitch filter configuration.

This function returns the currently configured slave SDA glitch filter.

Parameters

baseAddr	base address of the LPI2C module

Returns

number of cycles for the LPI2C slave SDA glitch filter Implements : LPI2C_HAL_SlaveGetSDAGlitchFilter_↩ Activity

3.48.5.135 LPI2C_HAL_SlaveGetSlaveBusyEvent()

```
static bool LPI2C_HAL_SlaveGetSlaveBusyEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the busy state of the slave.

This function returns true if the LPI2C slave is busy and false if the LPI2C slave is idle.

Parameters

baseAddr	base address of the LPI2C module

Returns

busy state of the slave Implements : LPI2C_HAL_SlaveGetSlaveBusyEvent_Activity

3.48.5.136 LPI2C_HAL_SlaveGetSMBusAlert()

```
static bool LPI2C_HAL_SlaveGetSMBusAlert (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured state for the SMBus Alert match.

This function returns the currently configured setting for SMBus Alert match.

Parameters

baseAddr	base address of the LPI2C module

Returns

specifies if match on SMBus Alert is enabled or disabled Implements : LPI2C_HAL_SlaveGetSMBusAlert_↩ Activity

3.48.5.137 LPI2C_HAL_SlaveGetSMBusAlertResponseEvent()

```
static bool LPI2C_HAL_SlaveGetSMBusAlertResponseEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the occurrence of an SMBus alert response.

This function checks for the detection of a SMBus Alert Response. This event is cleared by reading the received address - see function LPI2C_HAL_SlaveGetReceivedAddr().

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of an SMBus alert response Implements : LPI2C_HAL_SlaveGetSMBusAlertResponseEvent_↩ Activity

3.48.5.138 LPI2C_HAL_SlaveGetSoftwareReset()

```
static bool LPI2C_HAL_SlaveGetSoftwareReset (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the reset setting for the LPI2C slave.

This function returns the state of the LPI2C slave software reset bit.

Parameters

baseAddr	base address of the LPI2C module

Returns

the reset state of the LPI2C slave logic Implements : LPI2C_HAL_SlaveGetSoftwareReset_Activity

3.48.5.139 LPI2C_HAL_SlaveGetStartOfFrame()

```
static bool LPI2C_HAL_SlaveGetStartOfFrame (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if the current received data is the first in the current frame.

This function checks if the currently received data byte is the first byte since a (repeated) START or STOP condition.

Parameters

baseAddr	base address of the LPI2C module

Returns

specifies if the current received data is the first in the current frame Implements : LPI2C_HAL_SlaveGet↵ StartOfFrame_Activity

3.48.5.140 LPI2C_HAL_SlaveGetSTOPDetectEvent()

```
static bool LPI2C_HAL_SlaveGetSTOPDetectEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check the detection of a STOP condition.

This function checks for the detection of a STOP condition, after the LPI2C slave matched the last address byte.

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of a STOP condition Implements : LPI2C_HAL_SlaveGetSTOPDetectEvent_Activity

3.48.5.141 LPI2C_HAL_SlaveGetTransmitACKEvent()

```
static bool LPI2C_HAL_SlaveGetTransmitACKEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if transmitting ACK response is required.

This function checks if the LPI2C slave requests the software to provide an ACK or NACK response. This event is cleared by calling function LPI2C_HAL_SlaveSetTransmitNACK().

Parameters

baseAddr	base address of the LPI2C module

Returns

indicates whether or not ACK response is required Implements : LPI2C_HAL_SlaveGetTransmitACKEvent↩_Activity

3.48.5.142 LPI2C_HAL_SlaveGetTransmitDataEvent()

```
static bool LPI2C_HAL_SlaveGetTransmitDataEvent (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if transmit data is requested.

This function checks if the LPI2C slave requests data to transmit. The event is cleared by providing transmit data - see function LPI2C_HAL_SlaveTransmitData(). The event can also be automatically cleared if the LPI2C module detects a NACK or a repeated START or STOP condition - see function LPI2C_HAL_SlaveSetTxFlagConfig()

Parameters

baseAddr	base address of the LPI2C module

Returns

indication of a transmit data request Implements : LPI2C_HAL_SlaveGetTransmitDataEvent_Activity

3.48.5.143 LPI2C_HAL_SlaveGetTransmitNACK()

```
static lpi2c_slave_nack_transmit_t LPI2C_HAL_SlaveGetTransmitNACK (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured ACK/NACK transmission setting.

This function returns the currently configured setting for ACK/NACK response.

Parameters

baseAddr	base address of the LPI2C module

Returns

specifies whether ACK or NACK is configured to be transmitted Implements : LPI2C_HAL_SlaveGet↩TransmitNACK_Activity

3.48.5.144 LPI2C_HAL_SlaveGetTxDMA()

```
static bool LPI2C_HAL_SlaveGetTxDMA (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Check if slave transmit data DMA requests are enabled.

This function returns true if transmit data DMA requests are enabled.

Parameters

baseAddr	base address of the LPI2C module

Returns

enabled/disabled status of transmit data DMA requests Implements : LPI2C_HAL_SlaveGetTxDMA_Activity

3.48.5.145 LPI2C_HAL_SlaveGetTXDStall()

```
static bool LPI2C_HAL_SlaveGetTXDStall (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured state for clock stretching for data transmission.

This function returns the currently configured setting for data transmission clock stretching.

Parameters

baseAddr	base address of the LPI2C module

Returns

indicates if clock stretching is enabled or disabled Implements : LPI2C_HAL_SlaveGetTXDStall_Activity

3.48.5.146 LPI2C_HAL_SlaveGetTxFlagConfig()

```
static lpi2c_slave_txflag_config_t LPI2C_HAL_SlaveGetTxFlagConfig (
            const LPI2C_Type * baseAddr )  [inline], [static]
```

Return the configured settings for the transmit data flag.

This function returns the currently configured setting for the transmit data flag.

Parameters

baseAddr	base address of the LPI2C module

Returns

specifies if transmit data flag is automatically cleared when NACK is detected Implements : LPI2C_HAL_↩ SlaveGetTxFlagConfig_Activity

3.48.5.147 LPI2C_HAL_SlaveSetACKStall()

```
static void LPI2C_HAL_SlaveSetACKStall (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable or disable clock stretching for the sending of the ACK bit.

This function enables or disables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit and is therefore not compatible with high speed mode.

Parameters

baseAddr	base address of the LPI2C module
enable	enable or disable clock stretching Implements : LPI2C_HAL_SlaveSetACKStall_Activity

3.48.5.148 LPI2C_HAL_SlaveSetAddr0()

```
static void LPI2C_HAL_SlaveSetAddr0 (
            LPI2C_Type * baseAddr,
            uint16_t addr )  [inline], [static]
```

Configure the ADDR0 address for slave address match.

This function configures the ADDR0 value which is used to validate the received slave address. In 10-bit mode, the first address byte is compared to { 11110, ADDR0[10:9] } and the second address byte is compared to ADDR0[8:1]. In 7-bit mode, the address is compared to ADDR0[7:1] The formula used for address validation is configured with function LPI2C_HAL_SlaveSetAddrConfig().

Parameters

baseAddr	base address of the LPI2C module
addr	ADDR0 address for slave address match Implements : LPI2C_HAL_SlaveSetAddr0_Activity

3.48.5.149 LPI2C_HAL_SlaveSetAddr1()

```
static void LPI2C_HAL_SlaveSetAddr1 (
            LPI2C_Type * baseAddr,
            uint16_t addr )  [inline], [static]
```

Configure the ADDR1 address for slave address match.

This function configures the ADDR1 value which is used to validate the received slave address. In 10-bit mode, the first address byte is compared to { 11110, ADDR1[10:9] } and the second address byte is compared to ADDR1[8:1]. In 7-bit mode, the address is compared to ADDR1[7:1] The formula used for address validation is configured with function LPI2C_HAL_SlaveSetAddrConfig().

Parameters

baseAddr	base address of the LPI2C module
addr	ADDR1 address for slave address match Implements : LPI2C_HAL_SlaveSetAddr1_Activity

3.48.5.150 LPI2C_HAL_SlaveSetAddrConfig()

```
static void LPI2C_HAL_SlaveSetAddrConfig (
            LPI2C_Type * baseAddr,
            lpi2c_slave_addr_config_t configuration )  [inline], [static]
```

Control address match configuration.

This function configures the condition that will cause an address match to occur. See type lpi2c_slave_addr_↩ config_t for a description of available options.

Parameters

baseAddr	base address of the LPI2C module
configuration	configures the condition that will cause an address to match Implements : LPI2C_HAL_SlaveSetAddrConfig_Activity

3.48.5.151 LPI2C_HAL_SlaveSetAddrDMA()

```
static void LPI2C_HAL_SlaveSetAddrDMA (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable/disable slave address valid DMA requests.

This function enables or disables generation of Rx DMA requests when a valid address is received. Note that this DMA request is shared with slave receive data requests, so if both are used the receive data register should be configured to allow address reads - see function LPI2C_HAL_SlaveSetRxDataConfig()

Parameters

baseAddr	base address of the LPI2C module
enable	specifies whether to enable or disable address valid DMA requests Implements : LPI2C_HAL_SlaveSetAddrDMA_Activity

3.48.5.152 LPI2C_HAL_SlaveSetAddrStall()

```
static void LPI2C_HAL_SlaveSetAddrStall (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable or disable clock stretching for valid address reception.

This function enables or disables SCL clock stretching when the address valid flag is asserted. Clock stretching only occurs following the 9th bit and is therefore compatible with high speed mode.

Parameters

baseAddr	base address of the LPI2C module
enable	enable or disable clock stretching Implements : LPI2C_HAL_SlaveSetAddrStall_Activity

3.48.5.153 LPI2C_HAL_SlaveSetClockHoldTime()

```
static void LPI2C_HAL_SlaveSetClockHoldTime (
            LPI2C_Type * baseAddr,
            uint8_t cycles )  [inline], [static]
```

Configure the minimum clock hold time for the I2C slave.

This function configures the minimum clock hold time for the I2C slave, when clock stretching is enabled. The minimum hold time is equal to CLKHOLD+3 cycles. The I2C slave clock hold time is not affected by the PRESCALE configuration, and is disabled in high speed mode.

Parameters

baseAddr	base address of the LPI2C module
cycles	number of cycles for the minimum clock hold time for the I2C slave Implements : LPI2C_HAL_SlaveSetClockHoldTime_Activity

3.48.5.154 LPI2C_HAL_SlaveSetDataValidDelay()

```
static void LPI2C_HAL_SlaveSetDataValidDelay (
            LPI2C_Type * baseAddr,
            uint8_t cycles )  [inline], [static]
```

Configure the SDA data valid delay time for the I2C slave.

This function configures the SDA data valid delay time for the I2C slave equal to FILTSCL+DATAVD+3 cycles. This data valid delay must be configured to less than the minimum SCL low period. The I2C slave data valid delay time is not affected by the PRESCALE configuration, and is disabled in high speed mode.

Parameters

baseAddr	base address of the LPI2C module
cycles	number of cycles for the SDA data valid delay time for the I2C slave Implements : LPI2C_HAL_SlaveSetDataValidDelay_Activity

3.48.5.155 LPI2C_HAL_SlaveSetEnable()

```
static void LPI2C_HAL_SlaveSetEnable (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable or disable the LPI2C slave.

This function enables or disables the LPI2C module in slave mode.

Parameters

baseAddr	base address of the LPI2C module
enable	specifies whether to enable or disable the LPI2C slave Implements : LPI2C_HAL_SlaveSetEnable_Activity

3.48.5.156 LPI2C_HAL_SlaveSetFilterDoze()

```
static void LPI2C_HAL_SlaveSetFilterDoze (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Configure the slave filter in doze mode.

This function enables or disables the digital filter in doze mode.

Parameters

baseAddr	base address of the LPI2C module
enable	enable/disable the slave filter in doze mode Implements : LPI2C_HAL_SlaveSetFilterDoze_Activity

3.48.5.157 LPI2C_HAL_SlaveSetFilterEnable()

```
static void LPI2C_HAL_SlaveSetFilterEnable (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable/disable the slave filter.

This function enables or disables the digital filter and output delay counter for slave mode.

Parameters

baseAddr	base address of the LPI2C module
enable	enable/disable the slave filter Implements : LPI2C_HAL_SlaveSetFilterEnable_Activity

3.48.5.158 LPI2C_HAL_SlaveSetGeneralCall()

```
static void LPI2C_HAL_SlaveSetGeneralCall (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable or disable general call address.

This function enables or disables match general call address.

Parameters

baseAddr	base address of the LPI2C module
enable	enable or disable general call address Implements : LPI2C_HAL_SlaveSetGeneralCall_Activity

3.48.5.159 LPI2C_HAL_SlaveSetHighSpeedModeDetect()

```
static void LPI2C_HAL_SlaveSetHighSpeedModeDetect (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Control detection of the High-speed Mode master code.

This function enables or disables the detection of the High-speed Mode master code of slave address 0000_1XX, but does not cause an address match on this code. When set and any Hs-mode master code is detected, the slave filter and ACK stalls are disabled until the next STOP condition is detected.

Parameters

baseAddr	base address of the LPI2C module
enable	enable/disable the detection of the High-speed Mode master code Implements : LPI2C_HAL_SlaveSetHighSpeedModeDetect_Activity

3.48.5.160 LPI2C_HAL_SlaveSetIgnoreNACK()

```
static void LPI2C_HAL_SlaveSetIgnoreNACK (
            LPI2C_Type * baseAddr,
            lpi2c_slave_nack_config_t nack_config )  [inline], [static]
```

Control slave behaviour when NACK is detected.

This function controls the option to ignore received NACKs. When enabled, the LPI2C slave will continue transfers after a NACK is detected. This option is needed for Ultra-Fast mode.

Parameters

baseAddr	base address of the LPI2C module
nack_config	slave behaviour when NACK is detected Implements : LPI2C_HAL_SlaveSetIgnoreNACK_Activity

3.48.5.161 LPI2C_HAL_SlaveSetInt()

```
static void LPI2C_HAL_SlaveSetInt (
            LPI2C_Type * baseAddr,
            uint32_t interrupts,
            bool enable )  [inline], [static]
```

Enable or disable specified LPI2C slave interrupts.

This function can enable or disable one or more slave interrupt sources specified by the interrupts parameter.

Parameters

baseAddr	base address of the LPI2C module
interrupts	interrupts to be enabled or disabled; must be a bitwise or between one or more of the following constants: • LPI2C_HAL_SLAVE_SMBUS_ALERT_RESPONSE - SMBus Alert Response Interrupt • LPI2C_HAL_SLAVE_GENERAL_CALL - General Call Interrupt • LPI2C_HAL_SLAVE_ADDRESS_MATCH_1 - Address Match 1 Interrupt • LPI2C_HAL_SLAVE_ADDRESS_MATCH_0 - Address Match 0 Interrupt • LPI2C_HAL_SLAVE_FIFO_ERROR - FIFO Error Interrupt • LPI2C_HAL_SLAVE_BIT_ERROR - Bit Error Interrupt • LPI2C_HAL_SLAVE_STOP_DETECT - STOP Detect Interrupt • LPI2C_HAL_SLAVE_REPEATED_START - Repeated Start Interrupt • LPI2C_HAL_SLAVE_TRANSMIT_ACK - Transmit ACK Interrupt • LPI2C_HAL_SLAVE_ADDRESS_VALID - Address Valid Interrupt • LPI2C_HAL_SLAVE_RECEIVE_DATA - Receive Data Interrupt • LPI2C_HAL_SLAVE_TRANSMIT_DATA - Transmit Data Interrupt
enable	specifies whether to enable or disable specified interrupts Implements : LPI2C_HAL_SlaveSetInt_Activity

3.48.5.162 LPI2C_HAL_SlaveSetRxDataConfig()

```
static void LPI2C_HAL_SlaveSetRxDataConfig (
            LPI2C_Type * baseAddr,
            lpi2c_slave_rxdata_config_t configuration )  [inline], [static]
```

Control the functionality of the receive data register.

This function can configure the receive data register to allow the address to be read from it. When this option is enabled reading the data register when an address valid event is active will return the address and clear the address valid event. When the address valid event is not active data reads will behave normally (return received data and clear the receive data event).

Parameters

baseAddr	base address of the LPI2C module
configuration	controls if address can be read from the receive data register Implements : LPI2C_HAL_SlaveSetRxDataConfig_Activity

3.48.5.163 LPI2C_HAL_SlaveSetRxDMA()

```
static void LPI2C_HAL_SlaveSetRxDMA (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable/disable slave receive data DMA requests.

This function enables or disables generation of Rx DMA requests when received data is available.

Parameters

baseAddr	base address of the LPI2C module
enable	specifies whether to enable or disable receive data DMA requests Implements : LPI2C_HAL_SlaveSetRxDMA_Activity

3.48.5.164 LPI2C_HAL_SlaveSetRXStall()

```
static void LPI2C_HAL_SlaveSetRXStall (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable or disable clock stretching for data reception.

This function enables or disables SCL clock stretching when receive data flag is set during a slave-receive transfer. Clock stretching occurs following the 9th bit and is therefore compatible with high speed mode.

Parameters

baseAddr	base address of the LPI2C module
enable	enable or disable clock stretching Implements : LPI2C_HAL_SlaveSetRXStall_Activity

3.48.5.165 LPI2C_HAL_SlaveSetSCLGlitchFilter()

```
static void LPI2C_HAL_SlaveSetSCLGlitchFilter (
            LPI2C_Type * baseAddr,
            uint8_t cycles )  [inline], [static]
```

Configure the LPI2C slave SCL glitch filter.

This function configures the I2C slave digital glitch filters for SCL input, a configuration of 0 will disable the glitch filter. Glitches equal to or less than FILTSCL cycles long will be filtered out and ignored. The latency through the glitch filter is equal to FILTSCL+3 cycles and must be configured less than the minimum SCL low or high period. The glitch filter cycle count is not affected by the PRESCALE configuration, and is disabled in high speed mode.

Parameters

baseAddr	base address of the LPI2C module
cycles	number of cycles for the LPI2C slave SCL glitch filter Implements : LPI2C_HAL_SlaveSetSCLGlitchFilter_Activity

3.48.5.166 LPI2C_HAL_SlaveSetSDAGlitchFilter()

```
static void LPI2C_HAL_SlaveSetSDAGlitchFilter (
            LPI2C_Type * baseAddr,
            uint8_t cycles )  [inline], [static]
```

Configure the LPI2C slave SDA glitch filter.

This function configures the I2C slave digital glitch filters for SDA input. A configuration of 0 will disable the glitch filter. Glitches equal to or less than FILTSDA cycles long will be filtered out and ignored. The latency through the glitch filter is equal to FILTSDA+3 cycles and must be configured less than the minimum SCL low or high period. The glitch filter cycle count is not affected by the PRESCALE configuration, and is disabled in high speed mode.

Parameters

baseAddr	base address of the LPI2C module
cycles	number of cycles for the LPI2C slave SDA glitch filter Implements : LPI2C_HAL_SlaveSetSDAGlitchFilter_Activity

3.48.5.167 LPI2C_HAL_SlaveSetSMBusAlert()

```
static void LPI2C_HAL_SlaveSetSMBusAlert (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable or disable match on SMBus Alert.

This function enables or disables match on SMBus Alert.

Parameters

baseAddr	base address of the LPI2C module
enable	enable or disable match on SMBus Alert Implements : LPI2C_HAL_SlaveSetSMBusAlert_Activity

3.48.5.168 LPI2C_HAL_SlaveSetSoftwareReset()

```
static void LPI2C_HAL_SlaveSetSoftwareReset (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Set/clear the slave reset command.

Calling this function with enable parameter set to true will perform a software reset of the LPI2C slave.

Parameters

baseAddr	base address of the LPI2C module
enable	specifies the reset state of the LPI2C slave logic Implements : LPI2C_HAL_SlaveSetSoftwareReset_Activity

3.48.5.169 LPI2C_HAL_SlaveSetTransmitNACK()

```
static void LPI2C_HAL_SlaveSetTransmitNACK (
            LPI2C_Type * baseAddr,
            lpi2c_slave_nack_transmit_t nack )  [inline], [static]
```

Configure the ACK/NACK transmission after a received byte.

This function can be used to instruct the LPI2C slave whether to send an ACK or a NACK after receiving a byte. When ACK stall is enabled this function must be called after each matching address and after each received data byte. It can also be called when LPI2C Slave is disabled or idle to configure the default ACK/NACK.

Parameters

baseAddr	base address of the LPI2C module
nack	specifies whether to transmit ACK or NACK Implements : LPI2C_HAL_SlaveSetTransmitNACK_Activity

3.48.5.170 LPI2C_HAL_SlaveSetTxDMA()

```
static void LPI2C_HAL_SlaveSetTxDMA (
            LPI2C_Type * baseAddr,
            bool enable )  [inline], [static]
```

Enable/disable slave transmit data DMA requests.

This function enables or disables generation of Tx DMA requests when the module requires more data to transmit.

Parameters

baseAddr	base address of the LPI2C module
enable	specifies whether to enable or disable transmit data DMA requests Implements : LPI2C_HAL_SlaveSetTxDMA_Activity

3.48.5.171 LPI2C_HAL_SlaveSetTXDStall()

```
static void LPI2C_HAL_SlaveSetTXDStall (
            LPI2C_Type * baseAddr,
            bool enable ) [inline], [static]
```

Enable or disable clock stretching for data transmission.

This function enables or disables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer. Clock stretching occurs following the 9th bit and is therefore compatible with high speed mode.

Parameters

baseAddr	base address of the LPI2C module
enable	enable or disable clock stretching Implements : LPI2C_HAL_SlaveSetTXDStall_Activity

3.48.5.172 LPI2C_HAL_SlaveSetTxFlagConfig()

```
static void LPI2C_HAL_SlaveSetTxFlagConfig (
            LPI2C_Type * baseAddr,
            lpi2c_slave_txflag_config_t configuration ) [inline], [static]
```

Control the conditions for setting the transmit data flag.

This function controls the conditions that will trigger transmit data events. If this option is enabled the transmit data register is automatically emptied when a slave-transmit transfer is detected. This cause the transmit data event to be triggered whenever a slave-transmit transfer is detected and negate at the end of the slave-transmit transfer. If this option is disabled the transmit data event will assert whenever the transit data register is empty and negate when the transmit data register is full. This allows the transmit data register to be filled before a slave transmit transfer is detected, but can cause the transmit data register to be written before a NACK is detected on the last byte of a slave transmit transfer.

Parameters

baseAddr	base address of the LPI2C module
configuration	controls if transmit data flag is automatically cleared when NACK is detected Implements : LPI2C_HAL_SlaveSetTxFlagConfig_Activity

3.48.5.173 LPI2C_HAL_SlaveTransmitData()

```
static void LPI2C_HAL_SlaveTransmitData (
            LPI2C_Type * baseAddr,
            uint8_t data ) [inline], [static]
```

Provide data for the LPI2C slave transmitter.

This function provides one byte of data for the LPI2C slave to transmit. Calling this function clears the transmit data event.

Parameters

baseAddr	base address of the LPI2C module
data	data for the LPI2C slave transmitter Implements : LPI2C_HAL_SlaveTransmitData_Activity

3.49 LPIT Driver

3.49.1 Detailed Description

Low Power Interrupt Timer Peripheral Driver.

Hardware background

Each LPIT timer channel can be configured to run in one of 4 modes:
32-bit Periodic Counter: In this mode the counter will load and then decrement down to zero. It will then set the timer interrupt flag and assert the output pre-trigger.
Dual 16-bit Periodic Counter: In this mode, the counter will load and then the lower 16-bits will decrement down to zero, which will assert the output pre-trigger. The upper 16-bits will then decrement down to zero, which will negate the output pre-trigger and set the timer interrupt flag.
32-bit Trigger Accumulator: In this mode, the counter will load on the first trigger rising edge and then decrement down to zero on each trigger rising edge. It will then set the timer interrupt flag and assert the output pre-trigger.
32-bit Trigger Input Capture: In this mode, the counter will load with 0xFFFF_FFFF and then decrement down to zero. If a trigger rising edge is detected, it will store the inverse of the current counter value in the load value register, set the timer interrupt flag and assert the output pre-trigger.
In these modes, the timer channel operation is further controlled by Trigger Control bits (TSOT, TSOI, TROT) which control the load, reload, start and restart of the timer channels.

Driver consideration

The Driver uses structures for configuration. Each structure contains members that are specific to its respective functionality. There are **lpit_user_config_t** and **lpit_user_channel_config_t**.

Interrupt handling

Each LPIT timer channel has a corresponding interrupt handler. The LPIT Driver does not define interrupt handler internally. These interrupt handler methods can be defined by the user application. There are two ways to add an LPIT interrupt handler:

1. Using the weak symbols defined by start-up code. if the methods `LPIT`**`x`**`_Handler(void)` (x denotes instance number) are not defined, the linker use a default ISR. An error will be generated if methods with the same name are defined multiple times. This method works regardless of the placement of the interrupt vector table (Flash or RAM).
2. Using the Interrupt Manager's `INT_SYS_InstallHandler()` method. This can be used to dynamically change the ISR at run-time. This method works only if the interrupt vector table is located in RAM.

Clocking configuration

The LPIT Driver does not handle clock setup (from PCC) configuration. This is handled by the Clock Manager. The driver assumes that clock configurations have been made, so it is the user's responsibility to set up clocking and pin configurations correctly.

Basic operations

1. Pre-Initialization information of LPIT module

    - Before using the LPIT driver, the protocol clock of the module must be configured by the application using PCC module.
    - Configures Trigger MUX Control (TRGMUX) if want to use external trigger for LPIT module.
    - Configures different peripherals if want to use them in LPIT interrupt routine.
    - Provides configuration data structure to LPIT initialization API.

2. To initialize the LPIT module, just call the LPIT_DRV_Init() function with the user configuration data structure. This function configures LPIT module operation when MCU enters DEBUG and DOZE (Low power mode) modes and enables LPIT module. This function must be called firstly.
   In the following code, LPIT module is initialized to continue to run when MCU enters both Debug and DOZE modes.

```
#define BOARD_LPIT_INSTANCE 0U
/* LPIT module configuration stucture */
lpit_user_config_t lpitconfig =
{
    .enableRunInDebug = true,
    .enableRunInDoze  = true
};
/* Initializes the LPIT module. */
LPIT_DRV_Init(BOARD_LPIT_INSTANCE, &lpitconfig);
```

3. After calling the LPIT_DRV_Init() function, call LPIT_DRV_InitChannel() function with user channel configuration structure to initialize timer channel.
   This function configures timer channel chaining, timer channel mode, timer channel period, interrupt generation, trigger source, trigger select, reload on trigger, stop on interrupt and start on trigger. In the following code, timer channel is initialized with the channel chaining is disabled, interrupt generation is enabled, operation mode is 32 bit periodic counter mode, trigger source is external, reload on trigger is disabled, stop on interrupt is disabled, start on trigger is disabled and timer period is 0.5 second. Note that:

    - Trigger select is not effective if trigger source is external.

    - Timer channel period must be suitable for operation mode.

    - The timer channel 0 can not be chained.

    ```
    /* Channel 0 configuration structure */
    lpit_user_channel_config_t chnlconfig =
    {
        .timerMode = LPIT_PERIODIC_COUNTER,
        .periodUnits = LPTMR_PERIOD_UNITS_MICROSECONDS,
        .period = 1000000U,
        .triggerSource = LPIT_TRIGGER_SOURCE_INTERNAL,
        .triggerSelect = 1U,
        .enableReloadOnTrigger = false,
        .enableStopOnInterrupt = false,
        .enableStartOnTrigger = false,
        .chainChannel = false,
        .isInterruptEnabled = true
    };
    /* Initializes the channel 0 */
    LPIT_DRV_InitChannel(BOARD_LPIT_INSTANCE, 0, &chnlconfig);
    ```

4. To reconfigure timer channel period , just call LPIT_DRV_SetTimerPeriodByUs() or LPIT_DRV_SetTimer↵PeriodByCount() with corresponding new period. In the following code, the timer channel period is reconfigured with new period in count unit.

```
/* Reconfigures timer channel period with new period of 10000 count*/
LPIT_DRV_SetTimerPeriodByCount(BOARD_LPIT_INSTANCE, 0, 10000);
```

5. To start timer channel counting, just call LPIT_DRV_StartTimerChannels() with timer channels starting mask. In the following code, the timer channel 0 is started with the mask of 0x1U.

```
/* Starts channel 0 counting*/
LPIT_DRV_StartTimerChannels(BOARD_LPIT_INSTANCE, 0x1U);
```

6. To get interrupt flag of timer channel, just call LPIT_DRV_GetInterruptFlagTimerChannels() function with interrupt flag getting mask. In the following code, the interrupt flag of channel 0 is got with the mask of 0x1U.

```
/* Gets interrupt flag of channel 0 */
LPIT_DRV_GetInterruptFlagTimerChannels(BOARD_LPIT_INSTANCE, 0x1U);
```

7. To clear interrupt flag of timer channel, just call LPIT_DRV_ClearInterruptFlagTimerChannels() function with interrupt flag clearing mask. In the following code, the interrupt flag of channel 0 is cleared with the mask of 0x1U.

```
/* Clears interrupt flag of channel 0 */
LPIT_DRV_ClearInterruptFlagTimerChannels(BOARD_LPIT_INSTANCE, 0x1U)
      ;
```

8. To stop timer channel counting, just call LPIT_DRV_StopTimerChannels() with timer channels stopping mask. In the following code, the timer channel 0 is stopped with the mask of 0x1U.

```
/* Stops channel 0 counting*/
LPIT_DRV_StopTimerChannels(BOARD_LPIT_INSTANCE, 0x1U);
```

9. To disable LPIT module, just call LPIT_DRV_Deinit().

```
/* Disables LPIT module*/
LPIT_DRV_Deinit(BOARD_LPIT_INSTANCE);
```

Data Structures

- struct lpit_user_config_t

  LPIT configuration structure. More...
- struct lpit_user_channel_config_t

  Structure to configure the channel timer. More...

Macros

- #define MAX_PERIOD_COUNT (0xFFFFFFFFU)

  Max period in count of all operation mode except for dual 16 bit periodic counter mode.
- #define MAX_PERIOD_COUNT_IN_DUAL_16BIT_MODE (0x1FFFEU)

  Max period in count of dual 16 bit periodic counter mode.
- #define MAX_PERIOD_COUNT_16_BIT (0xFFFFU)

  Max count of 16 bit.

Enumerations

- enum lpit_period_units_t { LPIT_PERIOD_UNITS_COUNTS = 0x00U, LPIT_PERIOD_UNITS_MICROSE↩ CONDS = 0x01U }

  Unit options for LPIT period.

Initialization and De-initialization

- void LPIT_DRV_Init (uint32_t instance, const lpit_user_config_t ∗userConfig)

  Initializes the LPIT module.
- void LPIT_DRV_Deinit (uint32_t instance)

  De-Initializes the LPIT module.
- status_t LPIT_DRV_InitChannel (uint32_t instance, uint32_t channel, const lpit_user_channel_config_↩ t ∗userChannelConfig)

  Initializes the LPIT channel.

Timer Start and Stop

- void LPIT_DRV_StartTimerChannels (uint32_t instance, uint32_t mask)

  Starts the timer channel counting.
- void LPIT_DRV_StopTimerChannels (uint32_t instance, uint32_t mask)

  Stops the timer channel counting.

Timer Period

- status_t LPIT_DRV_SetTimerPeriodByUs (uint32_t instance, uint32_t channel, uint32_t periodUs)

  Sets the timer channel period in microseconds.
- status_t LPIT_DRV_SetTimerPeriodInDual16ModeByUs (uint32_t instance, uint32_t channel, uint16_↩ t periodHigh, uint16_t periodLow)

  Sets the timer channel period in microseconds.
- uint64_t LPIT_DRV_GetTimerPeriodByUs (uint32_t instance, uint32_t channel)

  Gets the timer channel period in microseconds.
- uint64_t LPIT_DRV_GetCurrentTimerUs (uint32_t instance, uint32_t channel)

  Gets the current timer channel counting value in microseconds.
- void LPIT_DRV_SetTimerPeriodByCount (uint32_t instance, uint32_t channel, uint32_t count)

  Sets the timer channel period in count unit.
- void LPIT_DRV_SetTimerPeriodInDual16ModeByCount (uint32_t instance, uint32_t channel, uint16_↩ t periodHigh, uint16_t periodLow)

  Sets the timer channel period in count unit.
- uint32_t LPIT_DRV_GetTimerPeriodByCount (uint32_t instance, uint32_t channel)

  Gets the current timer channel period in count unit.
- uint32_t LPIT_DRV_GetCurrentTimerCount (uint32_t instance, uint32_t channel)

  Gets the current timer channel counting value in count.

Interrupt

- uint32_t LPIT_DRV_GetInterruptFlagTimerChannels (uint32_t instance, uint32_t mask)

  Gets the current interrupt flag of timer channels.
- void LPIT_DRV_ClearInterruptFlagTimerChannels (uint32_t instance, uint32_t mask)

  Clears the interrupt flag of timer channels.

3.49.2 Data Structure Documentation

3.49.2.1 struct lpit_user_config_t

LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral to enable or disable LPIT module in DEBUG and DOZE mode Implements : lpit_user_config_t_Class

Data Fields

- bool enableRunInDebug
- bool enableRunInDoze

Field Documentation

3.49.2.1.1 enableRunInDebug

```
bool enableRunInDebug
```

True: Timer channels continue to run in debug mode False: Timer channels stop in debug mode

3.49.2.1.2 enableRunInDoze

```
bool enableRunInDoze
```

True: Timer channels continue to run in doze mode False: Timer channels stop in doze mode

3.49.2.2 struct lpit_user_channel_config_t

Structure to configure the channel timer.

This structure holds the configuration settings for the LPIT timer channel Implements : lpit_user_channel_config↩_t_Class

Data Fields

- lpit_timer_modes_t timerMode
- lpit_period_units_t periodUnits
- uint32_t period
- lpit_trigger_source_t triggerSource
- uint32_t triggerSelect
- bool enableReloadOnTrigger
- bool enableStopOnInterrupt
- bool enableStartOnTrigger
- bool chainChannel
- bool isInterruptEnabled

Field Documentation

3.49.2.2.1 chainChannel

```
bool chainChannel
```

Channel chaining enable

3.49.2.2.2 enableReloadOnTrigger

```
bool enableReloadOnTrigger
```

True: Timer channel will reload on selected trigger False: Timer channel will not reload on selected trigger

3.49.2.2.3 enableStartOnTrigger

```
bool enableStartOnTrigger
```

True: Timer channel starts to decrement when rising edge on selected trigger is detected. False: Timer starts to decrement immediately based on restart condition

3.49.2.2.4 enableStopOnInterrupt

```
bool enableStopOnInterrupt
```

True: Timer will stop after timeout False: Timer channel does not stop after timeout

3.49.2.2.5 isInterruptEnabled

```
bool isInterruptEnabled
```

Timer channel interrupt generation enable

3.49.2.2.6 period

```
uint32_t period
```

Period of timer channel

3.49.2.2.7 periodUnits

```
lpit_period_units_t periodUnits
```

Timer period value units

3.49.2.2.8 timerMode

```
lpit_timer_modes_t timerMode
```

Operation mode of timer channel

3.49.2.2.9 triggerSelect

```
uint32_t triggerSelect
```

Selects one trigger from the internal trigger sources this field makes sense if trigger source is internal

3.49.2.2.10 triggerSource

```
lpit_trigger_source_t triggerSource
```

Selects between internal and external trigger sources

3.49.3 Macro Definition Documentation

3.49.3.1 MAX_PERIOD_COUNT

```
#define MAX_PERIOD_COUNT (0xFFFFFFFFU)
```

Max period in count of all operation mode except for dual 16 bit periodic counter mode.

3.49.3.2 MAX_PERIOD_COUNT_16_BIT

```
#define MAX_PERIOD_COUNT_16_BIT (0xFFFFU)
```

Max count of 16 bit.

3.49.3.3 MAX_PERIOD_COUNT_IN_DUAL_16BIT_MODE

```
#define MAX_PERIOD_COUNT_IN_DUAL_16BIT_MODE (0x1FFFEU)
```

Max period in count of dual 16 bit periodic counter mode.

3.49.4 Enumeration Type Documentation

3.49.4.1 lpit_period_units_t

```
enum lpit_period_units_t
```

Unit options for LPIT period.

This is used to determine unit of timer period Implements : lpit_period_units_t_Class

Enumerator

LPIT_PERIOD_UNITS_COUNTS	Period value unit is count
LPIT_PERIOD_UNITS_MICROSECONDS	Period value unit is microsecond

3.49.5 Function Documentation

3.49.5.1 LPIT_DRV_ClearInterruptFlagTimerChannels()

```
void LPIT_DRV_ClearInterruptFlagTimerChannels (
            uint32_t instance,
            uint32_t mask )
```

Clears the interrupt flag of timer channels.

This function clears the interrupt flag of timer channels after their interrupt event occurred.

Parameters

in	*instance*	LPIT module instance number
in	*mask*	The interrupt flag clearing mask that decides which channels will be cleared interrupt flag • For example: – with mask = 0x01u then the interrupt flag of channel 0 only will be cleared – with mask = 0x02u then the interrupt flag of channel 1 only will be cleared – with mask = 0x03u then the interrupt flags of channel 0 and channel 1 will be cleared

3.49.5.2 LPIT_DRV_Deinit()

```
void LPIT_DRV_Deinit (
            uint32_t instance )
```

De-Initializes the LPIT module.

This function disables LPIT module. In order to use the LPIT module again, LPIT_DRV_Init must be called.

Parameters

in	*instance*	LPIT module instance number

3.49.5.3 LPIT_DRV_GetCurrentTimerCount()

```
uint32_t LPIT_DRV_GetCurrentTimerCount (
            uint32_t instance,
            uint32_t channel )
```

Gets the current timer channel counting value in count.

This function returns the real-time timer channel counting value, the value in a range from 0 to timer channel period. Need to make sure the running time does not exceed the timer channel period.

Parameters

in	*instance*	LPIT module instance number
in	*channel*	Timer channel number

Returns

Current timer channel counting value in count

3.49.5.4 LPIT_DRV_GetCurrentTimerUs()

```
uint64_t LPIT_DRV_GetCurrentTimerUs (
            uint32_t instance,
            uint32_t channel )
```

Gets the current timer channel counting value in microseconds.

This function returns an absolute time stamp in microseconds. 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. The return counting value here makes sense if the operation mode of timer channel is 32 bit periodic counter or dual 16 bit periodic counter or 32-bit trigger input capture. Need to make sure the running time will not exceed the timer channel period.

Parameters

`in`	*instance*	LPIT module instance number
`in`	*channel*	Timer channel number

Returns

Current timer channel counting value in microseconds

3.49.5.5 LPIT_DRV_GetInterruptFlagTimerChannels()

```
uint32_t LPIT_DRV_GetInterruptFlagTimerChannels (
            uint32_t instance,
            uint32_t mask )
```

Gets the current interrupt flag of timer channels.

This function gets the current interrupt flag of timer channels. In compare modes, the flag sets to 1 at the end of the timer period. In capture modes, the flag sets to 1 when the trigger asserts.

Parameters

`in`	*instance*	LPIT module instance number.
`in`	*mask*	The interrupt flag getting mask that decides which channels will be got interrupt flag. • For example: – with mask = 0x01u then the interrupt flag of channel 0 only will be got – with mask = 0x02u then the interrupt flag of channel 1 only will be got – with mask = 0x03u then the interrupt flags of channel 0 and channel 1 will be got

Returns

Current the interrupt flag of timer channels

3.49.5.6 LPIT_DRV_GetTimerPeriodByCount()

```
uint32_t LPIT_DRV_GetTimerPeriodByCount (
            uint32_t instance,
            uint32_t channel )
```

Gets the current timer channel period in count unit.

This function returns current period of timer channel given as argument.

Parameters

in	*instance*	LPIT module instance number
in	*channel*	Timer channel number

Returns

Timer channel period in count unit

3.49.5.7 LPIT_DRV_GetTimerPeriodByUs()

```
uint64_t LPIT_DRV_GetTimerPeriodByUs (
              uint32_t instance,
              uint32_t channel )
```

Gets the timer channel period in microseconds.

This function gets the timer channel period in microseconds. The returned period here makes sense if the operation mode of timer channel is 32 bit periodic counter or dual 16 bit periodic counter.

Parameters

in	*instance*	LPIT module instance number
in	*channel*	Timer channel number

Returns

Timer channel period in microseconds

3.49.5.8 LPIT_DRV_Init()

```
void LPIT_DRV_Init (
              uint32_t instance,
              const lpit_user_config_t * userConfig )
```

Initializes the LPIT module.

This function resets LPIT module, enables the LPIT module, configures LPIT module operation in Debug and DOZE mode. The LPIT configuration structure shall be passed as arguments. This configuration structure affects all timer channels. This function should be called before calling any other LPIT driver function.

This is an example demonstrating how to define a LPIT configuration structure:

```
lpit_user_config_t lpitInit =
{
     .enableRunInDebug = false,
     .enableRunInDoze = true
};
```

Parameters

in	*instance*	LPIT module instance number.
in	*userConfig*	Pointer to LPIT configuration structure.

3.49.5.9 LPIT_DRV_InitChannel()

```
status_t LPIT_DRV_InitChannel (
             uint32_t instance,
             uint32_t channel,
             const lpit_user_channel_config_t * userChannelConfig )
```

Initializes the LPIT channel.

This function initializes the LPIT timers by using a channel, this function configures timer channel chaining, timer channel mode, timer channel period, interrupt generation, trigger source, trigger select, reload on trigger, stop on interrupt and start on trigger. The timer channel number and its configuration structure shall be passed as arguments. Timer channels do not start counting by default after calling this function. The function LPIT_DRV_↩ StartTimerChannels must be called to start the timer channel counting. In order to re-configures the period, call the LPIT_DRV_SetTimerPeriodByUs or LPIT_DRV_SetTimerPeriodByCount.

This is an example demonstrating how to define a LPIT channel configuration structure:

```
lpit_user_channel_config_t lpitTestInit =
{
 .timerMode = LPIT_PERIODIC_COUNTER,
 .periodUnits = LPTMR_PERIOD_UNITS_MICROSECONDS,
 .period = 1000000U,
 .triggerSource = LPIT_TRIGGER_SOURCE_INTERNAL,
 .triggerSelect = 1U,
 .enableReloadOnTrigger = false,
 .enableStopOnInterrupt = false,
 .enableStartOnTrigger = false,
 .chainChannel = false,
 .isInterruptEnabled = true
};
```

Parameters

in	*instance*	LPIT module instance number
in	*channel*	Timer channel number
in	*userChannelConfig*	Pointer to LPIT channel configuration structure

Returns

Operation status
- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: The channel 0 is chained.
- STATUS_ERROR: The input period is invalid.

3.49.5.10 LPIT_DRV_SetTimerPeriodByCount()

```
void LPIT_DRV_SetTimerPeriodByCount (
             uint32_t instance,
             uint32_t channel,
             uint32_t count )
```

Sets the timer channel period in count unit.

This function sets the timer channel period in count unit. The counter period of a running timer channel can be modified by first setting a new load value, the value will be loaded after the timer channel expires. To abort the current cycle and start a timer channel period with the new value, the timer channel must be disabled and enabled again.

Parameters

in	*instance*	LPIT module instance number
in	*channel*	Timer channel number
in	*count*	Timer channel period in count unit

3.49.5.11 LPIT_DRV_SetTimerPeriodByUs()

```
status_t LPIT_DRV_SetTimerPeriodByUs (
            uint32_t instance,
            uint32_t channel,
            uint32_t periodUs )
```

Sets the timer channel period in microseconds.

This function sets the timer channel period in microseconds when timer channel mode is 32 bit periodic or dual 16 bit counter mode. The period range depends on the frequency of the LPIT functional clock and operation mode of timer channel. If the required period is out of range, use the suitable mode if applicable. This function is only valid for one single channel.

Parameters

in	*instance*	LPIT module instance number
in	*channel*	Timer channel number
in	*periodUs*	Timer channel period in microseconds

Returns

Operation status

- STATUS_SUCCESS: Input period of timer channel is valid.
- STATUS_ERROR: Input period of timer channel is invalid.

3.49.5.12 LPIT_DRV_SetTimerPeriodInDual16ModeByCount()

```
void LPIT_DRV_SetTimerPeriodInDual16ModeByCount (
            uint32_t instance,
            uint32_t channel,
            uint16_t periodHigh,
            uint16_t periodLow )
```

Sets the timer channel period in count unit.

This function sets the timer channel period in count unit when timer channel mode is dual 16 periodic counter mode. The counter period of a running timer channel can be modified by first setting a new load value, the value will be loaded after the timer channel expires. To abort the current cycle and start a timer channel period with the new value, the timer channel must be disabled and enabled again.

Parameters

in	*instance*	LPIT module instance number
in	*channel*	Timer channel number
in	*periodHigh*	Period of higher 16 bit in count unit
in	*periodLow*	Period of lower 16 bit in count unit

3.49.5.13 LPIT_DRV_SetTimerPeriodInDual16ModeByUs()

```
status_t LPIT_DRV_SetTimerPeriodInDual16ModeByUs (
            uint32_t instance,
            uint32_t channel,
            uint16_t periodHigh,
            uint16_t periodLow )
```

Sets the timer channel period in microseconds.

This function sets the timer channel period in microseconds when timer channel mode is dual 16 bit periodic counter mode. The period range depends on the frequency of the LPIT functional clock and operation mode of timer channel. If the required period is out of range, use the suitable mode if applicable. This function is only valid for one single channel.

Parameters

in	*instance*	LPIT module instance number
in	*channel*	Timer channel number
in	*periodHigh*	Period of higher 16 bit in microseconds
in	*periodLow*	Period of lower 16 bit in microseconds

Returns

Operation status
- STATUS_SUCCESS: Input period of timer channel is valid.
- STATUS_ERROR: Input period of timer channel is invalid.

3.49.5.14 LPIT_DRV_StartTimerChannels()

```
void LPIT_DRV_StartTimerChannels (
            uint32_t instance,
            uint32_t mask )
```

Starts the timer channel counting.

This function allows starting timer channels simultaneously . After calling this function, timer channels are going operate depend on mode and control bits which controls timer channel start, reload and restart.

Parameters

in	*instance*	LPIT module instance number
in	*mask*	Timer channels starting mask that decides which channels will be started • For example: – with mask = 0x01U then channel 0 will be started – with mask = 0x02U then channel 1 will be started – with mask = 0x03U then channel 0 and channel 1 will be started

3.49.5.15 LPIT_DRV_StopTimerChannels()

```
void LPIT_DRV_StopTimerChannels (
            uint32_t instance,
            uint32_t mask )
```

Stops the timer channel counting.

This function allows stop timer channels simultaneously from counting. Timer channels reload their periods respectively after the next time they call the LPIT_DRV_StartTimerChannels. Note that: In 32-bit Trigger Accumulator mode, the counter will load on the first trigger rising edge.

Parameters

in	*instance*	LPIT module instance number
in	*mask*	Timer channels stopping mask that decides which channels will be stopped • For example: – with mask = 0x01U then channel 0 will be stopped – with mask = 0x02U then channel 1 will be stopped – with mask = 0x03U then channel 0 and channel 1 will be stopped

3.50 LPIT HAL

3.50.1 Detailed Description

Low Power Interrupt Timer module Hardware Abstraction Layer.

This HAL provides low-level access to all hardware features of the LPIT.

Data Structures

- struct lpit_module_information_t

    Hardware information of LPIT module Implements : lpit_module_information_t_Class. More...

Enumerations

- enum lpit_timer_modes_t { LPIT_PERIODIC_COUNTER = 0x00U, LPIT_DUAL_PERIODIC_COUNTER = 0x01U, LPIT_TRIGGER_ACCUMULATOR = 0x02U, LPIT_INPUT_CAPTURE = 0x03U }

    Mode options available for the LPIT timer Implements : lpit_timer_modes_t_Class.
- enum lpit_trigger_source_t { LPIT_TRIGGER_SOURCE_EXTERNAL = 0x00U, LPIT_TRIGGER_SOURC↩ E_INTERNAL = 0x01U }

    Trigger source options.

Timer Initialization

- void LPIT_HAL_GetModuleInformation (const LPIT_Type ∗base, lpit_module_information_t ∗module↩ Infomation)

    Gets the information of LPIT module.
- static void LPIT_HAL_Enable (LPIT_Type ∗const base)

    Enables the LPIT module.
- static void LPIT_HAL_Disable (LPIT_Type ∗const base)

    Disables the LPIT module.
- static void LPIT_HAL_Reset (LPIT_Type ∗const base)

    Resets the LPIT module.

Timer Start and Stop

- static void LPIT_HAL_StartTimerChannels (LPIT_Type ∗const base, uint32_t mask)

    Starts the timer channel counting.
- static void LPIT_HAL_StopTimerChannels (LPIT_Type ∗const base, uint32_t mask)

    Stops the timer channel from counting.
- static bool LPIT_HAL_IsTimerChannelRunning (const LPIT_Type ∗base, uint32_t channel)

    Checks timer channel operation status.

Timer Period

- static void LPIT_HAL_SetTimerPeriodByCount (LPIT_Type ∗const base, uint32_t channel, uint32_t count)

    Sets the timer channel period in count unit.
- static uint32_t LPIT_HAL_GetTimerPeriodByCount (const LPIT_Type ∗base, uint32_t channel)

    Gets the timer channel period in count unit.
- static uint32_t LPIT_HAL_GetCurrentTimerCount (const LPIT_Type ∗base, uint32_t channel)

    Gets the current timer channel counting value.

Timer Interrupt

- static void LPIT_HAL_EnableInterruptTimerChannels (LPIT_Type ∗const base, uint32_t mask)

    Enables the interrupt generation for timer channels.
- static void LPIT_HAL_DisableInterruptTimerChannels (LPIT_Type ∗const base, uint32_t mask)

    Disables the interrupt generation for timer channels.
- static uint32_t LPIT_HAL_GetInterruptFlagTimerChannels (const LPIT_Type ∗base, uint32_t mask)

    Gets the interrupt flag of timer channels.
- static void LPIT_HAL_ClearInterruptFlagTimerChannels (LPIT_Type ∗const base, uint32_t mask)

    Clears the interrupt flag of timer channels.

Timer Configuration

- static void LPIT_HAL_SetTimerChannelModeCmd (LPIT_Type ∗const base, uint32_t channel, lpit_timer_↩ modes_t mode)

    Sets operation mode of timer channel.
- static lpit_timer_modes_t LPIT_HAL_GetTimerChannelModeCmd (const LPIT_Type ∗base, uint32_t channel)

    Gets current operation mode of timer channel.
- static void LPIT_HAL_SetTriggerSelectCmd (LPIT_Type ∗const base, uint32_t channel, uint32_t trigger↩ ChannelSelect)

    Sets internal trigger source for timer channel.
- static void LPIT_HAL_SetTriggerSourceCmd (LPIT_Type ∗const base, uint32_t channel, lpit_trigger_↩ source_t triggerSource)

    Sets trigger source of timer channel.
- static void LPIT_HAL_SetReloadOnTriggerCmd (LPIT_Type ∗const base, uint32_t channel, bool isReload↩ OnTrigger)

    Sets timer channel reload on trigger.
- static void LPIT_HAL_SetStopOnInterruptCmd (LPIT_Type ∗const base, uint32_t channel, bool isStopOn↩ Interrupt)

    Sets timer channel stop on interrupt.
- static void LPIT_HAL_SetStartOnTriggerCmd (LPIT_Type ∗const base, uint32_t channel, bool isStartOn↩ Trigger)

    Sets timer channel start on trigger.
- static void LPIT_HAL_SetTimerChannelChainCmd (LPIT_Type ∗const base, uint32_t channel, bool is↩ ChannelChained)

    Sets timer channel chaining.
- static void LPIT_HAL_SetTimerRunInDebugCmd (LPIT_Type ∗const base, bool isRunInDebug)

    Sets operation of LPIT in debug mode.
- static void LPIT_HAL_SetTimerRunInDozeCmd (LPIT_Type ∗const base, bool isRunInDoze)

    Sets operation of LPIT in DOZE mode.

3.50.2 Data Structure Documentation

3.50.2.1 struct lpit_module_information_t

Hardware information of LPIT module Implements : lpit_module_information_t_Class.

Data Fields

- uint32_t majorVersionNumber
- uint32_t minorVersionNumber
- uint32_t featureNumber
- uint32_t numberOfExternalTriggerInputs
- uint32_t numberOfTimerChannels

Field Documentation

3.50.2.1.1 featureNumber

```
uint32_t featureNumber
```

The feature set number

3.50.2.1.2 majorVersionNumber

```
uint32_t majorVersionNumber
```

The major version number for the LPIT module specification

3.50.2.1.3 minorVersionNumber

```
uint32_t minorVersionNumber
```

The minor version number for the LPIT module specification

3.50.2.1.4 numberOfExternalTriggerInputs

```
uint32_t numberOfExternalTriggerInputs
```

Number of external triggers implemented

3.50.2.1.5 numberOfTimerChannels

```
uint32_t numberOfTimerChannels
```

Number of timer channels implemented

3.50.3 Enumeration Type Documentation

3.50.3.1 lpit_timer_modes_t

```
enum lpit_timer_modes_t
```

Mode options available for the LPIT timer Implements : lpit_timer_modes_t_Class.

Enumerator

LPIT_PERIODIC_COUNTER	32-bit Periodic Counter
LPIT_DUAL_PERIODIC_COUNTER	Dual 16-bit Periodic Counter
LPIT_TRIGGER_ACCUMULATOR	32-bit Trigger Accumulator
LPIT_INPUT_CAPTURE	32-bit Trigger Input Capture

3.50.3.2 lpit_trigger_source_t

```
enum lpit_trigger_source_t
```

Trigger source options.

This is used for both internal and external trigger sources. The actual trigger options available is SoC specific, user should refer to the reference manual. Implements : lpit_trigger_source_t_Class

Enumerator

LPIT_TRIGGER_SOURCE_EXTERNAL	Use external trigger
LPIT_TRIGGER_SOURCE_INTERNAL	Use internal trigger

3.50.4 Function Documentation

3.50.4.1 LPIT_HAL_ClearInterruptFlagTimerChannels()

```
static void LPIT_HAL_ClearInterruptFlagTimerChannels (
            LPIT_Type *const base,
            uint32_t mask )  [inline], [static]
```

Clears the interrupt flag of timer channels.

This function clears current interrupt flag of timer channels.

Parameters

in	*base*	LPIT peripheral base address
in	*mask*	The interrupt flag clearing mask that decides which channels will be cleared interrupt flag. • For example: – with mask = 0x01u then the interrupt flag of channel 0 only will be cleared – with mask = 0x02u then the interrupt flag of channel 1 only will be cleared – with mask = 0x03u then the interrupt flags of channel 0 and channel 1 will be cleared Implements : LPIT_HAL_ClearInterruptFlagTimerChannels_Activity

3.50.4.2 LPIT_HAL_Disable()

```
static void LPIT_HAL_Disable (
            LPIT_Type *const base )  [inline], [static]
```

Disables the LPIT module.

This function disables functional clock of LPIT module (Note: it does not affect the system clock gating control).

Parameters

in	*base*	LPIT peripheral base address Implements : LPIT_HAL_Disable_Activity

3.50.4.3 LPIT_HAL_DisableInterruptTimerChannels()

```
static void LPIT_HAL_DisableInterruptTimerChannels (
            LPIT_Type *const base,
            uint32_t mask )  [inline], [static]
```

Disables the interrupt generation for timer channels.

This function allows disabling interrupt generation for timer channels simultaneously.

Parameters

in	*base*	LPIT peripheral base address
in	*mask*	The interrupt disabling mask that decides which channels will be disabled interrupt. • For example: – with mask = 0x01u then will disable interrupt for channel 0 only – with mask = 0x02u then will disable interrupt for channel 1 only – with mask = 0x03u then will disable interrupt for channel 0 and channel 1 Implements : LPIT_HAL_DisableInterruptTimerChannels_Activity

3.50.4.4 LPIT_HAL_Enable()

```
static void LPIT_HAL_Enable (
            LPIT_Type *const base )  [inline], [static]
```

Enables the LPIT module.

This function enables the functional clock of LPIT module (Note: this function does not un-gate the system clock gating control). It should be called before setup any timer channel.

Parameters

in	*base*	LPIT peripheral base address Implements : LPIT_HAL_Enable_Activity

3.50.4.5 LPIT_HAL_EnableInterruptTimerChannels()

```
static void LPIT_HAL_EnableInterruptTimerChannels (
            LPIT_Type *const base,
            uint32_t mask )  [inline], [static]
```

Enables the interrupt generation for timer channels.

This function allows enabling interrupt generation for timer channels simultaneously.

Parameters

in	*base*	LPIT peripheral base address
in	*mask*	The interrupt enabling mask that decides which channels will be enabled interrupt. • For example: – with mask = 0x01u then will enable interrupt for channel 0 only – with mask = 0x02u then will enable interrupt for channel 1 only – with mask = 0x03u then will enable interrupt for channel 0 and channel 1 Implements : LPIT_HAL_EnableInterruptTimerChannels_Activity

3.50.4.6 LPIT_HAL_GetCurrentTimerCount()

```
static uint32_t LPIT_HAL_GetCurrentTimerCount (
            const LPIT_Type * base,
            uint32_t channel )  [inline], [static]
```

Gets the current timer channel counting value.

This function returns the real-time timer channel counting value, the value in a range from 0 to timer channel period. Need to make sure the running time does not exceed the timer channel period.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number

Returns

Current timer channel counting value Implements : LPIT_HAL_GetCurrentTimerCount_Activity

3.50.4.7 LPIT_HAL_GetInterruptFlagTimerChannels()

```
static uint32_t LPIT_HAL_GetInterruptFlagTimerChannels (
            const LPIT_Type * base,
            uint32_t mask )  [inline], [static]
```

Gets the interrupt flag of timer channels.

This function gets current interrupt flag of timer channels.

Parameters

in	*base*	LPIT peripheral base address
in	*mask*	The interrupt flag getting mask that decides which channels will be got interrupt flag. • For example: – with mask = 0x01u then the interrupt flag of channel 0 only will be got – with mask = 0x02u then the interrupt flag of channel 1 only will be got – with mask = 0x03u then the interrupt flags of channel 0 and channel 1 will be got

Returns

> The interrupt flag of timer channels. Implements : LPIT_HAL_GetInterruptFlagTimerChannels_Activity

3.50.4.8 LPIT_HAL_GetModuleInformation()

```
void LPIT_HAL_GetModuleInformation (
            const LPIT_Type * base,
            lpit_module_information_t * moduleInfomation )
```

Gets the information of LPIT module.

This function gets the informations of LPIT module as: major version number, minor version number, feature number, number of external trigger inputs, number of timer channels.

Parameters

in	*base*	LPIT peripheral base address
out	*moduleInfomation*	Pointer to lpit_module_information_t structure which is used to contain LPIT information.

3.50.4.9 LPIT_HAL_GetTimerChannelModeCmd()

```
static lpit_timer_modes_t LPIT_HAL_GetTimerChannelModeCmd (
            const LPIT_Type * base,
            uint32_t channel )  [inline], [static]
```

Gets current operation mode of timer channel.

This function gets current operation mode of the timer channel given as argument.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number

Returns

> Operation mode of timer channel that is one of lpit_timer_modes_t Implements : LPIT_HAL_GetTimer←ChannelModeCmd_Activity

3.50.4.10 LPIT_HAL_GetTimerPeriodByCount()

```
static uint32_t LPIT_HAL_GetTimerPeriodByCount (
            const LPIT_Type * base,
            uint32_t channel )  [inline], [static]
```

Gets the timer channel period in count unit.

This function returns current period of timer channel given as argument.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number

Returns

Timer channel period in count unit Implements : LPIT_HAL_GetTimerPeriodByCount_Activity

3.50.4.11 LPIT_HAL_IsTimerChannelRunning()

```
static bool LPIT_HAL_IsTimerChannelRunning (
            const LPIT_Type * base,
            uint32_t channel )  [inline], [static]
```

Checks timer channel operation status.

This function checks to see whether the timer channel given as argument is running.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number

Returns

The timer channel operation status -True: the timer channel is running -False: the timer channel is not running Implements : LPIT_HAL_IsTimerChannelRunning_Activity

3.50.4.12 LPIT_HAL_Reset()

```
static void LPIT_HAL_Reset (
            LPIT_Type *const base )  [inline], [static]
```

Resets the LPIT module.

This function sets all LPIT registers to reset value, except the Module Control Register.

Parameters

in	*base*	LPIT peripheral base address Implements : LPIT_HAL_Reset_Activity

3.50.4.13 LPIT_HAL_SetReloadOnTriggerCmd()

```
static void LPIT_HAL_SetReloadOnTriggerCmd (
            LPIT_Type *const base,
            uint32_t channel,
            bool isReloadOnTrigger )  [inline], [static]
```

Sets timer channel reload on trigger.

This function sets the timer channel to reload/don't reload on trigger.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number
in	*isReloadOnTrigger*	Timer channel reload on trigger • True : timer channel will reload on trigger • False : timer channel will not reload on trigger Implements : LPIT_HAL_SetReloadOnTriggerCmd_Activity

3.50.4.14 LPIT_HAL_SetStartOnTriggerCmd()

```
static void LPIT_HAL_SetStartOnTriggerCmd (
            LPIT_Type *const base,
            uint32_t channel,
            bool isStartOnTrigger )  [inline], [static]
```

Sets timer channel start on trigger.

This function sets the timer channel to starts/don't start on trigger.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number
in	*isStartOnTrigger*	Timer channel start on trigger • True : Timer channel starts to decrement when rising edge on selected trigger is detected • False : Timer channel starts to decrement immediately based on restart condition (controlled by Timer Stop On Interrupt bit) Implements : LPIT_HAL_SetStartOnTriggerCmd_Activity

3.50.4.15 LPIT_HAL_SetStopOnInterruptCmd()

```
static void LPIT_HAL_SetStopOnInterruptCmd (
            LPIT_Type *const base,
            uint32_t channel,
            bool isStopOnInterrupt )  [inline], [static]
```

Sets timer channel stop on interrupt.

This function sets the timer channel to stop or don't stop after it times out.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number
in	*isStopOnInterrupt*	Timer channel stop on interrupt • True : Timer channel will stop after it times out • False : Timer channel will not stop after it times out Implements : LPIT_HAL_SetStopOnInterruptCmd_Activity

3.50.4.16 LPIT_HAL_SetTimerChannelChainCmd()

```
static void LPIT_HAL_SetTimerChannelChainCmd (
             LPIT_Type *const base,
             uint32_t channel,
             bool isChannelChained )  [inline], [static]
```

Sets timer channel chaining.

This function sets the timer channel to be chained or not chained.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number(Note: The timer channel 0 cannot be chained)
in	*isChannelChained*	Timer channel chaining • True : Timer channel is chained. Timer channel decrements on previous channel's timeout • False : Timer channel is not chained. Timer channel runs independently Implements : LPIT_HAL_SetTimerChannelChainCmd_Activity

3.50.4.17 LPIT_HAL_SetTimerChannelModeCmd()

```
static void LPIT_HAL_SetTimerChannelModeCmd (
             LPIT_Type *const base,
             uint32_t channel,
             lpit_timer_modes_t mode )  [inline], [static]
```

Sets operation mode of timer channel.

This function sets the timer channel operation mode which control how the timer channel decrements.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number
in	*mode*	Operation mode of timer channel that is member of lpit_timer_modes_t Implements : LPIT_HAL_SetTimerChannelModeCmd_Activity

3.50.4.18 LPIT_HAL_SetTimerPeriodByCount()

```
static void LPIT_HAL_SetTimerPeriodByCount (
             LPIT_Type *const base,
             uint32_t channel,
             uint32_t count )  [inline], [static]
```

Sets the timer channel period in count unit.

This function sets the timer channel period in count unit. The period range depends on the frequency of the LPIT functional clock and operation mode of timer channel. If the required period is out of range, use the suitable mode if applicable. Timer channel begins counting from the value that is set by this function. The counter period of a running timer channel can be modified by first setting a new load value, the value will be loaded after the timer channel expires. To abort the current cycle and start a timer channel period with the new value, the timer channel must be disabled and enabled again.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number
in	*count*	Timer channel period in count unit Implements : LPIT_HAL_SetTimerPeriodByCount_Activity

3.50.4.19 LPIT_HAL_SetTimerRunInDebugCmd()

```
static void LPIT_HAL_SetTimerRunInDebugCmd (
            LPIT_Type *const base,
            bool isRunInDebug )  [inline], [static]
```

Sets operation of LPIT in debug mode.

When the device enters debug mode, the timer channels 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 channel values), and continue the operation.

Parameters

in	*base*	LPIT peripheral base address
in	*isRunInDebug*	LPIT run in debug mode • True: LPIT continue to run when the device enters debug mode • False: LPIT stop when the device enters debug mode Implements : LPIT_HAL_SetTimerRunInDebugCmd_Activity

3.50.4.20 LPIT_HAL_SetTimerRunInDozeCmd()

```
static void LPIT_HAL_SetTimerRunInDozeCmd (
            LPIT_Type *const base,
            bool isRunInDoze )  [inline], [static]
```

Sets operation of LPIT in DOZE mode.

When the device enters debug mode, the timer channels may or may not be frozen, based on the configuration of this function. The LPIT must use an external or internal clock source which remains operating during DOZE modes(low power mode).

Parameters

in	*base*	LPIT peripheral base address
in	*isRunInDoze*	LPIT run in DOZE mode • True: LPIT continue to run when the device enters DOZE mode • False: LPIT channels stop when the device enters DOZE mode Implements : LPIT_HAL_SetTimerRunInDozeCmd_Activity

3.50.4.21 LPIT_HAL_SetTriggerSelectCmd()

```
static void LPIT_HAL_SetTriggerSelectCmd (
            LPIT_Type *const base,
            uint32_t channel,
            uint32_t triggerChannelSelect )  [inline], [static]
```

Sets internal trigger source for timer channel.

This function selects one trigger from the set of internal triggers that is generated by other timer channels. The selected trigger is used for starting and/or reloading the timer channel.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number
in	*triggerChannelSelect*	Number of the channel which is selected to be trigger source Implements : LPIT_HAL_SetTriggerSelectCmd_Activity

3.50.4.22 LPIT_HAL_SetTriggerSourceCmd()

```
static void LPIT_HAL_SetTriggerSourceCmd (
            LPIT_Type *const base,
            uint32_t channel,
            lpit_trigger_source_t triggerSource )  [inline], [static]
```

Sets trigger source of timer channel.

This function sets trigger source of the timer channel to be internal or external trigger.

Parameters

in	*base*	LPIT peripheral base address
in	*channel*	Timer channel number
in	*triggerSource*	Trigger source of timer channel(internal or external source) Implements : LPIT_HAL_SetTriggerSourceCmd_Activity

3.50.4.23 LPIT_HAL_StartTimerChannels()

```
static void LPIT_HAL_StartTimerChannels (
            LPIT_Type *const base,
            uint32_t mask )  [inline], [static]
```

Starts the timer channel counting.

This function allows starting timer channels simultaneously . After calling this function, timer channels are going operate depend on mode and control bits which controls timer channel start, reload and restart.

Parameters

in	*base*	LPIT peripheral base address

Parameters

in	*mask*	Timer channels starting mask that decides which channels will be started • For example: – with mask = 0x01U then channel 0 will be started – with mask = 0x02U then channel 1 will be started – with mask = 0x03U then channel 0 and channel 1 will be started Implements : LPIT_HAL_StartTimerChannels_Activity

3.50.4.24 LPIT_HAL_StopTimerChannels()

```
static void LPIT_HAL_StopTimerChannels (
            LPIT_Type *const base,
            uint32_t mask )  [inline], [static]
```

Stops the timer channel from counting.

This function allows stop timer channels simultaneously from counting. Timer channels reload their periods respectively after the next time they call the LPIT_DRV_StartTimerChannels. Note that: In 32-bit Trigger Accumulator mode, the counter will load on the first trigger rising edge.

Parameters

in	*base*	LPIT peripheral base address
in	*mask*	Timer channels stopping mask that decides which channels will be stopped • For example: – with mask = 0x01U then channel 0 will be stopped – with mask = 0x02U then channel 1 will be stopped – with mask = 0x03U then channel 0 and channel 1 will be stopped Implements : LPIT_HAL_StopTimerChannels_Activity

3.51 LPSPI Driver

3.51.1 Detailed Description

Low Power Serial Peripheral Interface Peripheral Driver.

Data Structures

- struct lpspi_master_config_t

    Data structure containing information about a device on the SPI bus. More...
- struct lpspi_state_t

    Runtime state structure for the LPSPI master driver. More...
- struct lpspi_slave_config_t

    User configuration structure for the SPI slave driver. Implements : lpspi_slave_config_t_Class. More...

Enumerations

- enum lpspi_transfer_type { LPSPI_USING_DMA = 0, LPSPI_USING_INTERRUPTS }

    Type of LPSPI transfer (based on interrupts or DMA). Implements : lpspi_transfer_type_Class.
- enum transfer_status_t { LPSPI_TRANSFER_OK = 0U, LPSPI_TRANSMIT_FAIL, LPSPI_RECEIVE_FAIL }

Functions

- void LPSPI_DRV_IRQHandler (uint32_t instance)

    The function LPSPI_DRV_IRQHandler passes IRQ control to either the master or slave driver.
- void LPSPI_DRV_FillupTxBuffer (uint32_t instance)

    The function LPSPI_DRV_FillupTxBuffer writes data in TX hardware buffer depending on driver state and number of bytes remained to send.
- void LPSPI_DRV_ReadRXBuffer (uint32_t instance)

    The function LPSPI_DRV_ReadRXBuffer reads data from RX hardware buffer and writes this data in RX software buffer.
- void LPSPI_DRV_DisableTEIEInterrupts (uint32_t instance)

    Disable the TEIE interrupts at the end of a transfer. Disable the interrupts and clear the status for transmit/receive errors.
- void LPSPI0_IRQHandler (void)

    This function is the implementation of LPSPI0 handler named in startup code.
- void LPSPI1_IRQHandler (void)

    This function is the implementation of LPSPI1 handler named in startup code.
- void LPSPI2_IRQHandler (void)

    This function is the implementation of LPSPI2 handler named in startup code.

Variables

- LPSPI_Type ∗ g_lpspiBase [LPSPI_INSTANCE_COUNT]

    Table of base pointers for SPI instances.
- IRQn_Type g_lpspiIrqId [LPSPI_INSTANCE_COUNT]

    Table to save LPSPI IRQ enumeration numbers defined in the CMSIS header file.
- lpspi_state_t ∗ g_lpspiStatePtr [LPSPI_INSTANCE_COUNT]

Initialization and shutdown

- status_t LPSPI_DRV_MasterInit (uint32_t instance, lpspi_state_t ∗lpspiState, const lpspi_master_config_t ∗spiConfig)

  Initializes a LPSPI instance for interrupt driven master mode operation.
- status_t LPSPI_DRV_MasterDeinit (uint32_t instance)

  Shuts down a LPSPI instance.
- status_t LPSPI_DRV_MasterSetDelay (uint32_t instance, uint32_t delayBetwenTransfers, uint32_t delayS←CKtoPCS, uint32_t delayPCStoSCK)

  Configures the LPSPI master mode bus timing delay options.

Bus configuration

- status_t LPSPI_DRV_MasterConfigureBus (uint32_t instance, const lpspi_master_config_t ∗spiConfig, uint32_t ∗calculatedBaudRate)

  Configures the LPSPI port physical parameters to access a device on the bus when the LSPI instance is configured for interrupt operation.

Blocking transfers

- status_t LPSPI_DRV_MasterTransferBlocking (uint32_t instance, const uint8_t ∗sendBuffer, uint8_←t ∗receiveBuffer, uint16_t transferByteCount, uint32_t timeout)

  Performs an interrupt driven blocking SPI master mode transfer.

Non-blocking transfers

- status_t LPSPI_DRV_MasterTransfer (uint32_t instance, const uint8_t ∗sendBuffer, uint8_t ∗receiveBuffer, uint16_t transferByteCount)

  Performs an interrupt driven non-blocking SPI master mode transfer.
- status_t LPSPI_DRV_MasterGetTransferStatus (uint32_t instance, uint32_t ∗bytesRemained)

  Returns whether the previous interrupt driven transfer is completed.
- status_t LPSPI_DRV_MasterAbortTransfer (uint32_t instance)

  Terminates an interrupt driven asynchronous transfer early.
- void LPSPI_DRV_MasterIRQHandler (uint32_t instance)

  Interrupt handler for LPSPI master mode. This handler uses the buffers stored in the lpspi_master_state_t structs to transfer data.

Initialization and shutdown

- status_t LPSPI_DRV_SlaveInit (uint32_t instance, lpspi_state_t ∗lpspiState, const lpspi_slave_config_←t ∗slaveConfig)

  Initializes a LPSPI instance for a slave mode operation, using interrupt mechanism.
- status_t LPSPI_DRV_SlaveDeinit (uint32_t instance)

  Shuts down an LPSPI instance interrupt mechanism.

Blocking transfers

- status_t LPSPI_DRV_SlaveTransferBlocking (uint32_t instance, const uint8_t ∗sendBuffer, uint8_t ∗receive←Buffer, uint16_t transferByteCount, uint32_t timeout)

  Transfers data on LPSPI bus using interrupt and a blocking call.

Non-blocking transfers

- void LPSPI_DRV_SlaveIRQHandler (uint32_t instance)

  Interrupt handler for LPSPI slave mode. This handler uses the buffers stored in the lpspi_master_state_t structs to transfer data.
- status_t LPSPI_DRV_SlaveTransfer (uint32_t instance, const uint8_t ∗sendBuffer, uint8_t ∗receiveBuffer, uint16_t transferByteCount)

  Starts the transfer data on LPSPI bus using an interrupt and a non-blocking call.
- status_t LPSPI_DRV_SlaveAbortTransfer (uint32_t instance)

  Aborts the transfer that started by a non-blocking call transfer function.
- status_t LPSPI_DRV_SlaveGetTransferStatus (uint32_t instance, uint32_t ∗bytesRemained)

  Returns whether the previous transfer is finished.

3.51.2 Data Structure Documentation

3.51.2.1 struct lpspi_master_config_t

Data structure containing information about a device on the SPI bus.

The user must populate these members to set up the LPSPI master and properly communicate with the SPI device. Implements : lpspi_master_config_t_Class

Data Fields

- uint32_t bitsPerSec
- lpspi_which_pcs_t whichPcs
- lpspi_signal_polarity_t pcsPolarity
- bool isPcsContinuous
- uint16_t bitcount
- uint32_t lpspiSrcClk
- lpspi_clock_phase_t clkPhase
- lpspi_sck_polarity_t clkPolarity
- bool lsbFirst
- lpspi_transfer_type transferType
- uint8_t rxDMAChannel
- uint8_t txDMAChannel

Field Documentation

3.51.2.1.1 bitcount

```
uint16_t bitcount
```

Number of bits/frame, minimum is 8-bits

3.51.2.1.2 bitsPerSec

```
uint32_t bitsPerSec
```

Baud rate in bits per second

3.51.2.1.3 clkPhase

```
lpspi_clock_phase_t clkPhase
```

Selects which phase of clock to capture data

3.51.2.1.4 clkPolarity

```
lpspi_sck_polarity_t clkPolarity
```

Selects clock polarity

3.51.2.1.5 isPcsContinuous

```
bool isPcsContinuous
```

Keeps PCS asserted until transfer complete

3.51.2.1.6 lpspiSrcClk

```
uint32_t lpspiSrcClk
```

Module source clock

3.51.2.1.7 lsbFirst

```
bool lsbFirst
```

Option to transmit LSB first

3.51.2.1.8 pcsPolarity

```
lpspi_signal_polarity_t pcsPolarity
```

PCS polarity

3.51.2.1.9 rxDMAChannel

```
uint8_t rxDMAChannel
```

Channel number for DMA rx channel. If DMA mode isn't used this field will be ignored.

3.51.2.1.10 transferType

```
lpspi_transfer_type transferType
```

Type of LPSPI transfer

3.51.2.1.11 txDMAChannel

```
uint8_t txDMAChannel
```

Channel number for DMA tx channel. If DMA mode isn't used this field will be ignored.

3.51.2.1.12 whichPcs

```
lpspi_which_pcs_t whichPcs
```

Selects which PCS to use

3.51.2.2 struct lpspi_state_t

Runtime state structure for the LPSPI master driver.

This structure holds data that is used by the LPSPI 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. The LPSPI master driver populates the members. Implements : lpspi_state_t_Class

Data Fields

- uint16_t bitsPerFrame
- uint16_t bytesPerFrame
- bool isPcsContinuous
- bool isBlocking
- uint32_t lpspiSrcClk
- volatile bool isTransferInProgress
- const uint8_t ∗ txBuff
- uint8_t ∗ rxBuff
- volatile uint16_t txCount
- volatile uint16_t rxCount
- volatile uint16_t txFrameCnt
- volatile uint16_t rxFrameCnt
- volatile bool lsb
- uint8_t fifoSize
- uint8_t rxDMAChannel
- uint8_t txDMAChannel
- lpspi_transfer_type transferType
- semaphore_t lpspiSemaphore
- transfer_status_t status

Field Documentation

3.51.2.2.1 bitsPerFrame

```
uint16_t bitsPerFrame
```

Number of bits per frame: 8- to 4096-bits; needed for TCR programming

3.51.2.2.2 bytesPerFrame

```
uint16_t bytesPerFrame
```

Number of bytes per frame: 1- to 512-bytes

3.51.2.2.3 fifoSize

```
uint8_t fifoSize
```

RX/TX fifo size

3.51.2.2.4 isBlocking

```
bool isBlocking
```

Save the transfer type

3.51.2.2.5 isPcsContinuous

```
bool isPcsContinuous
```

Option to keep chip select asserted until transfer complete; needed for TCR programming

3.51.2.2.6 isTransferInProgress

```
volatile bool isTransferInProgress
```

True if there is an active transfer

3.51.2.2.7 lpspiSemaphore

```
semaphore_t lpspiSemaphore
```

The semaphore used for blocking transfers

3.51.2.2.8 lpspiSrcClk

```
uint32_t lpspiSrcClk
```

Module source clock

3.51.2.2.9 lsb

```
volatile bool lsb
```

True if first bit is LSB and false if first bit is MSB

3.51.2.2.10 rxBuff

```
uint8_t* rxBuff
```

The buffer into which received bytes are placed

3.51.2.2.11 rxCount

```
volatile uint16_t rxCount
```

Number of bytes remaining to receive

3.51.2.2.12 rxDMAChannel

```
uint8_t rxDMAChannel
```

Channel number for DMA rx channel

3.51.2.2.13 rxFrameCnt

```
volatile uint16_t rxFrameCnt
```

Number of bytes from current frame which were already received

3.51.2.2.14 status

```
transfer_status_t status
```

The status of the current

3.51.2.2.15 transferType

```
lpspi_transfer_type transferType
```

Type of LPSPI transfer

3.51.2.2.16 txBuff

```
const uint8_t* txBuff
```

The buffer from which transmitted bytes are taken

3.51.2.2.17 txCount

```
volatile uint16_t txCount
```

Number of bytes remaining to send

3.51.2.2.18 txDMAChannel

```
uint8_t txDMAChannel
```

Channel number for DMA tx channel

3.51.2.2.19 txFrameCnt

```
volatile uint16_t txFrameCnt
```

Number of bytes from current frame which were already sent

3.51.2.3 struct lpspi_slave_config_t

User configuration structure for the SPI slave driver. Implements : lpspi_slave_config_t_Class.

Data Fields

- lpspi_signal_polarity_t pcsPolarity
- uint16_t bitcount
- lpspi_clock_phase_t clkPhase
- lpspi_which_pcs_t whichPcs
- lpspi_sck_polarity_t clkPolarity
- bool lsbFirst
- lpspi_transfer_type transferType
- uint8_t rxDMAChannel
- uint8_t txDMAChannel

Field Documentation

3.51.2.3.1 bitcount

```
uint16_t bitcount
```

Number of bits/frame, minimum is 8-bits

3.51.2.3.2 clkPhase

```
lpspi_clock_phase_t clkPhase
```

Selects which phase of clock to capture data

3.51.2.3.3 clkPolarity

```
lpspi_sck_polarity_t clkPolarity
```

Selects clock polarity

3.51.2.3.4 lsbFirst

```
bool lsbFirst
```

Option to transmit LSB first

3.51.2.3.5 pcsPolarity

```
lpspi_signal_polarity_t pcsPolarity
```

PCS polarity

3.51.2.3.6 rxDMAChannel

```
uint8_t rxDMAChannel
```

Channel number for DMA rx channel. If DMA mode isn't used this field will be ignored.

3.51.2.3.7 transferType

```
lpspi_transfer_type transferType
```

Type of LPSPI transfer

3.51.2.3.8 txDMAChannel

```
uint8_t txDMAChannel
```

Channel number for DMA tx channel. If DMA mode isn't used this field will be ignored.

3.51.2.3.9 whichPcs

```
lpspi_which_pcs_t whichPcs
```

3.51.3 Enumeration Type Documentation

3.51.3.1 lpspi_transfer_type

```
enum lpspi_transfer_type
```

Type of LPSPI transfer (based on interrupts or DMA). Implements : lpspi_transfer_type_Class.

Enumerator

LPSPI_USING_DMA	The driver will use DMA to perform SPI transfer
LPSPI_USING_INTERRUPTS	The driver will use interrupts to perform SPI transfer

3.51.3.2 transfer_status_t

```
enum transfer_status_t
```

Enumerator

LPSPI_TRANSFER_OK	Transfer OK
LPSPI_TRANSMIT_FAIL	Error during transmission
LPSPI_RECEIVE_FAIL	Error during reception

3.51.4 Function Documentation

3.51.4.1 LPSPI0_IRQHandler()

```
void LPSPI0_IRQHandler (
            void  )
```

This function is the implementation of LPSPI0 handler named in startup code.

It passes the instance to the shared LPSPI IRQ handler.

3.51.4.2 LPSPI1_IRQHandler()

```
void LPSPI1_IRQHandler (
            void  )
```

This function is the implementation of LPSPI1 handler named in startup code.

It passes the instance to the shared LPSPI IRQ handler.

3.51.4.3 LPSPI2_IRQHandler()

```
void LPSPI2_IRQHandler (
            void  )
```

This function is the implementation of LPSPI2 handler named in startup code.

It passes the instance to the shared LPSPI IRQ handler.

3.51.4.4 LPSPI_DRV_DisableTEIEInterrupts()

```
void LPSPI_DRV_DisableTEIEInterrupts (
            uint32_t instance )
```

Disable the TEIE interrupts at the end of a transfer. Disable the interrupts and clear the status for transmit/receive errors.

3.51.4.5 LPSPI_DRV_FillupTxBuffer()

```
void LPSPI_DRV_FillupTxBuffer (
            uint32_t instance )
```

The function LPSPI_DRV_FillupTxBuffer writes data in TX hardware buffer depending on driver state and number of bytes remained to send.

The function LPSPI_DRV_FillupTxBuffer writes data in TX hardware buffer depending on driver state and number of bytes remained to send.

3.51.4.6 LPSPI_DRV_IRQHandler()

```
void LPSPI_DRV_IRQHandler (
            uint32_t instance )
```

The function LPSPI_DRV_IRQHandler passes IRQ control to either the master or slave driver.

The address of the IRQ handlers are checked to make sure they are non-zero before they are called. If the IRQ handler's address is zero, it means that driver was not present in the link (because the IRQ handlers are marked as weak). This would actually be a program error, because it means the master/slave config for the IRQ was set incorrectly.

3.51.4.7 LPSPI_DRV_MasterAbortTransfer()

```
status_t LPSPI_DRV_MasterAbortTransfer (
            uint32_t instance )
```

Terminates an interrupt driven asynchronous transfer early.

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

Parameters

instance	The instance number of the LPSPI peripheral.

Returns

STATUS_SUCCESS The transfer was successful, or LPSPI_STATUS_NO_TRANSFER_IN_PROGRESS No transfer is currently in progress.

3.51.4.8 LPSPI_DRV_MasterConfigureBus()

```
status_t LPSPI_DRV_MasterConfigureBus (
            uint32_t instance,
            const lpspi_master_config_t * spiConfig,
            uint32_t * calculatedBaudRate )
```

Configures the LPSPI port physical parameters to access a device on the bus when the LSPI instance is configured for interrupt operation.

In this function, the term "spiConfig" is used to indicate the SPI device for which the LPSPI master is communicating. This is an optional function as the spiConfig parameters are normally configured in the initialization function or the transfer functions, where these various functions would call the configure bus function. This is an example to set up the lpspi_master_config_t structure to call the LPSPI_DRV_MasterConfigureBus function by passing in these parameters:

```
lpspi_master_config_t spiConfig1;   You can also declare spiConfig2, spiConfig3, etc
spiConfig1.bitsPerSec = 500000;
spiConfig1.whichPcs = LPSPI_PCS0;
spiConfig1.pcsPolarity = LPSPI_ACTIVE_LOW;
spiConfig1.isPcsContinuous = false;
spiConfig1.bitCount = 16;
spiConfig1.clkPhase = LPSPI_CLOCK_PHASE_1ST_EDGE;
spiConfig1.clkPolarity = LPSPI_ACTIVE_HIGH;
spiConfig1.lsbFirst= false;
spiConfig.transferType = LPSPI_USING_INTERRUPTS;
```

Parameters

instance	The instance number of the LPSPI peripheral.
spiConfig	Pointer to the spiConfig structure. This structure contains the settings for the SPI bus configuration. The SPI device parameters are the desired baud rate (in bits-per-sec), bits-per-frame, chip select attributes, clock attributes, and data shift direction.
calculatedBaudRate	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

STATUS_SUCCESS The transfer has completed successfully, or STATUS_ERROR if driver is error and needs to clean error.

3.51.4.9 LPSPI_DRV_MasterDeinit()

```
status_t LPSPI_DRV_MasterDeinit (
            uint32_t instance )
```

Shuts down a LPSPI instance.

This function resets the LPSPI peripheral, gates its clock, and disables the interrupt to the core. It first checks to see if a transfer is in progress and if so returns an error status.

Parameters

instance	The instance number of the LPSPI peripheral.

Returns

STATUS_SUCCESS The transfer has completed successfully, or STATUS_BUSY The transfer is still in progress. STATUS_ERROR if driver is error and needs to clean error.

3.51.4.10 LPSPI_DRV_MasterGetTransferStatus()

```
status_t LPSPI_DRV_MasterGetTransferStatus (
            uint32_t instance,
            uint32_t * bytesRemained )
```

Returns whether the previous interrupt driven transfer is completed.

When performing an a-sync (non-blocking) 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

instance	The instance number of the LPSPI peripheral.
bytesRemained	Pointer to a value that is filled in with the number of bytes that must be received.

Returns

STATUS_SUCCESS The transfer has completed successfully, or STATUS_BUSY The transfer is still in progress. framesTransferred is filled with the number of words that have been transferred so far.

3.51.4.11 LPSPI_DRV_MasterInit()

```
status_t LPSPI_DRV_MasterInit (
            uint32_t instance,
            lpspi_state_t * lpspiState,
            const lpspi_master_config_t * spiConfig )
```

Initializes a LPSPI instance for interrupt driven master mode operation.

This function uses an interrupt-driven method for transferring data. In this function, the term "spiConfig" is used to indicate the SPI device for which the LPSPI master is communicating. This function initializes the run-time state structure to track the ongoing transfers, un-gates the clock to the LPSPI module, resets the LPSPI module, configures the IRQ state structure, enables the module-level interrupt to the core, and enables the LPSPI module. This is an example to set up the lpspi_master_state_t and call the LPSPI_DRV_MasterInit function by passing in these parameters:

```
lpspi_master_state_t lpspiMasterState;  <- the user  allocates memory for this structure
lpspi_master_config_t spiConfig;  Can declare more configs for use in transfer
      functions
spiConfig.bitsPerSec = 500000;
spiConfig.whichPcs = LPSPI_PCS0;
spiConfig.pcsPolarity = LPSPI_ACTIVE_LOW;
spiConfig.isPcsContinuous = false;
spiConfig.bitCount = 16;
spiConfig.clkPhase = LPSPI_CLOCK_PHASE_1ST_EDGE;
spiConfig.clkPolarity = LPSPI_ACTIVE_HIGH;
spiConfig.lsbFirst= false;
spiConfig.transferType = LPSPI_USING_INTERRUPTS;
LPSPI_DRV_MasterInit(masterInstance, &lpspiMasterState, &spiConfig);
```

Parameters

instance	The instance number of the LPSPI peripheral.
lpspiState	The pointer to the LPSPI master driver state structure. The user passes the memory for this run-time state structure. The LPSPI master driver populates the members. This run-time state structure keeps track of the transfer in progress.
spiConfig	The data structure containing information about a device on the SPI bus

Returns

An error code or STATUS_SUCCESS.

3.51.4.12 LPSPI_DRV_MasterIRQHandler()

```
void LPSPI_DRV_MasterIRQHandler (
              uint32_t instance )
```

Interrupt handler for LPSPI master mode. This handler uses the buffers stored in the lpspi_master_state_t structs to transfer data.

Parameters

instance	The instance number of the LPSPI peripheral.

Interrupt handler for LPSPI master mode. This handler uses the buffers stored in the lpspi_master_state_t structs to transfer data.

3.51.4.13 LPSPI_DRV_MasterSetDelay()

```
status_t LPSPI_DRV_MasterSetDelay (
              uint32_t instance,
              uint32_t delayBetwenTransfers,
              uint32_t delaySCKtoPCS,
              uint32_t delayPCStoSCK )
```

Configures the LPSPI master mode bus timing delay options.

This function involves the LPSPI module's delay options to "fine tune" some of the signal timings and match the timing needs of a slower peripheral device. This is an optional function that can be called after the LPSPI module has been initialized for master mode. The timings are adjusted in terms of cycles of the baud rate clock. The bus timing delays that can be adjusted are listed below:

SCK to PCS Delay: Adjustable delay option between the last edge of SCK to the de-assertion of the PCS signal.

PCS to SCK Delay: Adjustable delay option between the assertion of the PCS signal to the first SCK edge.

Delay between Transfers: Adjustable delay option between the de-assertion of the PCS signal for a frame to the assertion of the PCS signal for the next frame.

Parameters

instance	The instance number of the LPSPI peripheral.
delayBetwenTransfers	Minimum delay between 2 transfers in microseconds
delaySCKtoPCS	Minimum delay between SCK and PCS
delayPCStoSCK	Minimum delay between PCS and SCK

Returns

STATUS_SUCCESS The transfer has completed successfully, or STATUS_ERROR if driver is error and needs to clean error.

3.51.4.14 LPSPI_DRV_MasterTransfer()

```
status_t LPSPI_DRV_MasterTransfer (
            uint32_t instance,
            const uint8_t * sendBuffer,
            uint8_t * receiveBuffer,
            uint16_t transferByteCount )
```

Performs an interrupt driven non-blocking SPI master mode transfer.

This function simultaneously sends and receives data on the SPI bus, as SPI is naturally a full-duplex bus. The function returns immediately after initiating the transfer. The user needs to check whether the transfer is complete using the LPSPI_DRV_MasterGetTransferStatus function. This function allows the user to optionally pass in a SPI configuration structure which allows the user to change the SPI bus attributes in conjunction with initiating a SPI transfer. The difference between passing in the SPI configuration structure here as opposed to the configure bus function is that the configure bus function returns the calculated baud rate where this function does not. The user can also call the configure bus function prior to the transfer in which case the user would simply pass in a NULL to the transfer function's device structure parameter.

Parameters

instance	The instance number of the LPSPI peripheral.
spiConfig	Pointer to the SPI configuration structure. This structure 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 LPSPI_DRV_MasterConfigureBus 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) is sent.
receiveBuffer	Pointer to the buffer where the received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.
transferByteCount	The number of bytes to send and receive.

Returns

STATUS_SUCCESS The transfer was successful, or STATUS_BUSY Cannot perform transfer because a transfer is already in progress

3.51.4.15 LPSPI_DRV_MasterTransferBlocking()

```
status_t LPSPI_DRV_MasterTransferBlocking (
            uint32_t instance,
            const uint8_t * sendBuffer,
            uint8_t * receiveBuffer,
            uint16_t transferByteCount,
            uint32_t timeout )
```

Performs an interrupt driven blocking SPI master mode transfer.

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. This function allows the user to optionally pass in a SPI configuration structure which allows the user to change the SPI bus attributes in conjunction with initiating a SPI transfer. The difference between passing in the SPI configuration structure here as opposed to the configure bus function is that the configure bus function returns the calculated baud rate where this function does not. The user can also call the configure bus function prior to the transfer in which case the user would simply pass in a NULL to the transfer function's device structure parameter.

Parameters

instance	The instance number of the LPSPI peripheral.
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) is sent.
receiveBuffer	Pointer to the buffer where the received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.
transferByteCount	The number of bytes to send and receive.
timeout	A timeout for the transfer in milliseconds. If the transfer takes longer than this amount of time, the transfer is aborted and a STATUS_TIMEOUT error returned.

Returns

STATUS_SUCCESS The transfer was successful, or STATUS_BUSY Cannot perform transfer because a transfer is already in progress, or STATUS_TIMEOUT The transfer timed out and was aborted.

3.51.4.16 LPSPI_DRV_ReadRXBuffer()

```
void LPSPI_DRV_ReadRXBuffer (
            uint32_t instance )
```

The function LPSPI_DRV_ReadRXBuffer reads data from RX hardware buffer and writes this data in RX software buffer.

The function LPSPI_DRV_ReadRXBuffer reads data from RX hardware buffer and writes this data in RX software buffer.

3.51.4.17 LPSPI_DRV_SlaveAbortTransfer()

```
status_t LPSPI_DRV_SlaveAbortTransfer (
            uint32_t instance )
```

Aborts the transfer that started by a non-blocking call transfer function.

This function stops the transfer which was started by the calling the SPI_DRV_SlaveTransfer() function.

Parameters

instance	The instance number of SPI peripheral

Returns

STATUS_SUCCESS if everything is OK.

3.51.4.18 LPSPI_DRV_SlaveDeinit()

```
status_t LPSPI_DRV_SlaveDeinit (
            uint32_t instance )
```

Shuts down an LPSPI instance interrupt mechanism.

Disables the LPSPI module, gates its clock, and changes the LPSPI slave driver state to NonInit for the LPSPI slave module which is initialized with interrupt mechanism. After de-initialization, the user can re-initialize the LPSPI slave module with other mechanisms.

Parameters

instance	The instance number of the LPSPI peripheral.

Returns

STATUS_SUCCESS if driver starts to send/receive data successfully. STATUS_ERROR if driver is error and needs to clean error. STATUS_BUSY if a transfer is in progress

3.51.4.19 LPSPI_DRV_SlaveGetTransferStatus()

```
status_t LPSPI_DRV_SlaveGetTransferStatus (
            uint32_t instance,
            uint32_t * bytesRemained )
```

Returns whether the previous transfer is finished.

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

instance	The instance number of the LPSPI peripheral.
bytesRemained	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.

Returns

STATUS_SUCCESS The transfer has completed successfully, or STATUS_BUSY The transfer is still in progress. STATUS_ERROR if driver is error and needs to clean error.

3.51.4.20 LPSPI_DRV_SlaveInit()

```
status_t LPSPI_DRV_SlaveInit (
            uint32_t instance,
            lpspi_state_t * lpspiState,
            const lpspi_slave_config_t * slaveConfig )
```

Initializes a LPSPI instance for a slave mode operation, using interrupt mechanism.

This function un-gates the clock to the LPSPI module, initializes the LPSPI for slave mode. After it is initialized, the LPSPI module is configured in slave mode and the user can start transmitting and receiving data by calling send, receive, and transfer functions. This function indicates LPSPI slave uses an interrupt mechanism.

Parameters

instance	The instance number of the LPSPI peripheral.
lpspiState	The pointer to the LPSPI slave driver state structure.
slaveConfig	The configuration structure lpspi_slave_user_config_t which configures the data bus format.

Returns

An error code or STATUS_SUCCESS.

3.51.4.21 LPSPI_DRV_SlaveIRQHandler()

```
void LPSPI_DRV_SlaveIRQHandler (
            uint32_t instance )
```

Interrupt handler for LPSPI slave mode. This handler uses the buffers stored in the lpspi_master_state_t structs to transfer data.

Parameters

instance	The instance number of the LPSPI peripheral.

3.51.4.22 LPSPI_DRV_SlaveTransfer()

```
status_t LPSPI_DRV_SlaveTransfer (
            uint32_t instance,
            const uint8_t * sendBuffer,
            uint8_t * receiveBuffer,
            uint16_t transferByteCount )
```

Starts the transfer data on LPSPI bus using an interrupt and a non-blocking call.

Parameters

instance	The instance number of LPSPI peripheral
sendBuffer	The pointer to data that user wants to transmit.
receiveBuffer	The pointer to data that user wants to store received data.
transferByteCount	The number of bytes to send and receive.

Returns

STATUS_SUCCESS if driver starts to send/receive data successfully. STATUS_ERROR if driver is error and needs to clean error. STATUS_BUSY if a transfer is in progress

3.51.4.23 LPSPI_DRV_SlaveTransferBlocking()

```
status_t LPSPI_DRV_SlaveTransferBlocking (
            uint32_t instance,
            const uint8_t * sendBuffer,
            uint8_t * receiveBuffer,
            uint16_t transferByteCount,
            uint32_t timeout )
```

Transfers data on LPSPI bus using interrupt and a blocking call.

This function checks the driver status and mechanism, and transmits/receives data through the LPSPI bus. If the sendBuffer is NULL, the transmit process is ignored. If the receiveBuffer is NULL, the receive process is ignored. If both the receiveBuffer and the sendBuffer are available, the transmit and the receive progress is processed. If only the receiveBuffer is available, the receive is processed. Otherwise, the transmit is processed. This function only returns when the processes are completed. This function uses an interrupt mechanism.

Parameters

instance	The instance number of LPSPI peripheral
sendBuffer	The pointer to data that user wants to transmit.
receiveBuffer	The pointer to data that user wants to store received data.
transferByteCount	The number of bytes to send and receive.
timeout	The maximum number of milliseconds that function waits before timed out reached.

Returns

STATUS_SUCCESS if driver starts to send/receive data successfully. STATUS_ERROR if driver is error and needs to clean error. STATUS_BUSY if a transfer is in progress STATUS_TIMEOUT if time out reached while transferring is in progress.

3.51.5 Variable Documentation

3.51.5.1 g_lpspiBase

```
LPSPI_Type* g_lpspiBase[LPSPI_INSTANCE_COUNT]
```

Table of base pointers for SPI instances.

3.51.5.2 g_lpspiIrqId

```
IRQn_Type g_lpspiIrqId[LPSPI_INSTANCE_COUNT]
```

Table to save LPSPI IRQ enumeration numbers defined in the CMSIS header file.

3.51.5.3 g_lpspiStatePtr

```
lpspi_state_t* g_lpspiStatePtr[LPSPI_INSTANCE_COUNT]
```

3.52 LPSPI HAL

3.52.1 Detailed Description

Low Power Serial Peripheral Interface Hardware Abstraction Layer.

Data Structures

- struct lpspi_tx_cmd_config_t

  LPSPI Transmit Command Register configuration structure. More...
- struct lpspi_init_config_t

  LPSPI initialization configuration structure. More...

Enumerations

- enum lpspi_prescaler_t {
  LPSPI_DIV_1 = 0U, LPSPI_DIV_2 = 1U, LPSPI_DIV_4 = 2U, LPSPI_DIV_8 = 3U,
  LPSPI_DIV_16 = 4U, LPSPI_DIV_32 = 5U, LPSPI_DIV_64 = 6U, LPSPI_DIV_128 = 7U }

  Prescaler values for LPSPI clock source. Implements : lpspi_prescaler_t_Class.
- enum lpspi_status_flag_t {
  LPSPI_TX_DATA_FLAG = LPSPI_SR_TDF_SHIFT, LPSPI_RX_DATA_FLAG = LPSPI_SR_RDF_SHIFT, LPSPI_WORD_COMPLETE = LPSPI_SR_WCF_SHIFT, LPSPI_FRAME_COMPLETE = LPSPI_SR_FC↩
  F_SHIFT,
  LPSPI_TRANSFER_COMPLETE = LPSPI_SR_TCF_SHIFT, LPSPI_TRANSMIT_ERROR = LPSPI_SR_T↩
  EF_SHIFT, LPSPI_RECEIVE_ERROR = LPSPI_SR_REF_SHIFT, LPSPI_DATA_MATCH = LPSPI_SR_D↩
  MF_SHIFT,
  LPSPI_MODULE_BUSY = LPSPI_SR_MBF_SHIFT, LPSPI_ALL_STATUS = 0x00003F00U }

  LPSPI status flags. Implements : lpspi_status_flag_t_Class.
- enum lpspi_sck_polarity_t { LPSPI_SCK_ACTIVE_HIGH = 0U, LPSPI_SCK_ACTIVE_LOW = 1U }

  LPSPI Clock Signal (SCK) Polarity configuration. Implements : lpspi_sck_polarity_t_Class.
- enum lpspi_signal_polarity_t { LPSPI_ACTIVE_HIGH = 1U, LPSPI_ACTIVE_LOW = 0U }

  LPSPI Signal (PCS and Host Request) Polarity configuration. Implements : lpspi_signal_polarity_t_Class.
- enum lpspi_host_request_select_t { LPSPI_HOST_REQ_EXT_PIN = 0U, LPSPI_HOST_REQ_INTERNA↩
  L_TRIGGER = 1U }

  LPSPI Host Request select configuration. Implements : lpspi_host_request_select_t_Class.
- enum lpspi_master_slave_mode_t { LPSPI_MASTER = 1U, LPSPI_SLAVE = 0U }

  LPSPI master or slave configuration. Implements : lpspi_master_slave_mode_t_Class.
- enum lpspi_which_pcs_t { LPSPI_PCS0 = 0U, LPSPI_PCS1 = 1U, LPSPI_PCS2 = 2U, LPSPI_PCS3 = 3U }

  LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure). Implements : lpspi_which_pcs_t_Class.
- enum lpspi_match_config_t {
  LPSPI_MATCH_DISABLED = 0x0U, LPSPI_1ST_WRD_EQUALS_M0_OR_M1 = 0x2U, LPSPI_ANY_↩
  WRD_EQUALS_M0_OR_M1 = 0x3U, LPSPI_1ST_WRD_EQUALS_M0_AND_2ND_WRD_EQUALS_M1 =
  0x4U,
  LPSPI_ANY_WRD_EQUALS_M0_AND_NXT_WRD_EQUALS_M1 = 0x5U, LPSPI_1ST_WRD_AND_M1↩
  _EQUALS_M0_AND_M1 = 0x6U, LPSPI_ANY_WRD_AND_M1_EQUALS_M0_AND_M1 = 0x7U }

  LPSPI Match configuration options. Implements : lpspi_match_config_t_Class.
- enum lpspi_pin_config_t { LPSPI_SDI_IN_SDO_OUT = 0U, LPSPI_SDI_IN_OUT = 1U, LPSPI_SDO_IN_↩
  OUT = 2U, LPSPI_SDI_OUT_SDO_IN = 3U }

  LPSPI pin (SDO and SDI) configuration. Implements : lpspi_pin_config_t_Class.
- enum lpspi_clock_phase_t { LPSPI_CLOCK_PHASE_1ST_EDGE = 0U, LPSPI_CLOCK_PHASE_2ND_E↩
  DGE = 1U }

LPSPI clock phase configuration. Implements : lpspi_clock_phase_t_Class.

- enum lpspi_data_out_config_t { LPSPI_DATA_OUT_RETAINED = 0U, LPSPI_DATA_OUT_TRISTATE = 1U }

  LPSPI data output configuration. Implements : lpspi_data_out_config_t_Class.
- enum lpspi_transfer_width_t { LPSPI_SINGLE_BIT_XFER = 0U, LPSPI_TWO_BIT_XFER = 1U, LPSPI_F←OUR_BIT_XFER = 2U }

  LPSPI transfer width configuration. Implements : lpspi_transfer_width_t_Class.
- enum lpspi_delay_type_t { LPSPI_SCK_TO_PCS = LPSPI_CCR_SCKPCS_SHIFT, LPSPI_PCS_TO_SCK = LPSPI_CCR_PCSSCK_SHIFT, LPSPI_BETWEEN_TRANSFER = LPSPI_CCR_DBT_SHIFT }

  LPSPI delay type selection Implements : lpspi_delay_type_t_Class.

Variables

- static const uint32_t s_baudratePrescaler [ ] = { 1, 2, 4, 8, 16, 32, 64, 128 }

Configuration

- void LPSPI_HAL_Init (LPSPI_Type ∗base)

  Resets the LPSPI internal logic and registers to their default settings.
- status_t LPSPI_HAL_Config (LPSPI_Type ∗base, const lpspi_init_config_t ∗config, lpspi_tx_cmd_config_t ∗txCmdCfgSet, uint32_t ∗actualBaudRate)

  Configures the LPSPI registers to a user defined configuration.
- void LPSPI_HAL_GetVersionId (const LPSPI_Type ∗base, uint32_t ∗major, uint32_t ∗minor, uint32_←t ∗feature)

  Gets the Major, Minor and Feature ID of the LPSPI module.
- static void LPSPI_HAL_Enable (LPSPI_Type ∗base)

  Enables the LPSPI module.
- static bool LPSPI_HAL_IsModuleEnabled (const LPSPI_Type ∗base)

  Check if LPSPI module is enabled.
- status_t LPSPI_HAL_Disable (LPSPI_Type ∗base)

  Disables the LPSPI module.
- status_t LPSPI_HAL_SetMasterSlaveMode (LPSPI_Type ∗base, lpspi_master_slave_mode_t mode)

  Configures the LPSPI for master or slave.
- static bool LPSPI_HAL_IsMaster (const LPSPI_Type ∗base)

  Returns whether the LPSPI module is in master mode.
- void LPSPI_HAL_GetFifoSizes (const LPSPI_Type ∗base, uint8_t ∗txFifoSize, uint8_t ∗rxFifoSize)

  Gets the TX and RX FIFO sizes of the LPSPI module.
- void LPSPI_HAL_SetFlushFifoCmd (LPSPI_Type ∗base, bool flushTxFifo, bool flushRxFifo)

  Flushes the LPSPI FIFOs.
- static void LPSPI_HAL_SetRxWatermarks (LPSPI_Type ∗base, uint32_t rxWater)

  Sets the RX FIFO watermark values.
- static void LPSPI_HAL_SetTxWatermarks (LPSPI_Type ∗base, uint32_t txWater)

  Sets the TX FIFO watermark values.

Status flags and Interrupt configuration

- static bool LPSPI_HAL_GetStatusFlag (const LPSPI_Type ∗base, lpspi_status_flag_t statusFlag)

  Gets the LPSPI status flag state.
- status_t LPSPI_HAL_ClearStatusFlag (LPSPI_Type ∗base, lpspi_status_flag_t statusFlag)

  Clears the LPSPI status flag.
- static void LPSPI_HAL_SetIntMode (LPSPI_Type ∗base, lpspi_status_flag_t interruptSrc, bool enable)

  Configures the LPSPI interrupts.
- static bool LPSPI_HAL_GetIntMode (const LPSPI_Type ∗base, lpspi_status_flag_t interruptSrc)

  Returns if the LPSPI interrupt request is enabled or disabled.

DMA configuration

- static void LPSPI_HAL_SetTxDmaCmd (LPSPI_Type ∗base, bool enable)

  Sets the LPSPI Transmit Data DMA configuration (enable or disable).
- static void LPSPI_HAL_SetRxDmaCmd (LPSPI_Type ∗base, bool enable)

  Sets the LPSPI Receive Data DMA configuration (enable or disable).

SPI Bus Configuration

- void LPSPI_HAL_SetHostRequestMode (LPSPI_Type ∗base, lpspi_host_request_select_t hostReqInput, lpspi_signal_polarity_t hostReqPol, bool enable)

  Configures the LPSPI Host Request input.
- status_t LPSPI_HAL_SetPcsPolarityMode (LPSPI_Type ∗base, lpspi_which_pcs_t whichPcs, lpspi_signal↩_polarity_t pcsPolarity)

  Configures the desired LPSPI PCS polarity.
- status_t LPSPI_HAL_SetMatchConfigMode (LPSPI_Type ∗base, lpspi_match_config_t matchCondition, bool rxDataMatchOnly, uint32_t match0, uint32_t match1)

  Configures the LPSPI data match configuration mode.
- status_t LPSPI_HAL_SetPinConfigMode (LPSPI_Type ∗base, lpspi_pin_config_t pinCfg, lpspi_data_out_↩config_t dataOutConfig, bool pcs3and2Enable)

  Configures the LPSPI SDO/SDI pin configuration mode.
- uint32_t LPSPI_HAL_SetBaudRate (LPSPI_Type ∗base, uint32_t bitsPerSec, uint32_t sourceClockInHz, uint32_t ∗tcrPrescaleValue)

  Sets the LPSPI baud rate in bits per second.
- status_t LPSPI_HAL_SetBaudRateDivisor (LPSPI_Type ∗base, uint32_t divisor)

  Configures the baud rate divisor manually (only the LPSPI_CCR[SCKDIV]).
- status_t LPSPI_HAL_SetDelay (LPSPI_Type ∗base, lpspi_delay_type_t whichDelay, uint32_t delay)

  Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

Data transfer

- void LPSPI_HAL_SetTxCommandReg (LPSPI_Type ∗base, const lpspi_tx_cmd_config_t ∗txCmdCfgSet)

  Sets the Transmit Command Register (TCR) parameters.
- static void LPSPI_HAL_WriteData (LPSPI_Type ∗base, uint32_t data)

  Writes data into the TX data buffer.
- void LPSPI_HAL_WriteDataBlocking (LPSPI_Type ∗base, uint32_t data)

  Writes a data into the TX data buffer and waits till complete to return.
- static uint32_t LPSPI_HAL_ReadData (const LPSPI_Type ∗base)

  Reads data from the data buffer.
- uint32_t LPSPI_HAL_ReadDataBlocking (const LPSPI_Type ∗base)

  Reads data from the data buffer but first waits till data is ready.
- static uint32_t LPSPI_HAL_ReadTxCount (const LPSPI_Type ∗base)

  Reads TX COUNT form the FIFO Status Register.
- static uint32_t LPSPI_HAL_ReadRxCount (const LPSPI_Type ∗base)

  Reads RX COUNT form the FIFO Status Register.
- static void LPSPI_HAL_ClearContCBit (LPSPI_Type ∗base)

  Clear CONTC bit form TCR Register.
- static void LPSPI_HAL_SetContCBit (LPSPI_Type ∗base)

  Set CONTC bit form TCR Register.
- static void LPSPI_HAL_SetClockPrescaler (LPSPI_Type ∗base, lpspi_prescaler_t prescaler)

  Configures the clock prescaler used for all LPSPI master logic.
- static lpspi_prescaler_t LPSPI_HAL_GetClockPrescaler (const LPSPI_Type ∗base)

  Get the clock prescaler used for all LPSPI master logic.

3.52.2 Data Structure Documentation

3.52.2.1 struct lpspi_tx_cmd_config_t

LPSPI Transmit Command Register configuration structure.

This structure contains the Transmit Command Register (TCR) settings. Any writes to the TCR will cause the entire TCR contents to be pushed to the TX FIFO. Therefore any updates to the TCR should include updates to all of the register bit fields to form a 32-bit write to the TCR. Implements : lpspi_tx_cmd_config_t_Class

Data Fields

- uint32_t frameSize
- lpspi_transfer_width_t width
- bool txMask
- bool rxMask
- bool contCmd
- bool contTransfer
- bool byteSwap
- bool lsbFirst
- lpspi_which_pcs_t whichPcs
- uint32_t preDiv
- lpspi_clock_phase_t clkPhase
- lpspi_sck_polarity_t clkPolarity

Field Documentation

3.52.2.1.1 byteSwap

```
bool byteSwap
```

Option to invoke the byte swap option in the FIFOs.

3.52.2.1.2 clkPhase

```
lpspi_clock_phase_t clkPhase
```

Selects which phase of clock to capture data.

3.52.2.1.3 clkPolarity

```
lpspi_sck_polarity_t clkPolarity
```

Selects clock polarity.

3.52.2.1.4 contCmd

```
bool contCmd
```

Master option to change cmd word within cont transfer.

3.52.2.1.5 contTransfer

```
bool contTransfer
```

Master option for continuous transfer.

3.52.2.1.6 frameSize

```
uint32_t frameSize
```

Number of bits/frame, minimum is 8-bits.

3.52.2.1.7 lsbFirst

```
bool lsbFirst
```

Option to transmit LSB first.

3.52.2.1.8 preDiv

```
uint32_t preDiv
```

Selects the baud rate prescaler divider TCR bit setting.

3.52.2.1.9 rxMask

```
bool rxMask
```

Option to mask the receive data (won't store in FIFO).

3.52.2.1.10 txMask

```
bool txMask
```

Option to mask the transmit data (won't load to FIFO).

3.52.2.1.11 whichPcs

```
lpspi_which_pcs_t whichPcs
```

Selects which PCS to use.

3.52.2.1.12 width

```
lpspi_transfer_width_t width
```

Transfer width, single, 2-bit, or 4-bit transfer.

3.52.2.2 struct lpspi_init_config_t

LPSPI initialization configuration structure.

This structure contains parameters for the user to fill in to configure the LPSPI. The user passes this structure into the LPSPI init function to configure it to their desired parameters. Example user code for:

- 60MHz assumed, check ref manual for exact value
- baudrate 500KHz
- master mode
- PCS is active low

```
lpspi_init_config_t lpspiCfg;
lpspiCfg.lpspiSrcClk = 60000000;
lpspiCfg.baudRate = 500000;
lpspiCfg.lpspiMode = LPSPI_MASTER;
lpspiCfg.pcsPol = LPSPI_ACTIVE_LOW;
```

Implements : lpspi_init_config_t_Class

Data Fields

- uint32_t lpspiSrcClk
- uint32_t baudRate
- lpspi_master_slave_mode_t lpspiMode
- lpspi_signal_polarity_t pcsPol

Field Documentation

3.52.2.2.1 baudRate

`uint32_t baudRate`

LPSPI baudrate

3.52.2.2.2 lpspiMode

`lpspi_master_slave_mode_t lpspiMode`

LPSPI master/slave mode

3.52.2.2.3 lpspiSrcClk

`uint32_t lpspiSrcClk`

LPSPI module clock

3.52.2.2.4 pcsPol

`lpspi_signal_polarity_t pcsPol`

LPSPI PCS polarity

3.52.3 Enumeration Type Documentation

3.52.3.1 lpspi_clock_phase_t

`enum lpspi_clock_phase_t`

LPSPI clock phase configuration. Implements : lpspi_clock_phase_t_Class.

Enumerator

LPSPI_CLOCK_PHASE_1ST_EDGE	Data captured on SCK 1st edge, changed on 2nd.
LPSPI_CLOCK_PHASE_2ND_EDGE	Data changed on SCK 1st edge, captured on 2nd.

3.52.3.2 lpspi_data_out_config_t

```
enum lpspi_data_out_config_t
```

LPSPI data output configuration. Implements : lpspi_data_out_config_t_Class.

Enumerator

LPSPI_DATA_OUT_RETAINED	Data out retains last value when chip select de-asserted
LPSPI_DATA_OUT_TRISTATE	Data out is tri-stated when chip select de-asserted

3.52.3.3 lpspi_delay_type_t

```
enum lpspi_delay_type_t
```

LPSPI delay type selection Implements : lpspi_delay_type_t_Class.

Enumerator

LPSPI_SCK_TO_PCS	SCK to PCS Delay
LPSPI_PCS_TO_SCK	PCS to SCK Delay
LPSPI_BETWEEN_TRANSFER	Delay between transfers

3.52.3.4 lpspi_host_request_select_t

```
enum lpspi_host_request_select_t
```

LPSPI Host Request select configuration. Implements : lpspi_host_request_select_t_Class.

Enumerator

LPSPI_HOST_REQ_EXT_PIN	Host Request is an ext pin.
LPSPI_HOST_REQ_INTERNAL_TRIGGER	Host Request is an internal trigger.

3.52.3.5 lpspi_master_slave_mode_t

```
enum lpspi_master_slave_mode_t
```

LPSPI master or slave configuration. Implements : lpspi_master_slave_mode_t_Class.

Enumerator

LPSPI_MASTER	LPSPI peripheral operates in master mode.
LPSPI_SLAVE	LPSPI peripheral operates in slave mode.

3.52.3.6 lpspi_match_config_t

```
enum lpspi_match_config_t
```

LPSPI Match configuration options. Implements : lpspi_match_config_t_Class.

Enumerator

LPSPI_MATCH_DISABLED	LPSPI Match Disabled.
LPSPI_1ST_WRD_EQUALS_M0_OR_M1	Match is enabled, if 1st data word equals MATCH0 OR MATCH1
LPSPI_ANY_WRD_EQUALS_M0_OR_M1	Match is enabled, if any data word equals MATCH0 OR MATCH1
LPSPI_1ST_WRD_EQUALS_M0_AND_2ND_WR↩D_EQUALS_M1	Match is enabled, if 1st data word equals MATCH0 AND 2nd data word equals MATCH1
LPSPI_ANY_WRD_EQUALS_M0_AND_NXT_WR↩D_EQUALS_M1	Match is enabled, if any data word equals MATCH0 AND the next data word equals MATCH1
LPSPI_1ST_WRD_AND_M1_EQUALS_M0_AND_↩M1	Match is enabled, if (1st data word AND MATCH1) equals (MATCH0 AND MATCH1)
LPSPI_ANY_WRD_AND_M1_EQUALS_M0_AND↩_M1	Match is enabled, if (any data word AND MATCH1) equals (MATCH0 AND MATCH1)

3.52.3.7 lpspi_pin_config_t

```
enum lpspi_pin_config_t
```

LPSPI pin (SDO and SDI) configuration. Implements : lpspi_pin_config_t_Class.

Enumerator

LPSPI_SDI_IN_SDO_OUT	LPSPI SDI input, SDO output.
LPSPI_SDI_IN_OUT	SDI is used for both input and output data.
LPSPI_SDO_IN_OUT	SDO is used for both input and output data.
LPSPI_SDI_OUT_SDO_IN	LPSPI SDO input, SDI output.

3.52.3.8 lpspi_prescaler_t

```
enum lpspi_prescaler_t
```

Prescaler values for LPSPI clock source. Implements : lpspi_prescaler_t_Class.

Enumerator

LPSPI_DIV_1	
LPSPI_DIV_2	
LPSPI_DIV_4	
LPSPI_DIV_8	
LPSPI_DIV_16	
LPSPI_DIV_32	
LPSPI_DIV_64	
LPSPI_DIV_128	

3.52.3.9 lpspi_sck_polarity_t

```
enum lpspi_sck_polarity_t
```

LPSPI Clock Signal (SCK) Polarity configuration. Implements : lpspi_sck_polarity_t_Class.

Enumerator

LPSPI_SCK_ACTIVE_HIGH	Signal is Active High (idles low).
LPSPI_SCK_ACTIVE_LOW	Signal is Active Low (idles high).

3.52.3.10 lpspi_signal_polarity_t

```
enum lpspi_signal_polarity_t
```

LPSPI Signal (PCS and Host Request) Polarity configuration. Implements : lpspi_signal_polarity_t_Class.

Enumerator

LPSPI_ACTIVE_HIGH	Signal is Active High (idles low).
LPSPI_ACTIVE_LOW	Signal is Active Low (idles high).

3.52.3.11 lpspi_status_flag_t

```
enum lpspi_status_flag_t
```

LPSPI status flags. Implements : lpspi_status_flag_t_Class.

Enumerator

LPSPI_TX_DATA_FLAG	TX data flag
LPSPI_RX_DATA_FLAG	RX data flag
LPSPI_WORD_COMPLETE	Word Complete flag
LPSPI_FRAME_COMPLETE	Frame Complete flag
LPSPI_TRANSFER_COMPLETE	Transfer Complete flag
LPSPI_TRANSMIT_ERROR	Transmit Error flag (FIFO underrun)
LPSPI_RECEIVE_ERROR	Receive Error flag (FIFO overrun)
LPSPI_DATA_MATCH	Data Match flag
LPSPI_MODULE_BUSY	Module Busy flag
LPSPI_ALL_STATUS	Used for clearing all w1c status flags

3.52.3.12 lpspi_transfer_width_t

```
enum lpspi_transfer_width_t
```

LPSPI transfer width configuration. Implements : lpspi_transfer_width_t_Class.

Enumerator

LPSPI_SINGLE_BIT_XFER	1-bit shift at a time, data out on SDO, in on SDI (normal mode)
LPSPI_TWO_BIT_XFER	2-bits shift out on SDO/SDI and in on SDO/SDI
LPSPI_FOUR_BIT_XFER	4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

3.52.3.13 lpspi_which_pcs_t

```
enum lpspi_which_pcs_t
```

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure). Implements : lpspi_which_pcs_t_↩ Class.

Enumerator

LPSPI_PCS0	PCS[0]
LPSPI_PCS1	PCS[1]
LPSPI_PCS2	PCS[2]
LPSPI_PCS3	PCS[3]

3.52.4 Function Documentation

3.52.4.1 LPSPI_HAL_ClearContCBit()

```
static void LPSPI_HAL_ClearContCBit (
            LPSPI_Type * base )  [inline], [static]
```

Clear CONTC bit form TCR Register.

This function clears the CONTC bit from the Transmit Command Register (TCR).

Parameters

base	Module base pointer of type LPSPI_Type. Implements : LPSPI_HAL_ClearContCBit_Activity

3.52.4.2 LPSPI_HAL_ClearStatusFlag()

```
status_t LPSPI_HAL_ClearStatusFlag (
            LPSPI_Type * base,
            lpspi_status_flag_t statusFlag )
```

Clears the LPSPI status flag.

This function clears the state of one of the LPSPI status flags as requested by the user. Note, the flag must be w1c capable, if not the function returns an error. w1c capable flags are: LPSPI_WORD_COMPLETE LPSPI↩ _FRAME_COMPLETE LPSPI_TRANSFER_COMPLETE LPSPI_TRANSMIT_ERROR LPSPI_RECEIVE_ERROR LPSPI_DATA_MATCH

Parameters

base	Module base pointer of type LPSPI_Type.
statusFlag	The status flag, of type lpspi_status_flag_t

Returns

STATUS_SUCCESS or LPSPI_STATUS_INVALID_PARAMETER

3.52.4.3 LPSPI_HAL_Config()

```
status_t LPSPI_HAL_Config (
             LPSPI_Type * base,
             const lpspi_init_config_t * config,
             lpspi_tx_cmd_config_t * txCmdCfgSet,
             uint32_t * actualBaudRate )
```

Configures the LPSPI registers to a user defined configuration.

Note, the LPSPI module must first be disabled prior to calling this function. It is recommended to first call the LP←SPI_HAL_Init function prior to calling this function. This function configures the LPSPI based on the configuration passed in by the user for normal SPI mode operation. Recommend single bit transfer: txCmd.width = LPSPI_SIN←GLE_BIT_XFER, otherwise you will have to call function LPSPI_HAL_SetPinConfigMode to change the pin config. This function sets the TX and RX FIFO watermarks to 0 such that the write blocking and read blocking functions can be used following the init.

Parameters

base	Module base pointer of type LPSPI_Type.
config	User configuration of type lpspi_init_config_t. Members of this struct are not modifiable by the function.
actualBaudRate	The actual, calculated baud rate passed back to the user
txCmdCfgSet	Structure that contains the Transmit Command Register (TCR) settings of type lpspi_tx_cmd_config_t. There is a member of this struct that is modifiable by the function.

Returns

This function returns STATUS_ERROR if it is detected that an error occured during the LPSPI setup, otherwise, if success, it returns STATUS_SUCCESS.

3.52.4.4 LPSPI_HAL_Disable()

```
status_t LPSPI_HAL_Disable (
             LPSPI_Type * base )
```

Disables the LPSPI module.

Parameters

base	Module base pointer of type LPSPI_Type.

Returns

This function returns STATUS_BUSY if it is detected that the Module Busy Flag (MBF) is set, otherwise, if success, it returns STATUS_SUCCESS.

3.52.4.5 LPSPI_HAL_Enable()

```
static void LPSPI_HAL_Enable (
             LPSPI_Type * base )  [inline], [static]
```

Enables the LPSPI module.

Parameters

base	Module base pointer of type LPSPI_Type. Implements : LPSPI_HAL_Enable_Activity

3.52.4.6 LPSPI_HAL_GetClockPrescaler()

```
static lpspi_prescaler_t LPSPI_HAL_GetClockPrescaler (
             const LPSPI_Type * base )  [inline], [static]
```

Get the clock prescaler used for all LPSPI master logic.

Parameters

base	Module base pointer of type LPSPI_Type.

Returns

Prescaler value for master logic. Implements : LPSPI_HAL_GetClockPrescaler_Activity

3.52.4.7 LPSPI_HAL_GetFifoSizes()

```
void LPSPI_HAL_GetFifoSizes (
             const LPSPI_Type * base,
             uint8_t * txFifoSize,
             uint8_t * rxFifoSize )
```

Gets the TX and RX FIFO sizes of the LPSPI module.

@ param base Module base pointer of type LPSPI_Type. @ param txFifoSize The TX FIFO size passed back to the user @ param rxFifoSize The RX FIFO size passed back to the user

3.52.4.8 LPSPI_HAL_GetIntMode()

```
static bool LPSPI_HAL_GetIntMode (
             const LPSPI_Type * base,
             lpspi_status_flag_t interruptSrc )  [inline], [static]
```

Returns if the LPSPI interrupt request is enabled or disabled.

Parameters

base	Module base pointer of type LPSPI_Type.
interruptSrc	The interrupt source, of type lpspi_status_flag_t

Returns

Returns if the interrupt source is enabled (true) or disabled (false) Implements : LPSPI_HAL_GetIntMode_↩ Activity

3.52.4.9 LPSPI_HAL_GetStatusFlag()

```
static bool LPSPI_HAL_GetStatusFlag (
             const LPSPI_Type * base,
             lpspi_status_flag_t statusFlag ) [inline], [static]
```

Gets the LPSPI status flag state.

This function returns the state of one of the LPSPI status flags as requested by the user.

Parameters

base	Module base pointer of type LPSPI_Type.
statusFlag	The status flag, of type lpspi_status_flag_t

Returns

State of the status flag: asserted (true) or not-asserted (false) Implements : LPSPI_HAL_GetStatusFlag_↩ Activity

3.52.4.10 LPSPI_HAL_GetVersionId()

```
void LPSPI_HAL_GetVersionId (
             const LPSPI_Type * base,
             uint32_t * major,
             uint32_t * minor,
             uint32_t * feature )
```

Gets the Major, Minor and Feature ID of the LPSPI module.

@ param base Module base pointer of type LPSPI_Type. @ param major The Major version number passed back to the user @ param minor The Minor version number passed back to the user @ param feature The Feature set number passed back to the user

3.52.4.11 LPSPI_HAL_Init()

```
void LPSPI_HAL_Init (
             LPSPI_Type * base )
```

Resets the LPSPI internal logic and registers to their default settings.

This function first performs a software reset of the LPSPI module which resets the internal LPSPI logic and most registers, then proceeds to manually reset all of the LPSPI registers to their default setting to ensuring these registers at programmed to their default value which includes disabling the module.

Parameters

Module	base pointer of type LPSPI_Type.

3.52.4.12 LPSPI_HAL_IsMaster()

```
static bool LPSPI_HAL_IsMaster (
            const LPSPI_Type * base )  [inline], [static]
```

Returns whether the LPSPI module is in master mode.

Parameters

base	Module base pointer of type LPSPI_Type.

Returns

Returns true if LPSPI in master mode or false if in slave mode. Implements : LPSPI_HAL_IsMaster_Activity

3.52.4.13 LPSPI_HAL_IsModuleEnabled()

```
static bool LPSPI_HAL_IsModuleEnabled (
            const LPSPI_Type * base )  [inline], [static]
```

Check if LPSPI module is enabled.

Parameters

Module	base pointer of type LPSPI_Type. Implements : LPSPI_HAL_IsModuleEnabled_Activity

3.52.4.14 LPSPI_HAL_ReadData()

```
static uint32_t LPSPI_HAL_ReadData (
            const LPSPI_Type * base )  [inline], [static]
```

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

base	Module base pointer of type LPSPI_Type.

Returns

The data read from the data buffer Implements : LPSPI_HAL_ReadData_Activity

3.52.4.15 LPSPI_HAL_ReadDataBlocking()

```
uint32_t LPSPI_HAL_ReadDataBlocking (
            const LPSPI_Type * base )
```

Reads data from the data buffer but first waits till data is ready.

This function reads the data from the Receive Data Register (RDR). However, before reading the data, it first waits till the read data ready status indicates the data is ready to be read. This function can be used for either master or slave mode. Note that it is required that the RX FIFO watermark be set to 0.

Parameters

base	Module base pointer of type LPSPI_Type.

Returns

The data read from the data buffer

3.52.4.16 LPSPI_HAL_ReadRxCount()

```
static uint32_t LPSPI_HAL_ReadRxCount (
            const LPSPI_Type * base ) [inline], [static]
```

Reads RX COUNT form the FIFO Status Register.

This function reads the RX COUNT field from the FIFO Status Register (FSR).

Parameters

base	Module base pointer of type LPSPI_Type.

Returns

The data read from the FIFO Status Register Implements : LPSPI_HAL_ReadRxCount_Activity

3.52.4.17 LPSPI_HAL_ReadTxCount()

```
static uint32_t LPSPI_HAL_ReadTxCount (
            const LPSPI_Type * base ) [inline], [static]
```

Reads TX COUNT form the FIFO Status Register.

This function reads the TX COUNT field from the FIFO Status Register (FSR).

Parameters

base	Module base pointer of type LPSPI_Type.

Returns

The data read from the FIFO Status Register Implements : LPSPI_HAL_ReadTxCount_Activity

3.52.4.18 LPSPI_HAL_SetBaudRate()

```
uint32_t LPSPI_HAL_SetBaudRate (
            LPSPI_Type * base,
            uint32_t bitsPerSec,
            uint32_t sourceClockInHz,
            uint32_t * tcrPrescaleValue )
```

Sets the LPSPI baud rate in bits per second.

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). Also note that the baud rate does not take into affect until the Transmit Control Register (TCR) is programmed with the PRESCALE value. Hence, this function returns the PRESCALE tcrPrescaleValue parameter for later programming in the TCR. It is up to the higher level peripheral driver to alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configure for master mode before configuring this.

Parameters

base	Module base pointer of type LPSPI_Type.
bitsPerSec	The desired baud rate in bits per second
sourceClockInHz	Module source input clock in Hertz
tcrPrescaleValue	The TCR PRESCALE value, needed by user to program the TCR

Returns

The actual calculated baud rate. This function may also return a "0" if the LPSPI is not configued for master mode or if the LPSPI module is not disabled.

3.52.4.19 LPSPI_HAL_SetBaudRateDivisor()

```
status_t LPSPI_HAL_SetBaudRateDivisor (
            LPSPI_Type * base,
            uint32_t divisor )
```

Configures the baud rate divisor manually (only the LPSPI_CCR[SCKDIV]).

This function allows the caller to manually set the baud rate divisor in the event that this divider is known and the caller does not wish to call the LPSPI_HAL_SetBaudRate function. Note that this only affects the LPSPI_CCR[S↩ CKDIV]). The Transmit Control Register (TCR) is programmed separately with the PRESCALE value. The valid range is 0x00 to 0xFF, if the user inputs outside of this range, an error is returned.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configure for master mode before configuring this.

Parameters

base	Module base pointer of type LPSPI_Type.
divisor	Desired baud rate divisor setting (0x00 to 0xFF)

Returns

STATUS_SUCCESS or LPSPI_STATUS_OUT_OF_RANGE if divisor > 0xFF

3.52.4.20 LPSPI_HAL_SetClockPrescaler()

```
static void LPSPI_HAL_SetClockPrescaler (
            LPSPI_Type * base,
            lpspi_prescaler_t prescaler )  [inline], [static]
```

Configures the clock prescaler used for all LPSPI master logic.

Parameters

base	Module base pointer of type LPSPI_Type.
prescaler	Prescaler value for master logic. Implements : LPSPI_HAL_SetClockPrescaler_Activity

3.52.4.21 LPSPI_HAL_SetContCBit()

```
static void LPSPI_HAL_SetContCBit (
            LPSPI_Type * base )  [inline], [static]
```

Set CONTC bit form TCR Register.

This function set the CONTC bit from the Transmit Command Register (TCR).

Parameters

base	Module base pointer of type LPSPI_Type. Implements : LPSPI_HAL_SetContCBit_Activity

3.52.4.22 LPSPI_HAL_SetDelay()

```
status_t LPSPI_HAL_SetDelay (
            LPSPI_Type * base,
            lpspi_delay_type_t whichDelay,
            uint32_t delay )
```

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the: SCK to PCS delay, or PCS to SCK delay, or Between transfer delay.

These delay names are available in type lpspi_delay_type_t.

The user passes which delay they want to configure along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configure for master mode before configuring this.

Parameters

base	Module base pointer of type LPSPI_Type.
whichDelay	The desired delay to configure, must be of type lpspi_delay_type_t
delay	The 8-bit delay value 0x00 to 0xFF (255). The delay is equal to: -delay + 1 cycles of the LPSPI baud rate clock (SCK to PCS and PCS to SCK) -delay + 2 cycles of the LPSPI baud rate clock (Between transfer delay)

Returns

Either STATUS_SUCCESS, LPSPI_STATUS_OUT_OF_RANGE, or STATUS_ERROR if LPSPI is not disabled or if is not set for master mode.

3.52.4.23 LPSPI_HAL_SetFlushFifoCmd()

```
void LPSPI_HAL_SetFlushFifoCmd (
            LPSPI_Type * base,
            bool flushTxFifo,
            bool flushRxFifo )
```

Flushes the LPSPI FIFOs.

Parameters

base	Module base pointer of type LPSPI_Type.
flushTxFifo	Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO
flushRxFifo	Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO

3.52.4.24 LPSPI_HAL_SetHostRequestMode()

```
void LPSPI_HAL_SetHostRequestMode (
            LPSPI_Type * base,
            lpspi_host_request_select_t hostReqInput,
            lpspi_signal_polarity_t hostReqPol,
            bool enable )
```

Configures the LPSPI Host Request input.

This function allows the user to configure the host request input pin as follows: Enable or disable the host request functionality. Select the polarity of the host request signal. Select the source of the host request (external signal or internal trigger).

Parameters

base	Module base pointer of type LPSPI_Type.
hostReqInput	Host request input source of type lpspi_host_request_select_t
hostReqPol	Host request polarity of type lpspi_signal_polarity_t
enable	Enable (true) or disable (false) the Host request feature

3.52.4.25 LPSPI_HAL_SetIntMode()

```
static void LPSPI_HAL_SetIntMode (
            LPSPI_Type * base,
            lpspi_status_flag_t interruptSrc,
            bool enable )  [inline], [static]
```

Configures the LPSPI interrupts.

Parameters

base	Module base pointer of type LPSPI_Type.
interruptSrc	The interrupt source, of type lpspi_status_flag_t
enable	Enable (true) or disable (false) the interrupt source Implements : LPSPI_HAL_SetIntMode_Activity

3.52.4.26 LPSPI_HAL_SetMasterSlaveMode()

```
status_t LPSPI_HAL_SetMasterSlaveMode (
            LPSPI_Type * base,
            lpspi_master_slave_mode_t mode )
```

Configures the LPSPI for master or slave.

Note that the LPSPI module must first be disabled before configuring this.

Parameters

base	Module base pointer of type LPSPI_Type.
mode	Mode setting (master or slave) of type lpspi_master_slave_mode_t

Returns

This function returns the error condition STATUS_ERROR if the module is not disabled else it returns STA↩ TUS_SUCCESS.

3.52.4.27 LPSPI_HAL_SetMatchConfigMode()

```
status_t LPSPI_HAL_SetMatchConfigMode (
            LPSPI_Type * base,
            lpspi_match_config_t matchCondition,
            bool rxDataMatchOnly,
            uint32_t match0,
            uint32_t match1 )
```

Configures the LPSPI data match configuration mode.

When enabled and configured to the desired condition of type lpspi_match_config_t, the LPSPI will assert the DMF status flag if the data match condition is met. Note that the LPSPI module must first be disabled before configuring this.

Parameters

base	Module base pointer of type LPSPI_Type.
matchCondition	Select condition for the data match (see lpspi_match_config_t)
rxDataMatchOnly	When enabled, all received data that does not cause RMF to set is discarded.
match0	Setting for Match0 value
match1	Setting for Match1 value

Returns

This function returns the error condition STATUS_ERROR if the module is not disabled else it returns STA↩ TUS_SUCCESS.

3.52.4.28 LPSPI_HAL_SetPcsPolarityMode()

```
status_t LPSPI_HAL_SetPcsPolarityMode (
            LPSPI_Type * base,
            lpspi_which_pcs_t whichPcs,
            lpspi_signal_polarity_t pcsPolarity )
```

Configures the desired LPSPI PCS polarity.

This function allows the user to configure the polarity of a particular PCS signal. Note that the LPSPI module must first be disabled before configuring this.

Parameters

base	Module base pointer of type LPSPI_Type.
whichPcs	Select which PCS to program, of type lpspi_which_pcs_t
pcsPolarity	Set PCS as active high or low, of type lpspi_signal_polarity_t

Returns

This function returns the error condition STATUS_ERROR if the module is not disabled else it returns STA↩ TUS_SUCCESS.

3.52.4.29 LPSPI_HAL_SetPinConfigMode()

```
status_t LPSPI_HAL_SetPinConfigMode (
            LPSPI_Type * base,
            lpspi_pin_config_t pinCfg,
            lpspi_data_out_config_t dataOutConfig,
            bool pcs3and2Enable )
```

Configures the LPSPI SDO/SDI pin configuration mode.

This function configures the pin mode of the LPSPI. For the SDI and SDO pins, the user can configure these pins as follows: SDI is used for input data and SDO for output data. SDO is used for input data and SDO for output data. SDI is used for input data and SDI for output data. SDO is used for input data and SDI for output data.

The user has the option to configure the output data as: Output data retains last value when chip select is de-asserted (default setting). Output data is tristated when chip select is de-asserted.

Finally, the user has the option to configure the PCS[3:2] pins as: Enabled for PCS operation (default setting). Disabled - this is need if the user wishes to configure the LPSPI mode for 4-bit transfers where these pins will be used as I/O data pins.

Note that the LPSPI module must first be disabled before configuring this.

Parameters

base	Module base pointer of type LPSPI_Type.
pinCfg	Select configuration for the SDO/SDI pins (see lpspi_pin_config_t)
dataOutConfig	Select data output config after chip select de-assertion
pcs3and2Enable	Enable or disable PCS[3:2]

Returns

This function returns the error condition STATUS_ERROR if the module is not disabled else it returns STA↩ TUS_SUCCESS.

3.52.4.30 LPSPI_HAL_SetRxDmaCmd()

```
static void LPSPI_HAL_SetRxDmaCmd (
             LPSPI_Type * base,
             bool enable )  [inline], [static]
```

Sets the LPSPI Receive Data DMA configuration (enable or disable).

Parameters

base	Module base pointer of type LPSPI_Type.
enable	Enable (true) or disable (false) the RX DMA request Implements : LPSPI_HAL_SetRxDmaCmd_Activity

3.52.4.31 LPSPI_HAL_SetRxWatermarks()

```
static void LPSPI_HAL_SetRxWatermarks (
             LPSPI_Type * base,
             uint32_t rxWater )  [inline], [static]
```

Sets the RX FIFO watermark values.

This function allows the user to set the RX FIFO watermarks.

Parameters

base	Module base pointer of type LPSPI_Type.
rxWater	The RX FIFO watermark value Implements : LPSPI_HAL_SetRxWatermarks_Activity

3.52.4.32 LPSPI_HAL_SetTxCommandReg()

```
void LPSPI_HAL_SetTxCommandReg (
             LPSPI_Type * base,
             const lpspi_tx_cmd_config_t * txCmdCfgSet )
```

Sets the Transmit Command Register (TCR) parameters.

The Transmit Command Register (TCR) contains multiple parameters that affect the transmission of data, such as clock phase and polarity, which PCS to use, whether or not the PCS remains asserted at the completion of a frame, etc. Any writes to this register results in an immediate push of the entire register and its contents to the TX FIFO. Hence, writes to this register should include all of the desired parameters written to the register at once. Hence, the user should fill in the members of the lpspi_tx_cmd_config_t data structure and pass this to the function.

Parameters

base	Module base pointer of type LPSPI_Type.
txCmdCfgSet	Structure that contains the Transmit Command Register (TCR) settings of type lpspi_tx_cmd_config_t

3.52.4.33 LPSPI_HAL_SetTxDmaCmd()

```
static void LPSPI_HAL_SetTxDmaCmd (
             LPSPI_Type * base,
             bool enable )  [inline], [static]
```

Sets the LPSPI Transmit Data DMA configuration (enable or disable).

Parameters

base	Module base pointer of type LPSPI_Type.
enable	Enable (true) or disable (false) the TX DMA request Implements : LPSPI_HAL_SetTxDmaCmd_Activity

3.52.4.34 LPSPI_HAL_SetTxWatermarks()

```
static void LPSPI_HAL_SetTxWatermarks (
             LPSPI_Type * base,
             uint32_t txWater )  [inline], [static]
```

Sets the TX FIFO watermark values.

This function allows the user to set the TX FIFO watermarks.

Parameters

base	Module base pointer of type LPSPI_Type.
txWater	The TX FIFO watermark value Implements : LPSPI_HAL_SetTxWatermarks_Activity

3.52.4.35 LPSPI_HAL_WriteData()

```
static void LPSPI_HAL_WriteData (
            LPSPI_Type * base,
            uint32_t data )  [inline], [static]
```

Writes data into the TX data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user will have to manage sending the data one 32-bit word at a time. Any writes to the TDR will result in an immediate push to the TX FIFO. This function can be used for either master or slave mode.

Parameters

base	Module base pointer of type LPSPI_Type.
data	The data word to be sent Implements : LPSPI_HAL_WriteData_Activity

3.52.4.36 LPSPI_HAL_WriteDataBlocking()

```
void LPSPI_HAL_WriteDataBlocking (
            LPSPI_Type * base,
            uint32_t data )
```

Writes a data into the TX data buffer and waits till complete to return.

This function writes the data to the Transmit Data Register (TDR) and waits for completion before returning. If the frame size exceeds 32-bits, the user will have to manage sending the data one 32-bit word at a time. This function can be used for either master or slave mode. Note that it is required that the TX FIFO watermark be set to 0.

Parameters

Module	base pointer of type LPSPI_Type.
data	The data word to be sent

3.52.5 Variable Documentation

3.52.5.1 s_baudratePrescaler

```
const uint32_t s_baudratePrescaler[] = { 1, 2, 4, 8, 16, 32, 64, 128 }  [static]
```

3.53 LPTMR Driver

3.53.1 Detailed Description

Low Power Timer Peripheral Driver.

The LPTMR is a configurable general-purpose 16-bit counter that has two operational modes: Timer and Pulse-↩ Counter.

Depending on the configured operational mode, the counter in the LPTMR can be incremented using a clock input (Timer mode) or an event counter (external events like button presses or internal events from different trigger souces).

Timer Mode

In Timer mode, the LPTMR increments the internal counter from a selectable clock source. An optional 16-bit prescaler can be configured.

Pulse-Counter Mode

In Pulse-Counter Mode, the LPTMR counter increments from a selectable trigger source, input pin, which can be an external event (like a button press) or internal events (like triggers from TRGMUX).

An optional 16-bit glitch-fiter can be configured to reject events that have a duration below a set period.

Initialization prerequisites

Before configuring the LPTMR, the peripheral clock must be must be enabled from the PCC module.

The peripheral clock must not be confused with the counter clock, which is selectable within the LPTMR.

Driver configuration

The LPTMR driver allows configuring the LPTMR for Pulse-Counter Mode or Timer Mode via the general configuration structure.

Configurable options:

- work mode (timer or pulse-counter)
- enable interrupts and DMA requests
- free running mode (overflow mode of the counter)
- compare value (interrupt generation on counter value)
- compare value measurement units (counter ticks or microseconds)
- input clock selection
- prescaler/glitch filter configuration
- enable bypass prescaler
- pin select (for pulse-counter mode)
- input pin and polarity (for pulse-counter mode)

```
/* LPTMR initialization of config structure */
lptmr_config_t config = {
  .workMode = LPTMR_WORKMODE_TIMER,
  .dmaRequest = false,
  .interruptEnable = false,
  .freeRun = false,
  .compareValue = 1000U,
  .counterUnits = LPTMR_COUNTER_UNITS_TICKS,
  .clockSelect = LPTMR_CLOCKSOURCE_SIRCDIV2,
  .prescaler = LPTMR_PRESCALE_2,
  .bypassPrescaler = false,
  .pinSelect = LPTMR_PINSELECT_TRGMUX,
  .pinPolarity = LPTMR_PINPOLARITY_RISING,
};

/* Enable peripheral clock for LPTMR */
PCC_HAL_SetClockSourceSel(PCC, PCC_LPTMR0_CLOCK,
      CLK_SRC_FIRC);
PCC_HAL_SetClockMode(PCC, PCC_LPTMR0_CLOCK, true);

/* Initialize the LPTMR and start the counter in a separate operation */
status = LPTMR_DRV_Init(0, &config, false);
LPTMR_DRV_StartCounter(0);
```

API

Some of the features exposed by the API are targeted specifically for Timer Mode or Pulse-Counter Mode. For example, configuring the Compare Value in microseconds makes sense only for Timer Mode, so therefor it is restricted for use in Pulse-Counter mode.

For any invalid configuration the functions will either return an error code or trigger DEV_ASSERT (if enabled). For more details, please refer to each function description.

Data Structures

- struct lptmr_config_t

  Defines the configuration structure for LPTMR. More...

Enumerations

- enum lptmr_counter_units_t { LPTMR_COUNTER_UNITS_TICKS = 0x00U, LPTMR_COUNTER_UNITS_↩ MICROSECONDS = 0x01U }

  Defines the LPTMR counter units available for configuring or reading the timer compare value.

LPTMR Driver Functions

- void LPTMR_DRV_InitConfigStruct (lptmr_config_t ∗const config)

  Initialize a configuration structure with default values.
- void LPTMR_DRV_Init (const uint32_t instance, const lptmr_config_t ∗const config, const bool startCounter)

  Initialize a LPTMR instance with values from an input configuration structure.
- void LPTMR_DRV_SetConfig (const uint32_t instance, const lptmr_config_t ∗const config)

  Configure a LPTMR instance.
- void LPTMR_DRV_GetConfig (const uint32_t instance, lptmr_config_t ∗const config)

  Get the current configuration of a LPTMR instance.
- void LPTMR_DRV_Deinit (const uint32_t instance)

  De-initialize a LPTMR instance.
- status_t LPTMR_DRV_SetCompareValueByCount (const uint32_t instance, const uint16_t compareValue↩ ByCount)

  Set the compare value in counter tick units, for a LPTMR instance.

- void LPTMR_DRV_GetCompareValueByCount (const uint32_t instance, uint16_t ∗const compareValueBy←Count)
  Get the compare value in counter tick units, of a LPTMR instance.
- status_t LPTMR_DRV_SetCompareValueByUs (const uint32_t instance, const uint32_t compareValueUs)
  Set the compare value for Timer Mode in microseconds, for a LPTMR instance.
- void LPTMR_DRV_GetCompareValueByUs (const uint32_t instance, uint32_t ∗const compareValueUs)
  Get the compare value in microseconds, of a LPTMR instance.
- bool LPTMR_DRV_GetCompareFlag (const uint32_t instance)
  Get the current state of the Compare Flag of a LPTMR instance.
- void LPTMR_DRV_ClearCompareFlag (const uint32_t instance)
  Clear the Compare Flag of a LPTMR instance.
- bool LPTMR_DRV_IsRunning (const uint32_t instance)
  Get the run state of a LPTMR instance.
- void LPTMR_DRV_SetInterrupt (const uint32_t instance, const bool enableInterrupt)
  Enable/disable the LPTMR interrupt.
- uint16_t LPTMR_DRV_GetCounterValueByCount (const uint32_t instance)
  Get the current counter value in counter tick units.
- void LPTMR_DRV_StartCounter (const uint32_t instance)
  Enable the LPTMR / Start the counter.
- void LPTMR_DRV_StopCounter (const uint32_t instance)
  Disable the LPTMR / Stop the counter.
- void LPTMR_DRV_SetPinConfiguration (const uint32_t instance, const lptmr_pinselect_t pinSelect, const lptmr_pinpolarity_t pinPolarity)
  Set the Input Pin configuration for Pulse Counter mode.

3.53.2 Data Structure Documentation

3.53.2.1 struct lptmr_config_t

Defines the configuration structure for LPTMR.

Implements : lptmr_config_t_Class

Data Fields

- bool dmaRequest
- bool interruptEnable
- bool freeRun
- lptmr_workmode_t workMode
- lptmr_clocksource_t clockSelect
- lptmr_prescaler_t prescaler
- bool bypassPrescaler
- uint32_t compareValue
- lptmr_counter_units_t counterUnits
- lptmr_pinselect_t pinSelect
- lptmr_pinpolarity_t pinPolarity

Field Documentation

3.53.2.1.1 bypassPrescaler

```
bool bypassPrescaler
```

Enable/Disable prescaler bypass

3.53.2.1.2 clockSelect

`lptmr_clocksource_t clockSelect`

Clock selection for Timer/Glitch filter

3.53.2.1.3 compareValue

`uint32_t compareValue`

Compare value

3.53.2.1.4 counterUnits

`lptmr_counter_units_t counterUnits`

Compare value units

3.53.2.1.5 dmaRequest

`bool dmaRequest`

Enable/Disable DMA requests

3.53.2.1.6 freeRun

`bool freeRun`

Enable/Disable Free Running Mode

3.53.2.1.7 interruptEnable

`bool interruptEnable`

Enable/Disable Interrupt

3.53.2.1.8 pinPolarity

`lptmr_pinpolarity_t pinPolarity`

Pin Polarity for Pulse-Counter

3.53.2.1.9 pinSelect

`lptmr_pinselect_t pinSelect`

Pin selection for Pulse-Counter

3.53.2.1.10 prescaler

`lptmr_prescaler_t prescaler`

Prescaler Selection

3.53.2.1.11 workMode

`lptmr_workmode_t workMode`

Time/Pulse Counter Mode

3.53.3 Enumeration Type Documentation

3.53.3.1 lptmr_counter_units_t

`enum lptmr_counter_units_t`

Defines the LPTMR counter units available for configuring or reading the timer compare value.

Implements : lptmr_counter_units_t_Class

Enumerator

LPTMR_COUNTER_UNITS_TICKS	
LPTMR_COUNTER_UNITS_MICROSECONDS	

3.53.4 Function Documentation

3.53.4.1 LPTMR_DRV_ClearCompareFlag()

```
void LPTMR_DRV_ClearCompareFlag (
            const uint32_t instance )
```

Clear the Compare Flag of a LPTMR instance.

Parameters

in	*instance*	- LPTMR instance number

3.53.4.2 LPTMR_DRV_Deinit()

```
void LPTMR_DRV_Deinit (
            const uint32_t instance )
```

De-initialize a LPTMR instance.

Parameters

in	*instance*	- LPTMR instance number

3.53.4.3 LPTMR_DRV_GetCompareFlag()

```
bool LPTMR_DRV_GetCompareFlag (
            const uint32_t instance )
```

Get the current state of the Compare Flag of a LPTMR instance.

Parameters

in	*instance*	- LPTMR instance number

Returns

the state of the Compare Flag

3.53.4.4 LPTMR_DRV_GetCompareValueByCount()

```
void LPTMR_DRV_GetCompareValueByCount (
            const uint32_t instance,
            uint16_t *const compareValueByCount )
```

Get the compare value in counter tick units, of a LPTMR instance.

Parameters

in	*instance*	- LPTMR instance number
out	*compareValueByCount*	- pointer to current compare value, in counter ticks

3.53.4.5 LPTMR_DRV_GetCompareValueByUs()

```
void LPTMR_DRV_GetCompareValueByUs (
            const uint32_t instance,
            uint32_t *const compareValueUs )
```

Get the compare value in microseconds, of a LPTMR instance.

Parameters

in	*instance*	- LPTMR instance number
out	*compareValueUs*	- pointer to current compare value, in microseconds

3.53.4.6 LPTMR_DRV_GetConfig()

```
void LPTMR_DRV_GetConfig (
            const uint32_t instance,
            lptmr_config_t *const config )
```

Get the current configuration of a LPTMR instance.

Parameters

in	*instance*	- LPTMR instance number
out	*config*	- pointer to the output configuration structure

3.53.4.7 LPTMR_DRV_GetCounterValueByCount()

```
uint16_t LPTMR_DRV_GetCounterValueByCount (
            const uint32_t instance )
```

Get the current counter value in counter tick units.

Parameters

in	*instance*	- LPTMR instance number

Returns

the current counter value

3.53.4.8 LPTMR_DRV_Init()

```
void LPTMR_DRV_Init (
            const uint32_t instance,
            const lptmr_config_t *const config,
            const bool startCounter )
```

Initialize a LPTMR instance with values from an input configuration structure.

When (counterUnits == LPTMR_COUNTER_UNITS_MICROSECONDS) the function will automatically configure the timer for the input compareValue in microseconds. The input params for 'prescaler' and 'bypassPrescaler' will be ignored - their values will be adapted by the function, to best fit the input compareValue (in microseconds) for the operating clock frequency.

LPTMR_COUNTER_UNITS_MICROSECONDS may only be used for LPTMR_WORKMODE_TIMER mode. Otherwise the function shall not convert 'compareValue' in ticks and this is likely to cause erroneous behavior.

When (counterUnits == LPTMR_COUNTER_UNITS_TICKS) the function will use the 'prescaler' and 'bypass↩Prescaler' provided in the input config structure.

Parameters

in	*instance*	- LPTMR instance number
in	*config*	- pointer to the input configuration structure
in	*startCounter*	- flag for starting the counter immediately after configuration

3.53.4.9 LPTMR_DRV_InitConfigStruct()

```
void LPTMR_DRV_InitConfigStruct (
            lptmr_config_t *const config )
```

Initialize a configuration structure with default values.

Parameters

out	*config*	- pointer to the configuration structure to be initialized

3.53.4.10 LPTMR_DRV_IsRunning()

```
bool LPTMR_DRV_IsRunning (
            const uint32_t instance )
```

Get the run state of a LPTMR instance.

Parameters

in	*instance*	- LPTMR instance number

Returns

the run state of the LPTMR instance:

- true: Timer/Counter started
- false: Timer/Counter stopped

3.53.4.11 LPTMR_DRV_SetCompareValueByCount()

```
status_t LPTMR_DRV_SetCompareValueByCount (
            const uint32_t instance,
            const uint16_t compareValueByCount )
```

Set the compare value in counter tick units, for a LPTMR instance.

Parameters

in	*instance*	- LPTMR instance number
in	*compareValueByCount*	- the compare value in counter ticks, to be written

Returns

One of the possible status codes:

- STATUS_SUCCESS: completed successfully
- STATUS_ERROR: cannot reconfigure compare value (TCF not set)
- STATUS_TIMEOUT: compare value greater then current counter value

3.53.4.12 LPTMR_DRV_SetCompareValueByUs()

```
status_t LPTMR_DRV_SetCompareValueByUs (
            const uint32_t instance,
            const uint32_t compareValueUs )
```

Set the compare value for Timer Mode in microseconds, for a LPTMR instance.

Parameters

in	*instance*	- LPTMR peripheral instance number
in	*compareValueUs*	- compare value in microseconds

Returns

One of the possible status codes:

- STATUS_SUCCESS: completed successfully
- STATUS_ERROR: cannot reconfigure compare value
- STATUS_TIMEOUT: compare value greater then current counter value

3.53.4.13 LPTMR_DRV_SetConfig()

```
void LPTMR_DRV_SetConfig (
            const uint32_t instance,
            const lptmr_config_t *const config )
```

Configure a LPTMR instance.

When (counterUnits == LPTMR_COUNTER_UNITS_MICROSECONDS) the function will automatically configure the timer for the input compareValue in microseconds. The input params for 'prescaler' and 'bypassPrescaler' will be ignored - their values will be adapted by the function, to best fit the input compareValue (in microseconds) for the operating clock frequency.

LPTMR_COUNTER_UNITS_MICROSECONDS may only be used for LPTMR_WORKMODE_TIMER mode. Otherwise the function shall not convert 'compareValue' in ticks and this is likely to cause erroneous behavior.

When (counterUnits == LPTMR_COUNTER_UNITS_TICKS) the function will use the 'prescaler' and 'bypass↩ Prescaler' provided in the input config structure.

Parameters

in	*instance*	- LPTMR instance number
in	*config*	- pointer to the input configuration structure

3.53.4.14 LPTMR_DRV_SetInterrupt()

```
void LPTMR_DRV_SetInterrupt (
            const uint32_t instance,
            const bool enableInterrupt )
```

Enable/disable the LPTMR interrupt.

Parameters

in	*instance*	- LPTMR instance number
in	*enableInterrupt*	- the new state of the LPTMR interrupt enable flag.

3.53.4.15 LPTMR_DRV_SetPinConfiguration()

```
void LPTMR_DRV_SetPinConfiguration (
            const uint32_t instance,
            const lptmr_pinselect_t pinSelect,
            const lptmr_pinpolarity_t pinPolarity )
```

Set the Input Pin configuration for Pulse Counter mode.

Parameters

in	*instance*	- LPTMR instance number
in	*pinSelect*	- LPTMR pin selection
in	*pinPolarity*	- polarity on which to increment counter (rising/falling edge)

3.53.4.16 LPTMR_DRV_StartCounter()

```
void LPTMR_DRV_StartCounter (
            const uint32_t instance )
```

Enable the LPTMR / Start the counter.

Parameters

in	*instance*	- LPTMR instance number

3.53.4.17 LPTMR_DRV_StopCounter()

```
void LPTMR_DRV_StopCounter (
             const uint32_t instance )
```

Disable the LPTMR / Stop the counter.

Parameters

in	*instance*	- LPTMR instance number

3.54 LPTMR HAL

3.54.1 Detailed Description

Low Power Timer Hardware Abstraction Layer.

This HAL provides low-level access to all hardware features of the LPTMR.

Enumerations

- enum lptmr_pinselect_t { LPTMR_PINSELECT_TRGMUX = 0x00u, LPTMR_PINSELECT_ALT1 = 0x01u, LPTMR_PINSELECT_ALT2 = 0x02u, LPTMR_PINSELECT_ALT3 = 0x03u }

  Pulse Counter Input selection Implements : lptmr_pinselect_t_Class.
- enum lptmr_pinpolarity_t { LPTMR_PINPOLARITY_RISING = 0u, LPTMR_PINPOLARITY_FALLING = 1u }

  Pulse Counter input polarity Implements : lptmr_pinpolarity_t_Class.
- enum lptmr_workmode_t { LPTMR_WORKMODE_TIMER = 0u, LPTMR_WORKMODE_PULSECOUNTER = 1u }

  Work Mode Implements : lptmr_workmode_t_Class.
- enum lptmr_prescaler_t {
  LPTMR_PRESCALE_2 = 0x00u, LPTMR_PRESCALE_4_GLITCHFILTER_2 = 0x01u, LPTMR_PRESCA↩LE_8_GLITCHFILTER_4 = 0x02u, LPTMR_PRESCALE_16_GLITCHFILTER_8 = 0x03u,
  LPTMR_PRESCALE_32_GLITCHFILTER_16 = 0x04u, LPTMR_PRESCALE_64_GLITCHFILTER_32 = 0x05u, LPTMR_PRESCALE_128_GLITCHFILTER_64 = 0x06u, LPTMR_PRESCALE_256_GLITCHFILT↩ER_128 = 0x07u,
  LPTMR_PRESCALE_512_GLITCHFILTER_256 = 0x08u, LPTMR_PRESCALE_1024_GLITCHFILTER_512 = 0x09u, LPTMR_PRESCALE_2048_GLITCHFILTER_1024 = 0x0Au, LPTMR_PRESCALE_4096_GLITC↩HFILTER_2048 = 0x0Bu,
  LPTMR_PRESCALE_8192_GLITCHFILTER_4096 = 0x0Cu, LPTMR_PRESCALE_16384_GLITCHFILTE↩R_8192 = 0x0Du, LPTMR_PRESCALE_32768_GLITCHFILTER_16384 = 0x0Eu, LPTMR_PRESCALE_↩65536_GLITCHFILTER_32768 = 0x0Fu }

  Prescaler Selection Implements : lptmr_prescaler_t_Class.
- enum lptmr_clocksource_t { LPTMR_CLOCKSOURCE_SIRCDIV2 = 0x00u, LPTMR_CLOCKSOURCE_1↩KHZ_LPO = 0x01u, LPTMR_CLOCKSOURCE_RTC = 0x02u, LPTMR_CLOCKSOURCE_PCC = 0x03u }

  Clock Source selection Implements : lptmr_clocksource_t_Class.

LPTMR HAL Functions

- void LPTMR_HAL_Init (LPTMR_Type ∗const base)

  Initialize the LPTMR instance to reset values.
- static bool LPTMR_HAL_GetDmaRequest (const LPTMR_Type ∗const base)

  Get the DMA Request Enable Flag.
- static void LPTMR_HAL_SetDmaRequest (LPTMR_Type ∗const base, bool enable)

  Configure the DMA Request Enable Flag state.
- static bool LPTMR_HAL_GetCompareFlag (const LPTMR_Type ∗const base)

  Get the Compare Flag state.
- static void LPTMR_HAL_ClearCompareFlag (LPTMR_Type ∗const base)

  Clear the Compare Flag.
- static bool LPTMR_HAL_GetInterruptEnable (const LPTMR_Type ∗const base)

  Get the Interrupt Enable state.
- static void LPTMR_HAL_SetInterrupt (LPTMR_Type ∗const base, bool enable)

  Configure the Interrupt Enable state.

- static lptmr_pinselect_t LPTMR_HAL_GetPinSelect (const LPTMR_Type ∗const base)

    Get the Pin select for Counter Mode.
- static void LPTMR_HAL_SetPinSelect (LPTMR_Type ∗const base, const lptmr_pinselect_t pinsel)

    Configure the Pin selection for Pulse Counter Mode.
- static lptmr_pinpolarity_t LPTMR_HAL_GetPinPolarity (const LPTMR_Type ∗const base)

    Get Pin Polarity for Pulse Counter Mode.
- static void LPTMR_HAL_SetPinPolarity (LPTMR_Type ∗const base, const lptmr_pinpolarity_t pol)

    Configure Pin Polarity for Pulse Counter Mode.
- static bool LPTMR_HAL_GetFreeRunning (const LPTMR_Type ∗const base)

    Get the Free Running state.
- static void LPTMR_HAL_SetFreeRunning (LPTMR_Type ∗const base, const bool enable)

    Configure the Free Running state.
- static lptmr_workmode_t LPTMR_HAL_GetWorkMode (const LPTMR_Type ∗const base)

    Get current Work Mode.
- static void LPTMR_HAL_SetWorkMode (LPTMR_Type ∗const base, const lptmr_workmode_t mode)

    Configure the Work Mode.
- static bool LPTMR_HAL_GetEnable (const LPTMR_Type ∗const base)

    Get the Enable state.
- static void LPTMR_HAL_Enable (LPTMR_Type ∗const base)

    Enable the LPTMR.
- static void LPTMR_HAL_Disable (LPTMR_Type ∗const base)

    Disable the LPTMR.
- static lptmr_prescaler_t LPTMR_HAL_GetPrescaler (const LPTMR_Type ∗const base)

    Get Prescaler/Glitch Filter divider value.
- static void LPTMR_HAL_SetPrescaler (LPTMR_Type ∗const base, const lptmr_prescaler_t presc)

    Configure the Prescaler/Glitch Filter divider value.
- static bool LPTMR_HAL_GetBypass (const LPTMR_Type ∗const base)

    Get the Prescaler/Glitch Filter Bypass enable state.
- static void LPTMR_HAL_SetBypass (LPTMR_Type ∗const base, const bool enable)

    Configure the Prescaler/Glitch Filter Bypass enable state.
- static lptmr_clocksource_t LPTMR_HAL_GetClockSelect (const LPTMR_Type ∗const base)

    Get the LPTMR input Clock selection.
- static void LPTMR_HAL_SetClockSelect (LPTMR_Type ∗const base, const lptmr_clocksource_t clocksel)

    Configure the LPTMR input Clock selection.
- static uint16_t LPTMR_HAL_GetCompareValue (const LPTMR_Type ∗const base)

    Get the Compare Value.
- static void LPTMR_HAL_SetCompareValue (LPTMR_Type ∗const base, const uint16_t compval)

    Configure the Compare Value.
- static uint16_t LPTMR_HAL_GetCounterValue (LPTMR_Type ∗const base)

    Get the current Counter Value.

3.54.2 Enumeration Type Documentation

3.54.2.1 lptmr_clocksource_t

```
enum lptmr_clocksource_t
```

Clock Source selection Implements : lptmr_clocksource_t_Class.

Enumerator

LPTMR_CLOCKSOURCE_SIRCDIV2	SIRC clock
LPTMR_CLOCKSOURCE_1KHZ_LPO	1kHz LPO clock
LPTMR_CLOCKSOURCE_RTC	RTC clock
LPTMR_CLOCKSOURCE_PCC	PCC configured clock

3.54.2.2 lptmr_pinpolarity_t

```
enum lptmr_pinpolarity_t
```

Pulse Counter input polarity Implements : lptmr_pinpolarity_t_Class.

Enumerator

LPTMR_PINPOLARITY_RISING	Count pulse on rising edge
LPTMR_PINPOLARITY_FALLING	Count pulse on falling edge

3.54.2.3 lptmr_pinselect_t

```
enum lptmr_pinselect_t
```

Pulse Counter Input selection Implements : lptmr_pinselect_t_Class.

Enumerator

LPTMR_PINSELECT_TRGMUX	Count pulses from TRGMUX trigger
LPTMR_PINSELECT_ALT1	Count pulses from pin alternative 1
LPTMR_PINSELECT_ALT2	Count pulses from pin alternative 2
LPTMR_PINSELECT_ALT3	Count pulses from pin alternative 3

3.54.2.4 lptmr_prescaler_t

```
enum lptmr_prescaler_t
```

Prescaler Selection Implements : lptmr_prescaler_t_Class.

Enumerator

LPTMR_PRESCALE_2	Timer mode: prescaler 2, Glitch filter mode: invalid
LPTMR_PRESCALE_4_GLITCHFILTER_2	Timer mode: prescaler 4, Glitch filter mode: 2 clocks
LPTMR_PRESCALE_8_GLITCHFILTER_4	Timer mode: prescaler 8, Glitch filter mode: 4 clocks
LPTMR_PRESCALE_16_GLITCHFILTER_8	Timer mode: prescaler 16, Glitch filter mode: 8 clocks
LPTMR_PRESCALE_32_GLITCHFILTER_16	Timer mode: prescaler 32, Glitch filter mode: 16 clocks
LPTMR_PRESCALE_64_GLITCHFILTER_32	Timer mode: prescaler 64, Glitch filter mode: 32 clocks
LPTMR_PRESCALE_128_GLITCHFILTER_64	Timer mode: prescaler 128, Glitch filter mode: 64 clocks

Enumerator

LPTMR_PRESCALE_256_GLITCHFILTER_128	Timer mode: prescaler 256, Glitch filter mode: 128 clocks
LPTMR_PRESCALE_512_GLITCHFILTER_256	Timer mode: prescaler 512, Glitch filter mode: 256 clocks
LPTMR_PRESCALE_1024_GLITCHFILTER_512	Timer mode: prescaler 1024, Glitch filter mode: 512 clocks
LPTMR_PRESCALE_2048_GLITCHFILTER_1024	Timer mode: prescaler 2048, Glitch filter mode: 1024 clocks
LPTMR_PRESCALE_4096_GLITCHFILTER_2048	Timer mode: prescaler 4096, Glitch filter mode: 2048 clocks
LPTMR_PRESCALE_8192_GLITCHFILTER_4096	Timer mode: prescaler 8192, Glitch filter mode: 4096 clocks
LPTMR_PRESCALE_16384_GLITCHFILTER_8192	Timer mode: prescaler 16384, Glitch filter mode: 8192 clocks
LPTMR_PRESCALE_32768_GLITCHFILTER_16384	Timer mode: prescaler 32768, Glitch filter mode: 16384 clocks
LPTMR_PRESCALE_65536_GLITCHFILTER_32768	Timer mode: prescaler 65536, Glitch filter mode: 32768 clocks

3.54.2.5 lptmr_workmode_t

```
enum lptmr_workmode_t
```

Work Mode Implements : lptmr_workmode_t_Class.

Enumerator

LPTMR_WORKMODE_TIMER	Timer
LPTMR_WORKMODE_PULSECOUNTER	Pulse counter

3.54.3 Function Documentation

3.54.3.1 LPTMR_HAL_ClearCompareFlag()

```
static void LPTMR_HAL_ClearCompareFlag (
             LPTMR_Type *const base )  [inline], [static]
```

Clear the Compare Flag.

This function clears the Compare Flag/Interrupt Pending state.

Parameters

in	*base*	- lptmr base pointer

Implements : LPTMR_HAL_ClearCompareFlag_Activity

3.54.3.2 LPTMR_HAL_Disable()

```
static void LPTMR_HAL_Disable (
             LPTMR_Type *const base )  [inline], [static]
```

Disable the LPTMR.

Disable the LPTMR. Stop the Counter/Timer and allow reconfiguration.

Parameters

in	*base*	- lptmr base pointer

Implements : LPTMR_HAL_Disable_Activity

3.54.3.3 LPTMR_HAL_Enable()

```
static void LPTMR_HAL_Enable (
             LPTMR_Type *const base )  [inline], [static]
```

Enable the LPTMR.

Enable the LPTMR. Starts the timer/counter.

Parameters

in	*base*	- lptmr base pointer

Implements : LPTMR_HAL_Enable_Activity

3.54.3.4 LPTMR_HAL_GetBypass()

```
static bool LPTMR_HAL_GetBypass (
             const LPTMR_Type *const base )  [inline], [static]
```

Get the Prescaler/Glitch Filter Bypass enable state.

This function checks whether the Prescaler/Glitch Filter Bypass is enabled.

Parameters

in	*base*	- lptmr base pointer

Returns

the Prescaler Bypass state
- true: Prescaler/Glitch Filter Bypass enabled
- false: Prescaler/Glitch Filter Bypass disabled

Implements : LPTMR_HAL_GetBypass_Activity

3.54.3.5 LPTMR_HAL_GetClockSelect()

```
static lptmr_clocksource_t LPTMR_HAL_GetClockSelect (
            const LPTMR_Type *const base )  [inline], [static]
```

Get the LPTMR input Clock selection.

This function returns the current configured input Clock for the LPTMR.

Parameters

in	*base*	- lptmr base pointer

Returns

the Clock source

- LPTMR_CLOCKSOURCE_SIRCDIV2: clock from SIRC DIV2
- LPTMR_CLOCKSOURCE_1KHZ_LPO: clock from 1kHz LPO
- LPTMR_CLOCKSOURCE_RTC: clock from RTC
- LPTMR_CLOCKSOURCE_PCC: clock from PCC

Implements : LPTMR_HAL_GetClockSelect_Activity

3.54.3.6 LPTMR_HAL_GetCompareFlag()

```
static bool LPTMR_HAL_GetCompareFlag (
            const LPTMR_Type *const base )  [inline], [static]
```

Get the Compare Flag state.

This function checks whether a Compare Match event has occurred or if there is an Interrupt Pending.

Parameters

in	*base*	- lptmr base pointer

Returns

the Compare Flag state

- true: Compare Match/Interrupt Pending asserted
- false: Compare Match/Interrupt Pending not asserted

Implements : LPTMR_HAL_GetCompareFlag_Activity

3.54.3.7 LPTMR_HAL_GetCompareValue()

```
static uint16_t LPTMR_HAL_GetCompareValue (
            const LPTMR_Type *const base )  [inline], [static]
```

Get the Compare Value.

This function returns the current Compare Value.

Parameters

in	*base*	- lptmr base pointer

Returns

the Compare Value

Implements : LPTMR_HAL_GetCompareValue_Activity

3.54.3.8 LPTMR_HAL_GetCounterValue()

```
static uint16_t LPTMR_HAL_GetCounterValue (
            LPTMR_Type *const base )  [inline], [static]
```

Get the current Counter Value.

This function returns the Counter Value.

Parameters

in	*base*	- lptmr base pointer

Returns

The Counter Value

Implements : LPTMR_HAL_GetCounterValue_Activity

3.54.3.9 LPTMR_HAL_GetDmaRequest()

```
static bool LPTMR_HAL_GetDmaRequest (
            const LPTMR_Type *const base )  [inline], [static]
```

Get the DMA Request Enable Flag.

This function checks whether a DMA Request feature of the LPTMR is enabled. The DMA Request is issued when a Compare Match is asserted. If enabled, the Compare Match/Interrupt Pending flag is cleared when the DMA controller is done.

Parameters

in	*base*	- lptmr base pointer

Returns

DMA Request enable

- true: enable DMA Request
- false: disable DMA Request

Implements : LPTMR_HAL_GetDmaRequest_Activity

3.54.3.10 LPTMR_HAL_GetEnable()

```
static bool LPTMR_HAL_GetEnable (
            const LPTMR_Type *const base )  [inline], [static]
```

Get the Enable state.

Prior to reconfiguring the LPTMR, it is necessary to disable it.

Parameters

in	*base*	- lptmr base pointer

Returns

The state of the LPTMR
- true: LPTMR enabled
- false: LPTMR disabled

Implements : LPTMR_HAL_GetEnable_Activity

3.54.3.11 LPTMR_HAL_GetFreeRunning()

```
static bool LPTMR_HAL_GetFreeRunning (
            const LPTMR_Type *const base )  [inline], [static]
```

Get the Free Running state.

This function checks whether the Free Running feature of the LPTMR is enabled or disabled.

Parameters

in	*base*	- lptmr base pointer

Returns

Free running mode state
- true: Free Running Mode enabled. Reset counter on 16-bit overflow
- false: Free Running Mode disabled. Reset counter on Compare Match.

Implements : LPTMR_HAL_GetFreeRunning_Activity

3.54.3.12 LPTMR_HAL_GetInterruptEnable()

```
static bool LPTMR_HAL_GetInterruptEnable (
            const LPTMR_Type *const base )  [inline], [static]
```

Get the Interrupt Enable state.

This function returns the Interrupt Enable state for the LPTMR. If enabled, an interrupt is generated when a Compare Match event occurs.

Parameters

in	*base*	- lptmr base pointer

Returns

Interrupt Enable state

- true: Interrupt enabled
- false: Interrupt disabled

Implements : LPTMR_HAL_GetInterruptEnable_Activity

3.54.3.13 LPTMR_HAL_GetPinPolarity()

```
static lptmr_pinpolarity_t LPTMR_HAL_GetPinPolarity (
             const LPTMR_Type *const base )  [inline], [static]
```

Get Pin Polarity for Pulse Counter Mode.

This function returns the configured pin polarity that triggers an increment in Pulse Counter Mode.

Parameters

in	*base*	- lptmr base pointer

Returns

the pin polarity for Pulse Counter Mode

- LPTMR_PINPOLARITY_RISING: count pulse on Rising Edge
- LPTMR_PINPOLARITY_FALLING: count pulse on Falling Edge

Implements : LPTMR_HAL_GetPinPolarity_Activity

3.54.3.14 LPTMR_HAL_GetPinSelect()

```
static lptmr_pinselect_t LPTMR_HAL_GetPinSelect (
             const LPTMR_Type *const base )  [inline], [static]
```

Get the Pin select for Counter Mode.

This function returns the configured Input Pin for Pulse Counter Mode.

Parameters

in	*base*	- lptmr base pointer

Returns

Input pin selection

- LPTMR_PINSELECT_TRGMUX: count pulses from TRGMUX output
- LPTMR_PINSELECT_ALT1: count pulses from pin alt 1
- LPTMR_PINSELECT_ALT2: count pulses from pin alt 2
- LPTMR_PINSELECT_ALT3: count pulses from pin alt 3

Implements : LPTMR_HAL_GetPinSelect_Activity

3.54.3.15 LPTMR_HAL_GetPrescaler()

```
static lptmr_prescaler_t LPTMR_HAL_GetPrescaler (
            const LPTMR_Type *const base )  [inline], [static]
```

Get Prescaler/Glitch Filter divider value.

This function returns the currently configured Prescaler/Glitch Filter divider value.

Parameters

in	*base*	- lptmr base pointer

Returns

The Prescaler/Glitch filter value

- LPTMR_PRESCALE_2: Timer mode: prescaler 2, Glitch filter mode: invalid
- LPTMR_PRESCALE_4_GLITCHFILTER_2: Timer mode: prescaler 4, Glitch filter mode: 2 clocks
- LPTMR_PRESCALE_8_GLITCHFILTER_4: Timer mode: prescaler 8, Glitch filter mode: 4 clocks
- LPTMR_PRESCALE_16_GLITCHFILTER_8: Timer mode: prescaler 16, Glitch filter mode: 8 clocks
- LPTMR_PRESCALE_32_GLITCHFILTER_16: Timer mode: prescaler 32, Glitch filter mode: 16 clocks
- LPTMR_PRESCALE_64_GLITCHFILTER_32: Timer mode: prescaler 64, Glitch filter mode: 32 clocks
- LPTMR_PRESCALE_128_GLITCHFILTER_64: Timer mode: prescaler 128, Glitch filter mode: 64 clocks
- LPTMR_PRESCALE_256_GLITCHFILTER_128: Timer mode: prescaler 256, Glitch filter mode: 128 clocks
- LPTMR_PRESCALE_512_GLITCHFILTER_256: Timer mode: prescaler 512, Glitch filter mode: 256 clocks
- LPTMR_PRESCALE_1024_GLITCHFILTER_512: Timer mode: prescaler 1024, Glitch filter mode: 512 clocks
- LPTMR_PRESCALE_2048_GLITCHFILTER_1024: Timer mode: prescaler 2048, Glitch filter mode↩ : 1024 clocks
- LPTMR_PRESCALE_4096_GLITCHFILTER_2048: Timer mode: prescaler 4096, Glitch filter mode↩ : 2048 clocks
- LPTMR_PRESCALE_8192_GLITCHFILTER_4096: Timer mode: prescaler 8192, Glitch filter mode↩ : 4096 clocks
- LPTMR_PRESCALE_16384_GLITCHFILTER_8192: Timer mode: prescaler 16384, Glitch filter mode: 8192 clocks
- LPTMR_PRESCALE_32768_GLITCHFILTER_16384: Timer mode: prescaler 32768, Glitch filter mode: 16384 clocks
- LPTMR_PRESCALE_65536_GLITCHFILTER_32768: Timer mode: prescaler 65536, Glitch filter mode: 32768 clocks

Implements : LPTMR_HAL_GetPrescaler_Activity

3.54.3.16 LPTMR_HAL_GetWorkMode()

```
static lptmr_workmode_t LPTMR_HAL_GetWorkMode (
             const LPTMR_Type *const base )  [inline], [static]
```

Get current Work Mode.

This function returns the currently configured Work Mode for the LPTMR.

Parameters

in	*base*	- lptmr base pointer

Returns

Work Mode

- LPTMR_WORKMODE_TIMER: LPTMR is in Timer Mode
- LPTMR_WORKMODE_PULSECOUNTER: LPTMR is in Pulse Counter Mode

Implements : LPTMR_HAL_GetWorkMode_Activity

3.54.3.17 LPTMR_HAL_Init()

```
void LPTMR_HAL_Init (
             LPTMR_Type *const base )
```

Initialize the LPTMR instance to reset values.

This function initializes the LPTMR instance to a known state (the register are written with their reset values from the Reference Manual).

Parameters

in	*base*	- lptmr base pointer

3.54.3.18 LPTMR_HAL_SetBypass()

```
static void LPTMR_HAL_SetBypass (
             LPTMR_Type *const base,
             const bool enable )  [inline], [static]
```

Configure the Prescaler/Glitch Filter Bypass enable state.

This function configures the Prescaler/Glitch filter Bypass. This feature can be configured only when the LPTMR is disabled.

Parameters

in	*base*	- lptmr base pointer
in	*enable*	- the new Prescaler/Glitch Filter Bypass state • true: Prescaler/Glitch Filter Bypass enabled • false: Prescaler/Glitch Filter Bypass disabled

Implements : LPTMR_HAL_SetBypass_Activity

3.54.3.19 LPTMR_HAL_SetClockSelect()

```
static void LPTMR_HAL_SetClockSelect (
            LPTMR_Type *const base,
            const lptmr_clocksource_t clocksel )  [inline], [static]
```

Configure the LPTMR input Clock selection.

This function configures a clock source for the LPTMR. This feature can be configured only when the LPTMR is disabled.

Parameters

in	*base*	- lptmr base pointer
in	*clocksel*	- new Clock Source • LPTMR_CLOCKSOURCE_SIRCDIV2: clock from SIRC DIV2 • LPTMR_CLOCKSOURCE_1KHZ_LPO: clock from 1kHz LPO • LPTMR_CLOCKSOURCE_RTC: clock from RTC • LPTMR_CLOCKSOURCE_PCC: clock from PCC

Implements : LPTMR_HAL_SetClockSelect_Activity

3.54.3.20 LPTMR_HAL_SetCompareValue()

```
static void LPTMR_HAL_SetCompareValue (
            LPTMR_Type *const base,
            const uint16_t compval )  [inline], [static]
```

Configure the Compare Value.

This function configures the Compare Value. If set to 0, the Compare Match event and the hardware trigger assert and remain asserted until the timer is disabled.

Parameters

in	*base*	- lptmr base pointer
in	*compval*	- the new Compare Value

Implements : LPTMR_HAL_SetCompareValue_Activity

3.54.3.21 LPTMR_HAL_SetDmaRequest()

```
static void LPTMR_HAL_SetDmaRequest (
            LPTMR_Type *const base,
            bool enable )  [inline], [static]
```

Configure the DMA Request Enable Flag state.

This function configures the DMA Request feature of the LPTMR. If enabled, a DMA Request is issued when the Compare Match event occurs. If enabled, the Compare Match/Interrupt Pending flag is cleared when the DMA controller is done.

Parameters

in	*base*	- lptmr base pointer
in	*enable*	- the new state of the DMA Request Enable Flag • true: enable DMA Request • false: disable DMA Request

Implements : LPTMR_HAL_SetDmaRequest_Activity

3.54.3.22 LPTMR_HAL_SetFreeRunning()

```
static void LPTMR_HAL_SetFreeRunning (
             LPTMR_Type *const base,
             const bool enable )  [inline], [static]
```

Configure the Free Running state.

This function configures the Free Running feature of the LPTMR. This feature can be configured only when the LPTMR is disabled.

Parameters

in	*base*	- lptmr base pointer
in	*enable*	- the new Free Running state • true: Free Running Mode enabled. Reset counter on 16-bit overflow • false: Free Running Mode disabled. Reset counter on Compare Match.

Implements : LPTMR_HAL_SetFreeRunning_Activity

3.54.3.23 LPTMR_HAL_SetInterrupt()

```
static void LPTMR_HAL_SetInterrupt (
             LPTMR_Type *const base,
             bool enable )  [inline], [static]
```

Configure the Interrupt Enable state.

This function configures the Interrupt Enable state for the LPTMR. If enabled, an interrupt is generated when a Compare Match event occurs.

Parameters

in	*base*	- lptmr base pointer
in	*enable*	- the new state for the interrupt • true: enable Interrupt • false: disable Interrupt

Implements : LPTMR_HAL_SetInterrupt_Activity

3.54.3.24 LPTMR_HAL_SetPinPolarity()

```
static void LPTMR_HAL_SetPinPolarity (
            LPTMR_Type *const base,
            const lptmr_pinpolarity_t pol )  [inline], [static]
```

Configure Pin Polarity for Pulse Counter Mode.

This function configures the pin polarity that triggers an increment in Pulse Counter Mode. This feature can be configured only when the LPTMR is disabled.

Parameters

in	*base*	- lptmr base pointer
in	*pol*	- the pin polarity to count in Pulse Counter Mode • LPTMR_PINPOLARITY_RISING: count pulse on Rising Edge • LPTMR_PINPOLARITY_FALLING: count pulse on Falling Edge

Implements : LPTMR_HAL_SetPinPolarity_Activity

3.54.3.25 LPTMR_HAL_SetPinSelect()

```
static void LPTMR_HAL_SetPinSelect (
            LPTMR_Type *const base,
            const lptmr_pinselect_t pinsel )  [inline], [static]
```

Configure the Pin selection for Pulse Counter Mode.

This function configures the input Pin selection for Pulse Counter Mode. This feature can be configured only when the LPTMR is disabled.

Parameters

in	*base*	- lptmr base pointer
in	*pinsel*	- pin selection • LPTMR_PINSELECT_TRGMUX: count pulses from TRGMUX output • LPTMR_PINSELECT_ALT1: count pulses from pin alt 1 • LPTMR_PINSELECT_ALT2: count pulses from pin alt 2 • LPTMR_PINSELECT_ALT3: count pulses from pin alt 3

Implements : LPTMR_HAL_SetPinSelect_Activity

3.54.3.26 LPTMR_HAL_SetPrescaler()

```
static void LPTMR_HAL_SetPrescaler (
            LPTMR_Type *const base,
            const lptmr_prescaler_t presc )  [inline], [static]
```

Configure the Prescaler/Glitch Filter divider value.

This function configures the value for the Prescaler/Glitch Filter. This feature can be configured only when the LPTMR is disabled.

Parameters

in	*base*	- lptmr base pointer
in	*presc*	- the new Prescaler value • LPTMR_PRESCALE_2: Timer mode: prescaler 2, Glitch filter mode: invalid • LPTMR_PRESCALE_4_GLITCHFILTER_2: Timer mode: prescaler 4, Glitch filter mode: 2 clocks • LPTMR_PRESCALE_8_GLITCHFILTER_4: Timer mode: prescaler 8, Glitch filter mode: 4 clocks • LPTMR_PRESCALE_16_GLITCHFILTER_8: Timer mode: prescaler 16, Glitch filter mode: 8 clocks • LPTMR_PRESCALE_32_GLITCHFILTER_16: Timer mode: prescaler 32, Glitch filter mode: 16 clocks • LPTMR_PRESCALE_64_GLITCHFILTER_32: Timer mode: prescaler 64, Glitch filter mode: 32 clocks • LPTMR_PRESCALE_128_GLITCHFILTER_64: Timer mode: prescaler 128, Glitch filter mode: 64 clocks • LPTMR_PRESCALE_256_GLITCHFILTER_128: Timer mode: prescaler 256, Glitch filter mode: 128 clocks • LPTMR_PRESCALE_512_GLITCHFILTER_256: Timer mode: prescaler 512, Glitch filter mode: 256 clocks • LPTMR_PRESCALE_1024_GLITCHFILTER_512: Timer mode: prescaler 1024, Glitch filter mode: 512 clocks • LPTMR_PRESCALE_2048_GLITCHFILTER_1024: Timer mode: prescaler 2048, Glitch filter mode: 1024 clocks • LPTMR_PRESCALE_4096_GLITCHFILTER_2048: Timer mode: prescaler 4096, Glitch filter mode: 2048 clocks • LPTMR_PRESCALE_8192_GLITCHFILTER_4096: Timer mode: prescaler 8192, Glitch filter mode: 4096 clocks • LPTMR_PRESCALE_16384_GLITCHFILTER_8192: Timer mode: prescaler 16384, Glitch filter mode: 8192 clocks • LPTMR_PRESCALE_32768_GLITCHFILTER_16384: Timer mode: prescaler 32768, Glitch filter mode: 16384 clocks • LPTMR_PRESCALE_65536_GLITCHFILTER_32768: Timer mode: prescaler 65536, Glitch filter mode: 32768 clocks

Implements : LPTMR_HAL_SetPrescaler_Activity

3.54.3.27 LPTMR_HAL_SetWorkMode()

```
static void LPTMR_HAL_SetWorkMode (
LPTMR_Type *const base,
const lptmr_workmode_t mode )  [inline], [static]
```

Configure the Work Mode.

This function configures the LPTMR to either Timer Mode or Pulse Counter Mode. This feature can be configured only when the LPTMR is disabled.

Parameters

in	*base*	- lptmr base pointer
in	*mode*	- new Work Mode • LPTMR_WORKMODE_TIMER: LPTMR set to Timer Mode • LPTMR_WORKMODE_PULSECOUNTER: LPTMR set to Pulse Counter Mode

Implements : LPTMR_HAL_SetWorkMode_Activity

3.55 LPUART Driver

3.55.1 Detailed Description

This module covers the functionality of the Low Power Universal Asynchronous Receiver-Transmitter (LPUART) peripheral driver.

The LPUART driver implements serial communication using the LPUART module in the S32144K processor.

Features

- Interrupt based
- Provides blocking and non-blocking transmit and receive functions
- Configurable baud rate
- 8/9/10 bits per char

Functionality

In order to use the LPUART driver it must be first initialized, using LPUART_DRV_Init() function. Once initialized, it cannot be initialized again for the same LPUART module instance until it is de-initialized, using LPUART_DRV_↩ Deinit(). The initialization function does the following operations:

- sets the baud rate
- sets parity/bit count/stop bits count
- initializes the state structure for the current instance
- enables receiver/transmitter for the current instance Different LPUART module instances can function independently of each other.

Interrupt-based communication

After initialization, a serial communication can be triggered by calling LPUART_DRV_SendData function; this will save the reference of the data buffer received as parameter in the internal tx buffer pointer, then copy the first byte to the data register. The hw tranceiver then automatically shifts the data and triggers a 'Transmit buffer empty' interrupt when all bits are shifted. The drivers interrupt handler takes care of transmitting the next byte in the buffer, by increasing the data pointer and decreasing the data size. The same sequence of operations is executed until all bytes in the tx buffer have been transmitted.

Similarly, data reception is triggered by calling LPUART_DRV_ReceiveData function, passing the rx buffer as parameter. When the tranceiver copies the received bits in the data register, the 'Receive buffer full' interrupt is triggered; the driver irq handler clears the flag by reading the received byte, saves it in the rx buffer, then increments the data pointer and decrements the data size. This is repeated untill all bytes are received.

The workflow applies to send/receive operations using blocking method (triggered by LPUART_DRV_SendData↩ Blocking and LPUART_DRV_ReceiveDataBlocking), with the single difference that the send/receive function will not return until the send/receive operation is complete (all bytes are successfully transferred or a timeout occured). The timeout for the blocking method is passed as parameter by the user.

If a user callback is installed for rx/tx, the callback has to take care of data handling and aborting the transfer when complete; the driver irq handler does not manipulate the buffers in this case. A target usecase here would be receiving an indefinite number of bytes; the user rx callback will be called by the driver each time a character is received and the application needs to call LPUART_DRV_AbortReceivingData in order to stop the reception.

DMA-based communication

In DMA operation, both blocking and non-blocking transmission methods confiure a DMA channel to copy data from the buffer to the data register (for tx), or viceversa (for rx). The driver assumes the DMA channel is already allocated and the proper requests are routed to it via DMAMUX. After configuring the DMA channel, the driver enables DMA requests for rx/tx, then the DMA engine takes care of moving data to/from the data buffer. In this scenario, the callback is only called when the full transmission is done, that is when the DMA channel finishes the number of loops configured in the transfer descriptor.

Important Notes

- Before using the LPUART driver the module clock must be configured
- The driver enables the interrupts for the corresponding LPUART module, but any interrupt priority must be done by the application
- The board specific configurations must be done prior to driver calls; the driver has no influence on the functionality of the rx/tx pins - they must be configured by application
- DMA module has to be initialized prior to LPUART usage in DMA mode; also, DMA channels need to be allocated for LPUART usage by the application (the driver only takes care of configuring the DMA channels received in the configuration structure)
- for 9/10 bits characters, the application must provide the appropriate buffers; the size of the tx/rx buffers in this scenario needs to be an even number, as the transferred characters will be split in two bytes (bit 8 for 9-bis chars and bits 8 & 9 for 10-bits chars will be stored in the subsequent byte). 9/10 bits chars are only supported in interrupt-based communications

Data Structures

- struct lpuart_state_t

  Runtime state of the LPUART driver. More...
- struct lpuart_user_config_t

  LPUART configuration structure. More...

Typedefs

- typedef void(∗ lpuart_rx_callback_t) (uint32_t instance, void ∗lpuartState)

  LPUART receive callback function type.
- typedef void(∗ lpuart_tx_callback_t) (uint32_t instance, void ∗lpuartState)

  LPUART transmit callback function type.

Enumerations

- enum lpuart_transfer_type_t { LPUART_USING_DMA = 0, LPUART_USING_INTERRUPTS }

  Type of LPUART transfer (based on interrupts or DMA).

Variables

- LPUART_Type ∗const g_lpuartBase [LPUART_INSTANCE_COUNT]

  Table of base addresses for LPUART instances.
- const IRQn_Type g_lpuartRxTxIrqId [LPUART_INSTANCE_COUNT]

  Table to save LPUART IRQ enumeration numbers defined in the CMSIS header file.
- const clock_names_t g_lpuartClkNames [LPUART_INSTANCE_COUNT]

  Table to save LPUART indexes in PCC register map for clock configuration.

LPUART Driver

- status_t LPUART_DRV_Init (uint32_t instance, lpuart_state_t ∗lpuartStatePtr, const lpuart_user_config_↩ t ∗lpuartUserConfig)

  Initializes an LPUART operation instance.
- status_t LPUART_DRV_Deinit (uint32_t instance)

  Shuts down the LPUART by disabling interrupts and transmitter/receiver.
- lpuart_rx_callback_t LPUART_DRV_InstallRxCallback (uint32_t instance, lpuart_rx_callback_t function, void ∗callbackParam)

  Installs callback function for the LPUART receive.
- lpuart_tx_callback_t LPUART_DRV_InstallTxCallback (uint32_t instance, lpuart_tx_callback_t function, void ∗callbackParam)

  Installs callback function for the LPUART transmit.
- status_t LPUART_DRV_SendDataBlocking (uint32_t instance, const uint8_t ∗txBuff, uint32_t txSize, uint32↩ _t timeout)

  Sends data out through the LPUART module using a blocking method.
- status_t LPUART_DRV_SendData (uint32_t instance, const uint8_t ∗txBuff, uint32_t txSize)

  Sends data out through the LPUART module using a non-blocking method. This enables an a-sync method for transmitting data. When used with a non-blocking receive, the LPUART can perform a full duplex operation. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete.
- status_t LPUART_DRV_GetTransmitStatus (uint32_t instance, uint32_t ∗bytesRemaining)

  Returns whether the previous transmit is complete.
- status_t LPUART_DRV_AbortSendingData (uint32_t instance)

  Terminates a non-blocking transmission early.
- status_t LPUART_DRV_ReceiveDataBlocking (uint32_t instance, uint8_t ∗rxBuff, uint32_t rxSize, uint32_t timeout)

  Gets data from the LPUART module by using a blocking method. Blocking means that the function does not return until the receive is complete.
- status_t LPUART_DRV_ReceiveData (uint32_t instance, uint8_t ∗rxBuff, uint32_t rxSize)

  Gets data from the LPUART module by using a non-blocking method. This enables an a-sync method for receiving data. When used with a non-blocking transmission, the LPUART can perform a full duplex operation. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the receive is complete.
- status_t LPUART_DRV_GetReceiveStatus (uint32_t instance, uint32_t ∗bytesRemaining)

  Returns whether the previous receive is complete.
- status_t LPUART_DRV_AbortReceivingData (uint32_t instance)

  Terminates a non-blocking receive early.

3.55.2 Data Structure Documentation

3.55.2.1 struct lpuart_state_t

Runtime state of the LPUART driver.

Note that the caller provides memory for the driver state structures during initialization because the driver does not statically allocate memory.

Implements : lpuart_state_t_Class

Data Fields

- const uint8_t ∗ txBuff
- uint8_t ∗ rxBuff
- volatile uint32_t txSize
- volatile uint32_t rxSize
- volatile bool isTxBusy
- volatile bool isRxBusy
- volatile bool isTxBlocking
- volatile bool isRxBlocking
- lpuart_bit_count_per_char_t bitCountPerChar
- lpuart_rx_callback_t rxCallback
- void ∗ rxCallbackParam
- lpuart_tx_callback_t txCallback
- void ∗ txCallbackParam
- lpuart_transfer_type_t transferType
- uint8_t rxDMAChannel
- uint8_t txDMAChannel
- semaphore_t rxComplete
- semaphore_t txComplete
- volatile status_t transmitStatus
- volatile status_t receiveStatus

Field Documentation

3.55.2.1.1 bitCountPerChar

```
lpuart_bit_count_per_char_t bitCountPerChar
```

number of bits in a char (8/9/10)

3.55.2.1.2 isRxBlocking

```
volatile bool isRxBlocking
```

True if receive is blocking transaction.

3.55.2.1.3 isRxBusy

```
volatile bool isRxBusy
```

True if there is an active receive.

3.55.2.1.4 isTxBlocking

```
volatile bool isTxBlocking
```

True if transmit is blocking transaction.

3.55.2.1.5 isTxBusy

```
volatile bool isTxBusy
```

True if there is an active transmit.

3.55.2.1.6 receiveStatus

```
volatile status_t receiveStatus
```

Status of last driver receive operation

3.55.2.1.7 rxBuff

```
uint8_t* rxBuff
```

The buffer of received data.

3.55.2.1.8 rxCallback

```
lpuart_rx_callback_t rxCallback
```

Callback to invoke for data receive Note: when the transmission is interrupt based, the callback is being called upon receiving a byte; when DMA transmission is used, the bytes are copied to the rx buffer by the DMA engine and the callback is called when all the bytes have been transferred.

3.55.2.1.9 rxCallbackParam

```
void* rxCallbackParam
```

Receive callback parameter pointer.

3.55.2.1.10 rxComplete

```
semaphore_t rxComplete
```

Synchronization object for blocking Rx timeout condition

3.55.2.1.11 rxDMAChannel

```
uint8_t rxDMAChannel
```

DMA channel number for DMA-based rx.

3.55.2.1.12 rxSize

```
volatile uint32_t rxSize
```

The remaining number of bytes to be received.

3.55.2.1.13 transferType

```
lpuart_transfer_type_t transferType
```

Type of LPUART transfer (interrupt/dma based)

3.55.2.1.14 transmitStatus

```
volatile status_t transmitStatus
```

Status of last driver transmit operation

3.55.2.1.15 txBuff

```
const uint8_t* txBuff
```

The buffer of data being sent.

3.55.2.1.16 txCallback

```
lpuart_tx_callback_t txCallback
```

Callback to invoke for data send Note: when the transmission is interrupt based, the callback is being called upon sending a byte; when DMA transmission is used, the bytes are copied to the tx buffer by the DMA engine and the callback is called when all the bytes have been transferred.

3.55.2.1.17 txCallbackParam

```
void* txCallbackParam
```

Transmit callback parameter pointer.

3.55.2.1.18 txComplete

```
semaphore_t txComplete
```

Synchronization object for blocking Tx timeout condition

3.55.2.1.19 txDMAChannel

```
uint8_t txDMAChannel
```

DMA channel number for DMA-based tx.

3.55.2.1.20 txSize

```
volatile uint32_t txSize
```

The remaining number of bytes to be transmitted.

3.55.2.2 struct lpuart_user_config_t

LPUART configuration structure.

Implements : lpuart_user_config_t_Class

Data Fields

- uint32_t baudRate
- lpuart_parity_mode_t parityMode
- lpuart_stop_bit_count_t stopBitCount
- lpuart_bit_count_per_char_t bitCountPerChar
- lpuart_transfer_type_t transferType
- uint8_t rxDMAChannel
- uint8_t txDMAChannel

Field Documentation

3.55.2.2.1 baudRate

```
uint32_t baudRate
```

LPUART baud rate

3.55.2.2.2 bitCountPerChar

```
lpuart_bit_count_per_char_t bitCountPerChar
```

number of bits in a character (8-default, 9 or 10); for 9/10 bits chars, users must provide appropriate buffers to the send/receive functions (bits 8/9 in subsequent bytes); for DMA transmission only 8-bit char is supported.

3.55.2.2.3 parityMode

```
lpuart_parity_mode_t parityMode
```

parity mode, disabled (default), even, odd

3.55.2.2.4 rxDMAChannel

```
uint8_t rxDMAChannel
```

Channel number for DMA rx channel. If DMA mode isn't used this field will be ignored.

3.55.2.2.5 stopBitCount

```
lpuart_stop_bit_count_t stopBitCount
```

number of stop bits, 1 stop bit (default) or 2 stop bits

3.55.2.2.6 transferType

```
lpuart_transfer_type_t transferType
```

Type of LPUART transfer (interrupt/dma based)

3.55.2.2.7 txDMAChannel

```
uint8_t txDMAChannel
```

Channel number for DMA tx channel. If DMA mode isn't used this field will be ignored.

3.55.3 Typedef Documentation

3.55.3.1 lpuart_rx_callback_t

```
typedef void(* lpuart_rx_callback_t) (uint32_t instance, void *lpuartState)
```

LPUART receive callback function type.

Implements : lpuart_rx_callback_t_Class

3.55.3.2 lpuart_tx_callback_t

```
typedef void(* lpuart_tx_callback_t) (uint32_t instance, void *lpuartState)
```

LPUART transmit callback function type.

Implements : lpuart_tx_callback_t_Class

3.55.4 Enumeration Type Documentation

3.55.4.1 lpuart_transfer_type_t

```
enum lpuart_transfer_type_t
```

Type of LPUART transfer (based on interrupts or DMA).

Implements : lpuart_transfer_type_t_Class

Enumerator

LPUART_USING_DMA	The driver will use DMA to perform UART transfer
LPUART_USING_INTERRUPTS	The driver will use interrupts to perform UART transfer

3.55.5 Function Documentation

3.55.5.1 LPUART_DRV_AbortReceivingData()

```
status_t LPUART_DRV_AbortReceivingData (
             uint32_t instance )
```

Terminates a non-blocking receive early.

Parameters

instance	LPUART instance number

Returns

Whether the receiving was successful or not.

3.55.5.2 LPUART_DRV_AbortSendingData()

```
status_t LPUART_DRV_AbortSendingData (
            uint32_t instance )
```

Terminates a non-blocking transmission early.

Parameters

instance	LPUART instance number

Returns

Whether the aborting is successful or not.

3.55.5.3 LPUART_DRV_Deinit()

```
status_t LPUART_DRV_Deinit (
            uint32_t instance )
```

Shuts down the LPUART by disabling interrupts and transmitter/receiver.

Parameters

instance	LPUART instance number

Returns

STATUS_SUCCESS if successful; STATUS_ERROR if an error occured

3.55.5.4 LPUART_DRV_GetReceiveStatus()

```
status_t LPUART_DRV_GetReceiveStatus (
            uint32_t instance,
            uint32_t * bytesRemaining )
```

Returns whether the previous receive is complete.

Parameters

instance	LPUART instance number
bytesRemaining	pointer to value that is filled with the number of bytes that still need to be received in the active transfer.

Returns

The receive status.

Return values

STATUS_SUCCESS	the receive has completed successfully.
STATUS_BUSY	the receive is still in progress. *bytesReceived* will be filled with the number of bytes that have been received so far.
STATUS_UART_ABORTED	The receive was aborted.
STATUS_TIMEOUT	A timeout was reached.
STATUS_ERROR	An error occured.

3.55.5.5 LPUART_DRV_GetTransmitStatus()

```
status_t LPUART_DRV_GetTransmitStatus (
            uint32_t instance,
            uint32_t * bytesRemaining )
```

Returns whether the previous transmit is complete.

Parameters

instance	LPUART instance number
bytesRemaining	Pointer to value that is populated with the number of bytes that have been sent in the active transfer

Returns

The transmit status.

Return values

STATUS_SUCCESS	The transmit has completed successfully.
STATUS_BUSY	The transmit is still in progress. *bytesTransmitted* will be filled with the number of bytes that have been transmitted so far.
STATUS_UART_ABORTED	The transmit was aborted.
STATUS_TIMEOUT	A timeout was reached.
STATUS_ERROR	An error occured.

3.55.5.6 LPUART_DRV_Init()

```
status_t LPUART_DRV_Init (
            uint32_t instance,
            lpuart_state_t * lpuartStatePtr,
            const lpuart_user_config_t * lpuartUserConfig )
```

Initializes an LPUART operation instance.

The caller provides memory for the driver state structures during initialization. The user must select the LPUART clock source in the application to initialize the LPUART.

Parameters

instance	LPUART instance number
lpuartUserConfig	user configuration structure of type lpuart_user_config_t
lpuartStatePtr	pointer to the LPUART driver state structure

Returns

STATUS_SUCCESS if successful; STATUS_ERROR if an error occured

3.55.5.7 LPUART_DRV_InstallRxCallback()

```
lpuart_rx_callback_t LPUART_DRV_InstallRxCallback (
            uint32_t instance,
            lpuart_rx_callback_t function,
            void * callbackParam )
```

Installs callback function for the LPUART receive.

Note

After a callback is installed, it bypasses part of the LPUART IRQHandler logic. Therefore, the callback needs to handle the indexes of txBuff and txSize.

Parameters

instance	The LPUART instance number.
function	The LPUART receive callback function.
rxBuff	The receive buffer used inside IRQHandler. This buffer must be kept as long as the callback is alive.
callbackParam	The LPUART receive callback parameter pointer.

Returns

Former LPUART receive callback function pointer.

3.55.5.8 LPUART_DRV_InstallTxCallback()

```
lpuart_tx_callback_t LPUART_DRV_InstallTxCallback (
            uint32_t instance,
            lpuart_tx_callback_t function,
            void * callbackParam )
```

Installs callback function for the LPUART transmit.

Note

After a callback is installed, it bypasses part of the LPUART IRQHandler logic. Therefore, the callback needs to handle the indexes of txBuff and txSize.

Parameters

instance	The LPUART instance number.
function	The LPUART transmit callback function.
txBuff	The transmit buffer used inside IRQHandler. This buffer must be kept as long as the callback is alive.
callbackParam	The LPUART transmit callback parameter pointer.

Returns

Former LPUART transmit callback function pointer.

3.55.5.9 LPUART_DRV_ReceiveData()

```
status_t LPUART_DRV_ReceiveData (
            uint32_t instance,
            uint8_t * rxBuff,
            uint32_t rxSize )
```

Gets data from the LPUART module by using a non-blocking method. This enables an a-sync method for receiving data. When used with a non-blocking transmission, the LPUART can perform a full duplex operation. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the receive is complete.

Parameters

instance	LPUART instance number
rxBuff	buffer containing 8-bit read data chars received
rxSize	the number of bytes to receive

Returns

STATUS_SUCCESS if successful; STATUS_BUSY if a resource is busy; STATUS_ERROR if an error occured

3.55.5.10 LPUART_DRV_ReceiveDataBlocking()

```
status_t LPUART_DRV_ReceiveDataBlocking (
            uint32_t instance,
            uint8_t * rxBuff,
            uint32_t rxSize,
            uint32_t timeout )
```

Gets data from the LPUART module by using a blocking method. Blocking means that the function does not return until the receive is complete.

Parameters

instance	LPUART instance number
rxBuff	buffer containing 8-bit read data chars received
rxSize	the number of bytes to receive
timeout	timeout value in milliseconds

Returns

STATUS_SUCCESS if successful; STATUS_TIMEOUT if the timeout was reached; STATUS_BUSY if a resource is busy; STATUS_ERROR if an error occured

3.55.5.11 LPUART_DRV_SendData()

```
status_t LPUART_DRV_SendData (
            uint32_t instance,
            const uint8_t * txBuff,
            uint32_t txSize )
```

Sends data out through the LPUART module using a non-blocking method. This enables an a-sync method for transmitting data. When used with a non-blocking receive, the LPUART can perform a full duplex operation. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete.

Parameters

instance	LPUART instance number
txBuff	source buffer containing 8-bit data chars to send
txSize	the number of bytes to send

Returns

STATUS_SUCCESS if successful; STATUS_BUSY if a resource is busy; STATUS_ERROR if an error occured

3.55.5.12 LPUART_DRV_SendDataBlocking()

```
status_t LPUART_DRV_SendDataBlocking (
            uint32_t instance,
            const uint8_t * txBuff,
            uint32_t txSize,
            uint32_t timeout )
```

Sends data out through the LPUART module using a blocking method.

Blocking means that the function does not return until the transmission is complete.

Parameters

instance	LPUART instance number
txBuff	source buffer containing 8-bit data chars to send
txSize	the number of bytes to send
timeout	timeout value in milliseconds

Returns

STATUS_SUCCESS if successful; STATUS_TIMEOUT if the timeout was reached; STATUS_BUSY if a resource is busy; STATUS_ERROR if an error occured

3.55.6 Variable Documentation

3.55.6.1 g_lpuartBase

```
LPUART_Type* const g_lpuartBase[LPUART_INSTANCE_COUNT]
```

Table of base addresses for LPUART instances.

3.55.6.2 g_lpuartClkNames

```
const clock_names_t g_lpuartClkNames[LPUART_INSTANCE_COUNT]
```

Table to save LPUART indexes in PCC register map for clock configuration.

3.55.6.3 g_lpuartRxTxIrqId

```
const IRQn_Type g_lpuartRxTxIrqId[LPUART_INSTANCE_COUNT]
```

Table to save LPUART IRQ enumeration numbers defined in the CMSIS header file.

3.56 LPUART HAL

3.56.1 Detailed Description

This module covers the functionality of the Low Power Universal Asynchronous Receiver-Transmitter (LPUART) hardware abstraction layer.

LPUART HAL provides the API for reading and writing register bit-fields belonging to the LPUART module. It also provides an initialization function for bringing the module to the reset state.

The LPUART HAL functions are used by LPUART and LIN drivers. Besides register access, the LPUART HAL also provide a function to set the communication baud-rate (needed by both drivers using LPUART module). Also, it provides basic serial communication functionality - send/receive functions based on polling method (writing/reading data to/from the buffer register when tx/rx flags are cleared).

For higher-level functionality, use the LPUART/LIN driver.

Data Structures

- struct lpuart_idle_line_config_t

  Structure for idle line configuration settings. More...

Macros

- #define LPUART_SHIFT (16U)
- #define LPUART_BAUD_REG_ID (1U)
- #define LPUART_STAT_REG_ID (2U)
- #define LPUART_CTRL_REG_ID (3U)
- #define LPUART_DATA_REG_ID (4U)
- #define LPUART_MATCH_REG_ID (5U)
- #define LPUART_MODIR_REG_ID (6U)
- #define LPUART_FIFO_REG_ID (7U)
- #define LPUART_WATER_REG_ID (8U)

Enumerations

- enum lpuart_stop_bit_count_t { LPUART_ONE_STOP_BIT = 0x0U, LPUART_TWO_STOP_BIT = 0x1U }

  LPUART number of stop bits.
- enum lpuart_parity_mode_t { LPUART_PARITY_DISABLED = 0x0U, LPUART_PARITY_EVEN = 0x2U, L↩ PUART_PARITY_ODD = 0x3U }

  LPUART parity mode.
- enum lpuart_bit_count_per_char_t { LPUART_8_BITS_PER_CHAR = 0x0U, LPUART_9_BITS_PER_CHAR = 0x1U, LPUART_10_BITS_PER_CHAR = 0x2U }

  LPUART number of bits in a character.
- enum lpuart_operation_config_t { LPUART_OPERATES = 0x0U, LPUART_STOPS = 0x1U }

  LPUART operation configuration constants.
- enum lpuart_wakeup_method_t { LPUART_IDLE_LINE_WAKE = 0x0U, LPUART_ADDR_MARK_WAKE = 0x1U }

  LPUART wakeup from standby method constants.
- enum lpuart_break_char_length_t { LPUART_BREAK_CHAR_10_BIT_MINIMUM = 0x0U, LPUART_BRE↩ AK_CHAR_13_BIT_MINIMUM = 0x1U }

LPUART break character length settings for transmit/detect.

- enum lpuart_singlewire_txdir_t { LPUART_SINGLE_WIRE_TX_DIR_IN = 0x0U, LPUART_SINGLE_WIRE↩ _TX_DIR_OUT = 0x1U }

  LPUART single-wire mode TX direction.
- enum lpuart_match_config_t { LPUART_ADDRESS_MATCH_WAKEUP = 0x0U, LPUART_IDLE_MATCH↩ _WAKEUP = 0x1U, LPUART_MATCH_ON_AND_MATCH_OFF = 0x2U, LPUART_ENABLE_RWU_ON_↩ DATA_MATCH = 0x3U }

  LPUART Configures the match addressing mode used.
- enum lpuart_ir_tx_pulsewidth_t { LPUART_IR_THREE_SIXTEENTH_WIDTH = 0x0U, LPUART_IR_ONE_↩ SIXTEENTH_WIDTH = 0x1U, LPUART_IR_ONE_THIRTYSECOND_WIDTH = 0x2U, LPUART_IR_ONE_↩ FOURTH_WIDTH = 0x3U }

  LPUART infra-red transmitter pulse width options.
- enum lpuart_idle_char_t {
  LPUART_1_IDLE_CHAR = 0x0U, LPUART_2_IDLE_CHAR = 0x1U, LPUART_4_IDLE_CHAR = 0x2U, LP↩ UART_8_IDLE_CHAR = 0x3U,
  LPUART_16_IDLE_CHAR = 0x4U, LPUART_32_IDLE_CHAR = 0x5U, LPUART_64_IDLE_CHAR = 0x6U,
  LPUART_128_IDLE_CHAR = 0x7U }

  LPUART Configures the number of idle characters that must be received before the IDLE flag is set.
- enum lpuart_cts_source_t { LPUART_CTS_SOURCE_PIN = 0x0U, LPUART_CTS_SOURCE_INVERTED↩ _RECEIVER_MATCH = 0x1U }

  LPUART Transmits the CTS Configuration. Configures the source of the CTS input.
- enum lpuart_cts_config_t { LPUART_CTS_SAMPLED_ON_EACH_CHAR = 0x0U, LPUART_CTS_SAMP↩ LED_ON_IDLE = 0x1U }

  LPUART Transmits CTS Source.Configures if the CTS state is checked at the start of each character or only when the transmitter is idle.
- enum lpuart_status_flag_t {
  LPUART_TX_DATA_REG_EMPTY, LPUART_TX_COMPLETE, LPUART_RX_DATA_REG_FULL, LPUA↩ RT_IDLE_LINE_DETECT,
  LPUART_RX_OVERRUN, LPUART_NOISE_DETECT, LPUART_FRAME_ERR, LPUART_PARITY_ERR,
  LPUART_LIN_BREAK_DETECT, LPUART_RX_ACTIVE_EDGE_DETECT, LPUART_RX_ACTIVE, LPUA↩ RT_NOISE_IN_CURRENT_WORD,
  LPUART_PARITY_ERR_IN_CURRENT_WORD, LPUART_MATCH_ADDR_ONE, LPUART_MATCH_AD↩ DR_TWO, LPUART_FIFO_TX_OF,
  LPUART_FIFO_RX_UF }

  LPUART status flags.
- enum lpuart_interrupt_t {
  LPUART_INT_LIN_BREAK_DETECT, LPUART_INT_RX_ACTIVE_EDGE, LPUART_INT_TX_DATA_RE↩ G_EMPTY, LPUART_INT_TX_COMPLETE,
  LPUART_INT_RX_DATA_REG_FULL, LPUART_INT_IDLE_LINE, LPUART_INT_RX_OVERRUN, LPUA↩ RT_INT_NOISE_ERR_FLAG,
  LPUART_INT_FRAME_ERR_FLAG, LPUART_INT_PARITY_ERR_FLAG, LPUART_INT_MATCH_ADDR↩ _ONE, LPUART_INT_MATCH_ADDR_TWO,
  LPUART_INT_FIFO_TXOF, LPUART_INT_FIFO_RXUF }

  LPUART interrupt configuration structure, default settings are 0 (disabled)

LPUART Common Configurations

- void LPUART_HAL_Init (LPUART_Type ∗base)

  Initializes the LPUART controller.
- static void LPUART_HAL_SetTransmitterCmd (LPUART_Type ∗base, bool enable)

  Enable/Disable the LPUART transmitter.
- static bool LPUART_HAL_GetTransmitterCmd (const LPUART_Type ∗base)

  Gets the LPUART transmitter enabled/disabled configuration.
- static void LPUART_HAL_SetReceiverCmd (LPUART_Type ∗base, bool enable)

Enable/Disable the LPUART receiver.
- static bool LPUART_HAL_GetReceiverCmd (const LPUART_Type ∗base)

  Gets the LPUART receiver enabled/disabled configuration.
- status_t LPUART_HAL_SetBaudRate (LPUART_Type ∗base, uint32_t sourceClockInHz, uint32_t desired←BaudRate)

  Configures the LPUART baud rate.
- static void LPUART_HAL_SetBaudRateDivisor (LPUART_Type ∗base, uint32_t baudRateDivisor)

  Sets the LPUART baud rate modulo divisor.
- static void LPUART_HAL_SetOversamplingRatio (LPUART_Type ∗base, uint32_t overSamplingRatio)

  Sets the LPUART baud rate oversampling ratio.
- static void LPUART_HAL_SetBothEdgeSamplingCmd (LPUART_Type ∗base, bool enable)

  Configures the LPUART baud rate both edge sampling.
- static bool LPUART_HAL_GetRxDataPolarity (const LPUART_Type ∗base)

  Returns whether the receive data is inverted or not.
- static void LPUART_HAL_SetRxDataPolarity (LPUART_Type ∗base, bool polarity)

  Sets whether the recevie data is inverted or not.
- void LPUART_HAL_SetBitCountPerChar (LPUART_Type ∗base, lpuart_bit_count_per_char_t bitCountPer←Char)

  Configures the number of bits per character in the LPUART controller.
- void LPUART_HAL_SetParityMode (LPUART_Type ∗base, lpuart_parity_mode_t parityModeType)

  Configures parity mode in the LPUART controller.
- static void LPUART_HAL_SetStopBitCount (LPUART_Type ∗base, lpuart_stop_bit_count_t stopBitCount)

  Configures the number of stop bits in the LPUART controller.
- static const volatile void ∗ LPUART_HAL_GetDataRegAddr (const LPUART_Type ∗base)

  Get LPUART tx/rx data register address.

LPUART Interrupts and DMA

- void LPUART_HAL_SetIntMode (LPUART_Type ∗base, lpuart_interrupt_t intSrc, bool enable)

  Configures the LPUART module interrupts.
- bool LPUART_HAL_GetIntMode (const LPUART_Type ∗base, lpuart_interrupt_t intSrc)

  Returns LPUART module interrupts state.
- static void LPUART_HAL_SetTxDmaCmd (LPUART_Type ∗base, bool enable)

  Configures DMA requests.
- static void LPUART_HAL_SetRxDmaCmd (LPUART_Type ∗base, bool enable)

  Configures DMA requests.
- static bool LPUART_HAL_IsTxDmaEnabled (const LPUART_Type ∗base)

  Gets the LPUART DMA request configuration.
- static bool LPUART_HAL_IsRxDmaEnabled (const LPUART_Type ∗base)

  Gets the LPUART DMA request configuration.

LPUART Transfer Functions

- static void LPUART_HAL_Putchar (LPUART_Type ∗base, uint8_t data)

  Sends the LPUART 8-bit character.
- void LPUART_HAL_Putchar9 (LPUART_Type ∗base, uint16_t data)

  Sends the LPUART 9-bit character.
- void LPUART_HAL_Putchar10 (LPUART_Type ∗base, uint16_t data)

  Sends the LPUART 10-bit character (Note: Feature available on select LPUART instances).
- static void LPUART_HAL_Getchar (const LPUART_Type ∗base, uint8_t ∗readData)

Gets the LPUART 8-bit character.
- void LPUART_HAL_Getchar9 (const LPUART_Type ∗base, uint16_t ∗readData)

  Gets the LPUART 9-bit character.
- void LPUART_HAL_Getchar10 (const LPUART_Type ∗base, uint16_t ∗readData)

  Gets the LPUART 10-bit character.
- void LPUART_HAL_SendDataPolling (LPUART_Type ∗base, const uint8_t ∗txBuff, uint32_t txSize)

  Send out multiple bytes of data using polling method.
- status_t LPUART_HAL_ReceiveDataPolling (LPUART_Type ∗base, uint8_t ∗rxBuff, uint32_t rxSize)

  Receive multiple bytes of data using polling method.

LPUART Status Flags

- bool LPUART_HAL_GetStatusFlag (const LPUART_Type ∗base, lpuart_status_flag_t statusFlag)

  LPUART get status flag.
- status_t LPUART_HAL_ClearStatusFlag (LPUART_Type ∗base, lpuart_status_flag_t statusFlag)

  LPUART clears an individual status flag.

LPUART Special Feature Configurations

- static void LPUART_HAL_SetIdleChar (LPUART_Type ∗base, lpuart_idle_char_t idleConfig)

  Configures the number of idle characters.
- static lpuart_idle_char_t LPUART_HAL_GetIdleChar (const LPUART_Type ∗base)

  Gets the number of idle characters for IDLE flag.
- static bool LPUART_HAL_IsCurrentDataWithNoise (const LPUART_Type ∗base)

  Checks whether the current data word was received with noise.
- static bool LPUART_HAL_IsCurrentDataWithFrameError (const LPUART_Type ∗base)

  Checks whether the current data word was received with frame error.
- static void LPUART_HAL_SetTxSpecialChar (LPUART_Type ∗base, uint8_t specialChar)

  Indicates a special character is to be transmitted.
- static bool LPUART_HAL_IsCurrentDataWithParityError (const LPUART_Type ∗base)

  Checks whether the current data word was received with parity error.
- static bool LPUART_HAL_IsReceiveBufferEmpty (const LPUART_Type ∗base)

  Checks whether the receive buffer is empty.
- static bool LPUART_HAL_WasPreviousReceiverStateIdle (const LPUART_Type ∗base)

  Checks whether the previous BUS state was idle before this byte is received.
- static void LPUART_HAL_SetWaitModeOperation (LPUART_Type ∗base, lpuart_operation_config_t mode)

  Configures the LPUART operation in wait mode (operates or stops operations in wait mode).
- static lpuart_operation_config_t LPUART_HAL_GetWaitModeOperation (const LPUART_Type ∗base)

  Gets the LPUART operation in wait mode.
- void LPUART_HAL_SetLoopbackCmd (LPUART_Type ∗base, bool enable)

  Configures the LPUART loopback operation (enable/disable loopback operation)
- void LPUART_HAL_SetSingleWireCmd (LPUART_Type ∗base, bool enable)

  Configures the LPUART single-wire operation (enable/disable single-wire mode).
- static void LPUART_HAL_SetTxdirInSinglewireMode (LPUART_Type ∗base, lpuart_singlewire_txdir_t direction)

  Configures the LPUART transmit direction while in single-wire mode.
- status_t LPUART_HAL_SetReceiverInStandbyMode (LPUART_Type ∗base)

  Places the LPUART receiver in standby mode.
- static void LPUART_HAL_PutReceiverInNormalMode (LPUART_Type ∗base)

  Places the LPUART receiver in a normal mode.

- static bool LPUART_HAL_IsReceiverInStandby (const LPUART_Type ∗base)

  Checks whether the LPUART receiver is in a standby mode.
- static void LPUART_HAL_SetReceiverWakeupMode (LPUART_Type ∗base, lpuart_wakeup_method_↩ t method)

  Sets the LPUART receiver wakeup method from standby mode.
- static lpuart_wakeup_method_t LPUART_HAL_GetReceiverWakeupMode (const LPUART_Type ∗base)

  Gets the LPUART receiver wakeup method from standby mode.
- void LPUART_HAL_SetIdleLineDetect (LPUART_Type ∗base, const lpuart_idle_line_config_t ∗config)

  LPUART idle-line detect operation configuration.
- static void LPUART_HAL_SetBreakCharTransmitLength (LPUART_Type ∗base, lpuart_break_char_length↩ _t length)

  LPUART break character transmit length configuration.
- static void LPUART_HAL_SetBreakCharDetectLength (LPUART_Type ∗base, lpuart_break_char_length_↩ t length)

  LPUART break character detect length configuration.
- static void LPUART_HAL_QueueBreakField (LPUART_Type ∗base)

  LPUART transmit sends break character configuration.
- static void LPUART_HAL_SetMatchAddressMode (LPUART_Type ∗base, lpuart_match_config_t config)

  Configures match address mode control.
- void LPUART_HAL_SetMatchAddressReg1 (LPUART_Type ∗base, bool enable, uint8_t value)

  Configures address match register 1.
- void LPUART_HAL_SetMatchAddressReg2 (LPUART_Type ∗base, bool enable, uint8_t value)

  Configures address match register 2.
- static void LPUART_HAL_SetSendMsbFirstCmd (LPUART_Type ∗base, bool enable)

  LPUART sends the MSB first configuration.
- static void LPUART_HAL_SetReceiveResyncCmd (LPUART_Type ∗base, bool enable)

  LPUART enable/disable re-sync of received data configuration.
- static void LPUART_HAL_SetCtsSource (LPUART_Type ∗base, lpuart_cts_source_t source)

  Transmits the CTS source configuration.
- static void LPUART_HAL_SetCtsMode (LPUART_Type ∗base, lpuart_cts_config_t mode)

  Transmits the CTS configuration.
- static void LPUART_HAL_SetTxCtsCmd (LPUART_Type ∗base, bool enable)

  Enable/Disable the transmitter clear-to-send.
- static void LPUART_HAL_SetRxRtsCmd (LPUART_Type ∗base, bool enable)

  Enable/Disable the receiver request-to-send.
- static void LPUART_HAL_SetTxRtsCmd (LPUART_Type ∗base, bool enable)

  Enable/Disable the transmitter request-to-send.
- static void LPUART_HAL_SetTxRtsPolarityMode (LPUART_Type ∗base, bool polarity)

  Configures the transmitter RTS polarity.
- static void LPUART_HAL_SetTxFIFOCmd (LPUART_Type ∗base, bool enable)

  Enable/Disable the transmitter FIFO.
- static void LPUART_HAL_SetRxFIFOCmd (LPUART_Type ∗base, bool enable)

  Enable/Disable the receiver FIFO.
- static void LPUART_HAL_SetRxIdleEmptyDuration (LPUART_Type ∗base, uint8_t duration)

  Enables the assertion of RDRF when the receiver is idle.
- static void LPUART_HAL_FlushTxFifoBuffer (LPUART_Type ∗base)

  Flush tx FIFO buffer.
- static void LPUART_HAL_FlushRxFifoBuffer (LPUART_Type ∗base)

  Flush rx FIFO buffer.
- static void LPUART_HAL_SetTxWatermark (LPUART_Type ∗base, uint8_t txWater)

  Sets the tx watermark.

- static void LPUART_HAL_SetRxWatermark (LPUART_Type ∗base, uint8_t rxWater)

    Sets the rx watermark.
- void LPUART_HAL_SetInfrared (LPUART_Type ∗base, bool enable, lpuart_ir_tx_pulsewidth_t pulseWidth)

    Configures the LPUART infrared operation.

3.56.2 Data Structure Documentation

3.56.2.1 struct lpuart_idle_line_config_t

Structure for idle line configuration settings.

Implements : lpuart_idle_line_config_t_Class

Data Fields

- unsigned idleLineType: 1
- unsigned rxWakeIdleDetect: 1

Field Documentation

3.56.2.1.1 idleLineType

```
unsigned idleLineType
```

ILT, Idle bit count start: 0 - after start bit (default), 1 - after stop bit

3.56.2.1.2 rxWakeIdleDetect

```
unsigned rxWakeIdleDetect
```

RWUID, Receiver Wake Up Idle Detect. IDLE status bit operation during receive standbyControls 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

3.56.3 Macro Definition Documentation

3.56.3.1 LPUART_BAUD_REG_ID

```
#define LPUART_BAUD_REG_ID (1U)
```

3.56.3.2 LPUART_CTRL_REG_ID

```
#define LPUART_CTRL_REG_ID (3U)
```

3.56.3.3 LPUART_DATA_REG_ID

```
#define LPUART_DATA_REG_ID (4U)
```

3.56.3.4 LPUART_FIFO_REG_ID

```
#define LPUART_FIFO_REG_ID (7U)
```

3.56.3.5 LPUART_MATCH_REG_ID

```
#define LPUART_MATCH_REG_ID (5U)
```

3.56.3.6 LPUART_MODIR_REG_ID

```
#define LPUART_MODIR_REG_ID (6U)
```

3.56.3.7 LPUART_SHIFT

```
#define LPUART_SHIFT (16U)
```

3.56.3.8 LPUART_STAT_REG_ID

```
#define LPUART_STAT_REG_ID (2U)
```

3.56.3.9 LPUART_WATER_REG_ID

```
#define LPUART_WATER_REG_ID (8U)
```

3.56.4 Enumeration Type Documentation

3.56.4.1 lpuart_bit_count_per_char_t

```
enum lpuart_bit_count_per_char_t
```

LPUART number of bits in a character.

Implements : lpuart_bit_count_per_char_t_Class

Enumerator

LPUART_8_BITS_PER_CHAR	8-bit data characters
LPUART_9_BITS_PER_CHAR	9-bit data characters
LPUART_10_BITS_PER_CHAR	10-bit data characters

3.56.4.2 lpuart_break_char_length_t

```
enum lpuart_break_char_length_t
```

LPUART break character length settings for transmit/detect.

The actual maximum bit times may vary depending on the LPUART instance.

Implements : lpuart_break_char_length_t_Class

Enumerator

LPUART_BREAK_CHAR_10_BIT_MINIMUM	LPUART 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)
LPUART_BREAK_CHAR_13_BIT_MINIMUM	LPUART break char length 13 bit times (if M = 0, SBNS = 0 or M10 = 0, SBNS = 1) or 14 (if M = 1, SBNS = 0 or M = 1, SBNS = 1) or 15 (if M10 = 1, SBNS = 1 or M10 = 1, SNBS = 0)

3.56.4.3 lpuart_cts_config_t

`enum lpuart_cts_config_t`

LPUART Transmits CTS Source.Configures if the CTS state is checked at the start of each character or only when the transmitter is idle.

Implements : lpuart_cts_config_t_Class

Enumerator

LPUART_CTS_SAMPLED_ON_EACH_CHAR	CTS input is sampled at the start of each character.
LPUART_CTS_SAMPLED_ON_IDLE	CTS input is sampled when the transmitter is idle.

3.56.4.4 lpuart_cts_source_t

`enum lpuart_cts_source_t`

LPUART Transmits the CTS Configuration. Configures the source of the CTS input.

Implements : lpuart_cts_source_t_Class

Enumerator

LPUART_CTS_SOURCE_PIN	CTS input is the LPUART_CTS pin.
LPUART_CTS_SOURCE_INVERTED_RECEIVER_MATCH	CTS input is the inverted Receiver Match result.

3.56.4.5 lpuart_idle_char_t

`enum lpuart_idle_char_t`

LPUART Configures the number of idle characters that must be received before the IDLE flag is set.

Implements : lpuart_idle_char_t_Class

Enumerator

LPUART_1_IDLE_CHAR	1 idle character
LPUART_2_IDLE_CHAR	2 idle character
LPUART_4_IDLE_CHAR	4 idle character
LPUART_8_IDLE_CHAR	8 idle character

Enumerator

LPUART_16_IDLE_CHAR	16 idle character
LPUART_32_IDLE_CHAR	32 idle character
LPUART_64_IDLE_CHAR	64 idle character
LPUART_128_IDLE_CHAR	128 idle character

3.56.4.6 lpuart_interrupt_t

```
enum lpuart_interrupt_t
```

LPUART interrupt configuration structure, default settings are 0 (disabled)

Implements : lpuart_interrupt_t_Class

Enumerator

LPUART_INT_LIN_BREAK_DETECT	LIN break detect.
LPUART_INT_RX_ACTIVE_EDGE	RX Active Edge.
LPUART_INT_TX_DATA_REG_EMPTY	Transmit data register empty.
LPUART_INT_TX_COMPLETE	Transmission complete.
LPUART_INT_RX_DATA_REG_FULL	Receiver data register full.
LPUART_INT_IDLE_LINE	Idle line.
LPUART_INT_RX_OVERRUN	Receiver Overrun.
LPUART_INT_NOISE_ERR_FLAG	Noise error flag.
LPUART_INT_FRAME_ERR_FLAG	Framing error flag.
LPUART_INT_PARITY_ERR_FLAG	Parity error flag.
LPUART_INT_MATCH_ADDR_ONE	Match address one flag.
LPUART_INT_MATCH_ADDR_TWO	Match address two flag.
LPUART_INT_FIFO_TXOF	Transmitter FIFO buffer interrupt
LPUART_INT_FIFO_RXUF	Receiver FIFO buffer interrupt

3.56.4.7 lpuart_ir_tx_pulsewidth_t

```
enum lpuart_ir_tx_pulsewidth_t
```

LPUART infra-red transmitter pulse width options.

Implements : lpuart_ir_tx_pulsewidth_t_Class

Enumerator

LPUART_IR_THREE_SIXTEENTH_WIDTH	3/16 pulse
LPUART_IR_ONE_SIXTEENTH_WIDTH	1/16 pulse
LPUART_IR_ONE_THIRTYSECOND_WIDTH	1/32 pulse
LPUART_IR_ONE_FOURTH_WIDTH	1/4 pulse

3.56.4.8 lpuart_match_config_t

```
enum lpuart_match_config_t
```

LPUART Configures the match addressing mode used.

Implements : lpuart_match_config_t_Class

Enumerator

LPUART_ADDRESS_MATCH_WAKEUP	Address Match Wakeup
LPUART_IDLE_MATCH_WAKEUP	Idle Match Wakeup
LPUART_MATCH_ON_AND_MATCH_OFF	Match On and Match Off
LPUART_ENABLE_RWU_ON_DATA_MATCH	Enables RWU on Data Match and Match On/Off for transmitter CTS input

3.56.4.9 lpuart_operation_config_t

```
enum lpuart_operation_config_t
```

LPUART operation configuration constants.

Implements : lpuart_operation_config_t_Class

Enumerator

LPUART_OPERATES	LPUART continues to operate normally.
LPUART_STOPS	LPUART stops operation.

3.56.4.10 lpuart_parity_mode_t

```
enum lpuart_parity_mode_t
```

LPUART parity mode.

Implements : lpuart_parity_mode_t_Class

Enumerator

LPUART_PARITY_DISABLED	parity disabled
LPUART_PARITY_EVEN	parity enabled, type even, bit setting: PE|PT = 10
LPUART_PARITY_ODD	parity enabled, type odd, bit setting: PE|PT = 11

3.56.4.11 lpuart_singlewire_txdir_t

```
enum lpuart_singlewire_txdir_t
```

LPUART single-wire mode TX direction.

Implements : lpuart_singlewire_txdir_t_Class

Enumerator

LPUART_SINGLE_WIRE_TX_DIR_IN	LPUART Single Wire mode TXDIR input
LPUART_SINGLE_WIRE_TX_DIR_OUT	LPUART Single Wire mode TXDIR output

3.56.4.12 lpuart_status_flag_t

```
enum lpuart_status_flag_t
```

LPUART status flags.

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

Implements : lpuart_status_flag_t_Class

Enumerator

LPUART_TX_DATA_REG_EMPTY	Tx data register empty flag, sets when Tx buffer is empty
LPUART_TX_COMPLETE	Transmission complete flag, sets when transmission activity complete
LPUART_RX_DATA_REG_FULL	Rx data register full flag, sets when the receive data buffer is full
LPUART_IDLE_LINE_DETECT	Idle line detect flag, sets when idle line detected
LPUART_RX_OVERRUN	Rx Overrun sets if new data is received before data is read
LPUART_NOISE_DETECT	Rx takes 3 samples of each received bit. If these differ, the flag sets
LPUART_FRAME_ERR	Frame error flag, sets if logic 0 was detected where stop bit expected
LPUART_PARITY_ERR	If parity enabled, sets upon parity error detection
LPUART_LIN_BREAK_DETECT	LIN break detect interrupt flag, sets when LIN break char detected
LPUART_RX_ACTIVE_EDGE_DETECT	Rx pin active edge interrupt flag, sets when active edge detected
LPUART_RX_ACTIVE	Receiver Active Flag (RAF), sets at beginning of valid start bit
LPUART_NOISE_IN_CURRENT_WORD	NOISY bit, sets if noise detected in current data word
LPUART_PARITY_ERR_IN_CURRENT_WORD	PARITYE bit, sets if noise detected in current data word
LPUART_MATCH_ADDR_ONE	Address one match flag
LPUART_MATCH_ADDR_TWO	Address two match flag
LPUART_FIFO_TX_OF	Transmitter FIFO buffer overflow
LPUART_FIFO_RX_UF	Receiver FIFO buffer underflow

3.56.4.13 lpuart_stop_bit_count_t

```
enum lpuart_stop_bit_count_t
```

LPUART number of stop bits.

Implements : lpuart_stop_bit_count_t_Class

Enumerator

LPUART_ONE_STOP_BIT	one stop bit
LPUART_TWO_STOP_BIT	two stop bits

3.56.4.14 lpuart_wakeup_method_t

```
enum lpuart_wakeup_method_t
```

LPUART wakeup from standby method constants.

Implements : lpuart_wakeup_method_t_Class

Enumerator

LPUART_IDLE_LINE_WAKE	Idle-line wakes the LPUART receiver from standby.
LPUART_ADDR_MARK_WAKE	Addr-mark wakes LPUART receiver from standby.

3.56.5 Function Documentation

3.56.5.1 LPUART_HAL_ClearStatusFlag()

```
status_t LPUART_HAL_ClearStatusFlag (
             LPUART_Type * base,
             lpuart_status_flag_t statusFlag )
```

LPUART clears an individual status flag.

This function clears an individual status flag (see lpuart_status_flag_t for list of status bits).

Parameters

base	LPUART base pointer
statusFlag	Desired LPUART status flag to clear

Returns

STATUS_SUCCESS if successful or STATUS_ERROR if an error occured

3.56.5.2 LPUART_HAL_FlushRxFifoBuffer()

```
static void LPUART_HAL_FlushRxFifoBuffer (
             LPUART_Type * base ) [inline], [static]
```

Flush rx FIFO buffer.

This function flushes the rx FIFO buffer. Note: This does not affect data that is in the receive shift register.

Parameters

base	LPUART base pointer Implements : LPUART_HAL_FlushRxFifoBuffer_Activity

3.56.5.3 LPUART_HAL_FlushTxFifoBuffer()

```
static void LPUART_HAL_FlushTxFifoBuffer (
             LPUART_Type * base )  [inline], [static]
```

Flush tx FIFO buffer.

This function flushes the tx FIFO buffer. Note: This does not affect data that is in the transmit shift register.

Parameters

base	LPUART base pointer Implements : LPUART_HAL_FlushTxFifoBuffer_Activity

3.56.5.4 LPUART_HAL_Getchar()

```
static void LPUART_HAL_Getchar (
             const LPUART_Type * base,
             uint8_t * readData )  [inline], [static]
```

Gets the LPUART 8-bit character.

This functions receives an 8-bit character.

Parameters

base	LPUART base pointer
readData	Data read from receive (8-bit) Implements : LPUART_HAL_Getchar_Activity

3.56.5.5 LPUART_HAL_Getchar10()

```
void LPUART_HAL_Getchar10 (
             const LPUART_Type * base,
             uint16_t * readData )
```

Gets the LPUART 10-bit character.

This functions receives a 10-bit character.

Parameters

base	LPUART base pointer
readData	Data read from receive (10-bit)

3.56.5.6 LPUART_HAL_Getchar9()

```
void LPUART_HAL_Getchar9 (
            const LPUART_Type * base,
            uint16_t * readData )
```

Gets the LPUART 9-bit character.

This functions receives a 9-bit character.

Parameters

base	LPUART base pointer
readData	Data read from receive (9-bit)

3.56.5.7 LPUART_HAL_GetDataRegAddr()

```
static const volatile void* LPUART_HAL_GetDataRegAddr (
            const LPUART_Type * base )  [inline], [static]
```

Get LPUART tx/rx data register address.

This function returns LPUART tx/rx data register address.

Parameters

base	LPUART base pointer.

Returns

LPUART tx/rx data register address. Implements : LPUART_HAL_GetDataRegAddr_Activity

3.56.5.8 LPUART_HAL_GetIdleChar()

```
static lpuart_idle_char_t LPUART_HAL_GetIdleChar (
            const LPUART_Type * base )  [inline], [static]
```

Gets the number of idle characters for IDLE flag.

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

Parameters

base	LPUART base pointer

Returns

idle characters configuration Implements : LPUART_HAL_GetIdleChar_Activity

3.56.5.9 LPUART_HAL_GetIntMode()

```
bool LPUART_HAL_GetIntMode (
            const LPUART_Type * base,
            lpuart_interrupt_t intSrc )
```

Returns LPUART module interrupts state.

This function returns whether a certain LPUART module interrupt is enabled or disabled.

Parameters

base	LPUART module base pointer.
intSrc	LPUART interrupt configuration data.

Returns

true: enable, false: disable.

3.56.5.10 LPUART_HAL_GetReceiverCmd()

```
static bool LPUART_HAL_GetReceiverCmd (
            const LPUART_Type * base )  [inline], [static]
```

Gets the LPUART receiver enabled/disabled configuration.

This function returns true if the LPUART receiver is enabled, or false, when the transmitter is disabled.

Parameters

base	LPUART base pointer

Returns

State of LPUART receiver enable(true)/disable(false) Implements : LPUART_HAL_GetReceiverCmd_Activity

3.56.5.11 LPUART_HAL_GetReceiverWakeupMode()

```
static lpuart_wakeup_method_t LPUART_HAL_GetReceiverWakeupMode (
            const LPUART_Type * base )  [inline], [static]
```

Gets the LPUART receiver wakeup method from standby mode.

This function returns the LPUART receiver wakeup method (idle line or addr-mark) from standby mode.

Parameters

base	LPUART base pointer

Returns

LPUART wakeup method: LPUART_IDLE_LINE_WAKE: 0 - Idle-line wake (default), LPUART_ADDR_MA↩ RK_WAKE: 1 - addr-mark wake Implements : LPUART_HAL_GetReceiverWakeupMode_Activity

3.56.5.12 LPUART_HAL_GetRxDataPolarity()

```
static bool LPUART_HAL_GetRxDataPolarity (
            const LPUART_Type * base )  [inline], [static]
```

Returns whether the receive data is inverted or not.

This function returns the polarity of the receive data.

Parameters

base	LPUART base pointer.

Returns

Rx data polarity; true: inverted, false: not inverted. Implements : LPUART_HAL_GetRxDataPolarity_Activity

3.56.5.13 LPUART_HAL_GetStatusFlag()

```
bool LPUART_HAL_GetStatusFlag (
            const LPUART_Type * base,
            lpuart_status_flag_t statusFlag )
```

LPUART get status flag.

This function returns the state of a status flag.

Parameters

base	LPUART base pointer
statusFlag	The status flag to query

Returns

Whether the current status flag is set(true) or not(false).

3.56.5.14 LPUART_HAL_GetTransmitterCmd()

```
static bool LPUART_HAL_GetTransmitterCmd (
            const LPUART_Type * base )  [inline], [static]
```

Gets the LPUART transmitter enabled/disabled configuration.

This function returns true if the LPUART transmitter is enabled, or false, when the transmitter is disabled.

Parameters

base	LPUART base pointer

Returns

State of LPUART transmitter enable(true)/disable(false) Implements : LPUART_HAL_GetTransmitterCmd_↩
Activity

3.56.5.15 LPUART_HAL_GetWaitModeOperation()

```
static lpuart_operation_config_t LPUART_HAL_GetWaitModeOperation (
             const LPUART_Type * base )  [inline], [static]
```

Gets the LPUART operation in wait mode.

This function returns the LPUART operation in wait mode (operates or stops operations in wait mode).

Parameters

base	LPUART base pointer

Returns

LPUART wait mode operation configuration - LPUART_OPERATES or LPUART_STOPS in wait mode Implements : LPUART_HAL_GetWaitModeOperation_Activity

3.56.5.16 LPUART_HAL_Init()

```
void LPUART_HAL_Init (
             LPUART_Type * base )
```

Initializes the LPUART controller.

This function Initializes the LPUART controller to known state.

Parameters

base	LPUART base pointer.

3.56.5.17 LPUART_HAL_IsCurrentDataWithFrameError()

```
static bool LPUART_HAL_IsCurrentDataWithFrameError (
             const LPUART_Type * base )  [inline], [static]
```

Checks whether the current data word was received with frame error.

This function returns whether the current data word was received with frame error.

Parameters

base	LPUART base pointer

Returns

The status of the FRETSC bit in the LPUART extended data register Implements : LPUART_HAL_IsCurrent↩ DataWithFrameError_Activity

3.56.5.18 LPUART_HAL_IsCurrentDataWithNoise()

```
static bool LPUART_HAL_IsCurrentDataWithNoise (
            const LPUART_Type * base )  [inline], [static]
```

Checks whether the current data word was received with noise.

This function returns whether the current data word was received with noise.

Parameters

base	LPUART base pointer.

Returns

The status of the NOISY bit in the LPUART extended data register Implements : LPUART_HAL_IsCurrent←DataWithNoise_Activity

3.56.5.19 LPUART_HAL_IsCurrentDataWithParityError()

```
static bool LPUART_HAL_IsCurrentDataWithParityError (
            const LPUART_Type * base )  [inline], [static]
```

Checks whether the current data word was received with parity error.

This function returns whether the current data word was received with parity error.

Parameters

base	LPUART base pointer

Returns

The status of the PARITYE bit in the LPUART extended data register Implements : LPUART_HAL_IsCurrent←DataWithParityError_Activity

3.56.5.20 LPUART_HAL_IsReceiveBufferEmpty()

```
static bool LPUART_HAL_IsReceiveBufferEmpty (
            const LPUART_Type * base )  [inline], [static]
```

Checks whether the receive buffer is empty.

This function returns whether the receive buffer is empty.

Parameters

base	LPUART base pointer

Returns

TRUE if the receive-buffer is empty, else FALSE. Implements : LPUART_HAL_IsReceiveBufferEmpty_Activity

3.56.5.21 LPUART_HAL_IsReceiverInStandby()

```
static bool LPUART_HAL_IsReceiverInStandby (
            const LPUART_Type * base )  [inline], [static]
```

Checks whether the LPUART receiver is in a standby mode.

This function returns whether the LPUART receiver is in a standby mode.

Parameters

base	LPUART base pointer

Returns

LPUART in normal more (0) or standby (1) Implements : LPUART_HAL_IsReceiverInStandby_Activity

3.56.5.22 LPUART_HAL_IsRxDmaEnabled()

```
static bool LPUART_HAL_IsRxDmaEnabled (
            const LPUART_Type * base )  [inline], [static]
```

Gets the LPUART DMA request configuration.

This function returns the LPUART Receive DMA request configuration.

Parameters

base	LPUART base pointer

Returns

Receives the DMA request configuration (enable: 1/disable: 0). Implements : LPUART_HAL_IsRxDma←
Enabled_Activity

3.56.5.23 LPUART_HAL_IsTxDmaEnabled()

```
static bool LPUART_HAL_IsTxDmaEnabled (
            const LPUART_Type * base )  [inline], [static]
```

Gets the LPUART DMA request configuration.

This function returns the LPUART Transmit DMA request configuration.

Parameters

base	LPUART base pointer

Returns

Transmit DMA request configuration (enable: 1/disable: 0) Implements : LPUART_HAL_IsTxDmaEnabled_↩ Activity

3.56.5.24 LPUART_HAL_Putchar()

```
static void LPUART_HAL_Putchar (
             LPUART_Type * base,
             uint8_t data )  [inline], [static]
```

Sends the LPUART 8-bit character.

This functions sends an 8-bit character.

Parameters

base	LPUART Instance
data	data to send (8-bit) Implements : LPUART_HAL_Putchar_Activity

3.56.5.25 LPUART_HAL_Putchar10()

```
void LPUART_HAL_Putchar10 (
             LPUART_Type * base,
             uint16_t data )
```

Sends the LPUART 10-bit character (Note: Feature available on select LPUART instances).

This functions sends a 10-bit character.

Parameters

base	LPUART Instance
data	data to send (10-bit)

3.56.5.26 LPUART_HAL_Putchar9()

```
void LPUART_HAL_Putchar9 (
             LPUART_Type * base,
             uint16_t data )
```

Sends the LPUART 9-bit character.

This functions sends a 9-bit character.

Parameters

base	LPUART Instance
data	data to send (9-bit)

3.56.5.27 LPUART_HAL_PutReceiverInNormalMode()

```
static void LPUART_HAL_PutReceiverInNormalMode (
            LPUART_Type * base )  [inline], [static]
```

Places the LPUART receiver in a normal mode.

This function places the LPUART receiver in a normal mode (disable standby mode operation).

Parameters

base	LPUART base pointer Implements : LPUART_HAL_PutReceiverInNormalMode_Activity

3.56.5.28 LPUART_HAL_QueueBreakField()

```
static void LPUART_HAL_QueueBreakField (
            LPUART_Type * base )  [inline], [static]
```

LPUART transmit sends break character configuration.

This function sets break character transmission in queue mode.

Parameters

base	LPUART base pointer Implements : LPUART_HAL_QueueBreakField_Activity

3.56.5.29 LPUART_HAL_ReceiveDataPolling()

```
status_t LPUART_HAL_ReceiveDataPolling (
            LPUART_Type * base,
            uint8_t * rxBuff,
            uint32_t rxSize )
```

Receive multiple bytes of data using polling method.

This function only supports 8-bit transaction.

Parameters

base	LPUART module base pointer.
rxBuff	The buffer pointer which saves the data to be received.
rxSize	Size of data need to be received in unit of byte.

Returns

STATUS_SUCCESS if the transaction is success or STATUS_UART_RX_OVERRUN if rx overrun.

3.56.5.30 LPUART_HAL_SendDataPolling()

```
void LPUART_HAL_SendDataPolling (
            LPUART_Type * base,
            const uint8_t * txBuff,
            uint32_t txSize )
```

Send out multiple bytes of data using polling method.

This function only supports 8-bit transaction.

Parameters

base	LPUART module base pointer.
txBuff	The buffer pointer which saves the data to be sent.
txSize	Size of data to be sent in unit of byte.

3.56.5.31 LPUART_HAL_SetBaudRate()

```
status_t LPUART_HAL_SetBaudRate (
            LPUART_Type * base,
            uint32_t sourceClockInHz,
            uint32_t desiredBaudRate )
```

Configures the LPUART baud rate.

This function configures the LPUART baud rate. 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

base	LPUART base pointer.
sourceClockInHz	LPUART source input clock in Hz.
desiredBaudRate	LPUART desired baud rate.

Returns

STATUS_SUCCESS if successful or STATUS_ERROR if an error occured

3.56.5.32 LPUART_HAL_SetBaudRateDivisor()

```
static void LPUART_HAL_SetBaudRateDivisor (
            LPUART_Type * base,
            uint32_t baudRateDivisor ) [inline], [static]
```

Sets the LPUART baud rate modulo divisor.

This function sets the LPUART baud rate modulo divisor.

Parameters

base	LPUART base pointer.
baudRateDivisor	The baud rate modulo division "SBR" Implements : LPUART_HAL_SetBaudRateDivisor_Activity

3.56.5.33 LPUART_HAL_SetBitCountPerChar()

```
void LPUART_HAL_SetBitCountPerChar (
             LPUART_Type * base,
             lpuart_bit_count_per_char_t bitCountPerChar )
```

Configures the number of bits per character in the LPUART controller.

This function 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. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

base	LPUART base pointer.
bitCountPerChar	Number of bits per char (8, 9, or 10, depending on the LPUART instance)

3.56.5.34 LPUART_HAL_SetBothEdgeSamplingCmd()

```
static void LPUART_HAL_SetBothEdgeSamplingCmd (
             LPUART_Type * base,
             bool enable ) [inline], [static]
```

Configures the LPUART baud rate both edge sampling.

This function configures the LPUART baud rate both edge sampling. (Note: Feature available on select LPUART instances used with baud rate programming) When enabled, the received data is sampled on both edges of the baud rate clock. This must be set when the oversampling ratio is between 4x and 7x. This function should only be called when the receiver is disabled.

Parameters

base	LPUART base pointer.
enable	Enable (1) or Disable (0) Both Edge Sampling Implements : LPUART_HAL_SetBothEdgeSamplingCmd_Activity

3.56.5.35 LPUART_HAL_SetBreakCharDetectLength()

```
static void LPUART_HAL_SetBreakCharDetectLength (
             LPUART_Type * base,
             lpuart_break_char_length_t length ) [inline], [static]
```

LPUART break character detect length configuration.

This function sets the LPUART detectable break character length.

Parameters

base	LPUART base pointer
length	LPUART break character length setting: 0 - minimum 10-bit times (default), 1 - minimum 13-bit times Implements : LPUART_HAL_SetBreakCharDetectLength_Activity

3.56.5.36 LPUART_HAL_SetBreakCharTransmitLength()

```
static void LPUART_HAL_SetBreakCharTransmitLength (
            LPUART_Type * base,
            lpuart_break_char_length_t length )  [inline], [static]
```

LPUART break character transmit length configuration.

This function configures the break char length. In some LPUART instances, the user should disable the transmitter before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

base	LPUART base pointer
length	LPUART break character length setting: 0 - minimum 10-bit times (default), 1 - minimum 13-bit times Implements : LPUART_HAL_SetBreakCharTransmitLength_Activity

3.56.5.37 LPUART_HAL_SetCtsMode()

```
static void LPUART_HAL_SetCtsMode (
            LPUART_Type * base,
            lpuart_cts_config_t mode )  [inline], [static]
```

Transmits the CTS configuration.

This function transmits the CTS configuration. Note: configures if the CTS state is checked at the start of each character or only when the transmitter is idle.

Parameters

base	LPUART base pointer
mode	LPUART CTS configuration Implements : LPUART_HAL_SetCtsMode_Activity

3.56.5.38 LPUART_HAL_SetCtsSource()

```
static void LPUART_HAL_SetCtsSource (
            LPUART_Type * base,
            lpuart_cts_source_t source )  [inline], [static]
```

Transmits the CTS source configuration.

This function transmits the CTS source configuration.

Parameters

base	LPUART base pointer
source	LPUART CTS source Implements : LPUART_HAL_SetCtsSource_Activity

3.56.5.39 LPUART_HAL_SetIdleChar()

```
static void LPUART_HAL_SetIdleChar (
            LPUART_Type * base,
            lpuart_idle_char_t idleConfig )  [inline], [static]
```

Configures the number of idle characters.

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

Parameters

base	LPUART base pointer
idleConfig	Idle characters configuration Implements : LPUART_HAL_SetIdleChar_Activity

3.56.5.40 LPUART_HAL_SetIdleLineDetect()

```
void LPUART_HAL_SetIdleLineDetect (
            LPUART_Type * base,
            const lpuart_idle_line_config_t * config )
```

LPUART idle-line detect operation configuration.

This function configures idle-line detect operation configuration (idle line bit-count start and wake up affect on IDLE status bit). 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

base	LPUART base pointer
config	LPUART configuration data for idle line detect operation

3.56.5.41 LPUART_HAL_SetInfrared()

```
void LPUART_HAL_SetInfrared (
            LPUART_Type * base,
            bool enable,
            lpuart_ir_tx_pulsewidth_t pulseWidth )
```

Configures the LPUART infrared operation.

This function configures the LPUART infrared operation.

Parameters

base	LPUART base pointer
enable	Enable (1) or disable (0) the infrared operation
pulseWidth	The transmit narrow pulse width of type lpuart_ir_tx_pulsewidth_t

3.56.5.42 LPUART_HAL_SetIntMode()

```
void LPUART_HAL_SetIntMode (
            LPUART_Type * base,
            lpuart_interrupt_t intSrc,
            bool enable )
```

Configures the LPUART module interrupts.

This function configures the LPUART module interrupts to enable/disable various interrupt sources.

Parameters

base	LPUART module base pointer.
intSrc	LPUART interrupt configuration data.
enable	true: enable, false: disable.

3.56.5.43 LPUART_HAL_SetLoopbackCmd()

```
void LPUART_HAL_SetLoopbackCmd (
            LPUART_Type * base,
            bool enable )
```

Configures the LPUART loopback operation (enable/disable loopback operation)

This function configures the LPUART loopback operation (enable/disable loopback operation). 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

base	LPUART base pointer
enable	LPUART loopback mode - disabled (0) or enabled (1)

3.56.5.44 LPUART_HAL_SetMatchAddressMode()

```
static void LPUART_HAL_SetMatchAddressMode (
            LPUART_Type * base,
            lpuart_match_config_t config ) [inline], [static]
```

Configures match address mode control.

This function configures match address mode control.

Parameters

base	LPUART base pointer
config	MATCFG: Configures the match addressing mode used. Implements : LPUART_HAL_SetMatchAddressMode_Activity

3.56.5.45 LPUART_HAL_SetMatchAddressReg1()

```
void LPUART_HAL_SetMatchAddressReg1 (
            LPUART_Type * base,
            bool enable,
            uint8_t value )
```

Configures address match register 1.

This function configures address match register 1. The MAEN bit must be cleared before configuring MA value, so the enable/disable and set value must be included inside one function.

Parameters

base	LPUART base pointer
enable	Match address model enable (true)/disable (false)
value	Match address value to program into match address register 1

3.56.5.46 LPUART_HAL_SetMatchAddressReg2()

```
void LPUART_HAL_SetMatchAddressReg2 (
            LPUART_Type * base,
            bool enable,
            uint8_t value )
```

Configures address match register 2.

This function configures address match register 2. The MAEN bit must be cleared before configuring MA value, so the enable/disable and set value must be included inside one function.

Parameters

base	LPUART base pointer
enable	Match address model enable (true)/disable (false)
value	Match address value to program into match address register 2

3.56.5.47 LPUART_HAL_SetOversamplingRatio()

```
static void LPUART_HAL_SetOversamplingRatio (
            LPUART_Type * base,
            uint32_t overSamplingRatio )  [inline], [static]
```

Sets the LPUART baud rate oversampling ratio.

This function sets the LPUART baud rate oversampling ratio. (Note: Feature available on select LPUART instances used together with baud rate programming) The oversampling ratio should be set between 4x (00011) and 32x (11111). Writing an invalid oversampling ratio results in an error and is set to a default 16x (01111) oversampling ratio. Disable the transmitter/receiver before calling this function.

Parameters

base	LPUART base pointer.
overSamplingRatio	The oversampling ratio "OSR" Implements : LPUART_HAL_SetOversamplingRatio_Activity

3.56.5.48 LPUART_HAL_SetParityMode()

```
void LPUART_HAL_SetParityMode (
            LPUART_Type * base,
            lpuart_parity_mode_t parityModeType )
```

Configures parity mode in the LPUART controller.

This function configures parity mode in the LPUART controller. 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

base	LPUART base pointer.
parityModeType	Parity mode (enabled, disable, odd, even - see parity_mode_t struct)

3.56.5.49 LPUART_HAL_SetReceiverCmd()

```
static void LPUART_HAL_SetReceiverCmd (
            LPUART_Type * base,
            bool enable ) [inline], [static]
```

Enable/Disable the LPUART receiver.

This function enables or disables the LPUART receiver, based on the parameter received.

Parameters

base	LPUART base pointer
enable	Enable(true) or disable(false) receiver. Implements : LPUART_HAL_SetReceiverCmd_Activity

3.56.5.50 LPUART_HAL_SetReceiveResyncCmd()

```
static void LPUART_HAL_SetReceiveResyncCmd (
            LPUART_Type * base,
            bool enable ) [inline], [static]
```

LPUART enable/disable re-sync of received data configuration.

This function enables or disables re-sync of received data, based on the parameter received.

Parameters

base	LPUART base pointer
enable	re-sync of received data word configuration: true - re-sync of received data word (default) false - disable the re-sync Implements : LPUART_HAL_SetReceiveResyncCmd_Activity

3.56.5.51 LPUART_HAL_SetReceiverInStandbyMode()

```
status_t LPUART_HAL_SetReceiverInStandbyMode (
            LPUART_Type * base )
```

Places the LPUART receiver in standby mode.

This function places the LPUART receiver in standby mode.

Parameters

base	LPUART base pointer

Returns

STATUS_SUCCESS if successful or STATUS_ERROR if an error occured

3.56.5.52 LPUART_HAL_SetReceiverWakeupMode()

```
static void LPUART_HAL_SetReceiverWakeupMode (
             LPUART_Type * base,
             lpuart_wakeup_method_t method )  [inline], [static]
```

Sets the LPUART receiver wakeup method from standby mode.

This function sets the LPUART receiver wakeup method (idle line or addr-mark) from standby mode.

Parameters

base	LPUART base pointer
method	LPUART wakeup method: 0 - Idle-line wake (default), 1 - addr-mark wake Implements : LPUART_HAL_SetReceiverWakeupMode_Activity

3.56.5.53 LPUART_HAL_SetRxDataPolarity()

```
static void LPUART_HAL_SetRxDataPolarity (
             LPUART_Type * base,
             bool polarity )  [inline], [static]
```

Sets whether the recevie data is inverted or not.

This function sets the polarity of the receive data.

Parameters

base	LPUART base pointer.
polarity	Rx Data polarity; true: inverted, false: not inverted. Implements : LPUART_HAL_SetRxDataPolarity_Activity

3.56.5.54 LPUART_HAL_SetRxDmaCmd()

```
static void LPUART_HAL_SetRxDmaCmd (
             LPUART_Type * base,
             bool enable )  [inline], [static]
```

Configures DMA requests.

This function configures DMA requests for LPUART Receiver.

Parameters

base	LPUART base pointer
enable	Receive DMA request configuration (enable: 1/disable: 0) Implements : LPUART_HAL_SetRxDmaCmd_Activity

3.56.5.55 LPUART_HAL_SetRxFIFOCmd()

```
static void LPUART_HAL_SetRxFIFOCmd (
            LPUART_Type * base,
            bool enable )  [inline], [static]
```

Enable/Disable the receiver FIFO.

This function enables or disables the receiver FIFO structure, based on the parameter received. Note: The size of the FIFO structure is indicated by RXFIFOSIZE.

Parameters

base	LPUART base pointer
enable	disable(0)/enable(1) receiver FIFO. Implements : LPUART_HAL_SetRxFIFOCmd_Activity

3.56.5.56 LPUART_HAL_SetRxIdleEmptyDuration()

```
static void LPUART_HAL_SetRxIdleEmptyDuration (
            LPUART_Type * base,
            uint8_t duration )  [inline], [static]
```

Enables the assertion of RDRF when the receiver is idle.

This function enables the assertion of RDRF when the receiver is idle for a number of idle characters and the FIFO is not empty.

Parameters

base	LPUART base pointer.
duration	The number of characters the receiver must be empty before RDRF assertion 0 - disabled, >0 - rx must be idle for 2^(duration-1) characters before RDRF assertion

Implements : LPUART_HAL_SetRxIdleEmptyDuration_Activity

3.56.5.57 LPUART_HAL_SetRxRtsCmd()

```
static void LPUART_HAL_SetRxRtsCmd (
            LPUART_Type * base,
            bool enable )  [inline], [static]
```

Enable/Disable the receiver request-to-send.

This function enables or disables the receiver request-to-send, based on the parameter received. Note: do not enable both Receiver RTS (RXRTSE) and Transmit RTS (TXRTSE).

Parameters

base	LPUART base pointer
enable	disable(0)/enable(1) receiver RTS. Implements : LPUART_HAL_SetRxRtsCmd_Activity

3.56.5.58 LPUART_HAL_SetRxWatermark()

```
static void LPUART_HAL_SetRxWatermark (
             LPUART_Type * base,
             uint8_t rxWater )  [inline], [static]
```

Sets the rx watermark.

This function sets the rx FIFO watermark. When the number of datawords in the receive FIFO/buffer is greater than the value in this register field, an interrupt or a DMA request is generated. Note: For proper operation, the value in RXWATER must be set to be less than the receive FIFO/buffer size and greater than 0.

Parameters

base	LPUART base pointer
rxWater	Rx FIFO Watermark Implements : LPUART_HAL_SetRxWatermark_Activity

3.56.5.59 LPUART_HAL_SetSendMsbFirstCmd()

```
static void LPUART_HAL_SetSendMsbFirstCmd (
             LPUART_Type * base,
             bool enable )  [inline], [static]
```

LPUART sends the MSB first configuration.

This function configures whether MSB is sent first. Note: 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

base	LPUART base pointer
enable	false - LSB (default, disabled), true - MSB (enabled) Implements : LPUART_HAL_SetSendMsbFirstCmd_Activity

3.56.5.60 LPUART_HAL_SetSingleWireCmd()

```
void LPUART_HAL_SetSingleWireCmd (
             LPUART_Type * base,
             bool enable )
```

Configures the LPUART single-wire operation (enable/disable single-wire mode).

This function 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. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

base	LPUART base pointer
enable	LPUART loopback mode - disabled (0) or enabled (1)

3.56.5.61 LPUART_HAL_SetStopBitCount()

```
static void LPUART_HAL_SetStopBitCount (
            LPUART_Type * base,
            lpuart_stop_bit_count_t stopBitCount )  [inline], [static]
```

Configures the number of stop bits in the LPUART controller.

This function configures the number of stop bits in the LPUART controller. 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

base	LPUART base pointer.
stopBitCount	Number of stop bits (1 or 2 - see lpuart_stop_bit_count_t struct) Implements : LPUART_HAL_SetStopBitCount_Activity

3.56.5.62 LPUART_HAL_SetTransmitterCmd()

```
static void LPUART_HAL_SetTransmitterCmd (
            LPUART_Type * base,
            bool enable )  [inline], [static]
```

Enable/Disable the LPUART transmitter.

This function enables or disables the LPUART transmitter, based on the parameter received.

Parameters

base	LPUART base pointer.
enable	Enable(true) or disable(false) transmitter. Implements : LPUART_HAL_SetTransmitterCmd_Activity

3.56.5.63 LPUART_HAL_SetTxCtsCmd()

```
static void LPUART_HAL_SetTxCtsCmd (
            LPUART_Type * base,
            bool enable )  [inline], [static]
```

Enable/Disable the transmitter clear-to-send.

This function controls the transmitter clear-to-send, based on the parameter received.

Parameters

base	LPUART base pointer
enable	disable(0)/enable(1) transmitter CTS. Implements : LPUART_HAL_SetTxCtsCmd_Activity

3.56.5.64 LPUART_HAL_SetTxdirInSinglewireMode()

```
static void LPUART_HAL_SetTxdirInSinglewireMode (
            LPUART_Type * base,
            lpuart_singlewire_txdir_t direction )  [inline], [static]
```

Configures the LPUART transmit direction while in single-wire mode.

This function configures the LPUART transmit direction while in single-wire mode.

Parameters

base	LPUART base pointer
direction	LPUART single-wire transmit direction - input or output Implements : LPUART_HAL_SetTxdirInSinglewireMode_Activity

3.56.5.65 LPUART_HAL_SetTxDmaCmd()

```
static void LPUART_HAL_SetTxDmaCmd (
            LPUART_Type * base,
            bool enable )  [inline], [static]
```

Configures DMA requests.

This function configures DMA requests for LPUART Transmitter.

Parameters

base	LPUART base pointer
enable	Transmit DMA request configuration (enable:1 /disable: 0) Implements : LPUART_HAL_SetTxDmaCmd_Activity

3.56.5.66 LPUART_HAL_SetTxFIFOCmd()

```
static void LPUART_HAL_SetTxFIFOCmd (
            LPUART_Type * base,
            bool enable )  [inline], [static]
```

Enable/Disable the transmitter FIFO.

This function enables or disables the transmitter FIFO structure, based on the parameter received. Note: The size of the FIFO structure is indicated by TXFIFOSIZE.

Parameters

base	LPUART base pointer
enable	disable(0)/enable(1) transmitter FIFO. Implements : LPUART_HAL_SetTxFIFOCmd_Activity

3.56.5.67 LPUART_HAL_SetTxRtsCmd()

```
static void LPUART_HAL_SetTxRtsCmd (
            LPUART_Type * base,
            bool enable )  [inline], [static]
```

Enable/Disable the transmitter request-to-send.

This function enables or disables the transmitter request-to-send, based on the parameter received. Note: do not enable both Receiver RTS (RXRTSE) and Transmit RTS (TXRTSE).

Parameters

base	LPUART base pointer
enable	disable(0)/enable(1) transmitter RTS. Implements : LPUART_HAL_SetTxRtsCmd_Activity

3.56.5.68 LPUART_HAL_SetTxRtsPolarityMode()

```
static void LPUART_HAL_SetTxRtsPolarityMode (
            LPUART_Type * base,
            bool polarity )  [inline], [static]
```

Configures the transmitter RTS polarity.

This function configures the transmitter RTS polarity.

Parameters

base	LPUART base pointer
polarity	Settings to choose RTS polarity (0=active low, 1=active high) Implements : LPUART_HAL_SetTxRtsPolarityMode_Activity

3.56.5.69 LPUART_HAL_SetTxSpecialChar()

```
static void LPUART_HAL_SetTxSpecialChar (
            LPUART_Type * base,
            uint8_t specialChar )  [inline], [static]
```

Indicates a special character is to be transmitted.

This function sets this bit to indicate a break or idle character is to be transmitted instead of the contents in DA↩ TA[T9:T0].

Parameters

base	LPUART base pointer
specialChar	-> 0 - break character, 1 - idle character Implements : LPUART_HAL_SetTxSpecialChar_Activity

3.56.5.70 LPUART_HAL_SetTxWatermark()

```
static void LPUART_HAL_SetTxWatermark (
            LPUART_Type * base,
            uint8_t txWater )  [inline], [static]
```

Sets the tx watermark.

This function sets the tx FIFO watermark. When the number of datawords in the transmit FIFO/buffer is equal to or less than the value in this register field, an interrupt or a DMA request is generated. Note: For proper operation, the value in TXWATER must be set to be less than the transmit FIFO/buffer size and greater than 0.

Parameters

base	LPUART base pointer
txWater	Tx FIFO Watermark Implements : LPUART_HAL_SetTxWatermark_Activity

3.56.5.71 LPUART_HAL_SetWaitModeOperation()

```
static void LPUART_HAL_SetWaitModeOperation (
             LPUART_Type * base,
             lpuart_operation_config_t mode )  [inline], [static]
```

Configures the LPUART operation in wait mode (operates or stops operations in wait mode).

This function configures the LPUART operation in wait mode (operates or stops operations in wait 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

base	LPUART base pointer
mode	LPUART wait mode operation - operates or stops to operate in wait mode. Implements : LPUART_HAL_SetWaitModeOperation_Activity

3.56.5.72 LPUART_HAL_WasPreviousReceiverStateIdle()

```
static bool LPUART_HAL_WasPreviousReceiverStateIdle (
             const LPUART_Type * base )  [inline], [static]
```

Checks whether the previous BUS state was idle before this byte is received.

This function returns whether the previous BUS state was idle before this byte is received.

Parameters

base	LPUART base pointer

Returns

TRUE if the previous BUS state was IDLE, else FALSE. Implements : LPUART_HAL_WasPreviousReceiver↩ StateIdle_Activity

3.57 Low Power Inter-Integrated Circuit (LPI2C)

3.57.1 Detailed Description

The LPI2C is a low power Inter-Integrated Circuit (I2C) module that supports an efficient interface to an I2C bus as a master and/or a slave.

Modules

- LPI2C Driver

  Low Power Inter-Integrated Circuit (LPI2C) Peripheral Driver.
- LPI2C HAL

  Low Power Inter-Integrated Circuit (LPI2C) Hardware Abstraction Layer.

3.58 Low Power Interrupt Timer (LPIT)

3.58.1 Detailed Description

The Low Power Periodic Interrupt Timer (LPIT) is a multi-channel timer module generating independent pre-trigger and trigger outputs. These timer channels can operate individually or can be chained together. The LPIT can operate in low power modes if configured to do so. The pre-trigger and trigger outputs can be used to trigger other modules on the device.

The S32 SDK provides both HAL and Peripheral Drivers for the Low Power Interrupt Timer (LPIT) module of S32K devices.

Modules

- LPIT Driver

    Low Power Interrupt Timer Peripheral Driver.
- LPIT HAL

    Low Power Interrupt Timer module Hardware Abstraction Layer.

3.59 Low Power Serial Peripheral Interface (LPSPI)

3.59.1 Detailed Description

Low Power Serial Peripheral Interface (LPSPI) Peripheral Driver.

The LPSPI driver allows communication on an SPI bus using the LPSPI module in the S32144K processor.

Features

- Interrupt based
- Master or slave operation
- Provides blocking and non-blocking transmit and receive functions
- RX and TX hardware buffers (4 words)
- 4 configurable chip select
- Configurable baud rate

How to integrate LPSPI in your application

In order to use the LPSPI driver it must be first initialized in either master or slave mode, using functions LPSP↵ I_DRV_MasterInit() or LPSPI_DRV_SlaveInit(). Once initialized, it cannot be initialized again for the same LPSPI module instance until it is de-initialized, using LPSPI_DRV_MasterDeinit() or LPSPI_DRV_SlaveDeinit(). Different LPSPI module instances can function independently of each other.

In each mode (master/slave) are available two types of transfers: blocking and non-blocking. The functions which initiate blocking transfers will configure the time out for transmission. If time expires LPSPI_MasterTransferBlocking/↵ LPSPI_SlaveTransferBlocking will return error and the transmission will be aborted.

Important Notes

- The driver enables the interrupts for the corresponding LPSPI module, but any interrupt priority setting must be done by the application.
- The watermarks will be set by the application.
- The driver will configure SCK to PCS delay, PCS to SCK delay, delay between transfers with default values. If you application needs other values for this parameters LPSPI_DRV_MasterSetDelay function can be uesd.

Example code

```
const lpspi_master_config_t Send_MasterConfig0 = {
    .bitsPerSec = 50000U,
    .whichPcs = LPSPI_PCS0,
    .pcsPolarity = LPSPI_ACTIVE_HIGH,
    .isPcsContinuous = false,
    .bitcount = 8U,
    .lpspiSrcClk = 8000000U,
    .clkPhase = LPSPI_CLOCK_PHASE_1ST_EDGE,
    .clkPolarity = LPSPI_SCK_ACTIVE_HIGH,
    .lsbFirst = false,
    .transferType = LPSPI_USING_INTERRUPTS,
};
const lpspi_slave_config_t Receive_SlaveConfig0 = {
    .pcsPolarity = LPSPI_ACTIVE_HIGH,
    .bitcount = 8U,
    .clkPhase = LPSPI_CLOCK_PHASE_1ST_EDGE,
    .whichPcs = LPSPI_PCS0,
    .clkPolarity = LPSPI_SCK_ACTIVE_HIGH,
    .lsbFirst = false,
    .transferType = LPSPI_USING_INTERRUPTS,
};
/* Initialize clock and pins */
LPSPI_DRV_MasterInit(0U, &masterState, &Send_MasterConfig0);
/* Set delay between transfer, PCStoSCK and SCKtoPCS to 10 microseconds. */
LPSPI_DRV_MasterSetDelay(0U, 10U, 10U, 10u);
/* Initialize LPSPI1 (Slave)*/
LPSPI_DRV_SlaveInit(1U, &slaveState, &Receive_SlaveConfig0);
/* Allocate memory */
masterDataSend = (uint8_t*)calloc(100, sizeof(uint8_t));
masterDataReceive = (uint8_t*)calloc(100, sizeof(uint8_t));
slaveDataSend = (uint8_t*)calloc(100, sizeof(uint8_t));
slaveDataReceive = (uint8_t*)calloc(100, sizeof(uint8_t));
bufferSize = 100U;
testStatus[0] = true;
LPSPI_DRV_SlaveTransfer(0U, slaveDataSend,
              slaveDataReceive, bufferSize);
LPSPI_DRV_MasterTransferBlocking(1U, &Send_MasterConfig0, masterDataSend,
              masterDataReceive, bufferSize, TIMEOUT);
```

Modules

- LPSPI Driver

  Low Power Serial Peripheral Interface Peripheral Driver.
- LPSPI HAL

  Low Power Serial Peripheral Interface Hardware Abstraction Layer.

3.60 Low Power Timer (LPTMR)

3.60.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Drivers for the Low Power Timer (LPTMR) module of S32 SDK devices.

The LPTMR is a configurable 16-bit counter that can operate in two functional modes:

- Timer mode with selectable prescaler and clock source (periodic or free-running).
- Pulse-Counter mode, with configurable glitch filter, that can count events (internal or external)

Modules

- LPTMR Driver

  Low Power Timer Peripheral Driver.
- LPTMR HAL

  Low Power Timer Hardware Abstraction Layer.

3.61 Low Power Universal Asynchronous Receiver-Transmitter (LPUART)

3.61.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Driver for the Low Power Universal Asynchronous Receiver-↩
Transmitter (LPUART) module of S32 SDK devices.
The LPUART module is used for serial communication, supporting LIN master and slave operation. These sections describe the S32 SDK software modules API that can be used for initializing and configuring the module, as well as initiating serial communications using the interrupt-based method.

Modules

- LPUART Driver

  This module covers the functionality of the Low Power Universal Asynchronous Receiver-Transmitter (LPUART) peripheral driver.
- LPUART HAL

  This module covers the functionality of the Low Power Universal Asynchronous Receiver-Transmitter (LPUART) hardware abstraction layer.

3.62 MPU Driver

3.62.1 Detailed Description

Memory Protection Unit Peripheral Driver.

Pre-Initialization information of MPU module

1. Before using the MPU driver the protocol clock of the module must be configured by the application using PCC module.
2. Bus fault or Hard fault exception must be configured to handle MPU access violation.

To initialize the MPU module, call the MPU_DRV_Init() function and provide the user configuration data structure. This function sets the configuration of the MPU module automatically and enables the MPU module.
Note that the configuration for region 0:

- Only access right for **CORE, DMA** can be **changed** (**DEBUG** is **ignored**).
- The **start address**, **end address**, **process identifier** and **process identifier mask** are **ignored**.

This is example code to configure the MPU driver:

```
/* Device instance number */
#define MEMPROTECT1 (0U)

/* Number of region in user configuration */
#define MPU_NUM_OF_REGION_CFG0 (3U)

/* Master access rights configuration 0 */
mpu_master_access_right_t AccessRightConfig0[] =
{
    /* CORE_0 */
    {
        .masterNum = FEATURE_MPU_MASTER_CORE,            /* Master number      */
        .accessRight = MPU_SUPERVISOR_USER_NONE,         /* Access right       */
        .processIdentifierEnable = false                 /* Disable identifier */
    },

    /* DEBUGGER_0 */
    {
        .masterNum = FEATURE_MPU_MASTER_DEBUGGER,        /* Master number      */
        .accessRight = MPU_SUPERVISOR_USER_RWX,          /* Access right       */
        .processIdentifierEnable = false                 /* Disable identifier */
    },

    /* DMA_0 */
    {
        .masterNum = FEATURE_MPU_MASTER_DMA,             /* Master number      */
        .accessRight = MPU_SUPERVISOR_USER_RW,           /* Access right       */
        .processIdentifierEnable = false                 /* Disable identifier */
    }
};

/* Master access rights configuration 1 */
mpu_master_access_right_t AccessRightConfig1[] =
{
    /* CORE_1 */
    {
        .masterNum = FEATURE_MPU_MASTER_CORE,            /* Master number      */
        .accessRight = MPU_SUPERVISOR_RWX_USER_NONE,     /* Access right
      */
        .processIdentifierEnable = false                 /* Disable identifier */
    },

    /* DEBUGGER_1 */
    {
        .masterNum = FEATURE_MPU_MASTER_DEBUGGER,        /* Master number      */
        .accessRight = MPU_SUPERVISOR_USER_RWX,          /* Access right       */
        .processIdentifierEnable = false                 /* Disable identifier */
    },

    /* DMA_1 */
    {
        .masterNum = FEATURE_MPU_MASTER_DMA,            /* Master number      */
        .accessRight = MPU_SUPERVISOR_USER_RW,          /* Access right       */
        .processIdentifierEnable = false                /* Disable identifier */
    }
};

/* Master access rights configuration 2 */
mpu_master_access_right_t AccessRightConfig2[] =
{
    /* CORE_2 */
    {
        .masterNum = FEATURE_MPU_MASTER_CORE,           /* Master number      */
        .accessRight = MPU_SUPERVISOR_USER_WX,          /* Access right       */
        .processIdentifierEnable = false                /* Disable identifier */
    },

    /* DEBUGGER_2 */
    {
        .masterNum = FEATURE_MPU_MASTER_DEBUGGER,       /* Master number      */
        .accessRight = MPU_SUPERVISOR_USER_RWX,         /* Access right       */
        .processIdentifierEnable = false                /* Disable identifier */
    },

    /* DMA_2 */
    {
        .masterNum = FEATURE_MPU_MASTER_DMA,            /* Master number      */
        .accessRight = MPU_SUPERVISOR_USER_RW,          /* Access right       */
        .processIdentifierEnable = false                /* Disable identifier */
    }
};

/* User configuration 0 */
const mpu_user_config_t memProtect1_UserConfig0[] =
{
    /* Region 0 */
    {
        .startAddr      = 0x00000000UL,          /* Memory region start address */
        .endAddr        = 0xFFFFFFFFUL,          /* Memory region end address   */
        .masterAccRight = AccessRightConfig0,    /* Master access right         */
        .processIdentifier  = 0U,                /* Process identifier          */
        .processIdMask  = 0U                     /* Process identifier mask     */
    },

    /* Region 1 */
    {
        .startAddr      = 0x00000000UL,          /* Memory region start address */
        .endAddr        = 0x00080000UL,          /* Memory region end address   */
        .masterAccRight = AccessRightConfig1,    /* Master access right         */
        .processIdentifier  = 0U,                /* Process identifier          */
        .processIdMask  = 0U                     /* Process identifier mask     */
    },

    /* Region 2 */
    {
        .startAddr      = 0x1FF00000UL,          /* Memory region start address */
        .endAddr        = 0x20000000UL,          /* Memory region end address   */
        .masterAccRight = AccessRightConfig2,    /* Master access right         */
        .processIdentifier  = 0U,                /* Process identifier          */
        .processIdMask  = 0U                     /* Process identifier mask     */
    }
};

/* Initializes the MPU module */
MPU_DRV_Init(MEMPROTECT1, MPU_NUM_OF_REGION_CFG0, memProtect1_UserConfig0);
```

After MPU initialization:
The MPU_DRV_Deinit() can be used to reset by default and disable MPU module.
The MPU_DRV_SetMasterAccessRights() can be used to change the access rights for special master ports and for special region numbers.
The MPU_DRV_SetRegionAddr() can be used to change the start/end address for a region.
The MPU_DRV_SetRegionConfig() can be used to set the whole region with the start/end address with access rights.
The MPU_DRV_GetDetailErrorAccessInfo() API is provided to get the error status of a special slave port.
When an error happens in this port, the MPU_DRV_GetDetailErrorAccessInfo() API is provided to get the detailed error information.

Data Structures

- struct mpu_master_access_right_t

  MPU master access rights. Implements : mpu_master_access_right_t_Class. More...
- struct mpu_user_config_t

  MPU user region configuration structure. This structure is used when calling the MPU_DRV_Init function. Implements : mpu_user_config_t_Class. More...

Macros

- #define MPU_USER_MASK (0x07U)
- #define MPU_USER_SHIFT (0U)
- #define MPU_SUPERVISOR_MASK (0x18U)
- #define MPU_SUPERVISOR_SHIFT (3U)
- #define MPU_W_MASK (0x20U)
- #define MPU_W_SHIFT (5U)
- #define MPU_R_MASK (0x40U)
- #define MPU_R_SHIFT (6U)

Enumerations

- enum mpu_access_rights_t {
  MPU_SUPERVISOR_RWX_USER_NONE = 0x00U, MPU_SUPERVISOR_RWX_USER_X = 0x01U, MPU_SUPERVISOR_RWX_USER_W = 0x02U, MPU_SUPERVISOR_RWX_USER_WX = 0x03U,
  MPU_SUPERVISOR_RWX_USER_R = 0x04U, MPU_SUPERVISOR_RWX_USER_RX = 0x05U, MPU_SUPERVISOR_RWX_USER_RW = 0x06U, MPU_SUPERVISOR_RWX_USER_RWX = 0x07U,
  MPU_SUPERVISOR_RX_USER_NONE = 0x08U, MPU_SUPERVISOR_RX_USER_X = 0x09U, MPU_SUPERVISOR_RX_USER_W = 0x0AU, MPU_SUPERVISOR_RX_USER_WX = 0x0BU,
  MPU_SUPERVISOR_RX_USER_R = 0x0CU, MPU_SUPERVISOR_RX_USER_RX = 0x0DU, MPU_SUPERVISOR_RX_USER_RW = 0x0EU, MPU_SUPERVISOR_RX_USER_RWX = 0x0FU,
  MPU_SUPERVISOR_RW_USER_NONE = 0x10U, MPU_SUPERVISOR_RW_USER_X = 0x11U, MPU_SUPERVISOR_RW_USER_W = 0x12U, MPU_SUPERVISOR_RW_USER_WX = 0x13U,
  MPU_SUPERVISOR_RW_USER_R = 0x14U, MPU_SUPERVISOR_RW_USER_RX = 0x15U, MPU_SUPERVISOR_RW_USER_RW = 0x16U, MPU_SUPERVISOR_RW_USER_RWX = 0x17U,
  MPU_SUPERVISOR_USER_NONE = 0x18U, MPU_SUPERVISOR_USER_X = 0x19U, MPU_SUPERVISOR_USER_W = 0x1AU, MPU_SUPERVISOR_USER_WX = 0x1BU,
  MPU_SUPERVISOR_USER_R = 0x1CU, MPU_SUPERVISOR_USER_RX = 0x1DU, MPU_SUPERVISOR_USER_RW = 0x1EU, MPU_SUPERVISOR_USER_RWX = 0x1FU,
  MPU_NONE = 0x80U, MPU_W = 0xA0U, MPU_R = 0xC0U, MPU_RW = 0xE0U }

  MPU access rights.

Code	Supervisor	User	Description
MPU_SUPERVISOR_RWX_USER_NONE	*r w x*	*- - -*	*Allow Read, write, execute in supervisor mode; no access in user mode*
MPU_SUPERVISOR_RWX_USER_X	*r w x*	*- - x*	*Allow Read, write, execute in supervisor mode; execute in user mode*
MPU_SUPERVISOR_RWX_USER_W	*r w x*	*- w -*	*Allow Read, write, execute in supervisor mode; write in user mode*
MPU_SUPERVISOR_RWX_USER_WX	*r w x*	*- w x*	*Allow Read, write, execute in supervisor mode; write and execute in user mode*
MPU_SUPERVISOR_RWX_USER_R	*r w x*	*r - -*	*Allow Read, write, execute in supervisor mode; read in user mode*
MPU_SUPERVISOR_RWX_USER_RX	*r w x*	*r - x*	*Allow Read, write, execute in supervisor mode; read and execute in user mode*

Code	Supervisor	User	Description
MPU_SUPERVISOR_RWX_USER_RW	*r w x*	*r w -*	*Allow Read, write, execute in supervisor mode; read and write in user mode*
MPU_SUPERVISOR_RWX_USER_RWX	*r w x*	*r w x*	*Allow Read, write, execute in supervisor mode; read, write and execute in user mode*
MPU_SUPERVISOR_RX_USER_NONE	*r - x*	*- - -*	*Allow Read, execute in supervisor mode; no access in user mode*
MPU_SUPERVISOR_RX_USER_X	*r - x*	*- - x*	*Allow Read, execute in supervisor mode; execute in user mode*
MPU_SUPERVISOR_RX_USER_W	*r - x*	*- w -*	*Allow Read, execute in supervisor mode; write in user mode*
MPU_SUPERVISOR_RX_USER_WX	*r - x*	*- w x*	*Allow Read, execute in supervisor mode; write and execute in user mode*
MPU_SUPERVISOR_RX_USER_R	*r - x*	*r - -*	*Allow Read, execute in supervisor mode; read in user mode*
MPU_SUPERVISOR_RX_USER_RX	*r - x*	*r - x*	*Allow Read, execute in supervisor mode; read and execute in user mode*
MPU_SUPERVISOR_RX_USER_RW	*r - x*	*r w -*	*Allow Read, execute in supervisor mode; read and write in user mode*
MPU_SUPERVISOR_RX_USER_RWX	*r - x*	*r w x*	*Allow Read, execute in supervisor mode; read, write and execute in user mode*
MPU_SUPERVISOR_RW_USER_NONE	*r w -*	*- - -*	*Allow Read, write in supervisor mode; no access in user mode*
MPU_SUPERVISOR_RW_USER_X	*r w -*	*- - x*	*Allow Read, write in supervisor mode; execute in user mode*
MPU_SUPERVISOR_RW_USER_W	*r w -*	*- w -*	*Allow Read, write in supervisor mode; write in user mode*
MPU_SUPERVISOR_RW_USER_WX	*r w -*	*- w x*	*Allow Read, write in supervisor mode; write and execute in user mode*
MPU_SUPERVISOR_RW_USER_R	*r w -*	*r - -*	*Allow Read, write in supervisor mode; read in user mode*
MPU_SUPERVISOR_RW_USER_RX	*r w -*	*r - x*	*Allow Read, write in supervisor mode; read and execute in user mode*
MPU_SUPERVISOR_RW_USER_RW	*r w -*	*r w -*	*Allow Read, write in supervisor mode; read and write in user mode*
MPU_SUPERVISOR_RW_USER_RWX	*r w -*	*r w x*	*Allow Read, write in supervisor mode; read, write and execute in user mode*
MPU_SUPERVISOR_USER_NONE	*- - -*	*- - -*	*No access allowed in user and supervisor modes*
MPU_SUPERVISOR_USER_X	*- - x*	*- - x*	*Execute operation is allowed in user and supervisor modes*
MPU_SUPERVISOR_USER_W	*- w -*	*- w -*	*Write operation is allowed in user and supervisor modes*
MPU_SUPERVISOR_USER_WX	*- w x*	*- w x*	*Write and execute operations are allowed in user and supervisor modes*
MPU_SUPERVISOR_USER_R	*r - -*	*r - -*	*Read operation is allowed in user and supervisor modes*
MPU_SUPERVISOR_USER_RX	*r - x*	*r - x*	*Read and execute operations are allowed in user and supervisor modes*
MPU_SUPERVISOR_USER_RW	*r w -*	*r w -*	*Read and write operations are allowed in user and supervisor modes*
MPU_SUPERVISOR_USER_RWX	*r w x*	*r w x*	*Read write and execute operations are allowed in user and supervisor modes*

Variables

- MPU_Type ∗const g_mpuBase [MPU_INSTANCE_COUNT]

  Table of base addresses for MPU instances.

MPU Driver API

- status_t MPU_DRV_Init (uint32_t instance, uint8_t regionCnt, const mpu_user_config_t ∗userConfigArr)

  The function sets the MPU regions according to user input and then enables the MPU. Please note that access rights for region 0 will always be configured and regionCnt takes values between 1 and the maximum region count supported by the hardware. e.g. In S32K144 the number of supported regions is 8. The user must make sure that the clock is enabled.
- void MPU_DRV_Deinit (uint32_t instance)

  De-initializes the MPU region by resetting and disabling MPU module.
- void MPU_DRV_SetRegionAddr (uint32_t instance, uint8_t regionNum, uint32_t startAddr, uint32_t endAddr)

  Sets the region start and end address.
- status_t MPU_DRV_SetRegionConfig (uint32_t instance, uint8_t regionNum, const mpu_user_config_↩ t ∗userConfigPtr)

  Sets the region configuration.
- status_t MPU_DRV_SetMasterAccessRights (uint32_t instance, uint8_t regionNum, const mpu_master_↩ access_right_t ∗accessRightsPtr)

  Configures access permission.
- void MPU_DRV_GetDetailErrorAccessInfo (uint32_t instance, uint8_t slavePortNum, mpu_access_err_info↩ _t ∗errInfoPtr)

  Gets the MPU access error detail information for a slave port.

3.62.2 Data Structure Documentation

3.62.2.1 struct mpu_master_access_right_t

MPU master access rights. Implements : mpu_master_access_right_t_Class.

Data Fields

- uint8_t masterNum
- mpu_access_rights_t accessRight

Field Documentation

3.62.2.1.1 accessRight

```
mpu_access_rights_t accessRight
```

Access right

3.62.2.1.2 masterNum

```
uint8_t masterNum
```

Master number

3.62.2.2 struct mpu_user_config_t

MPU user region configuration structure. This structure is used when calling the MPU_DRV_Init function. Implements : mpu_user_config_t_Class.

Data Fields

- uint32_t startAddr
- uint32_t endAddr
- const mpu_master_access_right_t ∗ masterAccRight

Field Documentation

3.62.2.2.1 endAddr

```
uint32_t endAddr
```

Memory region end address

3.62.2.2.2 masterAccRight

```
const mpu_master_access_right_t* masterAccRight
```

Access permission for masters

3.62.2.2.3 startAddr

```
uint32_t startAddr
```

Memory region start address

3.62.3 Macro Definition Documentation

3.62.3.1 MPU_R_MASK

```
#define MPU_R_MASK (0x40U)
```

3.62.3.2 MPU_R_SHIFT

```
#define MPU_R_SHIFT (6U)
```

3.62.3.3 MPU_SUPERVISOR_MASK

```
#define MPU_SUPERVISOR_MASK (0x18U)
```

3.62.3.4 MPU_SUPERVISOR_SHIFT

```
#define MPU_SUPERVISOR_SHIFT (3U)
```

3.62.3.5 MPU_USER_MASK

```
#define MPU_USER_MASK (0x07U)
```

3.62.3.6 MPU_USER_SHIFT

```
#define MPU_USER_SHIFT (0U)
```

3.62.3.7 MPU_W_MASK

```
#define MPU_W_MASK (0x20U)
```

3.62.3.8 MPU_W_SHIFT

```
#define MPU_W_SHIFT (5U)
```

3.62.4 Enumeration Type Documentation

3.62.4.1 mpu_access_rights_t

```
enum mpu_access_rights_t
```

MPU access rights.

Code	Supervisor	User	Description
MPU_SUPERVISOR_RWX_USER_NO↩ NE	r w x	- - -	Allow Read, write, execute in supervisor mode; no access in user mode
MPU_SUPERVISOR_RWX_USER_X	r w x	- - x	Allow Read, write, execute in supervisor mode; execute in user mode
MPU_SUPERVISOR_RWX_USER_W	r w x	- w -	Allow Read, write, execute in supervisor mode; write in user mode
MPU_SUPERVISOR_RWX_USER_WX	r w x	- w x	Allow Read, write, execute in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_RWX_USER_R	r w x	r - -	Allow Read, write, execute in supervisor mode; read in user mode
MPU_SUPERVISOR_RWX_USER_RX	r w x	r - x	Allow Read, write, execute in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_RWX_USER_RW	r w x	r w -	Allow Read, write, execute in supervisor mode; read and write in user mode
MPU_SUPERVISOR_RWX_USER_RWX	r w x	r w x	Allow Read, write, execute in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_RX_USER_NONE	r - x	- - -	Allow Read, execute in supervisor mode; no access in user mode
MPU_SUPERVISOR_RX_USER_X	r - x	- - x	Allow Read, execute in supervisor mode; execute in user mode
MPU_SUPERVISOR_RX_USER_W	r - x	- w -	Allow Read, execute in supervisor mode; write in user mode
MPU_SUPERVISOR_RX_USER_WX	r - x	- w x	Allow Read, execute in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_RX_USER_R	r - x	r - -	Allow Read, execute in supervisor mode; read in user mode
MPU_SUPERVISOR_RX_USER_RX	r - x	r - x	Allow Read, execute in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_RX_USER_RW	r - x	r w -	Allow Read, execute in supervisor mode; read and write in user mode

Code	Supervisor	User	Description
MPU_SUPERVISOR_RX_USER_RWX	r - x	r w x	Allow Read, execute in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_RW_USER_NONE	r w -	- - -	Allow Read, write in supervisor mode; no access in user mode
MPU_SUPERVISOR_RW_USER_X	r w -	- - x	Allow Read, write in supervisor mode; execute in user mode
MPU_SUPERVISOR_RW_USER_W	r w -	- w -	Allow Read, write in supervisor mode; write in user mode
MPU_SUPERVISOR_RW_USER_WX	r w -	- w x	Allow Read, write in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_RW_USER_R	r w -	r - -	Allow Read, write in supervisor mode; read in user mode
MPU_SUPERVISOR_RW_USER_RX	r w -	r - x	Allow Read, write in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_RW_USER_RW	r w -	r w -	Allow Read, write in supervisor mode; read and write in user mode
MPU_SUPERVISOR_RW_USER_RWX	r w -	r w x	Allow Read, write in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_USER_NONE	- - -	- - -	No access allowed in user and supervisor modes
MPU_SUPERVISOR_USER_X	- - x	- - x	Execute operation is allowed in user and supervisor modes
MPU_SUPERVISOR_USER_W	- w -	- w -	Write operation is allowed in user and supervisor modes
MPU_SUPERVISOR_USER_WX	- w x	- w x	Write and execute operations are allowed in user and supervisor modes
MPU_SUPERVISOR_USER_R	r - -	r - -	Read operation is allowed in user and supervisor modes
MPU_SUPERVISOR_USER_RX	r - x	r - x	Read and execute operations are allowed in user and supervisor modes
MPU_SUPERVISOR_USER_RW	r w -	r w -	Read and write operations are allowed in user and supervisor modes
MPU_SUPERVISOR_USER_RWX	r w x	r w x	Read write and execute operations are allowed in user and supervisor modes

Code	Read/Write permission	Description
MPU_NONE	- -	No Read/Write access permission
MPU_W	- w	Write access permission
MPU_R	r -	Read access permission
MPU_RW	r w	Read/Write access permission

Implements : mpu_access_rights_t_Class

Enumerator

MPU_SUPERVISOR_RWX_USER_NONE	0b00000000U : rwx|—
MPU_SUPERVISOR_RWX_USER_X	0b00000001U : rwx|–x
MPU_SUPERVISOR_RWX_USER_W	0b00000010U : rwx|-w-
MPU_SUPERVISOR_RWX_USER_WX	0b00000011U : rwx|-wx
MPU_SUPERVISOR_RWX_USER_R	0b00000100U : rwx|r–
MPU_SUPERVISOR_RWX_USER_RX	0b00000101U : rwx|r-x

Enumerator

MPU_SUPERVISOR_RWX_USER_RW	0b00000110U : rwx|rw-
MPU_SUPERVISOR_RWX_USER_RWX	0b00000111U : rwx|rwx
MPU_SUPERVISOR_RX_USER_NONE	0b00001000U : r-x|—
MPU_SUPERVISOR_RX_USER_X	0b00001001U : r-x|–x
MPU_SUPERVISOR_RX_USER_W	0b00001010U : r-x|-w-
MPU_SUPERVISOR_RX_USER_WX	0b00001011U : r-x|-wx
MPU_SUPERVISOR_RX_USER_R	0b00001100U : r-x|r–
MPU_SUPERVISOR_RX_USER_RX	0b00001101U : r-x|r-x
MPU_SUPERVISOR_RX_USER_RW	0b00001110U : r-x|rw-
MPU_SUPERVISOR_RX_USER_RWX	0b00001111U : r-x|rwx
MPU_SUPERVISOR_RW_USER_NONE	0b00010000U : rw-|—
MPU_SUPERVISOR_RW_USER_X	0b00010001U : rw-|–x
MPU_SUPERVISOR_RW_USER_W	0b00010010U : rw-|-w-
MPU_SUPERVISOR_RW_USER_WX	0b00010011U : rw-|-wx
MPU_SUPERVISOR_RW_USER_R	0b00010100U : rw-|r–
MPU_SUPERVISOR_RW_USER_RX	0b00010101U : rw-|r-x
MPU_SUPERVISOR_RW_USER_RW	0b00010110U : rw-|rw-
MPU_SUPERVISOR_RW_USER_RWX	0b00010111U : rw-|rwx
MPU_SUPERVISOR_USER_NONE	0b00011000U : —|—
MPU_SUPERVISOR_USER_X	0b00011001U : –x|–x
MPU_SUPERVISOR_USER_W	0b00011010U : -w-|-w-
MPU_SUPERVISOR_USER_WX	0b00011011U : -wx|-wx
MPU_SUPERVISOR_USER_R	0b00011100U : r–|r–
MPU_SUPERVISOR_USER_RX	0b00011101U : r-x|r-x
MPU_SUPERVISOR_USER_RW	0b00011110U : rw-|rw-
MPU_SUPERVISOR_USER_RWX	0b00011111U : rwx|rwx
MPU_NONE	0b10000000U : –
MPU_W	0b10100000U : w-
MPU_R	0b11000000U : -r
MPU_RW	0b11100000U : wr

3.62.5 Function Documentation

3.62.5.1 MPU_DRV_Deinit()

```
void MPU_DRV_Deinit (
            uint32_t instance )
```

De-initializes the MPU region by resetting and disabling MPU module.

Parameters

in	*instance*	The MPU peripheral instance number.

3.62.5.2 MPU_DRV_GetDetailErrorAccessInfo()

```
void MPU_DRV_GetDetailErrorAccessInfo (
            uint32_t instance,
            uint8_t slavePortNum,
            mpu_access_err_info_t * errInfoPtr )
```

Gets the MPU access error detail information for a slave port.

Parameters

in	*instance*	The MPU peripheral instance number.
in	*slavePortNum*	The slave port number to get Error Detail.
out	*errInfoPtr*	A pointer to access error info structure.

3.62.5.3 MPU_DRV_Init()

```
status_t MPU_DRV_Init (
            uint32_t instance,
            uint8_t regionCnt,
            const mpu_user_config_t * userConfigArr )
```

The function sets the MPU regions according to user input and then enables the MPU. Please note that access rights for region 0 will always be configured and regionCnt takes values between 1 and the maximum region count supported by the hardware. e.g. In S32K144 the number of supported regions is 8. The user must make sure that the clock is enabled.

Parameters

in	*instance*	The MPU peripheral instance number.
in	*regionCnt*	Number of configuration regions.
in	*userConfigArr*	The pointer to the array of MPU user configure structure, see mpu_user_config_t.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failed due to master number is out of range supported by hardware.

3.62.5.4 MPU_DRV_SetMasterAccessRights()

```
status_t MPU_DRV_SetMasterAccessRights (
            uint32_t instance,
            uint8_t regionNum,
            const mpu_master_access_right_t * accessRightsPtr )
```

Configures access permission.

Parameters

in	*instance*	The MPU peripheral instance number.
in	*regionNum*	The MPU region number.
in	*accessRightsPtr*	A pointer to access permission structure.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failed due to master number is out of range supported by hardware.

3.62.5.5 MPU_DRV_SetRegionAddr()

```
void MPU_DRV_SetRegionAddr (
             uint32_t instance,
             uint8_t regionNum,
             uint32_t startAddr,
             uint32_t endAddr )
```

Sets the region start and end address.

Parameters

in	*instance*	The MPU peripheral instance number.
in	*regionNum*	The region number.
in	*startAddr*	Region start address.
in	*endAddr*	Region end address.

3.62.5.6 MPU_DRV_SetRegionConfig()

```
status_t MPU_DRV_SetRegionConfig (
             uint32_t instance,
             uint8_t regionNum,
             const mpu_user_config_t * userConfigPtr )
```

Sets the region configuration.

Parameters

in	*instance*	The MPU peripheral instance number.
in	*regionNum*	The region number.
in	*userConfigPtr*	Region configuration structure pointer.

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failed due to master number is out of range supported by hardware.

3.62.6 Variable Documentation

3.62.6.1 g_mpuBase

```
MPU_Type* const g_mpuBase[MPU_INSTANCE_COUNT]
```

Table of base addresses for MPU instances.

3.63 MPU HAL

3.63.1 Detailed Description

Memory Protection Unit Hardware Abstraction Level.

This HAL provides low-level access to all hardware features of the MPU.

Data Structures

- struct mpu_access_err_info_t

  MPU detail error access info Implements : mpu_access_err_info_t_Class. More...
- struct mpu_low_masters_access_rights_t

  MPU access rights for masters which have separated privilege rights for user and supervisor mode accesses (e.g. master0∼2 in S32K144) Implements : mpu_low_masters_access_rights_t_Class. More...
- struct mpu_high_masters_access_rights_t

  MPU access rights for master which have only read and write permissions Implements : mpu_high_masters_↩ access_rights_t_Class. More...

Enumerations

- enum mpu_err_access_type_t { MPU_ERR_TYPE_READ = 0U, MPU_ERR_TYPE_WRITE = 1U }

  MPU access error Implements : mpu_err_access_type_t_Class.
- enum mpu_err_attributes_t { MPU_INSTRUCTION_ACCESS_IN_USER_MODE = 0U, MPU_DATA_ACC↩ ESS_IN_USER_MODE = 1U, MPU_INSTRUCTION_ACCESS_IN_SUPERVISOR_MODE = 2U, MPU_DA↩ TA_ACCESS_IN_SUPERVISOR_MODE = 3U }

  MPU access error attributes Implements : mpu_err_attributes_t_Class.
- enum mpu_supervisor_access_rights_t { MPU_SUPERVISOR_READ_WRITE_EXECUTE = 0U, MPU_S↩ UPERVISOR_READ_EXECUTE = 1U, MPU_SUPERVISOR_READ_WRITE = 2U, MPU_SUPERVISOR_↩ EQUAL_TO_USERMODE = 3U }

  MPU access rights in supervisor mode Implements : mpu_supervisor_access_rights_t_Class.
- enum mpu_user_access_rights_t {
  MPU_USER_NO_ACCESS_RIGHTS = 0U, MPU_USER_EXECUTE = 1U, MPU_USER_WRITE = 2U, M↩ PU_USER_WRITE_EXECUTE = 3U,
  MPU_USER_READ = 4U, MPU_USER_READ_EXECUTE = 5U, MPU_USER_READ_WRITE = 6U, MPU↩ _USER_READ_WRITE_EXECUTE = 7U }

  MPU access rights in user mode Implements : mpu_user_access_rights_t_Class.

Functions

- static void MPU_HAL_Enable (MPU_Type ∗const base)

  Enables the MPU module.
- static void MPU_HAL_Disable (MPU_Type ∗const base)

  Disables the MPU module.
- static bool MPU_HAL_IsEnable (const MPU_Type ∗const base)

  Checks whether the MPU module is enabled.
- static uint8_t MPU_HAL_GetHardwareRevision (const MPU_Type ∗const base)

  Gets MPU hardware revision level.
- bool MPU_HAL_GetSlavePortErrorStatus (const MPU_Type ∗const base, uint8_t slaveNum)

  Gets the error status of a specified slave port.

- void MPU_HAL_GetDetailErrorAccessInfo (MPU_Type ∗const base, uint8_t slaveNum, mpu_access_err_↩ info_t ∗errInfoPtr)

  Gets MPU detail error access info.
- void MPU_HAL_SetRegionAddr (MPU_Type ∗const base, uint8_t regionNum, uint32_t startAddr, uint32_t endAddr)

  Sets region start and end address. Please note that using this function will clear the valid bit of the region, and a further validation might be needed.
- void MPU_HAL_SetLowMasterAccessRights (MPU_Type ∗const base, uint8_t regionNum, uint8_t master↩ Num, const mpu_low_masters_access_rights_t ∗accessRightsPtr)

  Sets access permission for master which has separated privilege rights for user and supervisor mode accesses in a specific region. Please note that using this function will clear the valid bit of the region. In order to keep the region valid, the MPU_HAL_SetAlternateLowMasterAccessRights function can be used.
- void MPU_HAL_SetHighMasterAccessRights (MPU_Type ∗const base, uint8_t regionNum, uint8_t master↩ Num, const mpu_high_masters_access_rights_t ∗accessRightsPtr)

  Sets access permission for master which has only read and write permissions in a specific region. Please note that using this function will clear the valid bit of the region. In order to keep the region valid, the MPU_HAL_SetAlternate↩ HighMasterAccessRights function can be used.
- static void MPU_HAL_SetRegionValidCmd (MPU_Type ∗const base, uint8_t regionNum, bool enable)

  Sets the region valid value. When a region changed not by alternating registers should set the valid again.
- static void MPU_HAL_SetProcessIdentifierMask (MPU_Type ∗const base, uint8_t regionNum, uint8_↩ t processIdentifierMask)

  Sets the process identifier mask.
- static void MPU_HAL_SetProcessIdentifier (MPU_Type ∗const base, uint8_t regionNum, uint8_t process↩ Identifier)

  Sets the process identifier.
- void MPU_HAL_SetAlternateLowMasterAccessRights (MPU_Type ∗const base, uint8_t regionNum, uint8_t masterNum, const mpu_low_masters_access_rights_t ∗accessRightsPtr)

  Sets access permission for master which has separated privilege rights for user and supervisor mode accesses in a specific region by alternate register.
- void MPU_HAL_SetAlternateHighMasterAccessRights (MPU_Type ∗const base, uint8_t regionNum, uint8_t masterNum, const mpu_high_masters_access_rights_t ∗accessRightsPtr)

  Sets access permission for master which has only read and write permissions in a specific region by alternate register.
- void MPU_HAL_Init (MPU_Type ∗const base)

  Initializes the MPU module and all regions will be invalid after cleared access permission.

3.63.2 Data Structure Documentation

3.63.2.1 struct mpu_access_err_info_t

MPU detail error access info Implements : mpu_access_err_info_t_Class.

Data Fields

- uint8_t master
- mpu_err_attributes_t attributes
- mpu_err_access_type_t accessType
- uint16_t accessCtr
- uint32_t addr
- uint8_t processorIdentification

Field Documentation

3.63.2.1.1 accessCtr

```
uint16_t accessCtr
```

Access error control

3.63.2.1.2 accessType

```
mpu_err_access_type_t accessType
```

Access error type

3.63.2.1.3 addr

```
uint32_t addr
```

Access error address

3.63.2.1.4 attributes

```
mpu_err_attributes_t attributes
```

Access error attributes

3.63.2.1.5 master

```
uint8_t master
```

Access error master

3.63.2.1.6 processorIdentification

```
uint8_t processorIdentification
```

Access error processor identification

3.63.2.2 struct mpu_low_masters_access_rights_t

MPU access rights for masters which have separated privilege rights for user and supervisor mode accesses (e.g. master0∼2 in S32K144) Implements : mpu_low_masters_access_rights_t_Class.

Data Fields

- mpu_user_access_rights_t userAccessRights
- mpu_supervisor_access_rights_t superAccessRights
- bool processIdentifierEnable

Field Documentation

3.63.2.2.1 processIdentifierEnable

```
bool processIdentifierEnable
```

Enables or disables process identifier

3.63.2.2.2 superAccessRights

```
mpu_supervisor_access_rights_t superAccessRights
```

Master access rights in supervisor mode

3.63.2.2.3 userAccessRights

```
mpu_user_access_rights_t userAccessRights
```

Master access rights in user mode

3.63.2.3 struct mpu_high_masters_access_rights_t

MPU access rights for master which have only read and write permissions Implements : mpu_high_masters_↩ access_rights_t_Class.

Data Fields

- bool writeEnable
- bool readEnable

Field Documentation

3.63.2.3.1 readEnable

```
bool readEnable
```

Enables or disables read permission

3.63.2.3.2 writeEnable

```
bool writeEnable
```

Enables or disables write permission

3.63.3 Enumeration Type Documentation

3.63.3.1 mpu_err_access_type_t

```
enum mpu_err_access_type_t
```

MPU access error Implements : mpu_err_access_type_t_Class.

Enumerator

MPU_ERR_TYPE_READ	MPU error type: read
MPU_ERR_TYPE_WRITE	MPU error type: write

3.63.3.2 mpu_err_attributes_t

`enum mpu_err_attributes_t`

MPU access error attributes Implements : mpu_err_attributes_t_Class.

Enumerator

MPU_INSTRUCTION_ACCESS_IN_USER_MODE	Access instruction error in user mode
MPU_DATA_ACCESS_IN_USER_MODE	Access data error in user mode
MPU_INSTRUCTION_ACCESS_IN_SUPERVISOR_MODE	Access instruction error in supervisor mode
MPU_DATA_ACCESS_IN_SUPERVISOR_MODE	Access data error in supervisor mode

3.63.3.3 mpu_supervisor_access_rights_t

`enum mpu_supervisor_access_rights_t`

MPU access rights in supervisor mode Implements : mpu_supervisor_access_rights_t_Class.

Enumerator

MPU_SUPERVISOR_READ_WRITE_EXECUTE	Read write and execute operations are allowed in supervisor mode
MPU_SUPERVISOR_READ_EXECUTE	Read and execute operations are allowed in supervisor mode
MPU_SUPERVISOR_READ_WRITE	Read write operations are allowed in supervisor mode
MPU_SUPERVISOR_EQUAL_TO_USERMODE	Access permission equal to user mode

3.63.3.4 mpu_user_access_rights_t

`enum mpu_user_access_rights_t`

MPU access rights in user mode Implements : mpu_user_access_rights_t_Class.

Enumerator

MPU_USER_NO_ACCESS_RIGHTS	No access allowed in user mode
MPU_USER_EXECUTE	Execute operation is allowed in user mode
MPU_USER_WRITE	Write operation is allowed in user mode
MPU_USER_WRITE_EXECUTE	Write and execute operations are allowed in user mode
MPU_USER_READ	Read is allowed in user mode
MPU_USER_READ_EXECUTE	Read and execute operations are allowed in user mode
MPU_USER_READ_WRITE	Read and write operations are allowed in user mode
MPU_USER_READ_WRITE_EXECUTE	Read write and execute operations are allowed in user mode

3.63.4 Function Documentation

3.63.4.1 MPU_HAL_Disable()

```
static void MPU_HAL_Disable (
            MPU_Type *const base )  [inline], [static]
```

Disables the MPU module.

Parameters

in	*base*	The MPU peripheral base address. Implements : MPU_HAL_Disable_Activity

3.63.4.2 MPU_HAL_Enable()

```
static void MPU_HAL_Enable (
            MPU_Type *const base )  [inline], [static]
```

Enables the MPU module.

Parameters

in	*base*	The MPU peripheral base address. Implements : MPU_HAL_Enable_Activity

3.63.4.3 MPU_HAL_GetDetailErrorAccessInfo()

```
void MPU_HAL_GetDetailErrorAccessInfo (
            MPU_Type *const base,
            uint8_t slaveNum,
            mpu_access_err_info_t * errInfoPtr )
```

Gets MPU detail error access info.

Parameters

in	*base*	The MPU peripheral base address.
in	*slaveNum*	MPU slave port number.
out	*errInfoPtr*	The pointer to the MPU access error information.

3.63.4.4 MPU_HAL_GetHardwareRevision()

```
static uint8_t MPU_HAL_GetHardwareRevision (
            const MPU_Type *const base )  [inline], [static]
```

Gets MPU hardware revision level.

Parameters

in	*base*	The MPU peripheral base address.

Returns

Hardware revision level. Implements : MPU_HAL_GetHardwareRevision_Activity

3.63.4.5 MPU_HAL_GetSlavePortErrorStatus()

```
bool MPU_HAL_GetSlavePortErrorStatus (
            const MPU_Type *const base,
            uint8_t slaveNum )
```

Gets the error status of a specified slave port.

Parameters

in	*base*	The MPU peripheral base address.
in	*slaveNum*	MPU slave port number.

Returns

The slave ports error status:

- true: error happens in this slave port.
- false: error didn't happen in this slave port.

3.63.4.6 MPU_HAL_Init()

```
void MPU_HAL_Init (
            MPU_Type *const base )
```

Initializes the MPU module and all regions will be invalid after cleared access permission.

Parameters

in	*base*	The MPU peripheral base address.

3.63.4.7 MPU_HAL_IsEnable()

```
static bool MPU_HAL_IsEnable (
            const MPU_Type *const base ) [inline], [static]
```

Checks whether the MPU module is enabled.

Parameters

in	*base*	The MPU peripheral base address.

Returns

State of the module

- true: MPU module is enabled.
- false: MPU module is disabled. Implements : MPU_HAL_IsEnable_Activity

3.63.4.8 MPU_HAL_SetAlternateHighMasterAccessRights()

```
void MPU_HAL_SetAlternateHighMasterAccessRights (
            MPU_Type *const base,
            uint8_t regionNum,
            uint8_t masterNum,
            const mpu_high_masters_access_rights_t * accessRightsPtr )
```

Sets access permission for master which has only read and write permissions in a specific region by alternate register.

Parameters

in	*base*	The MPU peripheral base address.
in	*regionNum*	MPU region number.
in	*masterNum*	MPU master number.
in	*accessRightsPtr*	The pointer of master access rights see mpu_high_masters_access_rights_t.

3.63.4.9 MPU_HAL_SetAlternateLowMasterAccessRights()

```
void MPU_HAL_SetAlternateLowMasterAccessRights (
            MPU_Type *const base,
            uint8_t regionNum,
            uint8_t masterNum,
            const mpu_low_masters_access_rights_t * accessRightsPtr )
```

Sets access permission for master which has separated privilege rights for user and supervisor mode accesses in a specific region by alternate register.

Parameters

in	*base*	The MPU peripheral base address.
in	*regionNum*	MPU region number.
in	*masterNum*	MPU master number.
in	*accessRightsPtr*	The pointer of master access rights see mpu_low_masters_access_rights_t.

3.63.4.10 MPU_HAL_SetHighMasterAccessRights()

```
void MPU_HAL_SetHighMasterAccessRights (
            MPU_Type *const base,
            uint8_t regionNum,
            uint8_t masterNum,
            const mpu_high_masters_access_rights_t * accessRightsPtr )
```

Sets access permission for master which has only read and write permissions in a specific region. Please note that using this function will clear the valid bit of the region. In order to keep the region valid, the MPU_HAL_Set←AlternateHighMasterAccessRights function can be used.

Parameters

in	*base*	The MPU peripheral base address.
in	*regionNum*	MPU region number.
in	*masterNum*	MPU master number.
in	*accessRightsPtr*	The pointer of master access rights see mpu_high_masters_access_rights_t.

3.63.4.11 MPU_HAL_SetLowMasterAccessRights()

```
void MPU_HAL_SetLowMasterAccessRights (
            MPU_Type *const base,
            uint8_t regionNum,
            uint8_t masterNum,
            const mpu_low_masters_access_rights_t * accessRightsPtr )
```

Sets access permission for master which has separated privilege rights for user and supervisor mode accesses in a specific region. Please note that using this function will clear the valid bit of the region. In order to keep the region valid, the MPU_HAL_SetAlternateLowMasterAccessRights function can be used.

Parameters

in	*base*	The MPU peripheral base address.
in	*regionNum*	MPU region number.
in	*masterNum*	MPU master number.
in	*accessRightsPtr*	The pointer of master access rights see mpu_low_masters_access_rights_t.

3.63.4.12 MPU_HAL_SetProcessIdentifier()

```
static void MPU_HAL_SetProcessIdentifier (
            MPU_Type *const base,
            uint8_t regionNum,
            uint8_t processIdentifier )  [inline], [static]
```

Sets the process identifier.

Parameters

in	*base*	The MPU peripheral base address.
in	*regionNum*	MPU region number.
in	*processIdentifier*	Process identifier value. Implements : MPU_HAL_SetProcessIdentifier_Activity

3.63.4.13 MPU_HAL_SetProcessIdentifierMask()

```
static void MPU_HAL_SetProcessIdentifierMask (
            MPU_Type *const base,
            uint8_t regionNum,
            uint8_t processIdentifierMask )  [inline], [static]
```

Sets the process identifier mask.

Parameters

in	*base*	The MPU peripheral base address.
in	*regionNum*	MPU region number.
in	*processIdentifierMask*	Process identifier mask value. Implements : MPU_HAL_SetProcessIdentifierMask_Activity

3.63.4.14 MPU_HAL_SetRegionAddr()

```
void MPU_HAL_SetRegionAddr (
            MPU_Type *const base,
            uint8_t regionNum,
            uint32_t startAddr,
            uint32_t endAddr )
```

Sets region start and end address. Please note that using this function will clear the valid bit of the region, and a further validation might be needed.

Parameters

in	*base*	The MPU peripheral base address.
in	*regionNum*	MPU region number.
in	*startAddr*	Region start address.
in	*endAddr*	Region end address.

3.63.4.15 MPU_HAL_SetRegionValidCmd()

```
static void MPU_HAL_SetRegionValidCmd (
            MPU_Type *const base,
            uint8_t regionNum,
            bool enable )  [inline], [static]
```

Sets the region valid value. When a region changed not by alternating registers should set the valid again.

Parameters

in	*base*	The MPU peripheral base address.
in	*regionNum*	MPU region number.
in	*enable*	Enables or disables region. Implements : MPU_HAL_SetRegionValidCmd_Activity

3.64 Memory Protection Unit (MPU)

3.64.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Driver for the Memory Protection Unit (MPU) module of S32 SDK devices.

The memory protection unit (MPU) provides hardware access control for all memory references generated in the device.
The MPU concurrently monitors all system bus transactions and evaluates their appropriateness using pre-programmed region descriptors that define memory spaces and their access rights. Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

Hardware background

The MPU concurrently monitors all system bus transactions and evaluates their appropriateness using pre-programmed region descriptors that define memory spaces and their access rights. Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The MPU implements a two-dimensional hardware array of memory region descriptors and the crossbar slave ports to continuously monitor the legality of every memory reference generated by each bus master in the system.

The feature set includes:

- 8 program-visible 128-bit region descriptors, accessible by four 32-bit words each
  - Each region descriptor defines a modulo-32 byte space, aligned anywhere in memory
    - Region sizes can vary from 32 bytes to 4 Gbytes
  - Two access control permissions defined in a single descriptor word
    - Masters 0–3: read, write, and execute attributes for supervisor and user accesses
    - Masters 4–7: read and write attributes
  - Hardware-assisted maintenance of the descriptor valid bit minimizes coherency issues
  - Alternate programming model view of the access control permissions word
  - Priority given to granting permission over denying access for overlapping region descriptors
- Detects access protection errors if a memory reference does not hit in any memory region, or if the reference is illegal in all hit memory regions. If an access error occurs, the reference is terminated with an error response, and the MPU inhibits the bus cycle being sent to the targeted slave device
- Error registers, per slave port, capture the last faulting address, attributes, and other information
- Global MPU enable/disable control bit

Modules

- MPU Driver

  Memory Protection Unit Peripheral Driver.
- MPU HAL

  Memory Protection Unit Hardware Abstraction Level.

3.65 OS Interface (OSIF)

3.65.1 Detailed Description

OS Interface Layer (OSIF)

This module is for SDK internal use only. It provides an abstract OS interface to SDK drivers (even if no OS is present) for basic OS operations (mutex and semaphore handling and time delay service).

FreeRTOS

A compile-time symbol, USING_OS_FREERTOS, needs to be defined.

FreeRTOSConfig.h necessities

FreeRTOS configuration file needs to have these options activated:

- INCLUDE_xQueueGetMutexHolder
- INCLUDE_xTaskGetCurrentTaskHandle

Bare-metal

If no OS is present, the corresponding bare-metal version of OSIF needs to be linked.

Mutex operations are dummy operations (always return success) and semaphore is implemented as a simple counter.

Time delay is implemented using the core Systick timer.

Constraints

To correctly measure time, a hardware timer is required. The Systick is employed for this purpose. As a consequence, the Systick timer is a blocked resource to the user application.

Macros

- #define OSIF_WAIT_FOREVER 0xFFFFFFFFu

Functions

- void OSIF_TimeDelay (const uint32_t delay)

    Delays execution for a number of milliseconds.
- uint32_t OSIF_GetMilliseconds (void)

    Returns the number of miliseconds elapsed since starting the internal timer or starting the scheduler.
- status_t OSIF_MutexLock (const mutex_t ∗const pMutex, const uint32_t timeout)

    Waits for a mutex and locks it.
- status_t OSIF_MutexUnlock (const mutex_t ∗const pMutex)

    Unlocks a previously locked mutex.
- status_t OSIF_MutexCreate (mutex_t ∗const pMutex)

    Create an unlocked mutex.
- status_t OSIF_MutexDestroy (const mutex_t ∗const pMutex)

    Destroys a previously created mutex.
- status_t OSIF_SemaWait (semaphore_t ∗const pSem, const uint32_t timeout)

    Decrement a semaphore with timeout.
- status_t OSIF_SemaPost (semaphore_t ∗const pSem)

    Increment a semaphore.
- status_t OSIF_SemaCreate (semaphore_t ∗const pSem, const uint8_t initValue)

    Creates a semaphore with a given value.
- status_t OSIF_SemaDestroy (const semaphore_t ∗const pSem)

    Destroys a previously created semaphore.

3.65.2 Macro Definition Documentation

3.65.2.1 OSIF_WAIT_FOREVER

```
#define OSIF_WAIT_FOREVER 0xFFFFFFFFu
```

3.65.3 Function Documentation

3.65.3.1 OSIF_GetMilliseconds()

```
uint32_t OSIF_GetMilliseconds (
             void  )
```

Returns the number of miliseconds elapsed since starting the internal timer or starting the scheduler.

Returns

the number of miliseconds elapsed

3.65.3.2 OSIF_MutexCreate()

```
status_t OSIF_MutexCreate (
             mutex_t ∗const pMutex )
```

Create an unlocked mutex.

Parameters

in	*pMutex*	reference to the mutex object

Returns

One of the possible status codes:

- STATUS_SUCCESS: mutex created
- STATUS_ERROR: mutex could not be created

3.65.3.3 OSIF_MutexDestroy()

```
status_t OSIF_MutexDestroy (
            const mutex_t *const pMutex )
```

Destroys a previously created mutex.

Parameters

in	*pMutex*	reference to the mutex object

Returns

One of the possible status codes:

- STATUS_SUCCESS: mutex destroyed

3.65.3.4 OSIF_MutexLock()

```
status_t OSIF_MutexLock (
            const mutex_t *const pMutex,
            const uint32_t timeout )
```

Waits for a mutex and locks it.

Parameters

in	*pMutex*	reference to the mutex object
in	*timeout*	time-out value in milliseconds

Returns

One of the possible status codes:

- STATUS_SUCCESS: mutex lock operation success
- STATUS_ERROR: mutex already owned by current thread
- STATUS_TIMEOUT: mutex lock operation timed out

3.65.3.5 OSIF_MutexUnlock()

```
status_t OSIF_MutexUnlock (
             const mutex_t *const pMutex )
```

Unlocks a previously locked mutex.

Parameters

in	*pMutex*	reference to the mutex object

Returns

One of the possible status codes:

- STATUS_SUCCESS: mutex unlock operation success
- STATUS_ERROR: mutex unlock failed

3.65.3.6 OSIF_SemaCreate()

```
status_t OSIF_SemaCreate (
             semaphore_t *const pSem,
             const uint8_t initValue )
```

Creates a semaphore with a given value.

Parameters

in	*pSem*	reference to the semaphore object
in	*initValue*	initial value of the semaphore

Returns

One of the possible status codes:

- STATUS_SUCCESS: semaphore created
- STATUS_ERROR: semaphore could not be created

3.65.3.7 OSIF_SemaDestroy()

```
status_t OSIF_SemaDestroy (
             const semaphore_t *const pSem )
```

Destroys a previously created semaphore.

Parameters

in	*pSem*	reference to the semaphore object

Returns

One of the possible status codes:

- STATUS_SUCCESS: semaphore destroyed

3.65.3.8 OSIF_SemaPost()

```
status_t OSIF_SemaPost (
             semaphore_t *const pSem )
```

Increment a semaphore.

Parameters

in	*pSem*	reference to the semaphore object

Returns

One of the possible status codes:

- STATUS_SUCCESS: semaphore post operation success
- STATUS_ERROR: semaphore could not be incremented

3.65.3.9 OSIF_SemaWait()

```
status_t OSIF_SemaWait (
             semaphore_t *const pSem,
             const uint32_t timeout )
```

Decrement a semaphore with timeout.

Parameters

in	*pSem*	reference to the semaphore object
in	*timeout*	time-out value in milliseconds

Returns

One of the possible status codes:

- STATUS_SUCCESS: semaphore wait operation success
- STATUS_TIMEOUT: semaphore wait timed out

3.65.3.10 OSIF_TimeDelay()

```
void OSIF_TimeDelay (
             const uint32_t delay )
```

Delays execution for a number of milliseconds.

Parameters

`in`	*delay*	Time delay in milliseconds.

3.66 PDB Driver

3.66.1 Detailed Description

Programmable Delay Block Peripheral Driver.

Overview

This section describes the programming interface of the PDB Peripheral driver. The PDB peripheral driver configures the PDB (Programmable Delay Block). It handles the triggers for ADC and pulse out to the CMP and the PDB counter.

PDB Driver model building

There is one main PDB counter for all triggers. When the indicated external trigger input arrives, the PDB counter launches and is increased by setting clock. The counter trigger milestones for ADC and the PDB counter and wait for the PDB counter. Once the PDB counter hits each milestone, also called the critical delay value, the corresponding event is triggered and the trigger signal is sent out to trigger other peripherals. Therefore, the PDB module is a collector and manager of triggers.

PDB Initialization

The core feature of the PDB module is a programmable timer/counter. Additional features enable and set the milestone for the corresponding trigger. The user should provide a configuration suitable for the application requirements. Call the API of **PDB_DRV_Init()** function to initialize the PDB timer/counter.

All triggers share the same counter.

The basic timing/counting step is set when initializing the main PDB counter:

The basic timing/counting step = F_BusClkHz / pdb_timer_config_t.clkPreDiv / pdb_timer_config_t.clkPreMultFactor

The F_BusClkHz is the frequency of bus clock in Hertz. The "clkPreDiv" and "clkPreMultFactor" are in the pdb_timer_config_t structure. All triggering milestones are based on this step.

PDB Call diagram

Three kinds of typical use cases are designed for the PDB module.

- Normal Timer/Counter. Normal Timer/Counter is the basic case. The Timer/Counter starts after the PDB is initialized and the milestone for the PDB Timer/Counter is set. After it is triggered and when the counter hits the milestone, the interrupt request occurs if enabled. In continuous mode, when the counter hits the upper limit, it returns zero and counts again.
- Trigger for ADC module. When the ADC trigger is enabled, a delay value for ADC trigger is set as the milestone. At least two ADC channel groups are provided. Likewise, there are more than two pre-triggers for ADC. Each pre-trigger is related to one channel group and can be enabled separately in the PDB module. When the PDB counter hits the milestone for the ADC pre-trigger, it triggers the ADC's conversion on the indicated channel group. To maximize the feature, the ADC should be configured to enable the hardware trigger mode.
- Trigger for pulse out to the CMP module. The pulse-out trigger is attached to the main PDB counter. There are two milestones for each pulse out channel, a milestone for level high and for level low, which makes a sample window for the CMP module.

These are the examples to initialize and configure the PDB driver for typical use cases.

Normal Timer/Counter:

```
#define PDB_INSTANCE    0UL

static volatile uint32_t gPdbIntCounter = 0U;
static volatile uint32_t gPdbInstance = 0U;
static void PDB_ISR_Counter(void);
void PDB_TEST_NormalTimer(uint32_t instance)
{
    pdb_timer_config_t PdbTimerConfig;
    PdbTimerConfig.loadValueMode         = PDB_LOAD_VAL_IMMEDIATELY;
    PdbTimerConfig.seqErrIntEnable       = false;
    PdbTimerConfig.clkPreDiv             = PDB_CLK_PREDIV_BY_8;
    PdbTimerConfig.clkPreMultFactor      =
      PDB_CLK_PREMULT_FACT_AS_40;
    PdbTimerConfig.triggerInput          = PDB_SOFTWARE_TRIGGER;
    PdbTimerConfig.continuousModeEnable  = true;
    PdbTimerConfig.dmaEnable             = false;
    PdbTimerConfig.intEnable             = true;
    PDB_DRV_Init(instance, &PdbTimerConfig);
    PDB_DRV_SetTimerModulusValue(instance, 0xFFFU);
    PDB_DRV_SetValueForTimerInterrupt(instance, 0xFFU);
    PDB_DRV_LoadValuesCmd(instance);
    gPdbIntCounter = 0U;
    gPdbInstance = instance;
    PDB_DRV_SoftTriggerCmd(instance);
    while (gPdbIntCounter < 20U) {}
    PRINTF("PDB Timer's delay interrupt generated.\r\n");
    PDB_DRV_Deinit(instance);
    PRINTF("OK.\r\n");
}

void PDB_IRQHandler()
{
    PDB_DRV_ClearTimerIntFlag(PDB_INSTANCE);
    if (gPdbIntCounter >= 0xFFFFU)
    {
        gPdbIntCounter = 0U;
    }
    else
    {
        gPdbIntCounter++;
    }
}

#if PDB_INSTANCE < 1
    void PDB0_IRQHandler(void)
    {
        PDB_IRQHandler();
    }
#elif PDB_INSTANCE < 2
    void PDB1_IRQHandler(void)
    {
        PDB_IRQHandler();
    }
#endif
```

Trigger for ADC module:

```
void PDB_TEST_AdcPreTrigger(uint32_t instance)
{
    pdb_timer_config_t PdbTimerConfig;
    pdb_adc_pretrigger_config_t PdbAdcPreTriggerConfig;
    PdbTimerConfig.loadValueMode         = PDB_LOAD_VAL_IMMEDIATELY;
    PdbTimerConfig.seqErrIntEnable       = false;
    PdbTimerConfig.clkPreDiv             = PDB_CLK_PREDIV_BY_8;
    PdbTimerConfig.clkPreMultFactor      =
      PDB_CLK_PREMULT_FACT_AS_40;
    PdbTimerConfig.triggerInput          = PDB_SOFTWARE_TRIGGER;
    PdbTimerConfig.continuousModeEnable  = false;
    PdbTimerConfig.dmaEnable             = false;
    PdbTimerConfig.intEnable             = false;
    PDB_DRV_Init(instance, &PdbTimerConfig);

    PdbAdcPreTriggerConfig.adcPreTriggerIdx              = 0U;
    PdbAdcPreTriggerConfig.preTriggerEnable              = true;
    PdbAdcPreTriggerConfig.preTriggerOutputEnable        = true;
    PdbAdcPreTriggerConfig.preTriggerBackToBackEnable    = false;
    PDB_DRV_ConfigAdcPreTrigger(instance, 0U, &PdbAdcPreTriggerConfig);
    PDB_DRV_SetTimerModulusValue(instance, 0xFFFU);
    PDB_DRV_SetAdcPreTriggerDelayValue(instance, 0U, 0U, 0xFFU);
    PDB_DRV_LoadValuesCmd(instance);
    PDB_DRV_SoftTriggerCmd(instance);
    while (1U != PDB_DRV_GetAdcPreTriggerFlags(instance, 0U, 1U)) {}
    PDB_DRV_ClearAdcPreTriggerFlags(instance, 0U, 1U);
    PRINTF("PDB ADC PreTrigger generated.\r\n");
    PDB_DRV_Deinit(instance);
    PRINTF("OK.\r\n");
}
```

Data Structures

- struct pdb_adc_pretrigger_config_t

  Defines the type of structure for configuring ADC's pre_trigger. More...

Functions

- void PDB_DRV_Init (const uint32_t instance, const pdb_timer_config_t ∗userConfigPtr)

  Initializes the PDB counter and triggers input.
- void PDB_DRV_Deinit (const uint32_t instance)

  De-initializes the PDB module.
- void PDB_DRV_SoftTriggerCmd (const uint32_t instance)

  Triggers the PDB with a software trigger.
- uint32_t PDB_DRV_GetTimerValue (const uint32_t instance)

  Gets the current counter value in the PDB module.
- bool PDB_DRV_GetTimerIntFlag (const uint32_t instance)

  Gets the PDB interrupt flag.
- void PDB_DRV_ClearTimerIntFlag (const uint32_t instance)

  Clears the interrupt flag.
- void PDB_DRV_LoadValuesCmd (const uint32_t instance)

  Executes the command of loading values.
- void PDB_DRV_SetTimerModulusValue (const uint32_t instance, const uint32_t value)

  Sets the value of timer modulus.
- void PDB_DRV_SetValueForTimerInterrupt (const uint32_t instance, const uint32_t value)

  Sets the value for the timer interrupt.
- void PDB_DRV_ConfigAdcPreTrigger (const uint32_t instance, const uint32_t chn, const pdb_adc_↩ pretrigger_config_t ∗configPtr)

  Configures the ADC pre_trigger in the PDB module.
- uint32_t PDB_DRV_GetAdcPreTriggerFlags (const uint32_t instance, const uint32_t chn, const uint32_↩ t preChnMask)

  Gets the ADC pre_trigger flag in the PDB module.
- void PDB_DRV_ClearAdcPreTriggerFlags (const uint32_t instance, const uint32_t chn, const uint32_t pre↩ ChnMask)

  Clears the ADC pre_trigger flag in the PDB module.
- uint32_t PDB_DRV_GetAdcPreTriggerSeqErrFlags (const uint32_t instance, const uint32_t chn, const uint32_t preChnMask)

  Gets the ADC pre_trigger flag in the PDB module.
- void PDB_DRV_ClearAdcPreTriggerSeqErrFlags (const uint32_t instance, const uint32_t chn, const uint32↩ _t preChnMask)

  Clears the ADC pre_trigger flag in the PDB module.
- void PDB_DRV_SetAdcPreTriggerDelayValue (const uint32_t instance, const uint32_t chn, const uint32_t preChn, const uint32_t value)

Sets the ADC pre_trigger delay value in the PDB module.
- void PDB_DRV_SetCmpPulseOutEnable (const uint32_t instance, const uint32_t pulseChnMask, bool enable)

  Switches on/off the CMP pulse out in the PDB module.
- void PDB_DRV_SetCmpPulseOutDelayForHigh (const uint32_t instance, const uint32_t pulseChn, const uint32_t value)

  Sets the CMP pulse out delay value for high in the PDB module.
- void PDB_DRV_SetCmpPulseOutDelayForLow (const uint32_t instance, const uint32_t pulseChn, const uint32_t value)

  Sets the CMP pulse out delay value for low in the PDB module.

3.66.2 Data Structure Documentation

3.66.2.1 struct pdb_adc_pretrigger_config_t

Defines the type of structure for configuring ADC's pre_trigger.

Data Fields

- uint32_t adcPreTriggerIdx
- bool preTriggerEnable
- bool preTriggerOutputEnable
- bool preTriggerBackToBackEnable

Field Documentation

3.66.2.1.1 adcPreTriggerIdx

```
uint32_t adcPreTriggerIdx
```

Setting pre_trigger's index.

3.66.2.1.2 preTriggerBackToBackEnable

```
bool preTriggerBackToBackEnable
```

Enable the back to back mode for ADC pre_trigger.

3.66.2.1.3 preTriggerEnable

```
bool preTriggerEnable
```

Enable the pre_trigger.

3.66.2.1.4 preTriggerOutputEnable

```
bool preTriggerOutputEnable
```

Enable the pre_trigger output.

3.66.3 Function Documentation

3.66.3.1 PDB_DRV_ClearAdcPreTriggerFlags()

```
void PDB_DRV_ClearAdcPreTriggerFlags (
             const uint32_t instance,
             const uint32_t chn,
             const uint32_t preChnMask )
```

Clears the ADC pre_trigger flag in the PDB module.

This function clears the ADC pre_trigger flags in the PDB module.

Parameters

instance	PDB instance ID.
chn	ADC channel.
preChnMask	ADC pre_trigger channels mask.

3.66.3.2 PDB_DRV_ClearAdcPreTriggerSeqErrFlags()

```
void PDB_DRV_ClearAdcPreTriggerSeqErrFlags (
             const uint32_t instance,
             const uint32_t chn,
             const uint32_t preChnMask )
```

Clears the ADC pre_trigger flag in the PDB module.

This function clears the ADC pre_trigger sequence error flags in the PDB module.

Parameters

instance	PDB instance ID.
chn	ADC channel.
preChnMask	ADC pre_trigger channels mask.

3.66.3.3 PDB_DRV_ClearTimerIntFlag()

```
void PDB_DRV_ClearTimerIntFlag (
             const uint32_t instance )
```

Clears the interrupt flag.

This function clears the interrupt flag.

Parameters

instance	PDB instance ID.

3.66.3.4 PDB_DRV_ConfigAdcPreTrigger()

```
void PDB_DRV_ConfigAdcPreTrigger (
             const uint32_t instance,
             const uint32_t chn,
             const pdb_adc_pretrigger_config_t * configPtr )
```

Configures the ADC pre_trigger in the PDB module.

This function configures the ADC pre_trigger in the PDB module.

Parameters

instance	PDB instance ID.
chn	ADC channel.
configPtr	Pointer to the user configuration structure. See the "pdb_adc_pretrigger_config_t".

3.66.3.5 PDB_DRV_Deinit()

```
void PDB_DRV_Deinit (
            const uint32_t instance )
```

De-initializes the PDB module.

This function de-initializes the PDB module. Calling this function shuts down the PDB module and reduces the power consumption.

Parameters

instance	PDB instance ID.

3.66.3.6 PDB_DRV_GetAdcPreTriggerFlags()

```
uint32_t PDB_DRV_GetAdcPreTriggerFlags (
            const uint32_t instance,
            const uint32_t chn,
            const uint32_t preChnMask )
```

Gets the ADC pre_trigger flag in the PDB module.

This function gets the ADC pre_trigger flags in the PDB module.

Parameters

instance	PDB instance ID.
chn	ADC channel.
preChnMask	ADC pre_trigger channels mask.

Returns

Assertion of indicated flag.

3.66.3.7 PDB_DRV_GetAdcPreTriggerSeqErrFlags()

```
uint32_t PDB_DRV_GetAdcPreTriggerSeqErrFlags (
            const uint32_t instance,
            const uint32_t chn,
            const uint32_t preChnMask )
```

Gets the ADC pre_trigger flag in the PDB module.

This function gets the ADC pre_trigger flags in the PDB module.

Parameters

instance	PDB instance ID.
chn	ADC channel.
preChnMask	ADC pre_trigger channels mask.

Returns

Assertion of indicated flag.

3.66.3.8 PDB_DRV_GetTimerIntFlag()

```
bool PDB_DRV_GetTimerIntFlag (
             const uint32_t instance )
```

Gets the PDB interrupt flag.

This function gets the PDB interrupt flag. It is asserted if the PDB interrupt occurs.

Parameters

instance	PDB instance ID.

Returns

Assertion of indicated event.

3.66.3.9 PDB_DRV_GetTimerValue()

```
uint32_t PDB_DRV_GetTimerValue (
             const uint32_t instance )
```

Gets the current counter value in the PDB module.

This function gets the current counter value.

Parameters

instance	PDB instance ID.

Returns

Current PDB counter value.

3.66.3.10 PDB_DRV_Init()

```
void PDB_DRV_Init (
             const uint32_t instance,
             const pdb_timer_config_t * userConfigPtr )
```

Initializes the PDB counter and triggers input.

This function initializes the PDB counter and triggers the input. It resets PDB registers and enables the PDB clock. Therefore, it should be called before any other operation. After it is initialized, the PDB can act as a triggered timer, which enables other features in PDB module.

Parameters

instance	PDB instance ID.
userConfigPtr	Pointer to the user configuration structure. See the "pdb_user_config_t".

3.66.3.11 PDB_DRV_LoadValuesCmd()

```
void PDB_DRV_LoadValuesCmd (
             const uint32_t instance )
```

Executes the command of loading values.

This function executes the command of loading values.

Parameters

instance	PDB instance ID.

3.66.3.12 PDB_DRV_SetAdcPreTriggerDelayValue()

```
void PDB_DRV_SetAdcPreTriggerDelayValue (
             const uint32_t instance,
             const uint32_t chn,
             const uint32_t preChn,
             const uint32_t value )
```

Sets the ADC pre_trigger delay value in the PDB module.

This function sets Set the ADC pre_trigger delay value in the PDB module.

Parameters

instance	PDB instance ID.
chn	ADC channel.
preChn	ADC pre_channel.
value	Setting value.

3.66.3.13 PDB_DRV_SetCmpPulseOutDelayForHigh()

```
void PDB_DRV_SetCmpPulseOutDelayForHigh (
             const uint32_t instance,
             const uint32_t pulseChn,
             const uint32_t value )
```

Sets the CMP pulse out delay value for high in the PDB module.

This function sets the CMP pulse out delay value for high in the PDB module.

Parameters

instance	PDB instance ID.
pulseChn	Pulse channel.
value	Setting value.

3.66.3.14 PDB_DRV_SetCmpPulseOutDelayForLow()

```
void PDB_DRV_SetCmpPulseOutDelayForLow (
            const uint32_t instance,
            const uint32_t pulseChn,
            const uint32_t value )
```

Sets the CMP pulse out delay value for low in the PDB module.

This function sets the CMP pulse out delay value for low in the PDB module.

Parameters

instance	PDB instance ID.
pulseChn	Pulse channel.
value	Setting value.

3.66.3.15 PDB_DRV_SetCmpPulseOutEnable()

```
void PDB_DRV_SetCmpPulseOutEnable (
            const uint32_t instance,
            const uint32_t pulseChnMask,
            bool enable )
```

Switches on/off the CMP pulse out in the PDB module.

This function switches the CMP pulse on/off in the PDB module.

Parameters

instance	PDB instance ID.
pulseChnMask	Pulse channel mask.
enable	Switcher to assert the feature.

3.66.3.16 PDB_DRV_SetTimerModulusValue()

```
void PDB_DRV_SetTimerModulusValue (
            const uint32_t instance,
            const uint32_t value )
```

Sets the value of timer modulus.

This function sets the value of timer modulus.

Parameters

instance	PDB instance ID.
value	Setting value.

3.66.3.17 PDB_DRV_SetValueForTimerInterrupt()

```
void PDB_DRV_SetValueForTimerInterrupt (
            const uint32_t instance,
            const uint32_t value )
```

Sets the value for the timer interrupt.

This function sets the value for the timer interrupt.

Parameters

instance	PDB instance ID.
value	Setting value.

3.66.3.18 PDB_DRV_SoftTriggerCmd()

```
void PDB_DRV_SoftTriggerCmd (
            const uint32_t instance )
```

Triggers the PDB with a software trigger.

This function triggers the PDB with a software trigger. When the PDB is set to use the software trigger as input, calling this function triggers the PDB.

Parameters

instance	PDB instance ID.

3.67 PDB HAL

3.67.1 Detailed Description

Programmable Delay Block Hardware Abstraction Layer.

This HAL provides low-level access to all hardware features of the PDB.

Data Structures

- struct pdb_timer_config_t

  Defines the type of structure for basic timer in PDB. More...

Enumerations

- enum pdb_load_value_mode_t { PDB_LOAD_VAL_IMMEDIATELY = 0U, PDB_LOAD_VAL_AT_MODUL↩O_COUNTER = 1U, PDB_LOAD_VAL_AT_NEXT_TRIGGER = 2U, PDB_LOAD_VAL_AT_MODULO_CO↩UNTER_OR_NEXT_TRIGGER = 3U }

  Defines the type of value load mode for the PDB module.
- enum pdb_clk_prescaler_div_t {
  PDB_CLK_PREDIV_BY_1 = 0U, PDB_CLK_PREDIV_BY_2 = 1U, PDB_CLK_PREDIV_BY_4 = 2U, PDB_↩CLK_PREDIV_BY_8 = 3U,
  PDB_CLK_PREDIV_BY_16 = 4U, PDB_CLK_PREDIV_BY_32 = 5U, PDB_CLK_PREDIV_BY_64 = 6U, P↩DB_CLK_PREDIV_BY_128 = 7U }

  Defines the type of prescaler divider for the PDB counter clock. Implements : pdb_clk_prescaler_div_t_Class.
- enum pdb_trigger_src_t {
  PDB_TRIGGER_0 = 0U, PDB_TRIGGER_1 = 1U, PDB_TRIGGER_2 = 2U, PDB_TRIGGER_3 = 3U,
  PDB_TRIGGER_4 = 4U, PDB_TRIGGER_5 = 5U, PDB_TRIGGER_6 = 6U, PDB_TRIGGER_7 = 7U,
  PDB_TRIGGER_8 = 8U, PDB_TRIGGER_9 = 9U, PDB_TRIGGER_10 = 10U, PDB_TRIGGER_11 = 11U,
  PDB_TRIGGER_12 = 12U, PDB_TRIGGER_13 = 13U, PDB_TRIGGER_14 = 14U, PDB_SOFTWARE_T↩RIGGER = 15U }

  Defines the type of trigger source mode for the PDB.
- enum pdb_clk_prescaler_mult_factor_t { PDB_CLK_PREMULT_FACT_AS_1 = 0U, PDB_CLK_PREMUL↩T_FACT_AS_10 = 1U, PDB_CLK_PREMULT_FACT_AS_20 = 2U, PDB_CLK_PREMULT_FACT_AS_40 = 3U }

  Defines the type of the multiplication source mode for PDB.

Functions

- void PDB_HAL_Init (PDB_Type ∗const base)

  Resets the PDB registers to a known state.
- void PDB_HAL_ConfigTimer (PDB_Type ∗const base, const pdb_timer_config_t ∗const configPtr)

  Configure the PDB timer.
- static void PDB_HAL_SetSoftTriggerCmd (PDB_Type ∗const base)

  Triggers the PDB by software if enabled.
- static void PDB_HAL_Enable (PDB_Type ∗const base)

  Switches on to enable the PDB module.
- static void PDB_HAL_Disable (PDB_Type ∗const base)

  Switches to disable the PDB module.
- static bool PDB_HAL_GetTimerIntFlag (PDB_Type const ∗const base)

  Gets the PDB delay interrupt flag.

- static void PDB_HAL_ClearTimerIntFlag (PDB_Type ∗const base)

  Clears the PDB delay interrupt flag.
- static void PDB_HAL_SetLoadValuesCmd (PDB_Type ∗const base)

  Loads the delay registers value for the PDB module.
- static void PDB_HAL_SetTimerModulusValue (PDB_Type ∗const base, uint32_t value)

  Sets the modulus value for the PDB module.
- static uint32_t PDB_HAL_GetTimerValue (PDB_Type const ∗const base)

  Gets the PDB counter value of PDB timer.
- static void PDB_HAL_SetValueForTimerInterrupt (PDB_Type ∗const base, uint32_t value)

  Sets the interrupt delay milestone of the PDB counter.
- void PDB_HAL_SetAdcPreTriggerBackToBackEnable (PDB_Type ∗const base, uint32_t chn, uint32_t pre←ChnMask, bool enable)

  Switches to enable the pre-trigger back-to-back mode.
- void PDB_HAL_SetAdcPreTriggerOutputEnable (PDB_Type ∗const base, uint32_t chn, uint32_t preChn←Mask, bool enable)

  Switches to enable the pre-trigger output.
- void PDB_HAL_SetAdcPreTriggerEnable (PDB_Type ∗const base, uint32_t chn, uint32_t preChnMask, bool enable)

  Switches to enable the pre-trigger.
- static uint32_t PDB_HAL_GetAdcPreTriggerFlags (PDB_Type const ∗const base, uint32_t chn, uint32_←t preChnMask)

  Gets the flag which indicates whether the PDB counter has reached the pre-trigger delay value.
- void PDB_HAL_ClearAdcPreTriggerFlags (PDB_Type ∗const base, uint32_t chn, uint32_t preChnMask)

  Clears the flag which indicates that the PDB counter has reached the pre-trigger delay value.
- static uint32_t PDB_HAL_GetAdcPreTriggerSeqErrFlags (PDB_Type const ∗const base, uint32_t chn, uint32_t preChnMask)

  Gets the flag which indicates whether a sequence error is detected.
- void PDB_HAL_ClearAdcPreTriggerSeqErrFlags (PDB_Type ∗const base, uint32_t chn, uint32_t preChn←Mask)

  Clears the flag which indicates that a sequence error has been detected.
- void PDB_HAL_SetAdcPreTriggerDelayValue (PDB_Type ∗const base, uint32_t chn, uint32_t preChn, uint32_t value)

  Sets the pre-trigger delay value.
- void PDB_HAL_SetCmpPulseOutEnable (PDB_Type ∗const base, uint32_t pulseChnMask, bool enable)

  Switches to enable the pulse-out trigger.
- static void PDB_HAL_SetCmpPulseOutDelayForHigh (PDB_Type ∗const base, uint32_t pulseChn, uint32_t value)

  Sets the counter delay value for the pulse-out goes high.
- static void PDB_HAL_SetCmpPulseOutDelayForLow (PDB_Type ∗const base, uint32_t pulseChn, uint32_t value)

  Sets the counter delay value for the pulse-out goes low.

3.67.2 Data Structure Documentation

3.67.2.1 struct pdb_timer_config_t

Defines the type of structure for basic timer in PDB.

Data Fields

- pdb_load_value_mode_t loadValueMode
- bool seqErrIntEnable
- pdb_clk_prescaler_div_t clkPreDiv
- pdb_clk_prescaler_mult_factor_t clkPreMultFactor
- pdb_trigger_src_t triggerInput
- bool continuousModeEnable
- bool dmaEnable
- bool intEnable

Field Documentation

3.67.2.1.1 clkPreDiv

`pdb_clk_prescaler_div_t clkPreDiv`

Select the prescaler divider.

3.67.2.1.2 clkPreMultFactor

`pdb_clk_prescaler_mult_factor_t clkPreMultFactor`

Select multiplication factor for prescaler.

3.67.2.1.3 continuousModeEnable

`bool continuousModeEnable`

Enable the continuous mode.

3.67.2.1.4 dmaEnable

`bool dmaEnable`

Enable the dma for timer.

3.67.2.1.5 intEnable

`bool intEnable`

Enable the interrupt for timer.

3.67.2.1.6 loadValueMode

`pdb_load_value_mode_t loadValueMode`

Select the load mode.

3.67.2.1.7 seqErrIntEnable

`bool seqErrIntEnable`

Enable PDB Sequence Error Interrupt.

3.67.2.1.8 triggerInput

`pdb_trigger_src_t triggerInput`

Select the trigger input source.

3.67.3 Enumeration Type Documentation

3.67.3.1 pdb_clk_prescaler_div_t

`enum pdb_clk_prescaler_div_t`

Defines the type of prescaler divider for the PDB counter clock. Implements : pdb_clk_prescaler_div_t_Class.

Enumerator

PDB_CLK_PREDIV_BY_1	Counting divided by multiplication factor selected by MULT.
PDB_CLK_PREDIV_BY_2	Counting divided by multiplication factor selected by 2 times ofMULT.
PDB_CLK_PREDIV_BY_4	Counting divided by multiplication factor selected by 4 times ofMULT.
PDB_CLK_PREDIV_BY_8	Counting divided by multiplication factor selected by 8 times ofMULT.
PDB_CLK_PREDIV_BY_16	Counting divided by multiplication factor selected by 16 times ofMULT.
PDB_CLK_PREDIV_BY_32	Counting divided by multiplication factor selected by 32 times ofMULT.
PDB_CLK_PREDIV_BY_64	Counting divided by multiplication factor selected by 64 times ofMULT.
PDB_CLK_PREDIV_BY_128	Counting divided by multiplication factor selected by 128 times ofMULT.

3.67.3.2 pdb_clk_prescaler_mult_factor_t

```
enum pdb_clk_prescaler_mult_factor_t
```

Defines the type of the multiplication source mode for PDB.

Selects the multiplication factor of the prescaler divider for the PDB counter clock. Implements : pdb_clk_↩
prescaler_mult_factor_t_Class

Enumerator

PDB_CLK_PREMULT_FACT_AS_1	Multiplication factor is 1.
PDB_CLK_PREMULT_FACT_AS_10	Multiplication factor is 10.
PDB_CLK_PREMULT_FACT_AS_20	Multiplication factor is 20.
PDB_CLK_PREMULT_FACT_AS_40	Multiplication factor is 40.

3.67.3.3 pdb_load_value_mode_t

```
enum pdb_load_value_mode_t
```

Defines the type of value load mode for the PDB module.

Some timing related registers, such as the MOD, IDLY, CHnDLYm, INTx and POyDLY, buffer the setting values. Only the load operation is triggered. The setting value is loaded from a buffer and takes effect. There are four loading modes to fit different applications. Implements : pdb_load_value_mode_t_Class

Enumerator

PDB_LOAD_VAL_IMMEDIATELY	Loaded immediately after load operation.
PDB_LOAD_VAL_AT_MODULO_COUNTER	Loaded when counter hits the modulo after load operation.
PDB_LOAD_VAL_AT_NEXT_TRIGGER	Loaded when detecting an input trigger after load operation.
PDB_LOAD_VAL_AT_MODULO_COUNTER_OR↩ _NEXT_TRIGGER	Loaded when counter hits the modulo or detecting an input trigger after load operation.

3.67.3.4 pdb_trigger_src_t

```
enum pdb_trigger_src_t
```

Defines the type of trigger source mode for the PDB.

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger. Implements : pdb_trigger_src_t_Class

Enumerator

PDB_TRIGGER_0	Select trigger-In 0.
PDB_TRIGGER_1	Select trigger-In 1.
PDB_TRIGGER_2	Select trigger-In 2.
PDB_TRIGGER_3	Select trigger-In 3.
PDB_TRIGGER_4	Select trigger-In 4.
PDB_TRIGGER_5	Select trigger-In 5.
PDB_TRIGGER_6	Select trigger-In 6.
PDB_TRIGGER_7	Select trigger-In 7.
PDB_TRIGGER_8	Select trigger-In 8.
PDB_TRIGGER_9	Select trigger-In 8.
PDB_TRIGGER_10	Select trigger-In 10.
PDB_TRIGGER_11	Select trigger-In 11.
PDB_TRIGGER_12	Select trigger-In 12.
PDB_TRIGGER_13	Select trigger-In 13.
PDB_TRIGGER_14	Select trigger-In 14.
PDB_SOFTWARE_TRIGGER	Select software trigger.

3.67.4 Function Documentation

3.67.4.1 PDB_HAL_ClearAdcPreTriggerFlags()

```
void PDB_HAL_ClearAdcPreTriggerFlags (
            PDB_Type *const base,
            uint32_t chn,
            uint32_t preChnMask )
```

Clears the flag which indicates that the PDB counter has reached the pre-trigger delay value.

This function clears the flag which indicates that the PDB counter has reached the pre-trigger delay value.

Parameters

base	Register base address for the module.
chn	ADC instance index for trigger.
preChnMask	ADC channel group index mask for trigger.

3.67.4.2 PDB_HAL_ClearAdcPreTriggerSeqErrFlags()

```
void PDB_HAL_ClearAdcPreTriggerSeqErrFlags (
            PDB_Type *const base,
            uint32_t chn,
            uint32_t preChnMask )
```

Clears the flag which indicates that a sequence error has been detected.

This function clears the flag which indicates that the sequence error has been detected.

Parameters

base	Register base address for the module.
chn	ADC instance index for trigger.
preChnMask	ADC channel group index mask for trigger.

3.67.4.3 PDB_HAL_ClearTimerIntFlag()

```
static void PDB_HAL_ClearTimerIntFlag (
             PDB_Type *const base )  [inline], [static]
```

Clears the PDB delay interrupt flag.

This function clears PDB delay interrupt flag.

Parameters

base	Register base address for the module. Implements : PDB_HAL_ClearTimerIntFlag_Activity

3.67.4.4 PDB_HAL_ConfigTimer()

```
void PDB_HAL_ConfigTimer (
             PDB_Type *const base,
             const pdb_timer_config_t *const configPtr )
```

Configure the PDB timer.

This function configure the PDB's basic timer.

Parameters

base	Register base address for the module.
configPtr	Pointer to configuration structure, see to "pdb_timer_config_t".

3.67.4.5 PDB_HAL_Disable()

```
static void PDB_HAL_Disable (
             PDB_Type *const base )  [inline], [static]
```

Switches to disable the PDB module.

This function switches to disable the PDB module.

Parameters

base	Register base address for the module. Implements : PDB_HAL_Disable_Activity

3.67.4.6 PDB_HAL_Enable()

```
static void PDB_HAL_Enable (
            PDB_Type *const base ) [inline], [static]
```

Switches on to enable the PDB module.

This function switches on to enable the PDB module.

Parameters

base	Register base address for the module. Implements : PDB_HAL_Enable_Activity

3.67.4.7 PDB_HAL_GetAdcPreTriggerFlags()

```
static uint32_t PDB_HAL_GetAdcPreTriggerFlags (
            PDB_Type const *const base,
            uint32_t chn,
            uint32_t preChnMask ) [inline], [static]
```

Gets the flag which indicates whether the PDB counter has reached the pre-trigger delay value.

This function gets the flag which indicates the PDB counter has reached the pre-trigger delay value.

Parameters

base	Register base address for the module.
chn	ADC instance index for trigger.
preChnMask	ADC channel group index mask for trigger.

Returns

Flag mask. Indicated bit would be 1 if the event is asserted. Implements : PDB_HAL_GetAdcPreTrigger←Flags_Activity

3.67.4.8 PDB_HAL_GetAdcPreTriggerSeqErrFlags()

```
static uint32_t PDB_HAL_GetAdcPreTriggerSeqErrFlags (
            PDB_Type const *const base,
            uint32_t chn,
            uint32_t preChnMask ) [inline], [static]
```

Gets the flag which indicates whether a sequence error is detected.

This function gets the flag which indicates whether a sequence error is detected.

Parameters

base	Register base address for the module.
chn	ADC instance index for trigger.
preChnMask	ADC channel group index mask for trigger.

Returns

Flag mask. Indicated bit would be 1 if the event is asserted. Implements : PDB_HAL_GetAdcPreTrigger↩ SeqErrFlags_Activity

3.67.4.9 PDB_HAL_GetTimerIntFlag()

```
static bool PDB_HAL_GetTimerIntFlag (
            PDB_Type const *const base )  [inline], [static]
```

Gets the PDB delay interrupt flag.

This function gets the PDB delay interrupt flag.

Parameters

base	Register base address for the module.

Returns

Flat status, true if the flag is set. Implements : PDB_HAL_GetTimerIntFlag_Activity

3.67.4.10 PDB_HAL_GetTimerValue()

```
static uint32_t PDB_HAL_GetTimerValue (
            PDB_Type const *const base )  [inline], [static]
```

Gets the PDB counter value of PDB timer.

This function gets the PDB counter value of PDB timer.

Parameters

base	Register base address for the module.

Returns

The current counter value. Implements : PDB_HAL_GetTimerValue_Activity

3.67.4.11 PDB_HAL_Init()

```
void PDB_HAL_Init (
            PDB_Type *const base )
```

Resets the PDB registers to a known state.

This function resets the PDB registers to a known state. This state is defined in a reference manual and is power on reset value.

Parameters

base	Register base address for the module.

3.67.4.12 PDB_HAL_SetAdcPreTriggerBackToBackEnable()

```
void PDB_HAL_SetAdcPreTriggerBackToBackEnable (
            PDB_Type *const base,
            uint32_t chn,
            uint32_t preChnMask,
            bool enable )
```

Switches to enable the pre-trigger back-to-back mode.

This function switches to enable the pre-trigger back-to-back mode.

Parameters

base	Register base address for the module.
chn	ADC instance index for trigger.
preChnMask	ADC channel group index mask for trigger.
enable	Switcher to assert the feature.

3.67.4.13 PDB_HAL_SetAdcPreTriggerDelayValue()

```
void PDB_HAL_SetAdcPreTriggerDelayValue (
            PDB_Type *const base,
            uint32_t chn,
            uint32_t preChn,
            uint32_t value )
```

Sets the pre-trigger delay value.

This function sets the pre-trigger delay value.

Parameters

base	Register base address for the module.
chn	ADC instance index for trigger.
preChn	ADC channel group index for trigger.
value	Setting value for pre-trigger's delay value.

3.67.4.14 PDB_HAL_SetAdcPreTriggerEnable()

```
void PDB_HAL_SetAdcPreTriggerEnable (
            PDB_Type *const base,
            uint32_t chn,
            uint32_t preChnMask,
            bool enable )
```

Switches to enable the pre-trigger.

This function switches to enable the pre-trigger.

Parameters

base	Register base address for the module.
chn	ADC instance index for trigger.
preChnMask	ADC channel group index mask for trigger.
enable	Switcher to assert the feature.

3.67.4.15 PDB_HAL_SetAdcPreTriggerOutputEnable()

```
void PDB_HAL_SetAdcPreTriggerOutputEnable (
             PDB_Type *const base,
             uint32_t chn,
             uint32_t preChnMask,
             bool enable )
```

Switches to enable the pre-trigger output.

This function switches to enable pre-trigger output.

Parameters

base	Register base address for the module.
chn	ADC instance index for trigger.
preChnMask	ADC channel group index mask for trigger.
enable	Switcher to assert the feature.

3.67.4.16 PDB_HAL_SetCmpPulseOutDelayForHigh()

```
static void PDB_HAL_SetCmpPulseOutDelayForHigh (
             PDB_Type *const base,
             uint32_t pulseChn,
             uint32_t value )  [inline], [static]
```

Sets the counter delay value for the pulse-out goes high.

This function sets the counter delay value for the pulse-out goes high.

Parameters

base	Register base address for the module.
pulseChn	Pulse-out channel index for trigger.
value	Setting value for PDB delay . Implements : PDB_HAL_SetCmpPulseOutDelayForHigh_Activity

3.67.4.17 PDB_HAL_SetCmpPulseOutDelayForLow()

```
static void PDB_HAL_SetCmpPulseOutDelayForLow (
             PDB_Type *const base,
             uint32_t pulseChn,
             uint32_t value )  [inline], [static]
```

Sets the counter delay value for the pulse-out goes low.

This function sets the counter delay value for the pulse-out goes low.

Parameters

base	Register base address for the module.
pulseChn	Pulse-out channel index for trigger.
value	Setting value for PDB delay . Implements : PDB_HAL_SetCmpPulseOutDelayForLow_Activity

3.67.4.18 PDB_HAL_SetCmpPulseOutEnable()

```
void PDB_HAL_SetCmpPulseOutEnable (
             PDB_Type *const base,
             uint32_t pulseChnMask,
             bool enable )
```

Switches to enable the pulse-out trigger.

This function switches to enable the pulse-out trigger.

Parameters

base	Register base address for the module.
pulseChnMask	Pulse-out channle index mask for trigger.
enable	Switcher to assert the feature.

3.67.4.19 PDB_HAL_SetLoadValuesCmd()

```
static void PDB_HAL_SetLoadValuesCmd (
             PDB_Type *const base ) [inline], [static]
```

Loads the delay registers value for the PDB module.

This function sets the LDOK bit and loads the delay registers value. Writing one to this bit updates the internal registers MOD, IDLY, CHnDLYm and POyDLY with the values written to their buffers. The MOD, IDLY, CHnDLYm and POyDLY take effect according to the load mode settings.

After one is written to the LDOK bit, the values in the buffers of above mentioned registers are not effective and cannot be written until the values in the buffers are loaded into their internal registers. The LDOK can be written only when the the PDB is enabled or as alone with it. It is automatically cleared either when the values in the buffers are loaded into the internal registers or when the PDB is disabled.

Parameters

base	Register base address for the module. Implements : PDB_HAL_SetLoadValuesCmd_Activity

3.67.4.20 PDB_HAL_SetSoftTriggerCmd()

```
static void PDB_HAL_SetSoftTriggerCmd (
             PDB_Type *const base ) [inline], [static]
```

Triggers the PDB by software if enabled.

If enabled, this function triggers the PDB by using software.

Parameters

base	Register base address for the module. Implements : PDB_HAL_SetSoftTriggerCmd_Activity

3.67.4.21 PDB_HAL_SetTimerModulusValue()

```
static void PDB_HAL_SetTimerModulusValue (
             PDB_Type *const base,
             uint32_t value )  [inline], [static]
```

Sets the modulus value for the PDB module.

This function sets the modulus value for the PDB module. When the counter reaches the setting value, it is automatically reset to zero. When in continuous mode, the counter begins to increase again.

Parameters

base	Register base address for the module.
value	The setting value of upper limit for PDB counter. Implements : PDB_HAL_SetTimerModulusValue_Activity

3.67.4.22 PDB_HAL_SetValueForTimerInterrupt()

```
static void PDB_HAL_SetValueForTimerInterrupt (
             PDB_Type *const base,
             uint32_t value )  [inline], [static]
```

Sets the interrupt delay milestone of the PDB counter.

This function sets the interrupt delay milestone of the PDB counter. If enabled, a PDB interrupt is generated when the counter is equal to the setting value.

Parameters

base	Register base address for the module.
value	The setting value for interrupt delay milestone of PDB counter. Implements : PDB_HAL_SetValueForTimerInterrupt_Activity

3.68 Peripheral Clock Control (PCC)

3.68.1 Detailed Description

This module covers Peripheral Clock Control module functionality.

PCC HAL provides the API for reading and writing register bit-fields belonging to the PCC module. Clock selection for most modules is controlled by PCC. The clock to each module can be individually gated on and off using PCC.

The PCC hal component allows application to configure these peripheral clocking options:

- Clock gating
- Clock source selection
- Fractional clock divider values

This is an example to write a peripheral clock configuration:

```
Count of peripheral clock user configurations
#define NUM_OF_CONFIGURED_PERIPHERAL_CLOCKS_0 5U

Peripheral clocking configuration 0
peripheral_clock_config_t peripheralClockConfig0[
      NUM_OF_CONFIGURED_PERIPHERAL_CLOCKS_0] = {
    {
        .peripheral       = PCC_FLEXTMR3_CLOCK,
        .clkGate          = true,
        .clkSrc           = CLK_SRC_FIRC,
        .frac             = 0,                                       N/A
        .divider          = 0,                                       N/A
    },
    {
        .peripheral       = PCC_LPTMR0_CLOCK,
        .clkGate          = true,
        .clkSrc           = CLK_SRC_SOSC,
        .frac             = MULTIPLY_BY_TWO,
        .divider          = DIVIDE_BY_FOUR,
    },
    {
        .peripheral       = PCC_SIM0_CLOCK,
        .clkGate          = true,
        .clkSrc           = 0,                                       N/A
        .frac             = 0,                                       N/A
        .divider          = 0,                                       N/A
    },
    {
        .peripheral       = PCC_PORTA_CLOCK,
        .clkGate          = true,
        .clkSrc           = 0,                                       N/A
        .frac             = 0,                                       N/A
        .divider          = 0,                                       N/A
    },
    {
        .peripheral       = PCC_SCG0_CLOCK,
        .clkGate          = true,
        .clkSrc           = 0,                                       N/A
        .frac             = 0,                                       N/A
        .divider          = 0,                                       N/A
    },
};

pcc_config_t pccConfig =
{
    .peripheralClocks = peripheralClockConfig0,                  Peripheral clock control
      configurations
    .count = NUM_OF_CONFIGURED_PERIPHERAL_CLOCKS_0,              Number of the peripheral clock control
      configurations
};

write peripheral clocking configuration
PCC_HAL_SetPeripheralClockConfig(PCC, &pccConfig);
```

For higher-level functionality, use the Clock Manager driver.

Data Structures

- struct peripheral_clock_config_t

  PCC peripheral instance clock configuration. Implements peripheral_clock_config_t_Class. More...
- struct pcc_config_t

  PCC configuration. Implements pcc_config_t_Class. More...

Enumerations

- enum peripheral_clock_source_t {
  CLK_SRC_OFF = 0x00U, CLK_SRC_SOSC = 0x01U, CLK_SRC_SIRC = 0x02U, CLK_SRC_FIRC = 0x03U,
  CLK_SRC_SPLL = 0x06U }

  PCC clock source select Implements peripheral_clock_source_t_Class.
- enum peripheral_clock_frac_t { MULTIPLY_BY_ONE = 0x00U, MULTIPLY_BY_TWO = 0x01U }

  PCC fractional value select Implements peripheral_clock_frac_t_Class.
- enum peripheral_clock_divider_t {
  DIVIDE_BY_ONE = 0x00U, DIVIDE_BY_TWO = 0x01U, DIVIDE_BY_THREE = 0x02U, DIVIDE_BY_FOUR = 0x03U,
  DIVIDE_BY_FIVE = 0x04U, DIVIDE_BY_SIX = 0x05U, DIVIDE_BY_SEVEN = 0x06U, DIVIDE_BY_EIGTH = 0x07U }

  PCC divider value select Implements peripheral_clock_divider_t_Class.

Variables

- const uint16_t clockNameMappings [CLOCK_NAME_COUNT]

  Clock name mappings Constant array storing the mappings between clock names and peripheral clock control indexes. If there is no peripheral clock control index for a clock name, then the corresponding value is PCC_INVALI↩ D_INDEX.

System Clock.

- void PCC_HAL_SetPeripheralClockConfig (PCC_Type ∗const base, const pcc_config_t ∗const config)

  Set the peripheral clock configuration.
- static void PCC_HAL_SetClockMode (PCC_Type ∗const base, const clock_names_t clockName, const bool isClockEnabled)

  Enables/disables the clock for a given peripheral. For example, to enable the ADC0 clock, use like this:
- static void PCC_HAL_SetClockSourceSel (PCC_Type ∗const base, const clock_names_t clockName, const peripheral_clock_source_t clockSource)

  Selects the clock source for a given peripheral For example, to select the FIRC source for ADC clock, use like this:
- static void PCC_HAL_SetFracValueSel (PCC_Type ∗const base, const clock_names_t clockName, const peripheral_clock_frac_t fracValue)

  Selects the fractional value for a given peripheral For example, to configure MULTIPLY_BY_ONE as fractional value for WDOG, use like this:
- static void PCC_HAL_SetDividerValueSel (PCC_Type ∗const base, const clock_names_t clockName, const peripheral_clock_divider_t divValue)

  Selects the divider value for a given peripheral For example, to configure DIVIDE_BY_ONE as divider value for W↩ DOG, use like this:
- static bool PCC_HAL_GetClockMode (const PCC_Type ∗const base, const clock_names_t clockName)

  Gets the clock gate control mode.
- static peripheral_clock_source_t PCC_HAL_GetClockSourceSel (const PCC_Type ∗const base, const clock_names_t clockName)

Gets the selection of a clock source for a specific peripheral.
- static peripheral_clock_frac_t PCC_HAL_GetFracValueSel (const PCC_Type ∗const base, const clock_↩ names_t clockName)

    Gets the selection of the fractional value for a specific peripheral.
- static peripheral_clock_divider_t PCC_HAL_GetDividerSel (const PCC_Type ∗const base, const clock_↩ names_t clockName)

    Gets the selection of the divider value for a specific peripheral.
- static bool PCC_HAL_GetPeripheralMode (const PCC_Type ∗const base, const clock_names_t clockName)

    Tells whether a given peripheral is present or not.

3.68.2 Data Structure Documentation

3.68.2.1 struct peripheral_clock_config_t

PCC peripheral instance clock configuration. Implements peripheral_clock_config_t_Class.

Data Fields

- clock_names_t clockName
- bool clkGate
- peripheral_clock_source_t clkSrc
- peripheral_clock_frac_t frac
- peripheral_clock_divider_t divider

Field Documentation

3.68.2.1.1 clkGate

```
bool clkGate
```

Peripheral clock gate.

3.68.2.1.2 clkSrc

```
peripheral_clock_source_t clkSrc
```

Peripheral clock source.

3.68.2.1.3 clockName

```
clock_names_t clockName
```

3.68.2.1.4 divider

```
peripheral_clock_divider_t divider
```

Peripheral clock divider value.

3.68.2.1.5 frac

```
peripheral_clock_frac_t frac
```

Peripheral clock fractional value.

3.68.2.2 struct pcc_config_t

PCC configuration. Implements pcc_config_t_Class.

Data Fields

- uint32_t count
- peripheral_clock_config_t ∗ peripheralClocks

Field Documentation

3.68.2.2.1 count

```
uint32_t count
```

Number of peripherals to be configured.

3.68.2.2.2 peripheralClocks

```
peripheral_clock_config_t* peripheralClocks
```

Pointer to the peripheral clock configurations array.

3.68.3 Enumeration Type Documentation

3.68.3.1 peripheral_clock_divider_t

```
enum peripheral_clock_divider_t
```

PCC divider value select Implements peripheral_clock_divider_t_Class.

Enumerator

DIVIDE_BY_ONE	
DIVIDE_BY_TWO	
DIVIDE_BY_THREE	
DIVIDE_BY_FOUR	
DIVIDE_BY_FIVE	
DIVIDE_BY_SIX	
DIVIDE_BY_SEVEN	
DIVIDE_BY_EIGTH	

3.68.3.2 peripheral_clock_frac_t

```
enum peripheral_clock_frac_t
```

PCC fractional value select Implements peripheral_clock_frac_t_Class.

Enumerator

MULTIPLY_BY_ONE	
MULTIPLY_BY_TWO	

3.68.3.3 peripheral_clock_source_t

```
enum peripheral_clock_source_t
```

PCC clock source select Implements peripheral_clock_source_t_Class.

Enumerator

CLK_SRC_OFF	
CLK_SRC_SOSC	
CLK_SRC_SIRC	
CLK_SRC_FIRC	
CLK_SRC_SPLL	

3.68.4 Function Documentation

3.68.4.1 PCC_HAL_GetClockMode()

```
static bool PCC_HAL_GetClockMode (
            const PCC_Type *const base,
            const clock_names_t clockName )  [inline], [static]
```

Gets the clock gate control mode.

Parameters

in	*base*	pcc base pointer
in	*clockName*	is the name of the peripheral clock must be one of the following values (see the clock_names_t type from S32K144_clock_names.h) PCC_DMA0_CLOCK PCC_MPU0_CLOCK ... PCC_LPUART3_CLOCK

Returns

the clock gate control mode

- false : Clock is disabled
- true : Clock is enabled Implements PCC_HAL_GetClockMode_Activity

3.68.4.2 PCC_HAL_GetClockSourceSel()

```
static peripheral_clock_source_t PCC_HAL_GetClockSourceSel (
            const PCC_Type *const base,
            const clock_names_t clockName )  [inline], [static]
```

Gets the selection of a clock source for a specific peripheral.

Parameters

in	*base*	pcc base pointer
in	*clockName*	is the name of the peripheral clock must be one of the following values (see the clock_names_t type from S32K144_clock_names.h) PCC_DMA0_CLOCK PCC_MPU0_CLOCK ... PCC_LPUART3_CLOCK

Returns

the clock gate control mode

- PCC_CLK_SRC_OFF : Clock is disabled
- PCC_CLK_SRC_SOSC : OSCCLK - System Oscillator Bus Clock
- PCC_CLK_SRC_SIRC : SCGIRCLK - Slow IRC Clock
- PCC_CLK_SRC_FIRC : SCGFIRCLK - Fast IRC Clock
- PCC_CLK_SRC_SPLL : SCGPCLK - System PLL clock Implements PCC_HAL_GetClockSourceSel↩_Activity

3.68.4.3 PCC_HAL_GetDividerSel()

```
static peripheral_clock_divider_t PCC_HAL_GetDividerSel (
            const PCC_Type *const base,
            const clock_names_t clockName )  [inline], [static]
```

Gets the selection of the divider value for a specific peripheral.

Parameters

in	*base*	pcc base pointer
in	*clockName*	is the name of the peripheral clock must be one of the following values (see the clock_names_t type from S32K144_clock_names.h) PCC_DMA0_CLOCK PCC_MPU0_CLOCK ... PCC_LPUART3_CLOCK

Returns

the divider value

- PCC_DIVIDE_BY_ONE : Divide by 1
- PCC_DIVIDE_BY_TWO : Divide by 2
- PCC_DIVIDE_BY_THREE : Divide by 3
- PCC_DIVIDE_BY_FOUR : Divide by 4
- PCC_DIVIDE_BY_FIVE : Divide by 5
- PCC_DIVIDE_BY_SIX : Divide by 6
- PCC_DIVIDE_BY_SEVEN : Divide by 7
- PCC_DIVIDE_BY_EIGTH : Divide by 8 Implements PCC_HAL_GetDividerSel_Activity

3.68.4.4 PCC_HAL_GetFracValueSel()

```
static peripheral_clock_frac_t PCC_HAL_GetFracValueSel (
            const PCC_Type *const base,
            const clock_names_t clockName )  [inline], [static]
```

Gets the selection of the fractional value for a specific peripheral.

Parameters

in	*base*	pcc base pointer
in	*clockName*	is the name of the peripheral clock must be one of the following values (see the clock_names_t type from S32K144_clock_names.h) PCC_DMA0_CLOCK PCC_MPU0_CLOCK ... PCC_LPUART3_CLOCK

Returns

the fractional value

- PCC_MULTPCCnLY_BY_ONE : Fractional value is zero
- PCC_MULTPCCnLY_BY_TWO : Fractional value is one Implements PCC_HAL_GetFracValueSel_←Activity

3.68.4.5 PCC_HAL_GetPeripheralMode()

```
static bool PCC_HAL_GetPeripheralMode (
            const PCC_Type *const base,
            const clock_names_t clockName )  [inline], [static]
```

Tells whether a given peripheral is present or not.

Parameters

in	*base*	pcc base pointer
in	*clockName*	is the name of the peripheral clock must be one of the following values (see the clock_names_t type from S32K144_clock_names.h) PCC_DMA0_CLOCK PCC_MPU0_CLOCK ... PCC_LPUART3_CLOCK

Returns

is the value that tells whether the peripheral is present or not

- false : Peripheral is not present
- true : Peripheral is present Implements PCC_HAL_GetPeripheralMode_Activity

3.68.4.6 PCC_HAL_SetClockMode()

```
static void PCC_HAL_SetClockMode (
            PCC_Type *const base,
            const clock_names_t clockName,
            const bool isClockEnabled )  [inline], [static]
```

Enables/disables the clock for a given peripheral. For example, to enable the ADC0 clock, use like this:

```
PCC_HAL_SetClockMode(PCC, PCC_ADC0_CLOCK, true);
```

Parameters

in	*base*	pcc base pointer
in	*clockName*	is the name of the peripheral clock must be one of the following values (see the clock_names_t type from S32K144_clock_names.h) PCC_DMA0_CLOCK PCC_MPU0_CLOCK ... PCC_LPUART3_CLOCK
in	*isClockEnabled*	is the value of the command that enables/disables the clock Implements PCC_HAL_SetClockMode_Activity

3.68.4.7 PCC_HAL_SetClockSourceSel()

```
static void PCC_HAL_SetClockSourceSel (
            PCC_Type *const base,
            const clock_names_t clockName,
            const peripheral_clock_source_t clockSource )  [inline], [static]
```

Selects the clock source for a given peripheral For example, to select the FIRC source for ADC clock, use like this:

```
PCC_HAL_SetClockSourceSel(PCC, PCC_ADC0_CLOCK,
      CLK_SRC_FIRC);
```

Parameters

in	*base*	pcc base pointer
in	*clockName*	is the name of the peripheral clock must be one of the following values (see the clock_names_t type from S32K144_clock_names.h) PCC_DMA0_CLOCK PCC_MPU0_CLOCK ... PCC_LPUART3_CLOCK
in	*clockSource*	is the clock source to use for a given peripheral Implements PCC_HAL_SetClockSourceSel_Activity

3.68.4.8 PCC_HAL_SetDividerValueSel()

```
static void PCC_HAL_SetDividerValueSel (
            PCC_Type *const base,
            const clock_names_t clockName,
            const peripheral_clock_divider_t divValue )  [inline], [static]
```

Selects the divider value for a given peripheral For example, to configure DIVIDE_BY_ONE as divider value for WDOG, use like this:

```
PCC_HAL_SetDividerValueSel(PCC, PCC_WDOG_CLOCK,
      DIVIDE_BY_ONE);
```

Parameters

in	*base*	pcc base pointer
in	*clockName*	is the name of the peripheral clock must be one of the following values (see the clock_names_t type from S32K144_clock_names.h) PCC_DMA0_CLOCK PCC_MPU0_CLOCK ... PCC_LPUART3_CLOCK
in	*divValue*	is the divider value to use for a given peripheral Implements PCC_HAL_SetDividerValueSel_Activity

3.68.4.9 PCC_HAL_SetFracValueSel()

```
static void PCC_HAL_SetFracValueSel (
            PCC_Type *const base,
            const clock_names_t clockName,
            const peripheral_clock_frac_t fracValue )  [inline], [static]
```

Selects the fractional value for a given peripheral For example, to configure MULTIPLY_BY_ONE as fractional value for WDOG, use like this:

```
PCC_HAL_SetFracValueSel(PCC, PCC_WDOG_CLOCK,
     MULTIPLY_BY_ONE);
```

Parameters

in	*base*	pcc base pointer
in	*clockName*	is the name of the peripheral clock must be one of the following values (see the clock_names_t type from S32K144_clock_names.h) PCC_DMA0_CLOCK PCC_MPU0_CLOCK ... PCC_LPUART3_CLOCK
in	*fracValue*	is the fractional value to use for a given peripheral Implements PCC_HAL_SetFracValueSel_Activity

3.68.4.10 PCC_HAL_SetPeripheralClockConfig()

```
void PCC_HAL_SetPeripheralClockConfig (
            PCC_Type *const base,
            const pcc_config_t *const config )
```

Set the peripheral clock configuration.

This function sets the peripheral configuration.

Parameters

in	*base*	Register base address for the PCC instance.
in	*config*	Pointer to the configuration.

3.68.5 Variable Documentation

3.68.5.1 clockNameMappings

```
const uint16_t clockNameMappings[CLOCK_NAME_COUNT]
```

Clock name mappings Constant array storing the mappings between clock names and peripheral clock control indexes. If there is no peripheral clock control index for a clock name, then the corresponding value is PCC_INV↩ ALID_INDEX.

3.69 Pins

3.69.1 Detailed Description

The S32 SDK Pins Driver provides a set of API/services to configure the Port Control and Interrupt IP: PORT.

Hardware background

The Port Control and Interrupt (PORT) module provides support for port control, digital filtering, 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. There is one instance of the PORT module for each port. Not all pins within each port are implemented on a specific device. The PORT module has the following features:

- Pin interrupt
- Digital input filter
- Port control

Driver consideration

The Pins driver is developed on top of the PORT HAL. Pins Driver is designed for:

- Configuration of pin routing/muxing
- Assignment of signal names to pins (customize pin names)
- Configuration of pin functional/electrical properties

The graphical user interface for routing functionality consists of two main views: Collapsed View and Pins View.

The Collapsed view is designed for pin routing configuration. The Pin/Signal Selection column displays the list of pins that can be configured for selected Function or Signal. If not configured, "No pin routed" is displayed, which means no user requirement for configuration. You can see the popup menu for setting on/off automatic value. The Direction column contains the list of directions that can be configured for selected function or signal, if direction configuration is supported. For fixed direction signals, the direction text is displayed and grayed-out.

The Pins view allows the configuration of electrical properties of pins and displays all pins. It displays the pad configuration that is available in Processor view where each pin is associated with signal name and function. If you want to view specific pin, use Pin Filter. It filters the User Pin/Signal Name column as shown below. You can also sort the first two columns, Pin and User Pin/Signal Name.

Functional Properties view allows configuration of functional properties on pins. The user graphical interface is mapped over the port control features.

The Driver uses structures for configuration. The user application can use the default for most settings, changing only what is necessary. All port control features are part of data structure configuration.Please see pin_settings_config_t.

This is an example to write a pin configuration:

```
User number of configured pins
#define NUM_OF_CONFIGURED_PINS 1

Array of pin configuration structures
pin_settings_config_t g_pin_mux_InitConfigArr[NUM_OF_CONFIGURED_PINS] = {
  {
          .base           = PORTE,
          .pinPortIdx     = 16u,
          .pullConfig     = PORT_INTERNAL_PULL_UP_ENABLED,
          .passiveFilter = false,
          .driveSelect    = PORT_LOW_DRIVE_STRENGTH,
          .mux            = PORT_MUX_AS_GPIO,
          .pinLock        = false,
          .intConfig      = PORT_DMA_INT_DISABLED,
          .clearIntFlag  = false,
  }
};

write pins configuration
PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);
```

Modules

- Port Control and Interrupts (PORT)

    This module covers the functionality of the PORT peripheral.

Data Structures

- struct pin_settings_config_t

    Defines the converter configuration. More...

Enumerations

- enum port_data_direction_t { GPIO_INPUT_DIRECTION = 0x0U, GPIO_OUTPUT_DIRECTION = 0x1U, G←PIO_UNSPECIFIED_DIRECTION = 0x2U }

    Configures the port data direction Implements : port_data_direction_t_Class.

Pins DRV.

- status_t PINS_DRV_Init (const uint32_t pin_count, const pin_settings_config_t config[ ])

    Initializes the pins with the given configuration structure.

3.69.2 Data Structure Documentation

3.69.2.1 struct pin_settings_config_t

Defines the converter configuration.

This structure is used to configure the pins Implements : pin_settings_config_t_Class

Data Fields

- PORT_Type ∗ base
- uint32_t pinPortIdx
- port_pull_config_t pullConfig
- bool passiveFilter
- port_drive_strength_t driveSelect
- port_mux_t mux

    Configures the drive strength.
- bool pinLock

    Pin mux selection.
- port_interrupt_config_t intConfig
- bool clearIntFlag
- GPIO_Type ∗ gpioBase
- port_data_direction_t direction

Field Documentation

3.69.2.1.1 base

```
PORT_Type* base
```

Port base pointer.

3.69.2.1.2 clearIntFlag

`bool clearIntFlag`

Interrupt generation condition.

3.69.2.1.3 direction

`port_data_direction_t direction`

3.69.2.1.4 driveSelect

`port_drive_strength_t driveSelect`

3.69.2.1.5 gpioBase

`GPIO_Type* gpioBase`

Clears the interrupt status flag. GPIO base pointer.

3.69.2.1.6 intConfig

`port_interrupt_config_t intConfig`

Lock pin control register or not.

3.69.2.1.7 mux

`port_mux_t mux`

Configures the drive strength.

3.69.2.1.8 passiveFilter

`bool passiveFilter`

Passive filter configuration.

3.69.2.1.9 pinLock

`bool pinLock`

Pin mux selection.

3.69.2.1.10 pinPortIdx

`uint32_t pinPortIdx`

Port pin number.

3.69.2.1.11 pullConfig

`port_pull_config_t pullConfig`

Internal resistor pull feature selection.

3.69.3 Enumeration Type Documentation

3.69.3.1 port_data_direction_t

`enum port_data_direction_t`

Configures the port data direction Implements : port_data_direction_t_Class.

Enumerator

GPIO_INPUT_DIRECTION	General purpose input direction.
GPIO_OUTPUT_DIRECTION	General purpose output direction.
GPIO_UNSPECIFIED_DIRECTION	General purpose unspecified direction.

3.69.4 Function Documentation

3.69.4.1 PINS_DRV_Init()

```
status_t PINS_DRV_Init (
            const uint32_t pin_count,
            const pin_settings_config_t config[] )
```

Initializes the pins with the given configuration structure.

This function configures the pins with the options provided in the provided structure.

Parameters

in	*pin_count*	the number of configured pins in structure
in	*config*	the configuration structure

Returns

the status of the operation

3.70 Port Control and Interrupts (PORT)

3.70.1 Detailed Description

This module covers the functionality of the PORT peripheral.

PORT HAL provides the API for reading and writing register bit-fields belonging to the PORT module.

Enumerations

- enum port_pull_config_t { PORT_INTERNAL_PULL_NOT_ENABLED = 0U, PORT_INTERNAL_PULL_D↩ OWN_ENABLED = 1U, PORT_INTERNAL_PULL_UP_ENABLED = 2U }

  Internal resistor pull feature selection Implements : port_pull_config_t_Class.
- enum port_drive_strength_t { PORT_LOW_DRIVE_STRENGTH = 0U, PORT_HIGH_DRIVE_STRENGTH = 1U }

  Configures the drive strength. Implements : port_drive_strength_t_Class.
- enum port_mux_t {
  PORT_PIN_DISABLED = 0U, PORT_MUX_AS_GPIO = 1U, PORT_MUX_ALT2 = 2U, PORT_MUX_ALT3 = 3U,
  PORT_MUX_ALT4 = 4U, PORT_MUX_ALT5 = 5U, PORT_MUX_ALT6 = 6U, PORT_MUX_ALT7 = 7U }

  Pin mux selection Implements : port_mux_t_Class.
- enum port_digital_filter_clock_source_t { PORT_BUS_CLOCK = 0U, PORT_LPO_CLOCK = 1U }

  Digital filter clock source selection Implements : port_digital_filter_clock_source_t_Class.
- enum port_interrupt_config_t {
  PORT_DMA_INT_DISABLED = 0x0U, PORT_DMA_RISING_EDGE = 0x1U, PORT_DMA_FALLING_EDGE = 0x2U, PORT_DMA_EITHER_EDGE = 0x3U,
  PORT_INT_LOGIC_ZERO = 0x8U, PORT_INT_RISING_EDGE = 0x9U, PORT_INT_FALLING_EDGE = 0xAU, PORT_INT_EITHER_EDGE = 0xBU,
  PORT_INT_LOGIC_ONE = 0xCU }

  Configures the interrupt generation condition. Implements : port_interrupt_config_t_Class.

Configuration

- static void PORT_HAL_SetPullSel (PORT_Type ∗const base, const uint32_t pin, const port_pull_config_t pullConfig)

  Configures the internal resistor.
- static void PORT_HAL_SetPassiveFilterMode (PORT_Type ∗const base, const uint32_t pin, const bool is↩ PassiveFilterEnabled)

  Configures the passive filter if the pin is used as a digital input.
- static void PORT_HAL_SetDriveStrengthMode (PORT_Type ∗const base, const uint32_t pin, const port_↩ drive_strength_t driveSelect)

  Configures the drive strength if the pin is used as a digital output.
- static void PORT_HAL_SetMuxModeSel (PORT_Type ∗const base, const uint32_t pin, const port_mux_t mux)

  Configures the pin muxing.
- static void PORT_HAL_SetPinCtrlLockMode (PORT_Type ∗const base, const uint32_t pin, const bool is↩ PinLockEnabled)

  Locks or unlocks the pin control register bits[15:0].
- static void PORT_HAL_SetDigitalFilterMode (PORT_Type ∗const base, const uint32_t pin, const bool is↩ DigitalFilterEnabled)

  Enables or disables the digital filter in one single port. Each bit of the 32-bit register represents one pin.

- static void PORT_HAL_SetDigitalFilterClock (PORT_Type ∗const base, const port_digital_filter_clock_↩ source_t clockSource)

  Configures the clock source for the digital input filters. Changing the filter clock source should only be done after disabling all enabled filters. Every pin in one port uses the same clock source.
- static void PORT_HAL_SetDigitalFilterWidth (PORT_Type ∗const base, const uint8_t width)

  Configures the maximum size of the glitches (in clock cycles) that the digital filter absorbs for enabled digital filters. Glitches that are longer than this register setting (in clock cycles) pass through the digital filter, while glitches that are equal to or less than this register setting (in clock cycles) are filtered. Changing the filter length should only be done after disabling all enabled filters.
- void PORT_HAL_SetLowGlobalPinCtrlCmd (PORT_Type ∗const base, const uint16_t lowPinSelect, const uint16_t config)

  Configures the low half of the pin control register for the same settings. This function operates pin 0 -15 of one specific port.
- void PORT_HAL_SetHighGlobalPinCtrlCmd (PORT_Type ∗const base, const uint16_t highPinSelect, const uint16_t config)

  Configures the high half of pin control register for the same settings. This function operates pin 16 -31 of one specific port.

Interrupt

- static void PORT_HAL_SetPinIntSel (PORT_Type ∗const base, const uint32_t pin, const port_interrupt_↩ config_t intConfig)

  Configures the port pin interrupt/DMA request.
- static port_interrupt_config_t PORT_HAL_GetPinIntSel (const PORT_Type ∗const base, const uint32_t pin)

  Gets the current port pin interrupt/DMA request configuration.
- static bool PORT_HAL_GetPinIntMode (const PORT_Type ∗const base, const uint32_t pin)

  Reads the individual pin-interrupt status flag.
- static void PORT_HAL_ClearPinIntFlagCmd (PORT_Type ∗const base, const uint32_t pin)

  Clears the individual pin-interrupt status flag.
- static uint32_t PORT_HAL_GetPortIntFlag (const PORT_Type ∗const base)

  Reads the entire port interrupt status flag.
- static void PORT_HAL_ClearPortIntFlagCmd (PORT_Type ∗const base)

  Clears the entire port interrupt status flag.

3.70.2 Enumeration Type Documentation

3.70.2.1 port_digital_filter_clock_source_t

```
enum port_digital_filter_clock_source_t
```

Digital filter clock source selection Implements : port_digital_filter_clock_source_t_Class.

Enumerator

PORT_BUS_CLOCK	Digital filters are clocked by the bus clock.
PORT_LPO_CLOCK	Digital filters are clocked by the 1 kHz LPO clock.

3.70.2.2 port_drive_strength_t

```
enum port_drive_strength_t
```

Configures the drive strength. Implements : port_drive_strength_t_Class.

Enumerator

PORT_LOW_DRIVE_STRENGTH	low drive strength is configured.
PORT_HIGH_DRIVE_STRENGTH	high drive strength is configured.

3.70.2.3 port_interrupt_config_t

```
enum port_interrupt_config_t
```

Configures the interrupt generation condition. Implements : port_interrupt_config_t_Class.

Enumerator

PORT_DMA_INT_DISABLED	Interrupt/DMA request is disabled.
PORT_DMA_RISING_EDGE	DMA request on rising edge.
PORT_DMA_FALLING_EDGE	DMA request on falling edge.
PORT_DMA_EITHER_EDGE	DMA request on either edge.
PORT_INT_LOGIC_ZERO	Interrupt when logic zero.
PORT_INT_RISING_EDGE	Interrupt on rising edge.
PORT_INT_FALLING_EDGE	Interrupt on falling edge.
PORT_INT_EITHER_EDGE	Interrupt on either edge.
PORT_INT_LOGIC_ONE	Interrupt when logic one.

3.70.2.4 port_mux_t

```
enum port_mux_t
```

Pin mux selection Implements : port_mux_t_Class.

Enumerator

PORT_PIN_DISABLED	corresponding pin is disabled, but is used as an analog pin.
PORT_MUX_AS_GPIO	corresponding pin is configured as GPIO.
PORT_MUX_ALT2	chip-specific
PORT_MUX_ALT3	chip-specific
PORT_MUX_ALT4	chip-specific
PORT_MUX_ALT5	chip-specific
PORT_MUX_ALT6	chip-specific
PORT_MUX_ALT7	chip-specific

3.70.2.5 port_pull_config_t

```
enum port_pull_config_t
```

Internal resistor pull feature selection Implements : port_pull_config_t_Class.

Enumerator

PORT_INTERNAL_PULL_NOT_ENABLED	internal pull-down or pull-up resistor is not enabled.
PORT_INTERNAL_PULL_DOWN_ENABLED	internal pull-down resistor is enabled.
PORT_INTERNAL_PULL_UP_ENABLED	internal pull-up resistor is enabled.

3.70.3 Function Documentation

3.70.3.1 PORT_HAL_ClearPinIntFlagCmd()

```
static void PORT_HAL_ClearPinIntFlagCmd (
            PORT_Type *const base,
            const uint32_t pin )  [inline], [static]
```

Clears the individual pin-interrupt status flag.

Parameters

in	*base*	port base pointer
in	*pin*	port pin number Implements : PORT_HAL_ClearPinIntFlagCmd_Activity

3.70.3.2 PORT_HAL_ClearPortIntFlagCmd()

```
static void PORT_HAL_ClearPortIntFlagCmd (
            PORT_Type *const base )  [inline], [static]
```

Clears the entire port interrupt status flag.

Parameters

in	*base*	port base pointer Implements : PORT_HAL_ClearPortIntFlagCmd_Activity

3.70.3.3 PORT_HAL_GetPinIntMode()

```
static bool PORT_HAL_GetPinIntMode (
            const PORT_Type *const base,
            const uint32_t pin )  [inline], [static]
```

Reads the individual pin-interrupt status flag.

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

in	*base*	port base pointer
in	*pin*	port pin number

Returns

current pin interrupt status flag

- 0: interrupt is not detected.
- 1: interrupt is detected. Implements : PORT_HAL_GetPinIntMode_Activity

3.70.3.4 PORT_HAL_GetPinIntSel()

```
static port_interrupt_config_t PORT_HAL_GetPinIntSel (
            const PORT_Type *const base,
            const uint32_t pin )  [inline], [static]
```

Gets the current port pin interrupt/DMA request configuration.

Parameters

in	*base*	port base pointer
in	*pin*	port pin number

Returns

interrupt configuration

- PORT_INT_DISABLED : Interrupt/DMA request disabled.
- PORT_DMA_RISING_EDGE : DMA request on rising edge.
- PORT_DMA_FALLING_EDGE: DMA request on falling edge.
- PORT_DMA_EITHER_EDGE : DMA request on either edge.
- PORT_INT_LOGIC_ZERO : Interrupt when logic zero.
- PORT_INT_RISING_EDGE : Interrupt on rising edge.
- PORT_INT_FALLING_EDGE: Interrupt on falling edge.
- PORT_INT_EITHER_EDGE : Interrupt on either edge.
- PORT_INT_LOGIC_ONE : Interrupt when logic one. Implements : PORT_HAL_GetPinIntSel_Activity

3.70.3.5 PORT_HAL_GetPortIntFlag()

```
static uint32_t PORT_HAL_GetPortIntFlag (
            const PORT_Type *const base )  [inline], [static]
```

Reads the entire port interrupt status flag.

Parameters

in	*base*	port base pointer

Returns

all 32 pin interrupt status flags. For specific bit:

- 0: interrupt is not detected.
- 1: interrupt is detected. Implements : PORT_HAL_GetPortIntFlag_Activity

3.70.3.6 PORT_HAL_SetDigitalFilterClock()

```
static void PORT_HAL_SetDigitalFilterClock (
            PORT_Type *const base,
            const port_digital_filter_clock_source_t clockSource )  [inline], [static]
```

Configures the clock source for the digital input filters. Changing the filter clock source should only be done after disabling all enabled filters. Every pin in one port uses the same clock source.

Parameters

in	*base*	port base pointer
in	*clockSource*	chose which clock source to use for current port • PORT_BUS_CLOCK: digital filters are clocked by the bus clock. • PORT_LPO_CLOCK: digital filters are clocked by the 1 kHz LPO clock. Implements : PORT_HAL_SetDigitalFilterClock_Activity

3.70.3.7 PORT_HAL_SetDigitalFilterMode()

```
static void PORT_HAL_SetDigitalFilterMode (
            PORT_Type *const base,
            const uint32_t pin,
            const bool isDigitalFilterEnabled )  [inline], [static]
```

Enables or disables the digital filter in one single port. Each bit of the 32-bit register represents one pin.

Parameters

in	*base*	port base pointer
in	*pin*	port pin number
in	*isDigitalFilterEnabled*	digital filter enable/disable • false: digital filter is disabled on the corresponding pin. • true : digital filter is enabled on the corresponding pin. Implements : PORT_HAL_SetDigitalFilterMode_Activity

3.70.3.8 PORT_HAL_SetDigitalFilterWidth()

```
static void PORT_HAL_SetDigitalFilterWidth (
            PORT_Type *const base,
            const uint8_t width )  [inline], [static]
```

Configures the maximum size of the glitches (in clock cycles) that the digital filter absorbs for enabled digital filters. Glitches that are longer than this register setting (in clock cycles) pass through the digital filter, while glitches that are equal to or less than this register setting (in clock cycles) are filtered. Changing the filter length should only be done after disabling all enabled filters.

Parameters

in	*base*	port base pointer
in	*width*	configure digital filter width (should be less than 5 bits). Implements : PORT_HAL_SetDigitalFilterWidth_Activity

3.70.3.9 PORT_HAL_SetDriveStrengthMode()

```
static void PORT_HAL_SetDriveStrengthMode (
             PORT_Type *const base,
             const uint32_t pin,
             const port_drive_strength_t driveSelect )  [inline], [static]
```

Configures the drive strength if the pin is used as a digital output.

Parameters

in	*base*	port base pointer
in	*pin*	port pin number
in	*driveSelect*	drive strength selection • PORT_LOW_DRIVE_STRENGTH : low drive strength is configured. • PORT_HIGH_DRIVE_STRENGTH: high drive strength is configured. Implements : PORT_HAL_SetDriveStrengthMode_Activity

3.70.3.10 PORT_HAL_SetHighGlobalPinCtrlCmd()

```
void PORT_HAL_SetHighGlobalPinCtrlCmd (
             PORT_Type *const base,
             const uint16_t highPinSelect,
             const uint16_t config )
```

Configures the high half of pin control register for the same settings. This function operates pin 16 -31 of one specific port.

Parameters

in	*base*	port base pointer
in	*highPinSelect*	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.
in	*config*	value is written to a high half port control register bits[15:0].

3.70.3.11 PORT_HAL_SetLowGlobalPinCtrlCmd()

```
void PORT_HAL_SetLowGlobalPinCtrlCmd (
             PORT_Type *const base,
             const uint16_t lowPinSelect,
             const uint16_t config )
```

Configures the low half of the pin control register for the same settings. This function operates pin 0 -15 of one specific port.

Parameters

in	*base*	port base pointer
in	*lowPinSelect*	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.
in	*config*	value is written to a low half port control register bits[15:0].

3.70.3.12 PORT_HAL_SetMuxModeSel()

```
static void PORT_HAL_SetMuxModeSel (
            PORT_Type *const base,
            const uint32_t pin,
            const port_mux_t mux )  [inline], [static]
```

Configures the pin muxing.

Parameters

in	*base*	port base pointer
in	*pin*	port pin number
in	*mux*	pin muxing slot selection • PORT_PIN_DISABLED: Pin disabled. • PORT_MUX_AS_GPIO : Set as GPIO. • others : chip-specific. Implements : PORT_HAL_SetMuxModeSel_Activity

3.70.3.13 PORT_HAL_SetPassiveFilterMode()

```
static void PORT_HAL_SetPassiveFilterMode (
            PORT_Type *const base,
            const uint32_t pin,
            const bool isPassiveFilterEnabled )  [inline], [static]
```

Configures the passive filter if the pin is used as a digital input.

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

in	*base*	port base pointer
in	*pin*	port pin number
in	*isPassiveFilterEnabled*	passive filter configuration • false: passive filter is disabled. • true : passive filter is enabled. Implements : PORT_HAL_SetPassiveFilterMode_Activity

3.70.3.14 PORT_HAL_SetPinCtrlLockMode()

```
static void PORT_HAL_SetPinCtrlLockMode (
            PORT_Type *const base,
            const uint32_t pin,
            const bool isPinLockEnabled )  [inline], [static]
```

Locks or unlocks the pin control register bits[15:0].

Parameters

in	*base*	port base pointer
in	*pin*	port pin number
in	*isPinLockEnabled*	lock pin control register or not • false: pin control register bit[15:0] are not locked. • true : pin control register bit[15:0] are locked, cannot be updated till system reset. Implements : PORT_HAL_SetPinCtrlLockMode_Activity

3.70.3.15 PORT_HAL_SetPinIntSel()

```
static void PORT_HAL_SetPinIntSel (
            PORT_Type *const base,
            const uint32_t pin,
            const port_interrupt_config_t intConfig )  [inline], [static]
```

Configures the port pin interrupt/DMA request.

Parameters

in	*base*	port base pointer.
in	*pin*	port pin number
in	*intConfig*	interrupt configuration • PORT_INT_DISABLED : Interrupt/DMA request disabled. • PORT_DMA_RISING_EDGE : DMA request on rising edge. • PORT_DMA_FALLING_EDGE: DMA request on falling edge. • PORT_DMA_EITHER_EDGE : DMA request on either edge. • PORT_INT_LOGIC_ZERO : Interrupt when logic zero. • PORT_INT_RISING_EDGE : Interrupt on rising edge. • PORT_INT_FALLING_EDGE: Interrupt on falling edge. • PORT_INT_EITHER_EDGE : Interrupt on either edge. • PORT_INT_LOGIC_ONE : Interrupt when logic one. Implements : PORT_HAL_SetPinIntSel_Activity

3.70.3.16 PORT_HAL_SetPullSel()

```
static void PORT_HAL_SetPullSel (
            PORT_Type *const base,
            const uint32_t pin,
            const port_pull_config_t pullConfig )  [inline], [static]
```

Configures the internal resistor.

Pull configuration is valid in all digital pin muxing modes.

Parameters

in	*base*	port base pointer.
in	*pin*	port pin number
in	*pullConfig*	internal resistor pull feature selection • PORT_PULL_NOT_ENABLED: internal pull-down or pull-up resistor is not enabled. • PORT_PULL_DOWN_ENABLED: internal pull-down resistor is enabled. • PORT_PULL_UP_ENABLED: internal pull-up resistor is enabled. Implements : PORT_HAL_SetPullSel_Activity

3.71 Power Management Controller (PMC)

3.71.1 Detailed Description

This module covers the functionality of the Power Management Controller (PMC) peripheral.

PMC HAL provides the API for reading and writing register bit-fields belonging to the PMC module.

Data Structures

- struct pmc_lpo_clock_config_t

  PMC LPO configuration. More...
- struct pmc_config_t

  PMC configure structure. More...

Enumerations

- enum pmc_int_select_t { PMC_INT_LOW_VOLT_DETECT, PMC_INT_LOW_VOLT_WARN }

  Power management control interrupts Implements pmc_int_select_t_Class.

Power Management Controller Control APIs

- void PMC_HAL_SetLowVoltIntCmd (PMC_Type ∗const baseAddr, const pmc_int_select_t intSelect, const bool enable)

  Enables/Disables the low voltage-related interrupts.
- void PMC_HAL_SetLowVoltIntAckCmd (PMC_Type ∗const baseAddr, const pmc_int_select_t intSelect)

  Acknowledges the low voltage-related interrupts.
- bool PMC_HAL_GetLowVoltIntFlag (const PMC_Type ∗const baseAddr, const pmc_int_select_t intSelect)

  Gets the flag for the low voltage-related interrupts.
- static void PMC_HAL_SetLowVoltDetectResetCmd (PMC_Type ∗const baseAddr, const bool enable)

  Low-Voltage Detect Hardware Reset Enable/Disable (write once)
- static void PMC_HAL_SetBiasMode (PMC_Type ∗const baseAddr, const bool enable)

  Enables/Disables the Bias.
- static void PMC_HAL_SetLpoMode (PMC_Type ∗const baseAddr, const bool enable)

  Enables/Disables the Low Power Oscillator.
- static bool PMC_HAL_GetLpoMode (const PMC_Type ∗const baseAddr)

  Gets the Low Power Oscillator status.
- static uint8_t PMC_HAL_GetRegulatorStatus (const PMC_Type ∗const baseAddr)

  Gets the Regulator regulation status.
- static uint8_t PMC_HAL_GetLpoStatus (const PMC_Type ∗const baseAddr)

  Gets the Low Power Oscillator status.
- static void PMC_HAL_SetLpoTrimValue (PMC_Type ∗const baseAddr, const int8_t decimalValue)

  Low Power Oscillator Trimming Value.

3.71.2 Data Structure Documentation

3.71.2.1 struct pmc_lpo_clock_config_t

PMC LPO configuration.

Data Fields

- bool initialize
- bool enable
- int8_t trimValue

Field Documentation

3.71.2.1.1 enable

```
bool enable
```

Enable/disable LPO

3.71.2.1.2 initialize

```
bool initialize
```

Initialize or not the PMC LPO settings.

3.71.2.1.3 trimValue

```
int8_t trimValue
```

LPO trimming value

3.71.2.2 struct pmc_config_t

PMC configure structure.

Data Fields

- pmc_lpo_clock_config_t lpoClockConfig

Field Documentation

3.71.2.2.1 lpoClockConfig

```
pmc_lpo_clock_config_t lpoClockConfig
```

Low Power Clock configuration.

3.71.3 Enumeration Type Documentation

3.71.3.1 pmc_int_select_t

```
enum pmc_int_select_t
```

Power management control interrupts Implements pmc_int_select_t_Class.

Enumerator

PMC_INT_LOW_VOLT_DETECT	Low Voltage Detect Interrupt
PMC_INT_LOW_VOLT_WARN	Low Voltage Warning Interrupt

3.71.4 Function Documentation

3.71.4.1 PMC_HAL_GetLowVoltIntFlag()

```
bool PMC_HAL_GetLowVoltIntFlag (
             const PMC_Type *const baseAddr,
             const pmc_int_select_t intSelect )
```

Gets the flag for the low voltage-related interrupts.

This function gets the flag for the low voltage detection, warning, etc. interrupts

Parameters

in	*baseAddr*	Base address for current PMC instance.
in	*intSelect*	interrupt select

Returns

status Current low voltage interrupt flag

- true: Low-Voltage Interrupt flag is set
- false: Low-Voltage Interrupt flag is not set

3.71.4.2 PMC_HAL_GetLpoMode()

```
static bool PMC_HAL_GetLpoMode (
             const PMC_Type *const baseAddr ) [inline], [static]
```

Gets the Low Power Oscillator status.

This function gets the Low Power Oscillator status.

Parameters

in	*baseAddr*	Base address for current PMC instance.

Returns

value LPO status false - LPO is disabled true - LPO is enabled Implements PMC_HAL_GetLpoMode_Activity

3.71.4.3 PMC_HAL_GetLpoStatus()

```
static uint8_t PMC_HAL_GetLpoStatus (
             const PMC_Type *const baseAddr ) [inline], [static]
```

Gets the Low Power Oscillator status.

This function provides the current status of the Low Power Oscillator.

Parameters

in	*baseAddr*	Base address for current PMC instance.

Returns

value LPO status 0 - Low power oscillator in low phase. 1 - Low power oscillator in high phase. Implements PMC_HAL_GetLpoStatus_Activity

3.71.4.4 PMC_HAL_GetRegulatorStatus()

```
static uint8_t PMC_HAL_GetRegulatorStatus (
            const PMC_Type *const baseAddr )  [inline], [static]
```

Gets the Regulator regulation status.

This function provides the current status of the internal voltage regulator.

Parameters

in	*baseAddr*	Base address for current PMC instance.

Returns

value Regulation status 0 - Regulator is in low power mode or transition to/from. 1 - Regulator is in full performance mode. Implements PMC_HAL_GetRegulatorStatus_Activity

3.71.4.5 PMC_HAL_SetBiasMode()

```
static void PMC_HAL_SetBiasMode (
            PMC_Type *const baseAddr,
            const bool enable )  [inline], [static]
```

Enables/Disables the Bias.

This function enables/disables the Bias.

Parameters

in	*baseAddr*	Base address for current PMC instance.
in	*enable*	enable/disable the Bias. Implements PMC_HAL_SetBiasMode_Activity

3.71.4.6 PMC_HAL_SetLowVoltDetectResetCmd()

```
static void PMC_HAL_SetLowVoltDetectResetCmd (
            PMC_Type *const baseAddr,
            const bool enable )  [inline], [static]
```

Low-Voltage Detect Hardware Reset Enable/Disable (write once)

This function enables/disables 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. Any additional writes are ignored.

Parameters

in	*baseAddr*	Base address for current PMC instance.
in	*enable*	enable/disable the LVD hardware reset Implements PMC_HAL_SetLowVoltDetectResetCmd_Activity

3.71.4.7 PMC_HAL_SetLowVoltIntAckCmd()

```
void PMC_HAL_SetLowVoltIntAckCmd (
            PMC_Type *const baseAddr,
            const pmc_int_select_t intSelect )
```

Acknowledges the low voltage-related interrupts.

This function acknowledges the low voltage detection, warning, etc. interrupts

Parameters

in	*baseAddr*	Base address for current PMC instance.
in	*intSelect*	interrupt select

3.71.4.8 PMC_HAL_SetLowVoltIntCmd()

```
void PMC_HAL_SetLowVoltIntCmd (
            PMC_Type *const baseAddr,
            const pmc_int_select_t intSelect,
            const bool enable )
```

Enables/Disables the low voltage-related interrupts.

This function enables the low voltage detection, warning, etc. interrupts.

Parameters

in	*baseAddr*	Base address for current PMC instance.
in	*intSelect*	interrupt select
in	*enable*	enable/disable the interrupt

3.71.4.9 PMC_HAL_SetLpoMode()

```
static void PMC_HAL_SetLpoMode (
            PMC_Type *const baseAddr,
            const bool enable ) [inline], [static]
```

Enables/Disables the Low Power Oscillator.

This function enables/disables the Low Power Oscillator.

Parameters

in	*baseAddr*	Base address for current PMC instance.
in	*enable*	enable/disable the Low Power Oscillator. Implements PMC_HAL_SetLpoMode_Activity

3.71.4.10 PMC_HAL_SetLpoTrimValue()

```
static void PMC_HAL_SetLpoTrimValue (
             PMC_Type *const baseAddr,
             const int8_t decimalValue )  [inline], [static]
```

Low Power Oscillator Trimming Value.

This function sets the trimming value for the low power oscillator

Parameters

in	*baseAddr*	Base address for current PMC instance.
in	*value*	Trimming value Implements PMC_HAL_SetLpoTrimValue_Activity

3.72 Power Manager

3.72.1 Detailed Description

The S32 SDK Power Manager provides a set of API/services that enables applications to configure and select among various operational and low power modes.

Driver consideration

The Power Manager driver is developed on top of an appropriate HAL (SMC, MC_ME etc). The Power Manager provides API to handle the device power modes. It also supports run-time switching between multiple power modes. Each power mode is described by configuration structures with multiple power-related options. The Power Manager provides a notification mechanism for registered callbacks and API for static and dynamic callback registration.

The Driver uses structures for configuration. The actual format of the structure is defined by the underlying HAL that is being used (SMC or MC_ME). There is a power mode and a callback configuration structure. These structures may be generated using Processor Expert. The user application can use the default for most settings, changing only what is necessary.

This driver provides functions for initializing power manager and changing the power mode.

All methods that access the hardware layer will return an error code to signal if the operation succeeded or failed. The values are defined by the status_t enumeration, and the possible values include: success, switch error, callback notification errors, wrong clock setup error.

Hardware background

System mode controller (SMC) is passing the system into and out of all low-power Stop and Run modes. Controls the power, clocks and memories of the system to achieve the power consumption and functionality of that mode.

Driver consideration

Power mode entry and sleep-on-exit option are provided at initialization time through the power manager user configuration structure. The available power mode entries are the following ones: HSRUN, RUN, VLPR, WAIT, VLPW, VLPS, PSTOP1 and PSTOP2

This is an example of configuration:

```
power_manager_user_config_t pwrMan1_InitConfig0 = {
    .powerMode = POWER_MANAGER_HSRUN,
    .sleepOnExitOption = false,
    .sleepOnExitValue = false,
};

power_manager_user_config_t *powerConfigsArr[] = {
    &pwrMan1_InitConfig0
};

power_manager_callback_user_config_t * powerCallbacksConfigsArr[] = {(
      void *)0};

if (STATUS_SUCCESS != POWER_SYS_Init(&powerConfigsArr,1,&powerCallbacksConfigsArr,0)) {
    ...
}
else {
    ...
}

if (STATUS_SUCCESS != POWER_SYS_SetMode(0,
      POWER_MANAGER_POLICY_AGREEMENT)) {
    ...
}
else {
    ...
}
```

Modules

- Power Management Controller (PMC)

  This module covers the functionality of the Power Management Controller (PMC) peripheral.
- Reset Control Module (RCM)

  This module covers the functionality of the Reset Control Module (RCM) peripheral.
- System Mode Controller (SMC)

  This module covers the functionality of the System Mode Controller (SMC) peripheral.

Data Structures

- struct power_manager_notify_struct_t

  Power mode user configuration structure. More...
- struct power_manager_callback_user_config_t

  callback configuration structure More...
- struct power_manager_state_t

  Power manager internal state structure. More...

Macros

- #define POWER_SET_MODE_TIMEOUT 1000U

Typedefs

- typedef void power_manager_callback_data_t

  Callback-specific data.
- typedef status_t(∗ power_manager_callback_t) (power_manager_notify_struct_t ∗notify, power_manager_↩ callback_data_t ∗dataPtr)

  Callback prototype.

Enumerations

- enum power_manager_policy_t { POWER_MANAGER_POLICY_AGREEMENT, POWER_MANAGER_P↩ OLICY_FORCIBLE }

  Power manager policies.
- enum power_manager_notify_t { POWER_MANAGER_NOTIFY_RECOVER = 0x00U, POWER_MANAGE↩ R_NOTIFY_BEFORE = 0x01U, POWER_MANAGER_NOTIFY_AFTER = 0x02U }

  The PM notification type. Used to notify registered callbacks. Callback notifications can be invoked in following situations:
- enum power_manager_callback_type_t { POWER_MANAGER_CALLBACK_BEFORE = 0x01U, POWER↩ _MANAGER_CALLBACK_AFTER = 0x02U, POWER_MANAGER_CALLBACK_BEFORE_AFTER = 0x03U }

  The callback type indicates when a callback will be invoked.

Functions

- status_t POWER_SYS_Init (power_manager_user_config_t ∗(∗powerConfigsPtr)[ ], uint8_t configsNumber, power_manager_callback_user_config_t ∗(∗callbacksPtr)[ ], uint8_t callbacksNumber)

  Power manager initialization for operation.
- status_t POWER_SYS_Deinit (void)

  This function deinitializes the Power manager.
- status_t POWER_SYS_SetMode (uint8_t powerModeIndex, power_manager_policy_t policy)

  This function configures the power mode.
- status_t POWER_SYS_GetLastMode (uint8_t ∗powerModeIndexPtr)

  This function returns the last successfully set power mode.
- status_t POWER_SYS_GetLastModeConfig (power_manager_user_config_t ∗∗powerModePtr)

  This function returns the user configuration structure of the last successfully set power mode.
- power_manager_modes_t POWER_SYS_GetCurrentMode (void)

  This function returns currently running power mode.
- uint8_t POWER_SYS_GetErrorCallbackIndex (void)

  This function returns the last failed notification callback.
- power_manager_callback_user_config_t ∗ POWER_SYS_GetErrorCallback (void)

  This function returns the callback configuration structure for the last failed notification.
- status_t POWER_SYS_DoInit (void)
- status_t POWER_SYS_DoDeinit (void)
- status_t POWER_SYS_DoSetMode (const power_manager_user_config_t ∗const configPtr)
- static status_t POWER_SYS_WaitForModeStatus (smc_run_mode_t mode)
- static status_t POWER_SYS_SwitchToSleepingPowerMode (const power_manager_user_config_t ∗const configPtr)
- static status_t POWER_SYS_SwitchToRunningPowerMode (const power_manager_user_config_t ∗const configPtr)

Variables

- power_manager_state_t gPowerManagerState

  Power manager internal structure.

3.72.2 Data Structure Documentation

3.72.2.1 struct power_manager_notify_struct_t

Power mode user configuration structure.

This structure defines power mode with additional power options. This structure is implementation-defiend. Please refer to actual definition based on the underlying HAL (SMC, MC_ME etc). Applications may define multiple power modes and switch between them. A list of all defined power modes is passed to the Power manager during initialization as an array of references to structures of this type (see POWER_SYS_Init()). Power modes can be switched by calling POWER_SYS_SetMode(), which takes as argument the index of the reqested power mode in the list passed during manager initialization. The power mode currently in use can be retrieved by calling POWER_SYS_GetLast←Mode(), which provides the index of the current power mode, or by calling POWER_SYS_GetLastModeConfig(), which provides a pointer to the configuration structure of the current power mode. The members of the power mode configuration structure depend on power options available for a specific chip, and includes at least the power mode. The available power modes are chip-specific. See power_manager_modes_t defined in the underlying HAL for a list of all supported modes.

Power notification structure passed to registered callback function

Implements power_manager_notify_struct_t_Class

Data Fields

- power_manager_user_config_t ∗ targetPowerConfigPtr
- uint8_t targetPowerConfigIndex
- power_manager_policy_t policy
- power_manager_notify_t notifyType

Field Documentation

3.72.2.1.1 notifyType

```
power_manager_notify_t notifyType
```

Power mode notification type.

3.72.2.1.2 policy

```
power_manager_policy_t policy
```

Power mode transition policy.

3.72.2.1.3 targetPowerConfigIndex

```
uint8_t targetPowerConfigIndex
```

Target power configuration index.

3.72.2.1.4 targetPowerConfigPtr

```
power_manager_user_config_t* targetPowerConfigPtr
```

Pointer to target power configuration

3.72.2.2 struct power_manager_callback_user_config_t

callback configuration structure

This structure holds configuration of callbacks passed to the Power manager during its initialization. Structures of this type are expected to be statically allocated. This structure contains following application-defined data: callback - pointer to the callback function callbackType - specifies when the callback is called callbackData - pointer to the data passed to the callback Implements power_manager_callback_user_config_t_Class

Data Fields

- power_manager_callback_t callbackFunction
- power_manager_callback_type_t callbackType
- power_manager_callback_data_t ∗ callbackData

Field Documentation

3.72.2.2.1 callbackData

```
power_manager_callback_data_t* callbackData
```

3.72.2.2.2 callbackFunction

```
power_manager_callback_t callbackFunction
```

3.72.2.2.3 callbackType

```
power_manager_callback_type_t callbackType
```

3.72.2.3 struct power_manager_state_t

Power manager internal state structure.

Power manager internal structure. Contains data necessary for Power manager proper functionality. Stores references to registered power mode configurations, callbacks, and other internal data. This structure is statically allocated and initialized by POWER_SYS_Init(). Implements power_manager_state_t_Class

Data Fields

- power_manager_user_config_t ∗(∗ configs)[ ]
- uint8_t configsNumber
- power_manager_callback_user_config_t ∗(∗ staticCallbacks)[ ]
- uint8_t staticCallbacksNumber
- uint8_t errorCallbackIndex
- uint8_t currentConfig

Field Documentation

3.72.2.3.1 configs

```
power_manager_user_config_t∗(∗ configs)[]
```

Pointer to power configure table.

3.72.2.3.2 configsNumber

```
uint8_t configsNumber
```

Number of power configurations

3.72.2.3.3 currentConfig

```
uint8_t currentConfig
```

Index of current configuration.

3.72.2.3.4 errorCallbackIndex

```
uint8_t errorCallbackIndex
```

Index of callback returns error.

3.72.2.3.5 staticCallbacks

```
power_manager_callback_user_config_t*(* staticCallbacks)[]
```

Pointer to callback table.

3.72.2.3.6 staticCallbacksNumber

```
uint8_t staticCallbacksNumber
```

Max. number of callback configurations

3.72.3 Macro Definition Documentation

3.72.3.1 POWER_SET_MODE_TIMEOUT

```
#define POWER_SET_MODE_TIMEOUT 1000U
```

Timeout used for waiting to set new mode

3.72.4 Typedef Documentation

3.72.4.1 power_manager_callback_data_t

```
typedef void power_manager_callback_data_t
```

Callback-specific data.

Pointer to data of this type is passed during callback registration. The pointer is part of the power_manager_↩
callback_user_config_t structure and is passed to the callback during power mode change notifications. Implements power_manager_callback_data_t_Class

3.72.4.2 power_manager_callback_t

```
typedef status_t(* power_manager_callback_t) (power_manager_notify_struct_t *notify, power_↩
manager_callback_data_t *dataPtr)
```

Callback prototype.

Declaration of callback. It is common for all registered callbacks. Function pointer of this type is part of power↩
_manager_callback_user_config_t callback configuration structure. Depending on the callback type, the callback function is invoked during power mode change (see POWER_SYS_SetMode()) before the mode change, after it, or in both cases to notify about the change progress (see power_manager_callback_type_t). When called, the type of the notification is passed as parameter along with a pointer to power mode configuration structure (see power↩
_manager_notify_struct_t) and any data passed during the callback registration (see power_manager_callback_↩
data_t). When notified before a mode change, depending on the power mode change policy (see power_manager↩
_policy_t) the callback may deny the mode change by returning any error code other than STATUS_SUCCESS (see POWER_SYS_SetMode()).

Parameters

notify	Notification structure.
dataPtr	Callback data. Pointer to the data passed during callback registration. Intended to pass any driver or application data such as internal state information.

Returns

An error code or STATUS_SUCCESS. Implements power_manager_callback_t_Class

3.72.5 Enumeration Type Documentation

3.72.5.1 power_manager_callback_type_t

```
enum power_manager_callback_type_t
```

The callback type indicates when a callback will be invoked.

Used in the callback configuration structures (power_manager_callback_user_config_t) to specify when the registered callback will be called during power mode change initiated by POWER_SYS_SetMode().

Implements power_manager_callback_type_t_Class

Enumerator

POWER_MANAGER_CALLBACK_BEFORE	Before callback.
POWER_MANAGER_CALLBACK_AFTER	After callback.
POWER_MANAGER_CALLBACK_BEFORE_AFTER	Before-After callback.

3.72.5.2 power_manager_notify_t

```
enum power_manager_notify_t
```

The PM notification type. Used to notify registered callbacks. Callback notifications can be invoked in following situations:

- before a power mode change (Callback return value can affect POWER_SYS_SetMode() execution. Refer to the POWER_SYS_SetMode() and power_manager_policy_t documentation).
- after a successful change of the power mode.
- after an unsuccessful attempt to switch power mode, in order to recover to a working state. Implements power_manager_notify_t_Class

Enumerator

POWER_MANAGER_NOTIFY_RECOVER	Notify IP to recover to previous work state.
POWER_MANAGER_NOTIFY_BEFORE	Notify IP that the system will change the power setting.
POWER_MANAGER_NOTIFY_AFTER	Notify IP that the system has changed to a new power setting.

3.72.5.3 power_manager_policy_t

```
enum power_manager_policy_t
```

Power manager policies.

Defines whether the mode switch initiated by the POWER_SYS_SetMode() is agreed upon (depending on the result of notification callbacks), or forced. For POWER_MANAGER_POLICY_FORCIBLE the power mode is changed regardless of the callback results, while for POWER_MANAGER_POLICY_AGREEMENT policy any error code returned by one of the callbacks aborts the mode change. See also POWER_SYS_SetMode() description. Implements power_manager_policy_t_Class

Enumerator

POWER_MANAGER_POLICY_AGREEMENT	Power mode is changed if all of the callbacks return success.
POWER_MANAGER_POLICY_FORCIBLE	Power mode is changed regardless of the result of callbacks.

3.72.6 Function Documentation

3.72.6.1 POWER_SYS_Deinit()

```
status_t POWER_SYS_Deinit (
             void  )
```

This function deinitializes the Power manager.

Returns

An error code or STATUS_SUCCESS.

3.72.6.2 POWER_SYS_DoDeinit()

```
status_t POWER_SYS_DoDeinit (
             void  )
```

3.72.6.3 POWER_SYS_DoInit()

```
status_t POWER_SYS_DoInit (
             void  )
```

3.72.6.4 POWER_SYS_DoSetMode()

```
status_t POWER_SYS_DoSetMode (
             const power_manager_user_config_t *const configPtr )
```

3.72.6.5 POWER_SYS_GetCurrentMode()

```
power_manager_modes_t POWER_SYS_GetCurrentMode (
            void )
```

This function returns currently running power mode.

This function reads hardware settings and returns currently running power mode.

Returns

Currently used run power mode.

3.72.6.6 POWER_SYS_GetErrorCallback()

```
power_manager_callback_user_config_t* POWER_SYS_GetErrorCallback (
            void )
```

This function returns the callback configuration structure for the last failed notification.

This function returns a pointer to configuration structure of the last callback that failed during the power mode switch when POWER_SYS_SetMode() was called. If the last POWER_SYS_SetMode() call ended successfully, a NULL value is returned.

Returns

Pointer to the callback configuration which returns error.

3.72.6.7 POWER_SYS_GetErrorCallbackIndex()

```
uint8_t POWER_SYS_GetErrorCallbackIndex (
            void )
```

This function returns the last failed notification callback.

This function returns the index of the last callback that failed during the power mode switch when POWER_SY↩ S_SetMode() was called. The returned value represents the index in the array of registered callbacks. If the last POWER_SYS_SetMode() call ended successfully, a value equal to the number of registered callbacks is returned.

Returns

Callback index of last failed callback or value equal to callbacks count.

3.72.6.8 POWER_SYS_GetLastMode()

```
status_t POWER_SYS_GetLastMode (
            uint8_t * powerModeIndexPtr )
```

This function returns the last successfully set power mode.

This function returns index of power mode which was last set using POWER_SYS_SetMode(). If the power mode was entered even though some of the registered callbacks denied the mode change, or if any of the callbacks invoked after the entering/restoring run mode failed, then the return code of this function has STATUS_ERROR value.

Parameters

out	*powerModeIndexPtr*	Power mode which has been set represented as an index into array of power mode configurations passed to the POWER_SYS_Init().

Returns

An error code or STATUS_SUCCESS.

3.72.6.9 POWER_SYS_GetLastModeConfig()

```
status_t POWER_SYS_GetLastModeConfig (
            power_manager_user_config_t ** powerModePtr )
```

This function returns the user configuration structure of the last successfully set power mode.

This function returns a pointer to configuration structure which was last set using POWER_SYS_SetMode(). If the current power mode was entered even though some of the registered callbacks denied the mode change, or if any of the callbacks invoked after the entering/restoring run mode failed, then the return code of this function has STATUS_ERROR value.

Parameters

out	*powerModePtr*	Pointer to power mode configuration structure of the last set power mode.

Returns

An error code or STATUS_SUCCESS.

3.72.6.10 POWER_SYS_Init()

```
status_t POWER_SYS_Init (
            power_manager_user_config_t *(*) powerConfigsPtr[ ],
            uint8_t configsNumber,
            power_manager_callback_user_config_t *(*) callbacksPtr[ ],
            uint8_t callbacksNumber )
```

Power manager initialization for operation.

This function initializes the Power manager and its run-time state structure. Pointer to an array of Power mode configuration structures needs to be passed as a parameter along with a parameter specifying its size. At least one power mode configuration is required. Optionally, pointer to the array of predefined callbacks can be passed with its corresponding size parameter. For details about callbacks, refer to the power_manager_callback_user_config_t. As Power manager stores only pointers to arrays of these structures, they need to exist and be valid for the entire life cycle of Power manager.

Parameters

in	*powerConfigsPtr*	A pointer to an array of pointers to all power configurations which will be handled by Power manager.
in	*configsNumber*	Number of power configurations. Size of powerConfigsPtr array.
in	*callbacksPtr*	A pointer to an array of pointers to callback configurations. If there are no callbacks to register during Power manager initialization, use NULL value.
in	*callbacksNumber*	Number of registered callbacks. Size of callbacksPtr array.

Returns

An error code or STATUS_SUCCESS.

3.72.6.11 POWER_SYS_SetMode()

```
status_t POWER_SYS_SetMode (
             uint8_t powerModeIndex,
             power_manager_policy_t policy )
```

This function configures the power mode.

This function switches to one of the defined power modes. Requested mode number is passed as an input parameter. This function notifies all registered callback functions before the mode change (using POWER_MANAGER_CALLBACK_BEFORE set as callback type parameter), sets specific power options defined in the power mode configuration and enters the specified mode. In case of run modes (for example, Run, Very low power run, or High speed run), this function also invokes all registered callbacks after the mode change (using POWER_MANAGER_CALLBACK_AFTER). In case of sleep or deep sleep modes, if the requested mode is not exited through a reset, these notifications are sent after the core wakes up. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array (see callbacksPtr parameter of POWER_SYS_Init()). The same order is used for before and after switch notifications. The notifications before the power mode switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the power mode change, further execution of this function depends on mode change policy: the mode change is either forced (POWER_MANAGER_POLICY_FORCIBLE) or aborted (POWER_MANAGER_POLICY_AGREEMENT). When mode change is forced, the results of the before switch notifications are ignored. If agreement is requested, in case any callback returns an error code then further before switch notifications are cancelled and all already notified callbacks are re-invoked with POWER_MANAGER_CALLBACK_AFTER set as callback type parameter. The index of the callback which returned error code during pre-switch notifications is stored and can be obtained by using POWER_SYS_GetErrorCallback(). Any error codes during callbacks re-invocation (recover phase) are ignored. POWER_SYS_SetMode() returns an error code denoting the phase in which a callback failed. It is possible to enter any mode supported by the processor. Refer to the chip reference manual for the list of available power modes. If it is necessary to switch into an intermediate power mode prior to entering the requested mode (for example, when switching from Run into Very low power wait through Very low power run mode), then the intermediate mode is entered without invoking the callback mechanism.

Parameters

in	*powerModeIndex*	Requested power mode represented as an index into array of user-defined power mode configurations passed to the POWER_SYS_Init().
in	*policy*	Transaction policy

Returns

An error code or STATUS_SUCCESS.

3.72.6.12 POWER_SYS_SwitchToRunningPowerMode()

```
static status_t POWER_SYS_SwitchToRunningPowerMode (
             const power_manager_user_config_t *const configPtr )  [static]
```

3.72.6.13 POWER_SYS_SwitchToSleepingPowerMode()

```
static status_t POWER_SYS_SwitchToSleepingPowerMode (
             const power_manager_user_config_t *const configPtr )  [static]
```

3.72.6.14 POWER_SYS_WaitForModeStatus()

```
static status_t POWER_SYS_WaitForModeStatus (
             smc_run_mode_t mode )  [static]
```

3.72.7 Variable Documentation

3.72.7.1 gPowerManagerState

```
power_manager_state_t gPowerManagerState
```

Power manager internal structure.

3.73 Programmable Delay Block (PDB)

3.73.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Drivers for the Programmable Delay Block (PDB) module of S32 SDK devices.

The PDB is a configurable counter that can generate events (triggers) that can be used by the ADC to start conversions or routed through TRGMUX to other modules in the S32K144.

Modules

- PDB Driver

  Programmable Delay Block Peripheral Driver.
- PDB HAL

  Programmable Delay Block Hardware Abstraction Layer.

3.74 Real Time Clock Driver

3.74.1 Detailed Description

Real Time Clock Driver Peripheral Driver.

Data Structures

- struct rtc_timedate_t

  RTC Time Date structure Implements : rtc_timedate_t_Class. More...
- struct rtc_init_config_t

  RTC Initialization structure Implements : rtc_init_config_t_Class. More...
- struct rtc_alarm_config_t

  RTC alarm configuration Implements : rtc_alarm_config_t_Class. More...
- struct rtc_interrupt_config_t

  RTC interrupt configuration. It is used to configure interrupt other than Time Alarm and Time Seconds interrupt Implements : rtc_interrupt_config_t_Class. More...
- struct rtc_seconds_int_config_t

  RTC Seconds Interrupt Configuration Implements : rtc_seconds_int_config_t_Class. More...
- struct rtc_register_lock_config_t

  RTC Register Lock Configuration Implements : rtc_register_lock_config_t_Class. More...

Macros

- #define SECONDS_IN_A_DAY (86400UL)
- #define SECONDS_IN_A_HOUR (3600U)
- #define SECONDS_IN_A_MIN (60U)
- #define MINS_IN_A_HOUR (60U)
- #define HOURS_IN_A_DAY (24U)
- #define DAYS_IN_A_YEAR (365U)
- #define DAYS_IN_A_LEAP_YEAR (366U)
- #define YEAR_RANGE_START (1970U)
- #define YEAR_RANGE_END (2099U)

Functions

- status_t RTC_DRV_Init (uint32_t instance, const rtc_init_config_t ∗const rtcUserCfg)

  This function initializes the RTC instance with the settings provided by the user via the rtcUserCfg parameter. The user must ensure that clock is enabled for the RTC instance used. If the Control register is locked then this method returns STATUS_ERROR. In order to clear the CR Lock the user must perform a power-on reset.
- status_t RTC_DRV_Deinit (uint32_t instance)

  This function deinitializes the RTC instance. If the Control register is locked then this method returns STATUS_ER↩ ROR.
- status_t RTC_DRV_StartCounter (uint32_t instance)

  Start RTC instance counter. Before calling this function the user should use RTC_DRV_SetTimeDate to configure the start time.
- status_t RTC_DRV_StopCounter (uint32_t instance)

  Disable RTC instance counter.
- status_t RTC_DRV_GetCurrentTimeDate (uint32_t instance, rtc_timedate_t ∗const currentTime)

  Get current time and date from RTC instance.

- status_t RTC_DRV_SetTimeDate (uint32_t instance, const rtc_timedate_t ∗const time)

  Set time and date for RTC instance. The user must stop the counter before using this function. Otherwise it will return an error.
- status_t RTC_DRV_ConfigureRegisterLock (uint32_t instance, const rtc_register_lock_config_t ∗const lockConfig)

  This method configures register lock for the corresponding RTC instance. Remember that all the registers are unlocked only by software reset or power on reset. (Except for CR that is unlocked only by POR).
- void RTC_DRV_GetRegisterLock (uint32_t instance, rtc_register_lock_config_t ∗const lockConfig)

  Get which registers are locked for RTC instance.
- status_t RTC_DRV_ConfigureTimeCompensation (uint32_t instance, uint8_t compInterval, int8_t compensation)

  This method configures time compensation. Data is passed by the compInterval and compensation parameters. For more details regarding coefficient calculation see the Reference Manual.
- void RTC_DRV_GetTimeCompensation (uint32_t instance, uint8_t ∗compInterval, int8_t ∗compensation)

  This retrieves the time compensation coefficients and saves them on the variables referenced by the parameters.
- void RTC_DRV_ConfigureFaultInt (uint32_t instance, rtc_interrupt_config_t ∗const intConfig)

  This method configures fault interrupts such as:
- void RTC_DRV_ConfigureSecondsInt (uint32_t instance, rtc_seconds_int_config_t ∗const intConfig)

  This method configures the Time Seconds Interrupt with the configuration from the intConfig parameter.
- status_t RTC_DRV_ConfigureAlarm (uint32_t instance, rtc_alarm_config_t ∗const alarmConfig)

  This method configures the alarm with the configuration from the alarmConfig parameter.
- void RTC_DRV_GetAlarmConfig (uint32_t instance, rtc_alarm_config_t ∗alarmConfig)

  Get alarm configuration for RTC instance.
- bool RTC_DRV_IsAlarmPending (uint32_t instance)

  Check if alarm is pending.
- void RTC_DRV_ConvertSecondsToTimeDate (const uint32_t ∗seconds, rtc_timedate_t ∗const timeDate)

  Convert seconds to rtc_timedate_t structure.
- void RTC_DRV_ConvertTimeDateToSeconds (const rtc_timedate_t ∗const timeDate, uint32_t ∗const seconds)

  Convert seconds to rtc_timedate_t structure.
- bool RTC_DRV_IsYearLeap (uint16_t year)

  Check if the current year is leap.
- bool RTC_DRV_IsTimeDateCorrectFormat (const rtc_timedate_t ∗const timeDate)

  Check if the date time struct is configured properly.
- status_t RTC_DRV_GetNextAlarmTime (uint32_t instance, rtc_timedate_t ∗const alarmTime)

  Gets the next alarm time.
- void RTC_DRV_IRQHandler (uint32_t instance)

  This method is the API's Interrupt handler for generic and alarm IRQ. It will handle the alarm repetition and calls the user callbacks if they are not NULL.
- void RTC_DRV_SecondsIRQHandler (uint32_t instance)

  This method is the API's Interrupt handler for RTC Second interrupt. This ISR will call the user callback if defined.

3.74.2 Data Structure Documentation

3.74.2.1 struct rtc_timedate_t

RTC Time Date structure Implements : rtc_timedate_t_Class.

Data Fields

- uint16_t year
- uint16_t month
- uint16_t day
- uint16_t hour
- uint16_t minutes
- uint8_t seconds

Field Documentation

3.74.2.1.1 day

```
uint16_t day
```

Day

3.74.2.1.2 hour

```
uint16_t hour
```

Hour

3.74.2.1.3 minutes

```
uint16_t minutes
```

Minutes

3.74.2.1.4 month

```
uint16_t month
```

Month

3.74.2.1.5 seconds

```
uint8_t seconds
```

Seconds

3.74.2.1.6 year

```
uint16_t year
```

Year

3.74.2.2 struct rtc_init_config_t

RTC Initialization structure Implements : rtc_init_config_t_Class.

Data Fields

- uint8_t compensationInterval
- int8_t compensation
- rtc_clk_select_t clockSelect
- rtc_clk_out_config_t clockOutConfig
- bool updateEnable
- bool nonSupervisorAccessEnable

Field Documentation

3.74.2.2.1 clockOutConfig

```
rtc_clk_out_config_t clockOutConfig
```

RTC Clock Out Source

3.74.2.2.2 clockSelect

```
rtc_clk_select_t clockSelect
```

RTC Clock Select

3.74.2.2.3 compensation

```
int8_t compensation
```

Compensation Value

3.74.2.2.4 compensationInterval

```
uint8_t compensationInterval
```

Compensation Interval

3.74.2.2.5 nonSupervisorAccessEnable

```
bool nonSupervisorAccessEnable
```

Enable writes to the registers in non Supervisor Mode

3.74.2.2.6 updateEnable

```
bool updateEnable
```

Enable changing the Time Counter Enable bit even if the Status register is locked

3.74.2.3 struct rtc_alarm_config_t

RTC alarm configuration Implements : rtc_alarm_config_t_Class.

Data Fields

- rtc_timedate_t alarmTime
- uint32_t repetitionInterval
- uint32_t numberOfRepeats
- bool repeatForever
- bool alarmIntEnable
- void(∗ alarmCallback)(void ∗callbackParam)
- void ∗ callbackParams

Field Documentation

3.74.2.3.1 alarmCallback

```
void(* alarmCallback) (void *callbackParam)
```

Pointer to the user callback method.

3.74.2.3.2 alarmIntEnable

```
bool alarmIntEnable
```

Enable alarm interrupt

3.74.2.3.3 alarmTime

```
rtc_timedate_t alarmTime
```

Alarm time

3.74.2.3.4 callbackParams

```
void* callbackParams
```

Pointer to the callback parameters.

3.74.2.3.5 numberOfRepeats

```
uint32_t numberOfRepeats
```

Number of alarm repeats

3.74.2.3.6 repeatForever

```
bool repeatForever
```

Repeat forever if set, discard number of repeats

3.74.2.3.7 repetitionInterval

```
uint32_t repetitionInterval
```

Interval of repetition in sec

3.74.2.4 struct rtc_interrupt_config_t

RTC interrupt configuration. It is used to configure interrupt other than Time Alarm and Time Seconds interrupt Implements : rtc_interrupt_config_t_Class.

Data Fields

- bool overflowIntEnable
- bool timeInvalidIntEnable
- void(∗ rtcCallback)(void ∗callbackParam)
- void ∗ callbackParams

Field Documentation

3.74.2.4.1 callbackParams

```
void* callbackParams
```

Pointer to the callback parameters.

3.74.2.4.2 overflowIntEnable

```
bool overflowIntEnable
```

Enable Time Overflow Interrupt

3.74.2.4.3 rtcCallback

```
void(* rtcCallback) (void *callbackParam)
```

Pointer to the user callback method.

3.74.2.4.4 timeInvalidIntEnable

```
bool timeInvalidIntEnable
```

Enable Time Invalid Interrupt

3.74.2.5 struct rtc_seconds_int_config_t

RTC Seconds Interrupt Configuration Implements : rtc_seconds_int_config_t_Class.

Data Fields

- rtc_second_int_cfg_t secondIntConfig
- bool secondIntEnable
- void(∗ rtcSecondsCallback)(void ∗callbackParam)
- void ∗ secondsCallbackParams

Field Documentation

3.74.2.5.1 rtcSecondsCallback

```
void(* rtcSecondsCallback) (void *callbackParam)
```

Pointer to the user callback method.

3.74.2.5.2 secondIntConfig

```
rtc_second_int_cfg_t secondIntConfig
```

Seconds Interrupt frequency

3.74.2.5.3 secondIntEnable

```
bool secondIntEnable
```

Seconds Interrupt enable

3.74.2.5.4 secondsCallbackParams

```
void* secondsCallbackParams
```

Pointer to the callback parameters.

3.74.2.6 struct rtc_register_lock_config_t

RTC Register Lock Configuration Implements : rtc_register_lock_config_t_Class.

Data Fields

- bool lockRegisterLock
- bool statusRegisterLock
- bool controlRegisterLock
- bool timeCompensationRegisterLock

Field Documentation

3.74.2.6.1 controlRegisterLock

```
bool controlRegisterLock
```

Lock state of the Control Register

3.74.2.6.2 lockRegisterLock

```
bool lockRegisterLock
```

Lock state of the Lock Register

3.74.2.6.3 statusRegisterLock

```
bool statusRegisterLock
```

Lock state of the Status Register

3.74.2.6.4 timeCompensationRegisterLock

```
bool timeCompensationRegisterLock
```

Lock state of the Time Compensation Register

3.74.3 Macro Definition Documentation

3.74.3.1 DAYS_IN_A_LEAP_YEAR

```
#define DAYS_IN_A_LEAP_YEAR (366U)
```

3.74.3.2 DAYS_IN_A_YEAR

```
#define DAYS_IN_A_YEAR (365U)
```

3.74.3.3 HOURS_IN_A_DAY

```
#define HOURS_IN_A_DAY (24U)
```

3.74.3.4 MINS_IN_A_HOUR

```
#define MINS_IN_A_HOUR (60U)
```

3.74.3.5 SECONDS_IN_A_DAY

```
#define SECONDS_IN_A_DAY (86400UL)
```

3.74.3.6 SECONDS_IN_A_HOUR

```
#define SECONDS_IN_A_HOUR (3600U)
```

3.74.3.7 SECONDS_IN_A_MIN

```
#define SECONDS_IN_A_MIN (60U)
```

3.74.3.8 YEAR_RANGE_END

```
#define YEAR_RANGE_END (2099U)
```

3.74.3.9 YEAR_RANGE_START

```
#define YEAR_RANGE_START (1970U)
```

3.74.4 Function Documentation

3.74.4.1 RTC_DRV_ConfigureAlarm()

```
status_t RTC_DRV_ConfigureAlarm (
             uint32_t instance,
             rtc_alarm_config_t *const alarmConfig )
```

This method configures the alarm with the configuration from the alarmConfig parameter.

Parameters

in	*instance*	The number of the RTC instance used
in	*alarmConfig*	Pointer to the structure which holds the alarm configuration

Returns

STATUS_SUCCESS if the configuration is successful or STATUS_ERROR if the alarm time is invalid.

3.74.4.2 RTC_DRV_ConfigureFaultInt()

```
void RTC_DRV_ConfigureFaultInt (
            uint32_t instance,
            rtc_interrupt_config_t *const intConfig )
```

This method configures fault interrupts such as:

- Time Overflow Interrupt
- Time Invalid Interrupt with the user provided configuration struct intConfig.

Parameters

in	*instance*	The number of the RTC instance used
in	*intConfig*	Pointer to the structure which holds the configuration

Returns

None

3.74.4.3 RTC_DRV_ConfigureRegisterLock()

```
status_t RTC_DRV_ConfigureRegisterLock (
            uint32_t instance,
            const rtc_register_lock_config_t *const lockConfig )
```

This method configures register lock for the corresponding RTC instance. Remember that all the registers are unlocked only by software reset or power on reset. (Except for CR that is unlocked only by POR).

Parameters

in	*instance*	The number of the RTC instance used
in	*lockConfig*	Pointer to the lock configuration structure

Returns

STATUS_SUCCESS if the operation was successful, STATUS_ERROR if the Lock Register is locked.

3.74.4.4 RTC_DRV_ConfigureSecondsInt()

```
void RTC_DRV_ConfigureSecondsInt (
            uint32_t instance,
            rtc_seconds_int_config_t *const intConfig )
```

This method configures the Time Seconds Interrupt with the configuration from the intConfig parameter.

Parameters

in	*instance*	The number of the RTC instance used
in	*intConfig*	Pointer to the structure which holds the configuration

Returns

None

3.74.4.5 RTC_DRV_ConfigureTimeCompensation()

```
status_t RTC_DRV_ConfigureTimeCompensation (
            uint32_t instance,
            uint8_t compInterval,
            int8_t compensation )
```

This method configures time compensation. Data is passed by the compInterval and compensation parameters. For more details regarding coefficient calculation see the Reference Manual.

Parameters

in	*instance*	The number of the RTC instance used
in	*compInterval*	Compensation interval
in	*compensation*	Compensation value

Returns

STATUS_SUCCESS if the operation was successful, STATUS_ERROR if the TC Register is locked.

3.74.4.6 RTC_DRV_ConvertSecondsToTimeDate()

```
void RTC_DRV_ConvertSecondsToTimeDate (
            const uint32_t * seconds,
            rtc_timedate_t *const timeDate )
```

Convert seconds to rtc_timedate_t structure.

Parameters

in	*seconds*	Pointer to the seconds
out	*timeDate*	Pointer to the structure in which to store the result

Returns

None

3.74.4.7 RTC_DRV_ConvertTimeDateToSeconds()

```
void RTC_DRV_ConvertTimeDateToSeconds (
             const rtc_timedate_t *const timeDate,
             uint32_t *const seconds )
```

Convert seconds to rtc_timedate_t structure.

Parameters

in	*timeDate*	Pointer to the source struct
out	*seconds*	Pointer to the variable in which to store the result

Returns

None

3.74.4.8 RTC_DRV_Deinit()

```
status_t RTC_DRV_Deinit (
             uint32_t instance )
```

This function deinitializes the RTC instance. If the Control register is locked then this method returns STATUS_E↩
RROR.

Parameters

in	*instance*	The number of the RTC instance used

Returns

STATUS_SUCCESS if the operation was successful or STATUS_ERROR if Control register is locked.

3.74.4.9 RTC_DRV_GetAlarmConfig()

```
void RTC_DRV_GetAlarmConfig (
             uint32_t instance,
             rtc_alarm_config_t * alarmConfig )
```

Get alarm configuration for RTC instance.

Parameters

in	*instance*	The number of the RTC instance used
out	*alarmConfig*	Pointer to the structure in which to store the alarm configuration

Returns

None

3.74.4.10 RTC_DRV_GetCurrentTimeDate()

```
status_t RTC_DRV_GetCurrentTimeDate (
             uint32_t instance,
             rtc_timedate_t *const currentTime )
```

Get current time and date from RTC instance.

Parameters

in	*instance*	The number of the RTC instance used
out	*currentTime*	Pointer to the variable in which to store the result

Returns

STATUS_SUCCESS if the operation was successful, STATUS_ERROR if there was a problem.

3.74.4.11 RTC_DRV_GetNextAlarmTime()

```
status_t RTC_DRV_GetNextAlarmTime (
             uint32_t instance,
             rtc_timedate_t *const alarmTime )
```

Gets the next alarm time.

Parameters

in	*instance*	The number of the RTC instance used
out	*alarmTime*	Pointer to the variable in which to store the data

Returns

STATUS_SUCCESS if the next alarm time is valid, STATUS_ERROR if there is no new alarm or alarm configuration specified.

3.74.4.12 RTC_DRV_GetRegisterLock()

```
void RTC_DRV_GetRegisterLock (
             uint32_t instance,
             rtc_register_lock_config_t *const lockConfig )
```

Get which registers are locked for RTC instance.

Parameters

in	*instance*	The number of the RTC instance used
out	*lockConfig*	Pointer to the lock configuration structure in which to save the data

Returns

None

3.74.4.13 RTC_DRV_GetTimeCompensation()

```
void RTC_DRV_GetTimeCompensation (
             uint32_t instance,
             uint8_t * compInterval,
             int8_t * compensation )
```

This retrieves the time compensation coefficients and saves them on the variables referenced by the parameters.

Parameters

in	*instance*	The number of the RTC instance used
out	*compInterval*	Pointer to the variable in which to save the compensation interval
out	*compensation*	Pointer to the variable in which to save the compensation value

Returns

None

3.74.4.14 RTC_DRV_Init()

```
status_t RTC_DRV_Init (
             uint32_t instance,
             const rtc_init_config_t *const rtcUserCfg )
```

This function initializes the RTC instance with the settings provided by the user via the rtcUserCfg parameter. The user must ensure that clock is enabled for the RTC instance used. If the Control register is locked then this method returns STATUS_ERROR. In order to clear the CR Lock the user must perform a power-on reset.

Parameters

in	*instance*	The number of the RTC instance used
in	*rtcUserCfg*	Pointer to the user's configuration structure

Returns

STATUS_SUCCESS if the operation was successful, STATUS_ERROR if Control is locked.

3.74.4.15 RTC_DRV_IRQHandler()

```
void RTC_DRV_IRQHandler (
             uint32_t instance )
```

This method is the API's Interrupt handler for generic and alarm IRQ. It will handle the alarm repetition and calls the user callbacks if they are not NULL.

Parameters

in	*instance*	RTC instance used

Returns

None

3.74.4.16 RTC_DRV_IsAlarmPending()

```
bool RTC_DRV_IsAlarmPending (
            uint32_t instance )
```

Check if alarm is pending.

Parameters

in	*instance*	The number of the RTC instance used

Returns

True if the alarm has occurred, false if not

3.74.4.17 RTC_DRV_IsTimeDateCorrectFormat()

```
bool RTC_DRV_IsTimeDateCorrectFormat (
            const rtc_timedate_t *const timeDate )
```

Check if the date time struct is configured properly.

Parameters

in	*timeDate*	Structure to check to check

Returns

True if the time date is in the correct format, false if not

3.74.4.18 RTC_DRV_IsYearLeap()

```
bool RTC_DRV_IsYearLeap (
            uint16_t year )
```

Check if the current year is leap.

Parameters

in	*year*	Year to check

Returns

True if the year is leap, false if not

3.74.4.19 RTC_DRV_SecondsIRQHandler()

```
void RTC_DRV_SecondsIRQHandler (
             uint32_t instance )
```

This method is the API's Interrupt handler for RTC Second interrupt. This ISR will call the user callback if defined.

Parameters

in	*instance*	RTC instance used

Returns

None

3.74.4.20 RTC_DRV_SetTimeDate()

```
status_t RTC_DRV_SetTimeDate (
             uint32_t instance,
             const rtc_timedate_t *const time )
```

Set time and date for RTC instance. The user must stop the counter before using this function. Otherwise it will return an error.

Parameters

in	*instance*	The number of the RTC instance used
in	*time*	Pointer to the variable in which the time is stored

Returns

STATUS_SUCCESS if the operation was successful, STATUS_ERROR if the time provided was invalid or if the counter was not stopped.

3.74.4.21 RTC_DRV_StartCounter()

```
status_t RTC_DRV_StartCounter (
             uint32_t instance )
```

Start RTC instance counter. Before calling this function the user should use RTC_DRV_SetTimeDate to configure the start time.

Parameters

in	*instance*	The number of the RTC instance used

Returns

STATUS_SUCCESS if the operation was successful, STATUS_ERROR if the counter cannot be enabled or is already enabled.

3.74.4.22 RTC_DRV_StopCounter()

```
status_t RTC_DRV_StopCounter (
            uint32_t instance )
```

Disable RTC instance counter.

Parameters

in	*instance*	The number of the RTC instance used

Returns

STATUS_SUCCESS if the operation was successful, STATUS_ERROR if the counter could not be stopped.

3.75 Real Time Clock Driver (RTC)

3.75.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Driver for the Real Time Clock (RTC) module of S32 SDK devices. .

Hardware background

The Real Time Clock Module is a independent timer that keeps track of the exact date and time with no software overhead, with low power usage.

Features of the RTC module include:

- 32-bit seconds counter with roll-over protection and 32-bit alarm
- 16-bit prescaler with compensation that can correct errors between 0.12 ppm and 3906 ppm
- Option to increment prescaler using the LPO (prescaler increments by 32 every clock edge)
- Register write protection
- Lock register requires POR or software reset to enable write access
- Configurable 1, 2, 4, 8, 16, 32, 64 or 128 Hz square wave output with optional interrupt
- Alarm interrupt configured by the driver automatically refreshes alarm time configured by the user
- User interrupt handlers can be configured for all interrupts

How to use the RTC driver in your application

In order to be able to use the RTC in your application, the first thing to do is initializing it with the desired configuration. This is done by calling the **RTC_DRV_Init** function. One of the arguments passed to this function is the configuration which will be used for the RTC instance, specified by the **rtc_init_config_t** structure.

The **rtc_init_config_t** structure allows you to configure the following:

- RTC clock source (32 KHz clock or 1 KHz LPO clock)
- Clock Out pin configuration (Clock OUT pin source)
- Compensation (Interval and value)
- Update enable - this allows updates to Time Counter Enable bit if the Status Register under limited conditions
- Enable non supervisor writes to the registers

The **rtc_seconds_int_config_t** structure configures the **time seconds interrupt**. To setup an interrupt every seconds you have to configure the structure mentioned with the following parameters:

- Frequency of the interrupt
- Interrupt Handler
- If needed - interrupt handler parameters

An alarm is configured with **rtc_alarm_config_t** structure, which is described by the following parameters:

- Alarm time in date-time format
- Interval of alarm repeat in seconds
- Number of alarm repeats (use 0 if the alarm is not recursive)
- Repeat forever field (if set, the number of repeats field will be ignored)
- Alarm interrupt enable
- Alarm interrupt handler
- Alarm interrupt handler parameters

Note If the alarm interrupt is not enabled, the user must make the updates of the alarm time manually.

After the RTC_DRV_Init call and, if needed, alarm and other configurations the RTC counter is started by calling RTC_DRV_Enable, with start time as parameter in **rtc_timedate_t** format.

To get the current time and date you can call RTC_DRV_GetCurrentTimeDate function, this method will get the seconds from the Time Seconds Register and will convert into human readable format as rtc_timedate_t.

Example

```
void rtcAlarmCallback(void)
{
    rtc_timedate_t currentTime;
    RTC_DRV_GetCurrentTimeDate(0U, &currentTime);

    /* Do something with the time and date */
}

int main()
{
    /* rtcTimer1 configuration structure */
    const rtc_init_config_t rtcTimer1_Config0 =
    {
        /* Time compensation interval */
        .compensationInterval       =   0U,
        /* Time compensation value */
        .compensation               =   0,
        /* RTC Clock Source is 32 KHz crystal */
        .clockSelect                =   RTC_CLK_SRC_OSC_32KHZ,
        /* RTC Clock Out is 32 KHz clock */
        .clockOutConfig             =   RTC_CLKOUT_SRC_32KHZ,
        /* Update of the TCE bit is not allowed */
        .updateEnable               =   false,
        /* Non-supervisor mode write accesses are not supported and generate
         * a bus error.
         */
        .nonSupervisorAccessEnable  =   false
    };

    /* RTC Initial Time and Date */
    rtc_timedate_t       rtcStartTime =
    {
        /* Year */
        .year        =   2016U,
        /* Month */
        .month       =   01U,
        /* Day */
        .day         =   01U,
        /* Hour */
        .hour        =   00U,
        /* Minutes */
        .minutes     =   00U,
        /* Seconds */
        .seconds     =   00U
    };

    /* rtcTimer1 Alarm configuration 0 */
    rtc_alarm_config_t   alarmConfig0 =
    {
        /* Alarm Date */
        .alarmTime             =
            {
                /* Year    */
                .year       =  2016U,
                /* Month    */
                .month      =  01U,
                /* Day      */
                .day        =  01U,
                /* Hour     */
                .hour       =  00U,
                /* Minutes */
                .minutes    =  00U,
                /* Seconds */
                .seconds    =  03U,
            },

        /* Alarm repeat interval */
        .repetitionInterval  =       3UL,

        /* Number of alarm repeats */
        .numberOfRepeats     =       0UL,

        /* Repeat alarm forever */
        .repeatForever       =       true,

        /* Alarm interrupt disabled */
        .alarmIntEnable      =       true,

        /* Alarm interrupt handler */
        .alarmCallback       =       (void *)rtcAlarmCallback,

        /* Alarm interrupt handler parameters */
        .callbackParams      =       (void *)NULL
    };

        /* Call the init function */
        RTC_DRV_Init(0UL, &rtcInitConfig);

        /* Set the time and date */
        RTC_DRV_SetTimeDate(0UL, &rtcStartTime);

        /* Start RTC counter */
        RTC_DRV_StartCounter(0UL);

        /* Configure an alarm every 3 seconds */
        RTC_DRV_ConfigureAlarm(0UL, &rtcAlarmConfig0);

        while(1);
}
```

Important Notes

- Before using the RTC driver the module clock must be configured
- The driver enables the interrupts for the corresponding RTC module, but any interrupt priority must be done by the application
- The board specific configurations must be done prior to driver calls; the driver has no influence on the functionality of the clockout pin - they must be configured by application

Modules

- Real Time Clock Driver

  Real Time Clock Driver Peripheral Driver.
- Real Time Clock HAL

  Real Time Clock Hardware Abstraction Layer.

3.76 Real Time Clock HAL

3.76.1 Detailed Description

Real Time Clock Hardware Abstraction Layer.

Enumerations

- enum rtc_second_int_cfg_t {
  RTC_INT_1HZ = 0x00U, RTC_INT_2HZ = 0x01U, RTC_INT_4HZ = 0x02U, RTC_INT_8HZ = 0x03U,
  RTC_INT_16HZ = 0x04U, RTC_INT_32HZ = 0x05U, RTC_INT_64HZ = 0x06U, RTC_INT_128HZ = 0x07U }

  RTC Seconds interrupt configuration Implements : rtc_second_int_cfg_t_Class.
- enum rtc_clk_out_config_t { RTC_CLKOUT_DISABLED = 0x00U, RTC_CLKOUT_SRC_TSIC = 0x01U, R↩
  TC_CLKOUT_SRC_32KHZ = 0x02U }

  RTC CLKOUT pin configuration Implements : rtc_clk_out_config_t_Class.
- enum rtc_clk_select_t { RTC_CLK_SRC_OSC_32KHZ = 0x00U, RTC_CLK_SRC_LPO_1KHZ = 0x01U }

  RTC clock select Implements : rtc_clk_select_t_Class.
- enum rtc_lock_register_select_t { RTC_LOCK_REG_LOCK = 0x00U, RTC_STATUS_REG_LOCK = 0x01U,
  RTC_CTRL_REG_LOCK = 0x02U, RTC_TCL_REG_LOCK = 0x03U }

  RTC register lock Implements : rtc_lock_register_select_t_Class.

Configuration

- status_t RTC_HAL_Init (RTC_Type ∗const base)

  Initialize RTC instance.
- status_t RTC_HAL_Enable (RTC_Type ∗const base)

  Enable RTC instance counter.
- status_t RTC_HAL_Disable (RTC_Type ∗const base)

  Disable RTC instance counter.
- status_t RTC_HAL_ConfigureRegisterLock (RTC_Type ∗const base, rtc_lock_register_select_t registerTo↩
  Config)

  This function configures register lock status.
- bool RTC_HAL_IsRegisterLocked (const RTC_Type ∗const base, rtc_lock_register_select_t reg)

  This function gets register lock status.
- status_t RTC_HAL_ConfigureClockOut (RTC_Type ∗const base, rtc_clk_out_config_t config)

  This function configures the Clock Out pin source.
- static uint32_t RTC_HAL_GetTimeSecondsRegister (const RTC_Type ∗const base)

  Get Time Seconds Register Value.
- status_t RTC_HAL_SetTimeSecondsRegister (RTC_Type ∗const base, uint32_t seconds)

  Set Time Seconds Register.
- static uint16_t RTC_HAL_GetTimePrescalerRegister (const RTC_Type ∗const base)

  Get Time Prescaler Register.
- status_t RTC_HAL_SetTimePrescalerRegister (RTC_Type ∗const base, uint16_t value)

  Set Time Prescaler Register.
- static uint32_t RTC_HAL_GetTimeAlarmRegister (const RTC_Type ∗const base)

  Get Time Alarm Register.
- static void RTC_HAL_SetTimeAlarmRegister (RTC_Type ∗const base, uint32_t seconds)

  Set Time Alarm Register.
- static void RTC_HAL_GetTimeCompensation (const RTC_Type ∗const base, int8_t ∗compensationValue,
  uint8_t ∗compensationInterval)

Get Time Compensation Value and Interval.
- static void RTC_HAL_SetTimeCompensation (RTC_Type ∗const base, int8_t compensationValue, uint8_↩ t compensationInterval)

    Set Time Compensation.
- static void RTC_HAL_GetCurrentTimeCompensation (const RTC_Type ∗const base, int8_t ∗compensation↩ Value, uint8_t ∗compensationInterval)

    Get TimeCompensation Value and Interval.
- static void RTC_HAL_SetLPOSelect (RTC_Type ∗const base, rtc_clk_select_t clk_select)

    Select clock source for RTC prescaler.
- static rtc_clk_select_t RTC_HAL_GetLPOSelect (const RTC_Type ∗const base)

    Get the selected clock source for RTC prescaler.
- static void RTC_HAL_SetUpdateMode (RTC_Type ∗const base, bool updateEnable)

    Set Update Mode of the registers when locked.
- static bool RTC_HAL_GetUpdateMode (const RTC_Type ∗const base)

    Get the Update Mode of the registers when locked.
- static void RTC_HAL_SetNonSupervisorAccess (RTC_Type ∗const base, bool enable)

    Set Non-Supervisor access mode.
- static bool RTC_HAL_GetNonSupervisorAccess (const RTC_Type ∗const base)

    Get Non-Supervisor access mode.
- static void RTC_HAL_SetSoftwareReset (RTC_Type ∗const base)

    Trigger a software reset.
- static void RTC_HAL_ClearSoftwareReset (RTC_Type ∗const base)

    Clear Software reset flag.
- static void RTC_HAL_SetTimeCounterEnable (RTC_Type ∗const base, bool enable)

    Enable or disable the Time counter.
- static bool RTC_HAL_GetTimeCounterEnable (const RTC_Type ∗const base)

    Get the Time Counter Enable value.
- static bool RTC_HAL_GetTimeAlarmFlag (const RTC_Type ∗const base)

    Get the Time alarm flag.
- static bool RTC_HAL_GetTimeOverflowFlag (const RTC_Type ∗const base)

    Get Time Overflow Flag.
- static bool RTC_HAL_GetTimeInvalidFlag (const RTC_Type ∗const base)

    Get Time Invalid flag.
- static void RTC_HAL_LockRegisterLock (RTC_Type ∗const base)

    Lock the Lock Register.
- static bool RTC_HAL_GetLockRegisterLock (const RTC_Type ∗const base)

    Get the Lock Register Lock state.
- static void RTC_HAL_StatusRegisterLock (RTC_Type ∗const base)

    Lock the Status Register.
- static bool RTC_HAL_GetStatusRegisterLock (const RTC_Type ∗const base)

    Get the Status Register Lock state.
- static bool RTC_HAL_GetControlRegisterLock (const RTC_Type ∗const base)

    Get the Control Register Lock state.
- static void RTC_HAL_ControlRegisterLock (RTC_Type ∗const base)

    Lock the Control Register.
- static bool RTC_HAL_GetTimeCompensationLock (const RTC_Type ∗const base)

    Get the TimeCompensation Register Lock state.
- static void RTC_HAL_TimeCompensationLock (RTC_Type ∗const base)

    Lock the TimeCompensation Register.
- static void RTC_HAL_SetTimeSecondsIntConf (RTC_Type ∗const base, rtc_second_int_cfg_t intCfg)

    Configure Time Seconds interrupt.

- static rtc_second_int_cfg_t RTC_HAL_GetTimeSecondsIntConf (const RTC_Type ∗const base)

  Get Time Seconds interrupt configuration.
- static void RTC_HAL_SetTimeSecondsIntEnable (RTC_Type ∗const base, bool enable)

  Enable TimeSeconds interrupt.
- static bool RTC_HAL_GetTimeSecondsIntEnable (const RTC_Type ∗const base)

  Get the TimeSeconds interrupt enable status.
- static void RTC_HAL_SetTimeAlarmIntEnable (RTC_Type ∗const base, bool enable)

  Enable TimeAlarm interrupt.
- static bool RTC_HAL_GetTimeAlarmIntEnable (const RTC_Type ∗const base)

  Get the TimeAlarm interrupt enable status.
- static void RTC_HAL_SetTimeOverflowIntEnable (RTC_Type ∗const base, bool enable)

  Enable TimeOverflow interrupt.
- static bool RTC_HAL_GetTimeOverflowIntEnable (const RTC_Type ∗const base)

  Get the TimeAlarm interrupt enable status.
- static void RTC_HAL_SetTimeInvalidIntEnable (RTC_Type ∗const base, bool enable)

  Enable TimeInvalid interrupt.
- static bool RTC_HAL_GetTimeInvalidIntEnable (const RTC_Type ∗const base)

  Get the TimeInvalid interrupt enable status.

3.76.2 Enumeration Type Documentation

3.76.2.1 rtc_clk_out_config_t

```
enum rtc_clk_out_config_t
```

RTC CLKOUT pin configuration Implements : rtc_clk_out_config_t_Class.

Enumerator

RTC_CLKOUT_DISABLED	Clock out pin is disabled
RTC_CLKOUT_SRC_TSIC	Output on RTC_CLKOUT as configured on Time seconds interrupt
RTC_CLKOUT_SRC_32KHZ	Output on RTC_CLKOUT of the 32KHz clock

3.76.2.2 rtc_clk_select_t

```
enum rtc_clk_select_t
```

RTC clock select Implements : rtc_clk_select_t_Class.

Enumerator

RTC_CLK_SRC_OSC_32KHZ	RTC Prescaler increments using 32 KHz crystal
RTC_CLK_SRC_LPO_1KHZ	RTC Prescaler increments using 1KHz LPO

3.76.2.3 rtc_lock_register_select_t

```
enum rtc_lock_register_select_t
```

RTC register lock Implements : rtc_lock_register_select_t_Class.

Enumerator

RTC_LOCK_REG_LOCK	RTC Lock Register lock
RTC_STATUS_REG_LOCK	RTC Status Register lock
RTC_CTRL_REG_LOCK	RTC Control Register lock
RTC_TCL_REG_LOCK	RTC Time Compensation Reg lock

3.76.2.4 rtc_second_int_cfg_t

```
enum rtc_second_int_cfg_t
```

RTC Seconds interrupt configuration Implements : rtc_second_int_cfg_t_Class.

Enumerator

RTC_INT_1HZ	RTC seconds interrupt occurs at 1 Hz
RTC_INT_2HZ	RTC seconds interrupt occurs at 2 Hz
RTC_INT_4HZ	RTC seconds interrupt occurs at 4 Hz
RTC_INT_8HZ	RTC seconds interrupt occurs at 8 Hz
RTC_INT_16HZ	RTC seconds interrupt occurs at 16 Hz
RTC_INT_32HZ	RTC seconds interrupt occurs at 32 Hz
RTC_INT_64HZ	RTC seconds interrupt occurs at 64 Hz
RTC_INT_128HZ	RTC seconds interrupt occurs at 128 Hz

3.76.3 Function Documentation

3.76.3.1 RTC_HAL_ClearSoftwareReset()

```
static void RTC_HAL_ClearSoftwareReset (
            RTC_Type *const base )  [inline], [static]
```

Clear Software reset flag.

Parameters

in	*base*	RTC base pointer Implements : RTC_HAL_ClearSoftwareReset_Activity

3.76.3.2 RTC_HAL_ConfigureClockOut()

```
status_t RTC_HAL_ConfigureClockOut (
            RTC_Type *const base,
            rtc_clk_out_config_t config )
```

This function configures the Clock Out pin source.

Parameters

in	*base*	RTC base pointer
in	*config*	Source for the Clock Out pin

Returns

Returns the status of the operation, STATUS_SUCCESS if the configuration was successful, STATUS_ER↩ROR if the Control Register is locked.

3.76.3.3 RTC_HAL_ConfigureRegisterLock()

```
status_t RTC_HAL_ConfigureRegisterLock (
            RTC_Type *const base,
            rtc_lock_register_select_t registerToConfig )
```

This function configures register lock status.

Parameters

in	*base*	RTC base pointer
in	*registerToConfig*	Register to configure

Returns

STATUS_SUCCESS if the lock was successful or if the register was already locked, STATUS_ERROR if the Lock Register is already locked or if the registerToConfig parameter is not a valid register.

3.76.3.4 RTC_HAL_ControlRegisterLock()

```
static void RTC_HAL_ControlRegisterLock (
            RTC_Type *const base ) [inline], [static]
```

Lock the Control Register.

This method locks the Control Register. If the register is locked, it can be unlocked only with power-on reset(POR).

Parameters

in	*base*	RTC base pointer Implements : RTC_HAL_ControlRegisterLock_Activity

3.76.3.5 RTC_HAL_Disable()

```
status_t RTC_HAL_Disable (
            RTC_Type *const base )
```

Disable RTC instance counter.

Parameters

in	*base*	RTC base pointer

Returns

STATUS_SUCCESS if the operation was successful, STATUS_ERROR if the counter was not disabled.

3.76.3.6 RTC_HAL_Enable()

```
status_t RTC_HAL_Enable (
             RTC_Type *const base )
```

Enable RTC instance counter.

Parameters

in	*base*	RTC base pointer

Returns

STATUS_SUCCESS if the operation was successful, STATUS_ERROR if the counter is enabled or if the time invalid flag is set.

3.76.3.7 RTC_HAL_GetControlRegisterLock()

```
static bool RTC_HAL_GetControlRegisterLock (
             const RTC_Type *const base )  [inline], [static]
```

Get the Control Register Lock state.

Parameters

in	*base*	RTC base pointer

Returns

State of the Control register lock

- true if register is locked
- false if the register is not locked Implements : RTC_HAL_GetControlRegisterLock_Activity

3.76.3.8 RTC_HAL_GetCurrentTimeCompensation()

```
static void RTC_HAL_GetCurrentTimeCompensation (
             const RTC_Type *const base,
             int8_t * compensationValue,
             uint8_t * compensationInterval )  [inline], [static]
```

Get TimeCompensation Value and Interval.

Returns current value used by the compensation logic for the present second interval. Updated once a second if the CIC equals 0 with the contents of the TCR field. If the CIC does not equal zero then it is loaded with zero.

Parameters

in	*base*	RTC base pointer
out	*compensationValue*	- Current value which is subtracted from the counter valid range -128, +127
out	*compensationInterval*	Current Compensation interval at which the compensation value is added to the prescaler register

Returns

Current value used by the compensation logic for the present second interval Implements : RTC_HAL_Get↩CurrentTimeCompensation_Activity

3.76.3.9 RTC_HAL_GetLockRegisterLock()

```
static bool RTC_HAL_GetLockRegisterLock (
            const RTC_Type *const base )  [inline], [static]
```

Get the Lock Register Lock state.

Parameters

in	*base*	RTC base pointer

Returns

State of the lock register lock

- true if register is locked
- false if the register is not locked Implements : RTC_HAL_GetLockRegisterLock_Activity

3.76.3.10 RTC_HAL_GetLPOSelect()

```
static rtc_clk_select_t RTC_HAL_GetLPOSelect (
            const RTC_Type *const base )  [inline], [static]
```

Get the selected clock source for RTC prescaler.

When set, the RTC prescaler increments using the LPO 1kHz clock and not the RTC 32kHz crystal clock. The LPO increments the prescaler from bit TPR[5] (TPR[4:0] are ignored), supporting close to 1 second increment of the seconds register.

Parameters

in	*base*	RTC base pointer

Returns

Selected clock source for RTC Implements : RTC_HAL_GetLPOSelect_Activity

3.76.3.11 RTC_HAL_GetNonSupervisorAccess()

```
static bool RTC_HAL_GetNonSupervisorAccess (
            const RTC_Type *const base )  [inline], [static]
```

Get Non-Supervisor access mode.

Parameters

in	*base*	RTC base pointer

Returns

supervisor access mode

- true if Non-supervisor mode write accesses are supported.
- false if Non-supervisor mode write accesses are not supported. Implements : RTC_HAL_GetNon↵ SupervisorAccess_Activity

3.76.3.12 RTC_HAL_GetStatusRegisterLock()

```
static bool RTC_HAL_GetStatusRegisterLock (
            const RTC_Type *const base )  [inline], [static]
```

Get the Status Register Lock state.

Parameters

in	*base*	RTC base pointer

Returns

State of the status register lock

- true if register is locked
- false if the register is not locked Implements : RTC_HAL_GetStatusRegisterLock_Activity

3.76.3.13 RTC_HAL_GetTimeAlarmFlag()

```
static bool RTC_HAL_GetTimeAlarmFlag (
            const RTC_Type *const base )  [inline], [static]
```

Get the Time alarm flag.

The alarm flag is cleared after a write in Time Alarm Register

Parameters

in	*base*	RTC base pointer

Returns

: State of the alarm flag

- true if an alarm occurred
- false if an alarm was not occurred Implements : RTC_HAL_GetTimeAlarmFlag_Activity

3.76.3.14 RTC_HAL_GetTimeAlarmIntEnable()

```
static bool RTC_HAL_GetTimeAlarmIntEnable (
            const RTC_Type *const base )  [inline], [static]
```

Get the TimeAlarm interrupt enable status.

Parameters

in	*base*	RTC base pointer

Returns

The TimeAlarm interrupt enablement

- true if interrupt is enabled
- false if interrupt is disabled Implements : RTC_HAL_GetTimeAlarmIntEnable_Activity

3.76.3.15 RTC_HAL_GetTimeAlarmRegister()

```
static uint32_t RTC_HAL_GetTimeAlarmRegister (
             const RTC_Type *const base )  [inline], [static]
```

Get Time Alarm Register.

Parameters

in	*base*	RTC base pointer

Returns

Value in seconds of the Time Alarm Register Implements : RTC_HAL_GetTimeAlarmRegister_Activity

3.76.3.16 RTC_HAL_GetTimeCompensation()

```
static void RTC_HAL_GetTimeCompensation (
             const RTC_Type *const base,
             int8_t * compensationValue,
             uint8_t * compensationInterval )  [inline], [static]
```

Get Time Compensation Value and Interval.

The Time Prescaler register overflows at every 32768 - (compRegister) cycles. For example if the compRegister is -128 TPR overflows at 32768 - (-128) = 32896 cycles. Else if compRegister is 127 TPR overflows at 32641 cycles. This correction is made at the interval configured by Compensation Interval

Parameters

in	*base*	RTC base pointer
out	*compensationValue*	Compensation value
out	*compensationInterval*	Compensation Interval

Returns

None

Implements : RTC_HAL_GetTimeCompensation_Activity

3.76.3.17 RTC_HAL_GetTimeCompensationLock()

```
static bool RTC_HAL_GetTimeCompensationLock (
             const RTC_Type *const base )  [inline], [static]
```

Get the TimeCompensation Register Lock state.

Parameters

in	*base*	RTC base pointer

Returns

State of the Time Compensation register lock

- true if register is locked
- false if the register is not locked Implements : RTC_HAL_GetTimeCompensationLock_Activity

3.76.3.18 RTC_HAL_GetTimeCounterEnable()

```
static bool RTC_HAL_GetTimeCounterEnable (
             const RTC_Type *const base )  [inline], [static]
```

Get the Time Counter Enable value.

Parameters

in	*base*	RTC base pointer

Returns

State of the counter enable bit

- true if the counter is enabled
- false if the counter is disabled Implements : RTC_HAL_GetTimeCounterEnable_Activity

3.76.3.19 RTC_HAL_GetTimeInvalidFlag()

```
static bool RTC_HAL_GetTimeInvalidFlag (
             const RTC_Type *const base )  [inline], [static]
```

Get Time Invalid flag.

The time invalid flag is set on POR or software reset. The TSR and TPR do not increment and read as zero when this bit is set. This bit is cleared by writing the TSR register when the time counter is disabled.

Parameters

in	*base*	RTC base pointer

Returns

State of the time invalid flag

- true if TIF is set
- false if TIF is clear Implements : RTC_HAL_GetTimeInvalidFlag_Activity

3.76.3.20 RTC_HAL_GetTimeInvalidIntEnable()

```
static bool RTC_HAL_GetTimeInvalidIntEnable (
             const RTC_Type *const base )  [inline], [static]
```

Get the TimeInvalid interrupt enable status.

Parameters

in	*base*	RTC base pointer

Returns

The TimeInvalid interrupt enablement

- true if interrupt is enabled
- false if interrupt is disabled Implements : RTC_HAL_GetTimeInvalidIntEnable_Activity

3.76.3.21 RTC_HAL_GetTimeOverflowFlag()

```
static bool RTC_HAL_GetTimeOverflowFlag (
             const RTC_Type *const base )  [inline], [static]
```

Get Time Overflow Flag.

The TOF is set when Time Seconds Register overflows. Disable the counter and write TSR to clear this bit

Parameters

in	*base*	RTC base pointer

Returns

State of the Time overflow flag

- true if an overflow has occurred
- false if an overflow has not occurred Implements : RTC_HAL_GetTimeOverflowFlag_Activity

3.76.3.22 RTC_HAL_GetTimeOverflowIntEnable()

```
static bool RTC_HAL_GetTimeOverflowIntEnable (
             const RTC_Type *const base )  [inline], [static]
```

Get the TimeAlarm interrupt enable status.

Parameters

in	*base*	RTC base pointer

Returns

The TimeOverflow interrupt enablement

- true if interrupt is enabled
- false if interrupt is disabled Implements : RTC_HAL_GetTimeOverflowIntEnable_Activity

3.76.3.23 RTC_HAL_GetTimePrescalerRegister()

```
static uint16_t RTC_HAL_GetTimePrescalerRegister (
            const RTC_Type *const base )  [inline], [static]
```

Get Time Prescaler Register.

Parameters

in	*base*	RTC base pointer

Returns

Value stored in the Time Prescaler Register Implements : RTC_HAL_GetTimePrescalerRegister_Activity

3.76.3.24 RTC_HAL_GetTimeSecondsIntConf()

```
static rtc_second_int_cfg_t RTC_HAL_GetTimeSecondsIntConf (
            const RTC_Type *const base )  [inline], [static]
```

Get Time Seconds interrupt configuration.

Parameters

in	*base*	RTC base pointer

Returns

Time Seconds interrupt configuration Implements : RTC_HAL_GetTimeSecondsIntConf_Activity

3.76.3.25 RTC_HAL_GetTimeSecondsIntEnable()

```
static bool RTC_HAL_GetTimeSecondsIntEnable (
            const RTC_Type *const base )  [inline], [static]
```

Get the TimeSeconds interrupt enable status.

Parameters

in	*base*	RTC base pointer

Returns

The TimeSeconds interrupt enablement

- true if interrupt is enabled
- false if interrupt is disabled Implements : RTC_HAL_GetTimeSecondsIntEnable_Activity

3.76.3.26 RTC_HAL_GetTimeSecondsRegister()

```
static uint32_t RTC_HAL_GetTimeSecondsRegister (
            const RTC_Type *const base )  [inline], [static]
```

Get Time Seconds Register Value.

Parameters

in	*base*	RTC base pointer

Returns

Number of seconds passed Implements : RTC_HAL_GetTimeSecondsRegister_Activity

3.76.3.27 RTC_HAL_GetUpdateMode()

```
static bool RTC_HAL_GetUpdateMode (
            const RTC_Type *const base )  [inline], [static]
```

Get the Update Mode of the registers when locked.

Parameters

in	*base*	RTC base pointer

Returns

Update mode value

- true if writing of the registers when locked is enabled
- false if writing of the registers when locked is disabled Implements : RTC_HAL_GetUpdateMode_Activity

3.76.3.28 RTC_HAL_Init()

```
status_t RTC_HAL_Init (
            RTC_Type *const base )
```

Initialize RTC instance.

Parameters

in	*base*	RTC base pointer

Returns

STATUS_SUCCESS if the operation was successful, STATUS_ERROR if at least one register is locked.

3.76.3.29 RTC_HAL_IsRegisterLocked()

```
bool RTC_HAL_IsRegisterLocked (
            const RTC_Type *const base,
            rtc_lock_register_select_t reg )
```

This function gets register lock status.

Parameters

in	*base*	RTC base pointer
in	*reg*	The register for which to check lock status

Returns

Return true if the register is locked, false if it is not locked

3.76.3.30 RTC_HAL_LockRegisterLock()

```
static void RTC_HAL_LockRegisterLock (
            RTC_Type *const base ) [inline], [static]
```

Lock the Lock Register.

This method locks the Lock Register. If the register is locked, it can be unlocked only with power-on reset(POR) or a software reset.

Parameters

in	*base*	RTC base pointer Implements : RTC_HAL_LockRegisterLock_Activity

3.76.3.31 RTC_HAL_SetLPOSelect()

```
static void RTC_HAL_SetLPOSelect (
            RTC_Type *const base,
            rtc_clk_select_t clk_select ) [inline], [static]
```

Select clock source for RTC prescaler.

When set, the RTC prescaler increments using the LPO 1kHz clock and not the RTC 32kHz crystal clock. The LPO increments the prescaler from bit TPR[5] (TPR[4:0] are ignored), supporting close to 1 second increment of the seconds register.

Parameters

in	*base*	RTC base pointer
in	*clk_select*	clock source Implements : RTC_HAL_SetLPOSelect_Activity

3.76.3.32 RTC_HAL_SetNonSupervisorAccess()

```
static void RTC_HAL_SetNonSupervisorAccess (
            RTC_Type *const base,
            bool enable )  [inline], [static]
```

Set Non-Supervisor access mode.

Parameters

in	*base*	RTC base pointer
in	*enable*	supervisor access • if true Non-supervisor mode write accesses are supported. • if false Non-supervisor mode write accesses are not supported and generate a bus error. Implements : RTC_HAL_SetNonSupervisorAccess_Activity

3.76.3.33 RTC_HAL_SetSoftwareReset()

```
static void RTC_HAL_SetSoftwareReset (
            RTC_Type *const base )  [inline], [static]
```

Trigger a software reset.

Parameters

in	*base*	RTC base pointer Implements : RTC_HAL_SetSoftwareReset_Activity

3.76.3.34 RTC_HAL_SetTimeAlarmIntEnable()

```
static void RTC_HAL_SetTimeAlarmIntEnable (
            RTC_Type *const base,
            bool enable )  [inline], [static]
```

Enable TimeAlarm interrupt.

Parameters

in	*base*	RTC base pointer
in	*enable*	Write • true to enable the interrupt • false to disable it Implements : RTC_HAL_SetTimeAlarmIntEnable_Activity

3.76.3.35 RTC_HAL_SetTimeAlarmRegister()

```
static void RTC_HAL_SetTimeAlarmRegister (
            RTC_Type *const base,
            uint32_t seconds )  [inline], [static]
```

Set Time Alarm Register.

Parameters

in	*base*	RTC base pointer
in	*seconds*	Number of seconds at which the alarm is triggered. The TAR value is correct only if the value is greater than current time (Time seconds register) Implements : RTC_HAL_SetTimeAlarmRegister_Activity

3.76.3.36 RTC_HAL_SetTimeCompensation()

```
static void RTC_HAL_SetTimeCompensation (
            RTC_Type *const base,
            int8_t compensationValue,
            uint8_t compensationInterval )  [inline], [static]
```

Set Time Compensation.

Configure the frequency of the Time Seconds counter together with Compensation Interval register.

The Time Prescaler register overflows at every 32768 - (compValue) cycles. For example if the compValue is -128 TPR overflows at 32768 - (-128) = 32896 cycles

Else if compValue is 127 TPR overflows at 32641 cycles

The compensation interval in seconds from 1 to 256 is used 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

in	*base*	RTC base pointer
in	*compensationValue*	- the value which is subtracted from the counter valid range -128, +127
in	*compensationInterval*	Compensation interval at which the compensation value is added to the prescaler register

Returns

None Implements : RTC_HAL_SetTimeCompensation_Activity

3.76.3.37 RTC_HAL_SetTimeCounterEnable()

```
static void RTC_HAL_SetTimeCounterEnable (
            RTC_Type *const base,
            bool enable )  [inline], [static]
```

Enable or disable the Time counter.

When time counter is disabled the TSR register and TPR register are writable, but do not increment. When time counter is enabled the TSR register and TPR register are not writable, but increment.

Parameters

in	*base*	RTC base pointer
in	*enable*	: • true to enable the counter • false to disable the counter Implements : RTC_HAL_SetTimeCounterEnable_Activity

3.76.3.38 RTC_HAL_SetTimeInvalidIntEnable()

```
static void RTC_HAL_SetTimeInvalidIntEnable (
             RTC_Type *const base,
             bool enable )  [inline], [static]
```

Enable TimeInvalid interrupt.

Parameters

in	*base*	RTC base pointer
in	*enable*	Write • true to enable the interrupt • false to disable it Implements : RTC_HAL_SetTimeInvalidIntEnable_Activity

3.76.3.39 RTC_HAL_SetTimeOverflowIntEnable()

```
static void RTC_HAL_SetTimeOverflowIntEnable (
             RTC_Type *const base,
             bool enable )  [inline], [static]
```

Enable TimeOverflow interrupt.

Parameters

in	*base*	RTC base pointer
in	*enable*	Write • true to enable the interrupt • false to disable it Implements : RTC_HAL_SetTimeOverflowIntEnable_Activity

3.76.3.40 RTC_HAL_SetTimePrescalerRegister()

```
status_t RTC_HAL_SetTimePrescalerRegister (
             RTC_Type *const base,
             uint16_t value )
```

Set Time Prescaler Register.

This function along with SetTimeSecondsRegister will help you set the starting time at a specified value. The write will fail if the Time Counter is enabled and will return STATUS_ERROR, otherwise the return will be STATUS_S↵ UCCESS

Parameters

in	*base*	RTC base pointer
in	*value*	Number of RTC CLK IN periods

Returns

STATUS_SUCCESS if the write is succeeded or STATUS_ERROR if the counter is enabled.

3.76.3.41 RTC_HAL_SetTimeSecondsIntConf()

```
static void RTC_HAL_SetTimeSecondsIntConf (
             RTC_Type *const base,
             rtc_second_int_cfg_t intCfg )  [inline], [static]
```

Configure Time Seconds interrupt.

Parameters

in	*base*	RTC base pointer
in	*intCfg*	Select at which frequency the interrupt will occur. Implements : RTC_HAL_SetTimeSecondsIntConf_Activity

3.76.3.42 RTC_HAL_SetTimeSecondsIntEnable()

```
static void RTC_HAL_SetTimeSecondsIntEnable (
             RTC_Type *const base,
             bool enable )  [inline], [static]
```

Enable TimeSeconds interrupt.

Parameters

in	*base*	RTC base pointer
in	*enable*	Write: • true to enable the interrupt • false to disable it Implements : RTC_HAL_SetTimeSecondsIntEnable_Activity

3.76.3.43 RTC_HAL_SetTimeSecondsRegister()

```
status_t RTC_HAL_SetTimeSecondsRegister (
             RTC_Type *const base,
             uint32_t seconds )
```

Set Time Seconds Register.

This function along with SetTimePrescalerRegister will help you set the starting time at a specified value. The write will fail if the Time Counter is enabled and will return STATUS_ERROR, otherwise the return will be STATUS_S↵ UCCESS

Parameters

in	*base*	RTC base pointer
in	*seconds*	number of seconds passed

Returns

STATUS_SUCCESS if the write is succeeded or STATUS_ERROR if the counter is enabled.

3.76.3.44 RTC_HAL_SetUpdateMode()

```
static void RTC_HAL_SetUpdateMode (
             RTC_Type *const base,
             bool updateEnable )  [inline], [static]
```

Set Update Mode of the registers when locked.

Parameters

in	*base*	RTC base pointer
in	*updateEnable*	value to be written in the register field • true to enable writing of the registers when locked • false to disable writing of the registers when locked Implements : RTC_HAL_SetUpdateMode_Activity

3.76.3.45 RTC_HAL_StatusRegisterLock()

```
static void RTC_HAL_StatusRegisterLock (
             RTC_Type *const base )  [inline], [static]
```

Lock the Status Register.

This method locks the Status Register. If the register is locked, it can be unlocked only with power-on reset(POR) or a software reset.

Parameters

in	*base*	RTC base pointer Implements : RTC_HAL_StatusRegisterLock_Activity

3.76.3.46 RTC_HAL_TimeCompensationLock()

```
static void RTC_HAL_TimeCompensationLock (
             RTC_Type *const base )  [inline], [static]
```

Lock the TimeCompensation Register.

This method locks the TimeCompensation Register. If the register is locked, it can be unlocked only with power-on reset(POR) or a software reset.

Parameters

in	*base*	RTC base pointer Implements : RTC_HAL_TimeCompensationLock_Activity

3.77 Reset Control Module (RCM)

3.77.1 Detailed Description

This module covers the functionality of the Reset Control Module (RCM) peripheral.

RCM HAL provides the API for reading and writing register bit-fields belonging to the RCM module.

Data Structures

- struct rcm_version_info_t

  RCM module version number Implements rcm_version_info_t_Class. More...

Enumerations

- enum rcm_source_names_t {
  RCM_WAKEUP, RCM_LOW_VOLT_DETECT, RCM_LOSS_OF_CLK, RCM_LOSS_OF_LOCK,
  RCM_WATCH_DOG, RCM_EXTERNAL_PIN, RCM_POWER_ON, RCM_SJTAG,
  RCM_CORE_LOCKUP, RCM_SOFTWARE, RCM_SMDM_AP, RCM_STOP_MODE_ACK_ERR,
  RCM_TAMPERR, RCM_CORE1, RCM_SRC_NAME_MAX }

  System Reset Source Name definitions Implements rcm_source_names_t_Class.
- enum rcm_filter_run_wait_modes_t { RCM_FILTER_DISABLED, RCM_FILTER_BUS_CLK, RCM_FILTE↩ R_LPO_CLK, RCM_FILTER_RESERVED }

  Reset pin filter select in Run and Wait modes Implements rcm_filter_run_wait_modes_t_Class.
- enum rcm_reset_delay_time_t { RCM_10LPO_CYCLES_DELAY, RCM_34LPO_CYCLES_DELAY, RCM_↩ 130LPO_CYCLES_DELAY, RCM_514LPO_CYCLES_DELAY }

  Reset delay time Implements rcm_reset_delay_time_t_Class.

Reset Control Module APIs

- void RCM_HAL_GetVersion (const RCM_Type ∗const baseAddr, rcm_version_info_t ∗const versionInfo)

  Get the version of the RCM module.
- bool RCM_HAL_GetSrcIndicationFeatureAvailability (const RCM_Type ∗const baseAddr, const rcm_↩ source_names_t srcName)

  Checks the existence of the status indication feature for a reset source.
- bool RCM_HAL_GetSrcStatusCmd (const RCM_Type ∗const baseAddr, const rcm_source_names_t src↩ Name)

  Gets the reset source status.
- void RCM_HAL_SetResetIntCmd (RCM_Type ∗const baseAddr, const rcm_source_names_t resetInterrupt, const bool enable)

  Enables/disables a specified system reset interrupt.
- static void RCM_HAL_SetAllResetIntCmd (RCM_Type ∗const baseAddr, const bool enable)

  Enables/disables all system reset interrupts.
- bool RCM_HAL_GetStickySrcStatusCmd (const RCM_Type ∗const baseAddr, const rcm_source_names_t srcName)

  Gets the sticky reset source status.
- void RCM_HAL_ClearStickySrcStatus (RCM_Type ∗const baseAddr)

  Clear the sticky reset source status.
- static void RCM_HAL_SetFilterStopModeCmd (RCM_Type ∗const baseAddr, const bool enable)

  Sets the reset pin filter in stop mode.

- static bool RCM_HAL_GetFilterStopModeCmd (const RCM_Type ∗const baseAddr)
  Gets the reset pin filter in stop mode.
- static void RCM_HAL_SetFilterRunWaitMode (RCM_Type ∗const baseAddr, const rcm_filter_run_wait_↩ modes_t mode)
  Sets the reset pin filter in run and wait mode.
- static rcm_filter_run_wait_modes_t RCM_HAL_GetFilterRunWaitMode (const RCM_Type ∗const baseAddr)
  Gets the reset pin filter for stop mode.
- static void RCM_HAL_SetFilterWidth (RCM_Type ∗const baseAddr, const uint32_t width)
  Sets the reset pin filter width.
- static uint32_t RCM_HAL_GetFilterWidth (const RCM_Type ∗const baseAddr)
  Gets the reset pin filter for stop mode.
- static void RCM_HAL_SetResetDelayTimeValue (RCM_Type ∗const baseAddr, const rcm_reset_delay_↩ time_t value)
  Sets reset delay time.

3.77.2 Data Structure Documentation

3.77.2.1 struct rcm_version_info_t

RCM module version number Implements rcm_version_info_t_Class.

Data Fields

- uint32_t majorNumber
- uint32_t minorNumber
- uint32_t featureNumber

Field Documentation

3.77.2.1.1 featureNumber

```
uint32_t featureNumber
```

Feature Specification Number

3.77.2.1.2 majorNumber

```
uint32_t majorNumber
```

Major Version Number

3.77.2.1.3 minorNumber

```
uint32_t minorNumber
```

Minor Version Number

3.77.3 Enumeration Type Documentation

3.77.3.1 rcm_filter_run_wait_modes_t

```
enum rcm_filter_run_wait_modes_t
```

Reset pin filter select in Run and Wait modes Implements rcm_filter_run_wait_modes_t_Class.

Enumerator

RCM_FILTER_DISABLED	
RCM_FILTER_BUS_CLK	
RCM_FILTER_LPO_CLK	
RCM_FILTER_RESERVED	

3.77.3.2 rcm_reset_delay_time_t

```
enum rcm_reset_delay_time_t
```

Reset delay time Implements rcm_reset_delay_time_t_Class.

Enumerator

RCM_10LPO_CYCLES_DELAY	
RCM_34LPO_CYCLES_DELAY	
RCM_130LPO_CYCLES_DELAY	
RCM_514LPO_CYCLES_DELAY	

3.77.3.3 rcm_source_names_t

```
enum rcm_source_names_t
```

System Reset Source Name definitions Implements rcm_source_names_t_Class.

Enumerator

RCM_WAKEUP	
RCM_LOW_VOLT_DETECT	
RCM_LOSS_OF_CLK	
RCM_LOSS_OF_LOCK	
RCM_WATCH_DOG	
RCM_EXTERNAL_PIN	
RCM_POWER_ON	
RCM_SJTAG	
RCM_CORE_LOCKUP	
RCM_SOFTWARE	
RCM_SMDM_AP	
RCM_STOP_MODE_ACK_ERR	
RCM_TAMPERR	
RCM_CORE1	
RCM_SRC_NAME_MAX	

3.77.4 Function Documentation

3.77.4.1 RCM_HAL_ClearStickySrcStatus()

```
void RCM_HAL_ClearStickySrcStatus (
            RCM_Type *const baseAddr )
```

Clear the sticky reset source status.

This function clears all the sticky system reset flags.

Parameters

in	*baseAddr*	Register base address of RCM

3.77.4.2 RCM_HAL_GetFilterRunWaitMode()

```
static rcm_filter_run_wait_modes_t RCM_HAL_GetFilterRunWaitMode (
            const RCM_Type *const baseAddr )  [inline], [static]
```

Gets the reset pin filter for stop mode.

This function gets the reset pin filter enable setting for stop mode.

Parameters

in	*baseAddr*	Register base address of RCM

Returns

mode for reset filter in run/wait mode Implements RCM_HAL_GetFilterRunWaitMode_Activity

3.77.4.3 RCM_HAL_GetFilterStopModeCmd()

```
static bool RCM_HAL_GetFilterStopModeCmd (
            const RCM_Type *const baseAddr )  [inline], [static]
```

Gets the reset pin filter in stop mode.

This function gets the reset pin filter enable setting in stop mode.

Parameters

in	*baseAddr*	Register base address of RCM

Returns

enable true/false to enable or disable the filter in stop mode Implements RCM_HAL_GetFilterStopMode↩
Cmd_Activity

3.77.4.4 RCM_HAL_GetFilterWidth()

```
static uint32_t RCM_HAL_GetFilterWidth (
            const RCM_Type *const baseAddr )  [inline], [static]
```

Gets the reset pin filter for stop mode.

This function gets the reset pin filter width.

Parameters

in	*baseAddr*	Register base address of RCM

Returns

width reset filter width Implements RCM_HAL_GetFilterWidth_Activity

3.77.4.5 RCM_HAL_GetSrcIndicationFeatureAvailability()

```
bool RCM_HAL_GetSrcIndicationFeatureAvailability (
             const RCM_Type *const baseAddr,
             const rcm_source_names_t srcName )
```

Checks the existence of the status indication feature for a reset source.

This function checks the existence of the status indication feature for a specified source.

Parameters

in	*baseAddr*	Register base address of RCM
in	*srcName*	reset source name

Returns

status true or false for specified reset source

3.77.4.6 RCM_HAL_GetSrcStatusCmd()

```
bool RCM_HAL_GetSrcStatusCmd (
             const RCM_Type *const baseAddr,
             const rcm_source_names_t srcName )
```

Gets the reset source status.

This function gets the current reset source status for a specified source.

Parameters

in	*baseAddr*	Register base address of RCM
in	*srcName*	reset source name

Returns

status true or false for specified reset source

3.77.4.7 RCM_HAL_GetStickySrcStatusCmd()

```
bool RCM_HAL_GetStickySrcStatusCmd (
             const RCM_Type *const baseAddr,
             const rcm_source_names_t srcName )
```

Gets the sticky reset source status.

This function gets the current reset source status that have not been cleared by software for a specified source.

Parameters

in	*baseAddr*	Register base address of RCM
in	*srcName*	reset source name

Returns

status true or false for specified reset source

3.77.4.8 RCM_HAL_GetVersion()

```
void RCM_HAL_GetVersion (
            const RCM_Type *const baseAddr,
            rcm_version_info_t *const versionInfo )
```

Get the version of the RCM module.

Parameters

in	*baseAddr*	base address of the RCM module
out	*versionInfo*	Device Version Number

3.77.4.9 RCM_HAL_SetAllResetIntCmd()

```
static void RCM_HAL_SetAllResetIntCmd (
            RCM_Type *const baseAddr,
            const bool enable ) [inline], [static]
```

Enables/disables all system reset interrupts.

This function enables/disables all system reset interrupts.

Parameters

in	*baseAddr*	Register base address of RCM
in	*enable*	enable or disable the filter in stop mode Implements RCM_HAL_SetAllResetIntCmd_Activity

3.77.4.10 RCM_HAL_SetFilterRunWaitMode()

```
static void RCM_HAL_SetFilterRunWaitMode (
            RCM_Type *const baseAddr,
            const rcm_filter_run_wait_modes_t mode ) [inline], [static]
```

Sets the reset pin filter in run and wait mode.

This function sets the reset pin filter enable setting in run/wait mode.

Parameters

in	*baseAddr*	Register base address of RCM
in	*mode*	to be set for reset filter in run/wait mode Implements RCM_HAL_SetFilterRunWaitMode_Activity

3.77.4.11 RCM_HAL_SetFilterStopModeCmd()

```
static void RCM_HAL_SetFilterStopModeCmd (
            RCM_Type *const baseAddr,
            const bool enable )  [inline], [static]
```

Sets the reset pin filter in stop mode.

This function sets the reset pin filter enable setting in stop mode.

Parameters

in	*baseAddr*	Register base address of RCM
in	*enable*	enable or disable the filter in stop mode Implements RCM_HAL_SetFilterStopModeCmd_Activity

3.77.4.12 RCM_HAL_SetFilterWidth()

```
static void RCM_HAL_SetFilterWidth (
            RCM_Type *const baseAddr,
            const uint32_t width )  [inline], [static]
```

Sets the reset pin filter width.

This function sets the reset pin filter width.

Parameters

in	*baseAddr*	Register base address of RCM
in	*width*	to be set for reset filter width Implements RCM_HAL_SetFilterWidth_Activity

3.77.4.13 RCM_HAL_SetResetDelayTimeValue()

```
static void RCM_HAL_SetResetDelayTimeValue (
            RCM_Type *const baseAddr,
            const rcm_reset_delay_time_t value )  [inline], [static]
```

Sets reset delay time.

This function configures the maximum reset delay time from when the interrupt is asserted.

Parameters

in	*baseAddr*	Register base address of RCM
in	*value*	Reset delay time Implements RCM_HAL_SetResetDelayTimeValue_Activity

3.77.4.14 RCM_HAL_SetResetIntCmd()

```
void RCM_HAL_SetResetIntCmd (
            RCM_Type *const baseAddr,
            const rcm_source_names_t resetInterrupt,
            const bool enable )
```

Enables/disables a specified system reset interrupt.

This function will enable/disable the specified system reset interrupt.

Parameters

in	*baseAddr*	Register base address of RCM
in	*resetInterrupt*	Reset source name
in	*enable*	true or false for the specified reset interrupt

3.78 Sim_hal_s32k144

3.78.1 Detailed Description

This module cover SIM module functionality implemented for S32K144.

Important Notes

The SIM_HAL API provides methods for setting and getting bit fields defined in the SIM module. The are a few exception to this:

1. SetLpoClks - the register is write once, as such it makes sense to configure all bit fields in a single write
2. Read only bit fields cannot be configured, as such there is not Set method for them
3. InitTraceClock - Trace clock configuration bit fields requires specific order

Data Structures

- struct sim_clock_out_config_t

    SIM ClockOut configuration. Implements sim_clock_out_config_t_Class. More...
- struct sim_lpo_clock_config_t

    SIM LPO Clocks configuration. Implements sim_lpo_clock_config_t_Class. More...
- struct sim_plat_gate_config_t

    SIM Platform Gate Clock configuration. Implements sim_plat_gate_config_t_Class. More...
- struct sim_tclk_config_t

    SIM Platform Gate Clock configuration. Implements sim_tclk_config_t_Class. More...
- struct sim_trace_clock_config_t

    SIM Debug Trace clock configuration. Implements sim_trace_clock_config_t_Class. More...
- struct sim_clock_config_t

    SIM configure structure. Implements sim_clock_config_t_Class. More...

Macros

- #define NUMBER_OF_TCLK_INPUTS 3U

Enumerations

- enum sim_adc_supply_src_t {
  ADC_SUPPLY_VDD = 0U, ADC_SUPPLY_VDDA = 1U, ADC_SUPPLY_VREFH = 2U, ADC_SUPPLY_V←DD_3V = 3U,
  ADC_SUPPLY_VDD_FLASH_3V = 4U, ADC_SUPPLY_LV = 5U }

    Internal supplies monitored by ADC_SUPPLY Implements sim_adc_supply_src_t_Class.
- enum clock_trace_src_t { CLOCK_TRACE_SRC_CORE_CLK, CLOCK_TRACE_SRC_PLATFORM_CLK }

    Debug trace clock source select Implements clock_trace_src_t_Class.
- enum sim_pdb_bb_src_t { PDB_BACK_TO_BACK_OPTION_0, PDB_BACK_TO_BACK_OPTION_1 }

    PDB back-to-back select Implements sim_pdb_bb_src_t_Class.

- enum sim_clkout_src_t {
SIM_CLKOUT_SEL_SYSTEM_SCG_CLKOUT = 0U, SIM_CLKOUT_SEL_SYSTEM_SOSC_DIV2_CLK = 2U, SIM_CLKOUT_SEL_SYSTEM_SIRC_DIV2_CLK = 4U, SIM_CLKOUT_SEL_SYSTEM_FIRC_DIV2_↩
CLK = 6U,
SIM_CLKOUT_SEL_SYSTEM_SPLL_DIV2_CLK = 8U, SIM_CLKOUT_SEL_SYSTEM_LPO_128K_CLK = 10U, SIM_CLKOUT_SEL_SYSTEM_LPO_CLK = 12U, SIM_CLKOUT_SEL_SYSTEM_RTC_CLK = 14U }

  SIM CLKOUT select Implements sim_clkout_src_t_Class.
- enum sim_clkout_div_t {
SIM_CLKOUT_DIV_BY_1 = 0U, SIM_CLKOUT_DIV_BY_2 = 1U, SIM_CLKOUT_DIV_BY_3 = 2U, SIM_C↩
LKOUT_DIV_BY_4 = 3U,
SIM_CLKOUT_DIV_BY_5 = 4U, SIM_CLKOUT_DIV_BY_6 = 5U, SIM_CLKOUT_DIV_BY_7 = 6U, SIM_C↩
LKOUT_DIV_BY_8 = 7U }

  SIM CLKOUT divider Implements sim_clkout_div_t_Class.
- enum sim_ftm_clk_sel_t { SIM_FTM_CLK_SEL_00 = 0U, SIM_FTM_CLK_SEL_01 = 1U, SIM_FTM_CLK_↩
SEL_10 = 2U, SIM_FTM_CLK_SEL_11 = 3U }

  SIM FlexTimer external clock select Implements sim_ftm_clk_sel_t_Class.
- enum sim_rtc_clk_sel_src_t { SIM_RTCCLK_SEL_SOSCDIV1_CLK, SIM_RTCCLK_SEL_LPO_32K, SIM↩
_RTCCLK_SEL_RTC_CLKIN, SIM_RTCCLK_SEL_FIRCDIV1_CLK }

  SIM CLK32KSEL clock source select Implements sim_rtc_clk_sel_src_t_Class.
- enum sim_lpoclk_sel_src_t { SIM_LPO_CLK_SEL_LPO_128K, SIM_LPO_CLK_SEL_NO_CLOCK, SIM_L↩
PO_CLK_SEL_LPO_32K, SIM_LPO_CLK_SEL_LPO_1K }

  SIM LPOCLKSEL clock source select Implements sim_lpoclk_sel_src_t_Class.
- enum sim_adc_pretrg_sel_t { SIM_ADC_PRETRG_SEL_PDB, SIM_ADC_PRETRG_SEL_TRGMUX, SIM↩
_ADC_PRETRG_SEL_SOFTWARE, SIM_ADC_PRETRG_SEL_RESERVED }

  SIM ADCx pre-trigger select Implements sim_adc_pretrg_sel_t_Class.
- enum sim_adc_sw_pretrg_sel_t {
SIM_ADC_SW_PRETRG_SEL_DISABLED = 0U, SIM_ADC_SW_PRETRG_SEL_RESERVED0 = 1U, SI↩
M_ADC_SW_PRETRG_SEL_RESERVED1 = 2U, SIM_ADC_SW_PRETRG_SEL_RESERVED2 = 3U,
SIM_ADC_SW_PRETRG_SEL_0 = 4U, SIM_ADC_SW_PRETRG_SEL_1 = 5U, SIM_ADC_SW_PRETRG↩
_SEL_2 = 6U, SIM_ADC_SW_PRETRG_SEL_3 = 7U }

  SIM ADCx software pre-trigger select Implements sim_adc_sw_pretrg_sel_t_Class.
- enum sim_adc_trg_sel_t { SIM_ADC_TRG_SEL_PDB = 0U, SIM_ADC_TRG_SEL_TRGMUX = 1U }

  SIM ADCx trigger select Implements sim_adc_trg_sel_t_Class.
- enum sim_ftm_ch_out_src_t { SIM_FTM_CH_OUT_SRC_0, SIM_FTM_CH_OUT_SRC_1 }

  SIM FlexTimer x channel y output source select Implements sim_ftm_ch_out_src_t_Class.
- enum sim_ftm_ch_src_t { SIM_FTM_CH_SRC_0 = 0U, SIM_FTM_CH_SRC_1 = 1U, SIM_FTM_CH_SRC_2 = 2U, SIM_FTM_CH_SRC_3 = 3U }

  SIM FlexTimer x channel y input source select Implements sim_ftm_ch_src_t_Class.
- enum sim_ram_size_t { SIM_RAM_SIZE_48KB = 0xDU, SIM_RAM_SIZE_64KB = 0xFU }

  SIM RAM size Implements sim_ram_size_t_Class.
- enum sim_package_t { SIM_PACKAGE_64_LQFP = 0x3U, SIM_PACKAGE_100_LQFP = 0x4U, SIM_PA↩
CKAGE_100_MAP_BGA = 0x8U }

  SIM Package Implements sim_package_t_Class.
- enum sim_features_t { SIM_FEATURE_FLEXIO = (1U << 5U), SIM_FEATURE_ISO_CAN_FD = (1U << 6U), SIM_FEATURE_SECURITY = (1U << 7U) }

  SIM Features Implements sim_features_t_Class.
- enum sim_eee_sram_size_t {
SIM_EEE_SRAM_SIZE_4KB = 0x2U, SIM_EEE_SRAM_SIZE_2KB = 0x3U, SIM_EEE_SRAM_SIZE_1KB = 0x4U, SIM_EEE_SRAM_SIZE_512B = 0x5U,
SIM_EEE_SRAM_SIZE_256B = 0x6U, SIM_EEE_SRAM_SIZE_128B = 0x7U, SIM_EEE_SRAM_SIZE_64B = 0x8U, SIM_EEE_SRAM_SIZE_32B = 0x9U }

  SIM EEE SRAM Size Implements sim_eee_sram_size_t_Class.

- enum sim_flexnvm_partition_t {
  SIM_DEPART_0000 = 0x0U, SIM_DEPART_0011 = 0x3U, SIM_DEPART_0100 = 0x4U, SIM_DEPART_↩
  1000 = 0x8U,
  SIM_DEPART_1010 = 0xAU, SIM_DEPART_1011 = 0xBU, SIM_DEPART_1100 = 0xCU, SIM_DEPART_↩
  1111 = 0xFU }

  SIM FlexNVM partition Implements sim_flexnvm_partition_t_Class.

Functions

- static void SIM_HAL_SetSramLRetentionCmd (SIM_Type ∗base, bool setting)

  Set SRAM L Retention setting.
- static bool SIM_HAL_GetSramLRetentionCmd (const SIM_Type ∗base)

  Get SRAM L Retention setting.
- static void SIM_HAL_SetSramURetentionCmd (SIM_Type ∗base, bool setting)

  Set SRAM U Retention setting.
- static bool SIM_HAL_GetSramURetentionCmd (const SIM_Type ∗base)

  Get SRAM U Retention setting.
- static void SIM_HAL_SetAdcSupplyEnCmd (SIM_Type ∗base, bool setting)

  Set ADC Supply Enable setting.
- static bool SIM_HAL_GetAdcSupplyEnCmd (const SIM_Type ∗base)

  Get ADC Supply Enable setting.
- static void SIM_HAL_SetAdcSupplySrc (SIM_Type ∗base, sim_adc_supply_src_t setting)

  Set ADC Supply Enable setting.
- static sim_adc_supply_src_t SIM_HAL_GetAdcSupplySrc (const SIM_Type ∗base)

  Get ADC supply source.
- static void SIM_HAL_SetPdbBackToBackSrc (SIM_Type ∗base, sim_pdb_bb_src_t setting)

  Set PDB back-to-back selection.
- static sim_pdb_bb_src_t SIM_HAL_GetPdbBackToBackSrc (const SIM_Type ∗base)

  Get PDB back-to-back selection.
- void SIM_HAL_GetClkoutDefaultConfig (sim_clock_out_config_t ∗config)

  Get the default SIM CLKOUT clock configuration.
- void SIM_HAL_GetClkoutConfig (const SIM_Type ∗base, sim_clock_out_config_t ∗config)

  Get the SIM CLKOUT clock configuration.
- void SIM_HAL_InitClkout (SIM_Type ∗base, const sim_clock_out_config_t ∗config)

  Initialize SIM CLKOUT.
- static void SIM_HAL_DeinitClkout (SIM_Type ∗base)

  De-initialize SIM CLKOUT.
- static void SIM_HAL_SetAdcInterleaveSel (SIM_Type ∗base, uint8_t setting)

  Set ADC interleave channel select.
- static uint8_t SIM_HAL_GetAdcInterleaveSel (const SIM_Type ∗base)

  Get ADC interleave channel select.
- void SIM_HAL_SetFtmExternalClkPinMode (SIM_Type ∗base, uint32_t instance, sim_ftm_clk_sel_t select)

  Sets the FlexTimer x external clock pin select setting.
- sim_ftm_clk_sel_t SIM_HAL_GetFtmExternalClkPinMode (const SIM_Type ∗base, uint32_t instance)

  Gets the FlexTimer x external clock pin select setting.
- void SIM_HAL_SetFtmFaultSelMode (SIM_Type ∗base, uint32_t instance, uint8_t select)

  Sets the FlexTimer x faults select settings.
- uint8_t SIM_HAL_GetFtmFaultSelMode (const SIM_Type ∗base, uint32_t instance)

  Gets the FlexTimer x faults select settings.
- static sim_rtc_clk_sel_src_t SIM_HAL_GetRtcClkSrc (const SIM_Type ∗base)

Get the clock selection of RTCCLKSEL.
- static void SIM_HAL_SetLpoClocks (SIM_Type ∗base, sim_lpo_clock_config_t setting)

  Set the clock selection of LPOCLKSEL.
- static sim_lpoclk_sel_src_t SIM_HAL_GetLpoClkSrc (const SIM_Type ∗base)

  Get the clock selection of LPOCLKSEL.
- static bool SIM_HAL_GetLpo32kClkEnCmd (const SIM_Type ∗base)

  Gets the 32 kHz LPO clock Control.
- static bool SIM_HAL_GetLpo1kClkEnCmd (const SIM_Type ∗base)

  Gets the 1 kHz LPO clock Control.
- uint32_t SIM_HAL_GetLpoFreq (const SIM_Type ∗base)

  Get SIM LPO clock frequency (LPO_CLOCK).
- uint32_t SIM_HAL_GetLpo128KFreq (const SIM_Type ∗base)

  Get SIM LPO 128KHz clock frequency (LPO_128K_CLOCK).
- uint32_t SIM_HAL_GetLpo32KFreq (const SIM_Type ∗base)

  Get SIM LPO 32KHz clock frequency (LPO_32K_CLOCK).
- uint32_t SIM_HAL_GetLpo1KFreq (const SIM_Type ∗base)

  Get SIM LPO 1KHz clock frequency (LPO_1K_CLOCK).
- void SIM_HAL_SetAdcPreTriggerMode (SIM_Type ∗base, uint32_t instance, sim_adc_pretrg_sel_t select)

  Sets the ADCx pre-trigger select setting.
- sim_adc_pretrg_sel_t SIM_HAL_GetAdcPreTriggerMode (const SIM_Type ∗base, uint32_t instance)

  Gets the ADCx pre-trigger select setting.
- void SIM_HAL_SetAdcSwPreTriggerMode (SIM_Type ∗base, uint32_t instance, sim_adc_sw_pretrg_sel_↩
  t select)

  Sets the ADCx software pre-trigger select setting.
- sim_adc_sw_pretrg_sel_t SIM_HAL_GetAdcSwPreTriggerMode (const SIM_Type ∗base, uint32_t instance)

  Gets the ADCx software pre-trigger select setting.
- void SIM_HAL_SetAdcTriggerMode (SIM_Type ∗base, uint32_t instance, sim_adc_trg_sel_t select)

  Sets the ADCx trigger select setting.
- sim_adc_trg_sel_t SIM_HAL_GetAdcTriggerMode (const SIM_Type ∗base, uint32_t instance)

  Gets the ADCx trigger select setting.
- static void SIM_HAL_SetFtmGlobalLoad (SIM_Type ∗base, bool enable)

  Configure the FTM Global Load from the FTM Option Register 1.
- static bool SIM_HAL_GetFtmGlobalLoad (const SIM_Type ∗base)

  Gets the FTM Global Load from the FTM Option Register 1.
- void SIM_HAL_SetFtmChOutSrcMode (SIM_Type ∗base, uint32_t instance, uint8_t channel, sim_ftm_ch_↩
  out_src_t select)

  Sets the FlexTimer x channel y output source select setting.
- sim_ftm_ch_out_src_t SIM_HAL_GetFtmChOutSrcMode (const SIM_Type ∗base, uint32_t instance, uint8_t channel)

  Gets the FlexTimer x channel y output source select setting.
- void SIM_HAL_SetFtmChSrcMode (SIM_Type ∗base, uint32_t instance, uint8_t channel, sim_ftm_ch_src_t select)

  Sets the FlexTimer x channel y input source select setting.
- sim_ftm_ch_src_t SIM_HAL_GetFtmChSrcMode (const SIM_Type ∗base, uint32_t instance, uint8_t channel)

  Gets the FlexTimer x channel y input source select setting.
- void SIM_HAL_SetFtmSyncCmd (SIM_Type ∗base, uint32_t instance, bool sync)

  Set FlexTimer x hardware trigger 0 software synchronization.
- static bool SIM_HAL_GetFtmSyncCmd (const SIM_Type ∗base, uint32_t instance)

  Get FlexTimer x hardware trigger software synchronization setting.
- static void SIM_HAL_SetObeCtrlCmd (SIM_Type ∗base, uint32_t instance, bool setting)

  Sets FTM channel state.

- static bool SIM_HAL_GetObeCtrlCmd (const SIM_Type ∗base, uint32_t instance)

  Gets FTM channel state.
- static uint32_t SIM_HAL_GetGeneration (const SIM_Type ∗base)

  Gets the product series Generation from System Device ID register (SIM_SDID).
- static uint32_t SIM_HAL_GetSubSeries (const SIM_Type ∗base)

  Gets the sub-series in the System Device ID register (SIM_SDID).
- static uint32_t SIM_HAL_GetDerivate (const SIM_Type ∗base)

  Gets the Derivate from the System Device ID register (SIM_SDID).
- static sim_ram_size_t SIM_HAL_GetRamSize (const SIM_Type ∗base)

  Gets RAM size.
- static uint32_t SIM_HAL_GetRevId (const SIM_Type ∗base)

  Gets the Revision ID in the System Device ID register (SIM_SDID).
- static sim_package_t SIM_HAL_GetPackage (const SIM_Type ∗base)

  Gets the Package in System Device ID register (SIM_SDID).
- static uint32_t SIM_HAL_GetFeatures (const SIM_Type ∗base)

  Gets the Features from System Device ID register (SIM_SDID).
- static void SIM_HAL_SetEimClockGate (SIM_Type ∗base, bool enable)

  Set the EIM Clock Gate from the Platform Clock Gating Control Register.
- static bool SIM_HAL_GetEimClockGate (const SIM_Type ∗base)

  Gets the EIM Clock Gate from the Platform Clock Gating Control Register.
- static void SIM_HAL_SetErmClockGate (SIM_Type ∗base, bool enable)

  Set the ERM Clock Gate from the Platform Clock Gating Control Register.
- static bool SIM_HAL_GetErmClockGate (const SIM_Type ∗base)

  Gets the ERM Clock Gate from the Platform Clock Gating Control Register.
- static void SIM_HAL_SetDmaClockGate (SIM_Type ∗base, bool enable)

  Set the DMA Clock Gate from the Platform Clock Gating Control Register.
- static bool SIM_HAL_GetDmaClockGate (const SIM_Type ∗base)

  Gets the DMA Clock Gate from the Platform Clock Gating Control Register.
- static void SIM_HAL_SetMpuClockGate (SIM_Type ∗base, bool enable)

  Configure the MPU Clock Gating from the Platform Clock Gating Control Register.
- static bool SIM_HAL_GetMpuClockGate (const SIM_Type ∗base)

  Gets the MPU Clock Gating from the Platform Clock Gating Control Register.
- static void SIM_HAL_SetMscmClockGate (SIM_Type ∗base, bool enable)

  Configure the MSCM Clock Gating from the Platform Clock Gating Control Register.
- static bool SIM_HAL_GetMscmClockGate (const SIM_Type ∗base)

  Gets the MSCM Clock Gating from the Platform Clock Gating Control Register.
- static sim_eee_sram_size_t SIM_HAL_GetEeeSramSize (const SIM_Type ∗base)

  Gets the EEE SRAM size in the Flash Configuration Register 1.
- static sim_flexnvm_partition_t SIM_HAL_GetFlexNvmPartition (const SIM_Type ∗base)

  Gets the FlexNVM partition in the Flash Configuration Register 1.
- static uint32_t SIM_HAL_GetUniqueIdHigh (const SIM_Type ∗base)

  Gets the UID127_96 from Unique Identification Register High.
- static uint32_t SIM_HAL_GetUniqueIdMidHigh (const SIM_Type ∗base)

  Gets the UID95_64 from Unique Identification Register Mid High.
- static uint32_t SIM_HAL_GetUniqueIdMidLow (const SIM_Type ∗base)

  Gets the UID63_32 from Unique Identification Register Mid Low.
- static uint32_t SIM_HAL_GetUniqueIdLow (const SIM_Type ∗base)

  Gets the UID31_0 from Unique Identification Register Low.
- void SIM_HAL_GetTraceClockDefaultConfig (sim_trace_clock_config_t ∗config)

  Get the default Debug Trace clock configuration.
- void SIM_HAL_InitTraceClock (SIM_Type ∗base, const sim_trace_clock_config_t ∗config)

Initialize SIM Debug Trace.
- static void SIM_HAL_DeinitTraceClock (SIM_Type ∗base)

  De-initialize SIM Debug Trace.
- static void SIM_HAL_SetSwTriggerTrgmux (SIM_Type ∗base, bool disable)

  Sets the Software Trigger bit to TRGMUX setting.
- static uint32_t SIM_HAL_GetSwTriggerTrgmux (const SIM_Type ∗base)

  Gets the Software Trigger bit to TRGMUX.
- static void SIM_HAL_SetTClkFreq (SIM_Type ∗base, uint8_t index, uint32_t frequency)

  Sets the TClk Frequency.
- static uint32_t SIM_HAL_GetTClkFreq (SIM_Type ∗base, uint8_t index)

  Gets the TClk Frequency.

Variables

- uint32_t g_TClkFreq [NUMBER_OF_TCLK_INPUTS]

3.78.2 Data Structure Documentation

3.78.2.1 struct sim_clock_out_config_t

SIM ClockOut configuration. Implements sim_clock_out_config_t_Class.

Data Fields

- bool initialize
- bool enable
- sim_clkout_src_t source
- sim_clkout_div_t divider

Field Documentation

3.78.2.1.1 divider

```
sim_clkout_div_t divider
```

SIM ClockOut divide ratio.

3.78.2.1.2 enable

```
bool enable
```

SIM ClockOut enable.

3.78.2.1.3 initialize

```
bool initialize
```

Initialize or not the ClockOut clock.

3.78.2.1.4 source

```
sim_clkout_src_t source
```

SIM ClockOut source select.

3.78.2.2 struct sim_lpo_clock_config_t

SIM LPO Clocks configuration. Implements sim_lpo_clock_config_t_Class.

Data Fields

- bool initialize
- sim_rtc_clk_sel_src_t sourceRtcClk
- sim_lpoclk_sel_src_t sourceLpoClk
- bool enableLpo32k
- bool enableLpo1k

Field Documentation

3.78.2.2.1 enableLpo1k

```
bool enableLpo1k
```

MSCM Clock Gating Control enable.

3.78.2.2.2 enableLpo32k

```
bool enableLpo32k
```

MSCM Clock Gating Control enable.

3.78.2.2.3 initialize

```
bool initialize
```

Initialize or not the LPO clock.

3.78.2.2.4 sourceLpoClk

```
sim_lpoclk_sel_src_t sourceLpoClk
```

LPO clock source select.

3.78.2.2.5 sourceRtcClk

```
sim_rtc_clk_sel_src_t sourceRtcClk
```

RTC_CLK source select.

3.78.2.3 struct sim_plat_gate_config_t

SIM Platform Gate Clock configuration. Implements sim_plat_gate_config_t_Class.

Data Fields

- bool initialize
- bool enableMscm
- bool enableMpu
- bool enableDma
- bool enableErm
- bool enableEim

Field Documentation

3.78.2.3.1 enableDma

```
bool enableDma
```

DMA Clock Gating Control enable.

3.78.2.3.2 enableEim

```
bool enableEim
```

EIM Clock Gating Control enable.

3.78.2.3.3 enableErm

```
bool enableErm
```

ERM Clock Gating Control enable.

3.78.2.3.4 enableMpu

```
bool enableMpu
```

MPU Clock Gating Control enable.

3.78.2.3.5 enableMscm

```
bool enableMscm
```

MSCM Clock Gating Control enable.

3.78.2.3.6 initialize

```
bool initialize
```

Initialize or not the Trace clock.

3.78.2.4 struct sim_tclk_config_t

SIM Platform Gate Clock configuration. Implements sim_tclk_config_t_Class.

Data Fields

- bool initialize
- uint32_t tclkFreq [NUMBER_OF_TCLK_INPUTS]

Field Documentation

3.78.2.4.1 initialize

```
bool initialize
```

Initialize or not the Trace clock.

3.78.2.4.2 tclkFreq

```
uint32_t tclkFreq[NUMBER_OF_TCLK_INPUTS]
```

TCLKx frequency.

3.78.2.5 struct sim_trace_clock_config_t

SIM Debug Trace clock configuration. Implements sim_trace_clock_config_t_Class.

Data Fields

- bool initialize
- bool divEnable
- clock_trace_src_t source
- uint8_t divider
- bool divFraction

Field Documentation

3.78.2.5.1 divEnable

```
bool divEnable
```

Trace clock divider enable.

3.78.2.5.2 divFraction

```
bool divFraction
```

Trace clock divider fraction.

3.78.2.5.3 divider

```
uint8_t divider
```

Trace clock divider divisor.

3.78.2.5.4 initialize

```
bool initialize
```

Initialize or not the Trace clock.

3.78.2.5.5 source

```
clock_trace_src_t source
```

Trace clock select.

3.78.2.6 struct sim_clock_config_t

SIM configure structure. Implements sim_clock_config_t_Class.

Data Fields

- sim_clock_out_config_t clockOutConfig
- sim_lpo_clock_config_t lpoClockConfig
- sim_tclk_config_t tclkConfig
- sim_plat_gate_config_t platGateConfig
- sim_trace_clock_config_t traceClockConfig

Field Documentation

3.78.2.6.1 clockOutConfig

```
sim_clock_out_config_t clockOutConfig
```

Clock Out configuration.

3.78.2.6.2 lpoClockConfig

```
sim_lpo_clock_config_t lpoClockConfig
```

Low Power Clock configuration.

3.78.2.6.3 platGateConfig

```
sim_plat_gate_config_t platGateConfig
```

Platform Gate Clock configuration.

3.78.2.6.4 tclkConfig

```
sim_tclk_config_t tclkConfig
```

Platform Gate Clock configuration.

3.78.2.6.5 traceClockConfig

```
sim_trace_clock_config_t traceClockConfig
```

Trace clock configuration.

3.78.3 Macro Definition Documentation

3.78.3.1 NUMBER_OF_TCLK_INPUTS

```
#define NUMBER_OF_TCLK_INPUTS 3U
```

3.78.4 Enumeration Type Documentation

3.78.4.1 clock_trace_src_t

```
enum clock_trace_src_t
```

Debug trace clock source select Implements clock_trace_src_t_Class.

Enumerator

CLOCK_TRACE_SRC_CORE_CLK	core clock
CLOCK_TRACE_SRC_PLATFORM_CLK	platform clock

3.78.4.2 sim_adc_pretrg_sel_t

```
enum sim_adc_pretrg_sel_t
```

SIM ADCx pre-trigger select Implements sim_adc_pretrg_sel_t_Class.

Enumerator

SIM_ADC_PRETRG_SEL_PDB	PDB pre-trigger
SIM_ADC_PRETRG_SEL_TRGMUX	TRGMUX pre-trigger
SIM_ADC_PRETRG_SEL_SOFTWARE	Software pre-trigger
SIM_ADC_PRETRG_SEL_RESERVED	Reserved

3.78.4.3 sim_adc_supply_src_t

```
enum sim_adc_supply_src_t
```

Internal supplies monitored by ADC_SUPPLY Implements sim_adc_supply_src_t_Class.

Enumerator

ADC_SUPPLY_VDD	5V input VDD supply
ADC_SUPPLY_VDDA	5V input analog supply
ADC_SUPPLY_VREFH	ADC Reference Supply
ADC_SUPPLY_VDD_3V	3.3V Oscillator Regulator Output
ADC_SUPPLY_VDD_FLASH_3V	3.3V flash regulator output
ADC_SUPPLY_LV	1.2V core regulator output

3.78.4.4 sim_adc_sw_pretrg_sel_t

```
enum sim_adc_sw_pretrg_sel_t
```

SIM ADCx software pre-trigger select Implements sim_adc_sw_pretrg_sel_t_Class.

Enumerator

SIM_ADC_SW_PRETRG_SEL_DISABLED	Software pre-trigger disabled
SIM_ADC_SW_PRETRG_SEL_RESERVED0	Reserved
SIM_ADC_SW_PRETRG_SEL_RESERVED1	Reserved
SIM_ADC_SW_PRETRG_SEL_RESERVED2	Reserved
SIM_ADC_SW_PRETRG_SEL_0	Software pre-trigger 0
SIM_ADC_SW_PRETRG_SEL_1	Software pre-trigger 1
SIM_ADC_SW_PRETRG_SEL_2	Software pre-trigger 2
SIM_ADC_SW_PRETRG_SEL_3	Software pre-trigger 3

3.78.4.5 sim_adc_trg_sel_t

```
enum sim_adc_trg_sel_t
```

SIM ADCx trigger select Implements sim_adc_trg_sel_t_Class.

Enumerator

SIM_ADC_TRG_SEL_PDB	PDB output
SIM_ADC_TRG_SEL_TRGMUX	TRGMUX output

3.78.4.6 sim_clkout_div_t

```
enum sim_clkout_div_t
```

SIM CLKOUT divider Implements sim_clkout_div_t_Class.

Enumerator

SIM_CLKOUT_DIV_BY↩_1	Divided by 1
SIM_CLKOUT_DIV_BY↩_2	Divided by 2

Enumerator

SIM_CLKOUT_DIV_BY↩_3	Divided by 3
SIM_CLKOUT_DIV_BY↩_4	Divided by 4
SIM_CLKOUT_DIV_BY↩_5	Divided by 5
SIM_CLKOUT_DIV_BY↩_6	Divided by 6
SIM_CLKOUT_DIV_BY↩_7	Divided by 7
SIM_CLKOUT_DIV_BY↩_8	Divided by 8

3.78.4.7 sim_clkout_src_t

```
enum sim_clkout_src_t
```

SIM CLKOUT select Implements sim_clkout_src_t_Class.

Enumerator

SIM_CLKOUT_SEL_SYSTEM_SCG_CLKOUT	SCG CLKOUT
SIM_CLKOUT_SEL_SYSTEM_SOSC_DIV2_CLK	SOSC DIV2 CLK
SIM_CLKOUT_SEL_SYSTEM_SIRC_DIV2_CLK	SIRC DIV2 CLK
SIM_CLKOUT_SEL_SYSTEM_FIRC_DIV2_CLK	FIRC DIV2 CLK
SIM_CLKOUT_SEL_SYSTEM_SPLL_DIV2_CLK	SPLL DIV2 CLK
SIM_CLKOUT_SEL_SYSTEM_LPO_128K_CLK	LPO CLK 128 Khz
SIM_CLKOUT_SEL_SYSTEM_LPO_CLK	LPO CLK as selected by SIM LPO CLK Select
SIM_CLKOUT_SEL_SYSTEM_RTC_CLK	RTC CLK as selected by SIM CLK 32 KHz Select

3.78.4.8 sim_eee_sram_size_t

```
enum sim_eee_sram_size_t
```

SIM EEE SRAM Size Implements sim_eee_sram_size_t_Class.

Enumerator

SIM_EEE_SRAM_SIZE_4KB	SIM EEE SRAM size - 4 KB
SIM_EEE_SRAM_SIZE_2KB	SIM EEE SRAM size - 2 KB
SIM_EEE_SRAM_SIZE_1KB	SIM EEE SRAM size - 1 KB
SIM_EEE_SRAM_SIZE_512B	SIM EEE SRAM size - 512 Bytes
SIM_EEE_SRAM_SIZE_256B	SIM EEE SRAM size - 256 Bytes
SIM_EEE_SRAM_SIZE_128B	SIM EEE SRAM size - 128 Bytes
SIM_EEE_SRAM_SIZE_64B	SIM EEE SRAM size - 64 Bytes
SIM_EEE_SRAM_SIZE_32B	SIM EEE SRAM size - 32 Bytes

3.78.4.9 sim_features_t

```
enum sim_features_t
```

SIM Features Implements sim_features_t_Class.

Enumerator

SIM_FEATURE_FLEXIO	FlexIO
SIM_FEATURE_ISO_CAN_FD	ISO CAN-FD
SIM_FEATURE_SECURITY	Security

3.78.4.10 sim_flexnvm_partition_t

```
enum sim_flexnvm_partition_t
```

SIM FlexNVM partition Implements sim_flexnvm_partition_t_Class.

Enumerator

SIM_DEPART_0000	Data flash 64 KByte, EEPROM backup 0 KByte
SIM_DEPART_0011	Data flash 32 KByte, EEPROM backup 32 KByte
SIM_DEPART_0100	Data flash 0 KByte, EEPROM backup 64 KByte
SIM_DEPART_1000	Data flash 0 KByte, EEPROM backup 64 KByte
SIM_DEPART_1010	Data flash 16 KByte, EEPROM backup 48 KByte
SIM_DEPART_1011	Data flash 32 KByte, EEPROM backup 32 KByte
SIM_DEPART_1100	Data flash 64 KByte, EEPROM backup 0 KByte
SIM_DEPART_1111	Data flash 64 KByte, EEPROM backup 0 KByte

3.78.4.11 sim_ftm_ch_out_src_t

```
enum sim_ftm_ch_out_src_t
```

SIM FlexTimer x channel y output source select Implements sim_ftm_ch_out_src_t_Class.

Enumerator

SIM_FTM_CH_OUT_SRC↩_0	FlexTimer x channel y output source 0.
SIM_FTM_CH_OUT_SRC↩_1	FlexTimer x channel y output source 1.

3.78.4.12 sim_ftm_ch_src_t

```
enum sim_ftm_ch_src_t
```

SIM FlexTimer x channel y input source select Implements sim_ftm_ch_src_t_Class.

Enumerator

SIM_FTM_CH_SRC↵_0	FlexTimer x channel y input source 0.
SIM_FTM_CH_SRC↵_1	FlexTimer x channel y input source 1.
SIM_FTM_CH_SRC↵_2	FlexTimer x channel y input source 2.
SIM_FTM_CH_SRC↵_3	FlexTimer x channel y input source 3.

3.78.4.13 sim_ftm_clk_sel_t

```
enum sim_ftm_clk_sel_t
```

SIM FlexTimer external clock select Implements sim_ftm_clk_sel_t_Class.

Enumerator

SIM_FTM_CLK_SEL_00	FTM external clock driven by TCLK0 pin.
SIM_FTM_CLK_SEL_01	FTM external clock driven by TCLK1 pin.
SIM_FTM_CLK_SEL_10	FTM external clock driven by TCLK2 pin.
SIM_FTM_CLK_SEL_11	No clock input

3.78.4.14 sim_lpoclk_sel_src_t

```
enum sim_lpoclk_sel_src_t
```

SIM LPOCLKSEL clock source select Implements sim_lpoclk_sel_src_t_Class.

Enumerator

SIM_LPO_CLK_SEL_LPO_128K	
SIM_LPO_CLK_SEL_NO_CLOCK	
SIM_LPO_CLK_SEL_LPO_32K	
SIM_LPO_CLK_SEL_LPO_1K	

3.78.4.15 sim_package_t

```
enum sim_package_t
```

SIM Package Implements sim_package_t_Class.

Enumerator

SIM_PACKAGE_64_LQFP	SIM Package - 64 LQFP
SIM_PACKAGE_100_LQFP	SIM Package - 100 LQFP
SIM_PACKAGE_100_MAP_BGA	SIM Package - 100 MAP BGA

3.78.4.16 sim_pdb_bb_src_t

```
enum sim_pdb_bb_src_t
```

PDB back-to-back select Implements sim_pdb_bb_src_t_Class.

Enumerator

PDB_BACK_TO_BACK_OPTION↩ _0	PDBx ch0 back-to-back operation with ADCx COCO[7:0]
PDB_BACK_TO_BACK_OPTION↩ _1	Ch0 of PDBx back-to-back operation with COCO[7:0] of ADCx

3.78.4.17 sim_ram_size_t

```
enum sim_ram_size_t
```

SIM RAM size Implements sim_ram_size_t_Class.

Enumerator

SIM_RAM_SIZE_48KB	SIM Ram size - 48KB
SIM_RAM_SIZE_64KB	SIM Ram size - 64KB

3.78.4.18 sim_rtc_clk_sel_src_t

```
enum sim_rtc_clk_sel_src_t
```

SIM CLK32KSEL clock source select Implements sim_rtc_clk_sel_src_t_Class.

Enumerator

SIM_RTCCLK_SEL_SOSCDIV1_CLK	
SIM_RTCCLK_SEL_LPO_32K	
SIM_RTCCLK_SEL_RTC_CLKIN	
SIM_RTCCLK_SEL_FIRCDIV1_CLK	

3.78.5 Function Documentation

3.78.5.1 SIM_HAL_DeinitClkout()

```
static void SIM_HAL_DeinitClkout (
             SIM_Type * base )  [inline], [static]
```

De-initialize SIM CLKOUT.

This function disables the SIM CLKOUT.

Parameters

in	*base*	Register base address for the SIM instance. Implements SIM_HAL_DeinitClkout_Activity

3.78.5.2 SIM_HAL_DeinitTraceClock()

```
static void SIM_HAL_DeinitTraceClock (
            SIM_Type * base )  [inline], [static]
```

De-initialize SIM Debug Trace.

This function disables the SIM Debug Trace clock.

Parameters

in	*base*	Register base address for the SIM instance. Implements SIM_HAL_DeinitTraceClock_Activity

3.78.5.3 SIM_HAL_GetAdcInterleaveSel()

```
static uint8_t SIM_HAL_GetAdcInterleaveSel (
            const SIM_Type * base )  [inline], [static]
```

Get ADC interleave channel select.

This function gets value of ADC interleave channel select.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Current value. Implements SIM_HAL_GetAdcInterleaveSel_Activity

3.78.5.4 SIM_HAL_GetAdcPreTriggerMode()

```
sim_adc_pretrg_sel_t SIM_HAL_GetAdcPreTriggerMode (
            const SIM_Type * base,
            uint32_t instance )
```

Gets the ADCx pre-trigger select setting.

This function gets the ADCx pre-trigger select setting.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.

Returns

select ADCax pre-trigger select setting

3.78.5.5 SIM_HAL_GetAdcSupplyEnCmd()

```
static bool SIM_HAL_GetAdcSupplyEnCmd (
              const SIM_Type * base )  [inline], [static]
```

Get ADC Supply Enable setting.

This function gets internal supply monitoring on ADC0 channel AD21 setting.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Current selection. Implements SIM_HAL_GetAdcSupplyEnCmd_Activity

3.78.5.6 SIM_HAL_GetAdcSupplySrc()

```
static sim_adc_supply_src_t SIM_HAL_GetAdcSupplySrc (
              const SIM_Type * base )  [inline], [static]
```

Get ADC supply source.

This function gets ADC supply source.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Current selection. Implements SIM_HAL_GetAdcSupplySrc_Activity

3.78.5.7 SIM_HAL_GetAdcSwPreTriggerMode()

```
sim_adc_sw_pretrg_sel_t SIM_HAL_GetAdcSwPreTriggerMode (
              const SIM_Type * base,
              uint32_t instance )
```

Gets the ADCx software pre-trigger select setting.

This function gets the ADCx software pre-trigger select setting.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.

Returns

select ADCx pre-trigger select setting

3.78.5.8 SIM_HAL_GetAdcTriggerMode()

```
sim_adc_trg_sel_t SIM_HAL_GetAdcTriggerMode (
                const SIM_Type * base,
                uint32_t instance )
```

Gets the ADCx trigger select setting.

This function gets the ADCx trigger select setting.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.

Returns

ADCx trigger select setting

3.78.5.9 SIM_HAL_GetClkoutConfig()

```
void SIM_HAL_GetClkoutConfig (
                const SIM_Type * base,
                sim_clock_out_config_t * config )
```

Get the SIM CLKOUT clock configuration.

This function gets the CLKOUT clock configuration.

Parameters

in	*base*	Register base address for the SIM instance.
in	*config*	Pointer to the configuration structure.

3.78.5.10 SIM_HAL_GetClkoutDefaultConfig()

```
void SIM_HAL_GetClkoutDefaultConfig (
                sim_clock_out_config_t * config )
```

Get the default SIM CLKOUT clock configuration.

This function gets the default CLKOUT clock configuration.

Parameters

out	*config*	Pointer to the configuration structure.

3.78.5.11 SIM_HAL_GetDerivate()

```
static uint32_t SIM_HAL_GetDerivate (
            const SIM_Type * base )  [inline], [static]
```

Gets the Derivate from the System Device ID register (SIM_SDID).

This function gets the Derivate from System Device ID register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Id Derivate Implements SIM_HAL_GetDerivate_Activity

3.78.5.12 SIM_HAL_GetDmaClockGate()

```
static bool SIM_HAL_GetDmaClockGate (
            const SIM_Type * base )  [inline], [static]
```

Gets the DMA Clock Gate from the Platform Clock Gating Control Register.

This function gets the DMA Clock Gate in the Platform Clock Gating Control Register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

DMA Clock Gating Implements SIM_HAL_GetDmaClockGate_Activity

3.78.5.13 SIM_HAL_GetEeeSramSize()

```
static sim_eee_sram_size_t SIM_HAL_GetEeeSramSize (
            const SIM_Type * base )  [inline], [static]
```

Gets the EEE SRAM size in the Flash Configuration Register 1.

This function gets the EEE SRAM size in the Flash Configuration Register 1.

Parameters

in	*base*	Base address for current SIM instance.

Returns

EEE SRAM size Implements SIM_HAL_GetEeeSramSize_Activity

3.78.5.14 SIM_HAL_GetEimClockGate()

```
static bool SIM_HAL_GetEimClockGate (
            const SIM_Type * base )  [inline], [static]
```

Gets the EIM Clock Gate from the Platform Clock Gating Control Register.

This function gets the EIM Clock Gate in the Platform Clock Gating Control Register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

EIM Clock Gating Implements SIM_HAL_GetEimClockGate_Activity

3.78.5.15 SIM_HAL_GetErmClockGate()

```
static bool SIM_HAL_GetErmClockGate (
            const SIM_Type * base )  [inline], [static]
```

Gets the ERM Clock Gate from the Platform Clock Gating Control Register.

This function gets the ERM Clock Gate in the Platform Clock Gating Control Register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

ERM Clock Gating Implements SIM_HAL_GetErmClockGate_Activity

3.78.5.16 SIM_HAL_GetFeatures()

```
static uint32_t SIM_HAL_GetFeatures (
            const SIM_Type * base )  [inline], [static]
```

Gets the Features from System Device ID register (SIM_SDID).

This function gets the Features from System Device ID register. See sim_features_t enumeration.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Id features. Implements SIM_HAL_GetFeatures_Activity

3.78.5.17 SIM_HAL_GetFlexNvmPartition()

```
static sim_flexnvm_partition_t SIM_HAL_GetFlexNvmPartition (
             const SIM_Type * base )  [inline], [static]
```

Gets the FlexNVM partition in the Flash Configuration Register 1.

This function gets the FlexNVM partition in the Flash Configuration Register 1

Parameters

in	*base*	Base address for current SIM instance.

Returns

FlexNVM partition setting Implements SIM_HAL_GetFlexNvmPartition_Activity

3.78.5.18 SIM_HAL_GetFtmChOutSrcMode()

```
sim_ftm_ch_out_src_t SIM_HAL_GetFtmChOutSrcMode (
             const SIM_Type * base,
             uint32_t instance,
             uint8_t channel )
```

Gets the FlexTimer x channel y output source select setting.

This function gets the FlexTimer x channel y output source select setting.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.
in	*channel*	FlexTimer channel y

Returns

select FlexTimer x channel y output source select setting

3.78.5.19 SIM_HAL_GetFtmChSrcMode()

```
sim_ftm_ch_src_t SIM_HAL_GetFtmChSrcMode (
             const SIM_Type * base,
             uint32_t instance,
             uint8_t channel )
```

Gets the FlexTimer x channel y input source select setting.

This function gets the FlexTimer x channel y input source select setting.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.
in	*channel*	FlexTimer channel y

Returns

select FlexTimer x channel y input source select setting

3.78.5.20 SIM_HAL_GetFtmExternalClkPinMode()

```
sim_ftm_clk_sel_t SIM_HAL_GetFtmExternalClkPinMode (
            const SIM_Type * base,
            uint32_t instance )
```

Gets the FlexTimer x external clock pin select setting.

This function gets the FlexTimer x external clock pin select setting.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.

Returns

select FlexTimer x external clock pin select setting

3.78.5.21 SIM_HAL_GetFtmFaultSelMode()

```
uint8_t SIM_HAL_GetFtmFaultSelMode (
            const SIM_Type * base,
            uint32_t instance )
```

Gets the FlexTimer x faults select settings.

This function gets the FlexTimer x faults select settings.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.

Returns

select FlexTimer x faults select settings.

3.78.5.22 SIM_HAL_GetFtmGlobalLoad()

```
static bool SIM_HAL_GetFtmGlobalLoad (
            const SIM_Type * base ) [inline], [static]
```

Gets the FTM Global Load from the FTM Option Register 1.

This function gets the FTM Global Load from the FTM Option Register 1.

Parameters

in	*base*	Base address for current SIM instance.

Returns

enable FTM global load setting Implements SIM_HAL_GetFtmGlobalLoad_Activity

3.78.5.23 SIM_HAL_GetFtmSyncCmd()

```
static bool SIM_HAL_GetFtmSyncCmd (
            const SIM_Type * base,
            uint32_t instance )  [inline], [static]
```

Get FlexTimer x hardware trigger software synchronization setting.

This function gets FlexTimer x hardware trigger software synchronization. FTMxSYNCBIT.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	device instance.

Returns

enable hardware trigger software synchronization setting Implements SIM_HAL_GetFtmSyncCmd_Activity

3.78.5.24 SIM_HAL_GetGeneration()

```
static uint32_t SIM_HAL_GetGeneration (
            const SIM_Type * base )  [inline], [static]
```

Gets the product series Generation from System Device ID register (SIM_SDID).

This function gets the product series Generation from System Device ID register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Id generation Implements SIM_HAL_GetGeneration_Activity

3.78.5.25 SIM_HAL_GetLpo128KFreq()

```
uint32_t SIM_HAL_GetLpo128KFreq (
            const SIM_Type * base )
```

Get SIM LPO 128KHz clock frequency (LPO_128K_CLOCK).

Parameters

in	*base*	Register base address for the SIM instance.

Returns

Clock frequency, if clock is invalid, return 0.

3.78.5.26 SIM_HAL_GetLpo1kClkEnCmd()

```
static bool SIM_HAL_GetLpo1kClkEnCmd (
            const SIM_Type * base )  [inline], [static]
```

Gets the 1 kHz LPO clock Control.

This function gets the 1 kHz LPO clock enable setting.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Current selection. Implements SIM_HAL_GetLpo1kClkEnCmd_Activity

3.78.5.27 SIM_HAL_GetLpo1KFreq()

```
uint32_t SIM_HAL_GetLpo1KFreq (
            const SIM_Type * base )
```

Get SIM LPO 1KHz clock frequency (LPO_1K_CLOCK).

Parameters

in	*base*	Register base address for the SIM instance.

Returns

Clock frequency, if clock is invalid, return 0.

3.78.5.28 SIM_HAL_GetLpo32kClkEnCmd()

```
static bool SIM_HAL_GetLpo32kClkEnCmd (
            const SIM_Type * base )  [inline], [static]
```

Gets the 32 kHz LPO clock Control.

This function gets the 32 kHz LPO clock enable setting.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Current selection. Implements SIM_HAL_GetLpo32kClkEnCmd_Activity

3.78.5.29 SIM_HAL_GetLpo32KFreq()

```
uint32_t SIM_HAL_GetLpo32KFreq (
            const SIM_Type * base )
```

Get SIM LPO 32KHz clock frequency (LPO_32K_CLOCK).

Parameters

in	*base*	Register base address for the SIM instance.

Returns

Clock frequency, if clock is invalid, return 0.

3.78.5.30 SIM_HAL_GetLpoClkSrc()

```
static sim_lpoclk_sel_src_t SIM_HAL_GetLpoClkSrc (
            const SIM_Type * base ) [inline], [static]
```

Get the clock selection of LPOCLKSEL.

This function gets the clock selection of LPOCLKSEL.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Current selection. Implements SIM_HAL_GetLpoClkSrc_Activity

3.78.5.31 SIM_HAL_GetLpoFreq()

```
uint32_t SIM_HAL_GetLpoFreq (
            const SIM_Type * base )
```

Get SIM LPO clock frequency (LPO_CLOCK).

Parameters

in	*base*	Register base address for the SIM instance.

Returns

Clock frequency, if clock is invalid, return 0.

3.78.5.32 SIM_HAL_GetMpuClockGate()

```
static bool SIM_HAL_GetMpuClockGate (
            const SIM_Type * base )  [inline], [static]
```

Gets the MPU Clock Gating from the Platform Clock Gating Control Register.

This function gets the MPU Clock Gating in the Platform Clock Gating Control Register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

MPU Clock Gating Implements SIM_HAL_GetMpuClockGate_Activity

3.78.5.33 SIM_HAL_GetMscmClockGate()

```
static bool SIM_HAL_GetMscmClockGate (
            const SIM_Type * base )  [inline], [static]
```

Gets the MSCM Clock Gating from the Platform Clock Gating Control Register.

This function gets the MSCM Clock Gating in the Platform Clock Gating Control Register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

MSCM Clock Gating Implements SIM_HAL_GetMscmClockGate_Activity

3.78.5.34 SIM_HAL_GetObeCtrlCmd()

```
static bool SIM_HAL_GetObeCtrlCmd (
            const SIM_Type * base,
            uint32_t instance )  [inline], [static]
```

Gets FTM channel state.

This function gets FTM current selection.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.

Returns

Current selection. Implements SIM_HAL_GetObeCtrlCmd_Activity

3.78.5.35 SIM_HAL_GetPackage()

```
static sim_package_t SIM_HAL_GetPackage (
            const SIM_Type * base ) [inline], [static]
```

Gets the Package in System Device ID register (SIM_SDID).

This function gets the Package in System Device ID register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Id package Implements SIM_HAL_GetPackage_Activity

3.78.5.36 SIM_HAL_GetPdbBackToBackSrc()

```
static sim_pdb_bb_src_t SIM_HAL_GetPdbBackToBackSrc (
            const SIM_Type * base ) [inline], [static]
```

Get PDB back-to-back selection.

This function gets PDB back-to-back selection.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Current selection. Implements SIM_HAL_GetPdbBackToBackSrc_Activity

3.78.5.37 SIM_HAL_GetRamSize()

```
static sim_ram_size_t SIM_HAL_GetRamSize (
            const SIM_Type * base ) [inline], [static]
```

Gets RAM size.

This function gets the RAM size. The field specifies the amount of system RAM available on the device.

Parameters

in	*base*	Base address for current SIM instance.

Returns

RAM size on the device Implements SIM_HAL_GetRamSize_Activity

3.78.5.38 SIM_HAL_GetRevId()

```
static uint32_t SIM_HAL_GetRevId (
             const SIM_Type * base )  [inline], [static]
```

Gets the Revision ID in the System Device ID register (SIM_SDID).

This function gets the Revision ID in System Device ID register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

id Revision ID Implements SIM_HAL_GetRevId_Activity

3.78.5.39 SIM_HAL_GetRtcClkSrc()

```
static sim_rtc_clk_sel_src_t SIM_HAL_GetRtcClkSrc (
             const SIM_Type * base )  [inline], [static]
```

Get the clock selection of RTCCLKSEL.

This function gets the clock selection of RTCCLKSEL.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Current selection. Implements SIM_HAL_GetRtcClkSrc_Activity

3.78.5.40 SIM_HAL_GetSramLRetentionCmd()

```
static bool SIM_HAL_GetSramLRetentionCmd (
             const SIM_Type * base )  [inline], [static]
```

Get SRAM L Retention setting.

This function gets SRAM L Retention setting.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Current selection. Implements SIM_HAL_GetSramLRetentionCmd_Activity

3.78.5.41 SIM_HAL_GetSramURetentionCmd()

```
static bool SIM_HAL_GetSramURetentionCmd (
            const SIM_Type * base )  [inline], [static]
```

Get SRAM U Retention setting.

This function gets SRAM U Retention setting.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Current selection. Implements SIM_HAL_GetSramURetentionCmd_Activity

3.78.5.42 SIM_HAL_GetSubSeries()

```
static uint32_t SIM_HAL_GetSubSeries (
            const SIM_Type * base )  [inline], [static]
```

Gets the sub-series in the System Device ID register (SIM_SDID).

This function gets the sub-series in System Device ID register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Id sub-series Implements SIM_HAL_GetSubSeries_Activity

3.78.5.43 SIM_HAL_GetSwTriggerTrgmux()

```
static uint32_t SIM_HAL_GetSwTriggerTrgmux (
            const SIM_Type * base )  [inline], [static]
```

Gets the Software Trigger bit to TRGMUX.

This function gets the Software Trigger bit to TRGMUX in Miscellaneous Control register.

Parameters

in	*base*	Base address for current SIM instance.

Returns

Software Trigger bit setting Implements SIM_HAL_GetSwTriggerTrgmux_Activity

3.78.5.44 SIM_HAL_GetTClkFreq()

```
static uint32_t SIM_HAL_GetTClkFreq (
            SIM_Type * base,
            uint8_t index )  [inline], [static]
```

Gets the TClk Frequency.

This function gets the TClk Frequency.

Parameters

in	*base*	Base address for current SIM instance.
in	*index*	Index of the TClk.

Returns

frequency The configured frequency of the specified TClk Implements SIM_HAL_GetTClkFreq_Activity

3.78.5.45 SIM_HAL_GetTraceClockDefaultConfig()

```
void SIM_HAL_GetTraceClockDefaultConfig (
            sim_trace_clock_config_t * config )
```

Get the default Debug Trace clock configuration.

This function gets the default Debug Trace clock configuration.

Parameters

in	*config*	Pointer to the configuration structure.

3.78.5.46 SIM_HAL_GetUniqueIdHigh()

```
static uint32_t SIM_HAL_GetUniqueIdHigh (
            const SIM_Type * base )  [inline], [static]
```

Gets the UID127_96 from Unique Identification Register High.

This function gets the UID127_96 from Unique Identification Register High.

Parameters

in	*base*	Base address for current SIM instance.

Returns

UID127_96 setting Implements SIM_HAL_GetUniqueIdHigh_Activity

3.78.5.47 SIM_HAL_GetUniqueIdLow()

```
static uint32_t SIM_HAL_GetUniqueIdLow (
            const SIM_Type * base )  [inline], [static]
```

Gets the UID31_0 from Unique Identification Register Low.

This function gets the UID31_0 from Unique Identification Register Low.

Parameters

in	*base*	Base address for current SIM instance.

Returns

UID31_0 setting Implements SIM_HAL_GetUniqueIdLow_Activity

3.78.5.48 SIM_HAL_GetUniqueIdMidHigh()

```
static uint32_t SIM_HAL_GetUniqueIdMidHigh (
            const SIM_Type * base )  [inline], [static]
```

Gets the UID95_64 from Unique Identification Register Mid High.

This function gets the UID95_64 from Unique Identification Register Mid High.

Parameters

in	*base*	Base address for current SIM instance.

Returns

UID95_64 setting Implements SIM_HAL_GetUniqueIdMidHigh_Activity

3.78.5.49 SIM_HAL_GetUniqueIdMidLow()

```
static uint32_t SIM_HAL_GetUniqueIdMidLow (
            const SIM_Type * base )  [inline], [static]
```

Gets the UID63_32 from Unique Identification Register Mid Low.

This function gets the UID63_32 from Unique Identification Register Mid Low.

Parameters

in	*base*	Base address for current SIM instance.

Returns

UID63_32 setting Implements SIM_HAL_GetUniqueIdMidLow_Activity

3.78.5.50 SIM_HAL_InitClkout()

```
void SIM_HAL_InitClkout (
            SIM_Type * base,
            const sim_clock_out_config_t * config )
```

Initialize SIM CLKOUT.

This function enables the SIM CLKOUT clock according to the configuration.

Parameters

in	*base*	Register base address for the SIM instance.
in	*config*	Pointer to the configuration structure.

3.78.5.51 SIM_HAL_InitTraceClock()

```
void SIM_HAL_InitTraceClock (
            SIM_Type * base,
            const sim_trace_clock_config_t * config )
```

Initialize SIM Debug Trace.

This function enables the SIM Debug Trace clock according to the configuration.

Parameters

in	*base*	Register base address for the SIM instance.
in	*config*	Pointer to the configuration structure.

3.78.5.52 SIM_HAL_SetAdcInterleaveSel()

```
static void SIM_HAL_SetAdcInterleaveSel (
            SIM_Type * base,
            uint8_t setting )  [inline], [static]
```

Set ADC interleave channel select.

This function sets value of ADC interleave channel select.

Parameters

in	*base*	Base address for current SIM instance.
in	*setting*	The value to set. Implements SIM_HAL_SetAdcInterleaveSel_Activity

3.78.5.53 SIM_HAL_SetAdcPreTriggerMode()

```
void SIM_HAL_SetAdcPreTriggerMode (
            SIM_Type * base,
            uint32_t instance,
            sim_adc_pretrg_sel_t select )
```

Sets the ADCx pre-trigger select setting.

This function selects the ADCx pre-trigger source.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.
in	*select*	Pre-trigger select setting for ADCx

3.78.5.54 SIM_HAL_SetAdcSupplyEnCmd()

```
static void SIM_HAL_SetAdcSupplyEnCmd (
            SIM_Type * base,
            bool setting )  [inline], [static]
```

Set ADC Supply Enable setting.

This function sets internal supply monitoring on ADC0 channel AD21

Parameters

in	*base*	Base address for current SIM instance.
in	*setting*	The value to set. Implements SIM_HAL_SetAdcSupplyEnCmd_Activity

3.78.5.55 SIM_HAL_SetAdcSupplySrc()

```
static void SIM_HAL_SetAdcSupplySrc (
            SIM_Type * base,
            sim_adc_supply_src_t setting )  [inline], [static]
```

Set ADC Supply Enable setting.

This function sets internal supply monitoring on ADC0 channel AD21 setting.

Parameters

in	*base*	Base address for current SIM instance.
in	*setting*	The value to set. Implements SIM_HAL_SetAdcSupplySrc_Activity

3.78.5.56 SIM_HAL_SetAdcSwPreTriggerMode()

```
void SIM_HAL_SetAdcSwPreTriggerMode (
            SIM_Type * base,
            uint32_t instance,
            sim_adc_sw_pretrg_sel_t select )
```

Sets the ADCx software pre-trigger select setting.

This function selects the ADCx software pre-trigger source.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.
in	*select*	pre-trigger select setting for ADCx

3.78.5.57 SIM_HAL_SetAdcTriggerMode()

```
void SIM_HAL_SetAdcTriggerMode (
             SIM_Type * base,
             uint32_t instance,
             sim_adc_trg_sel_t select )
```

Sets the ADCx trigger select setting.

This function selects the ADCx trigger source

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.
in	*select*	Trigger select setting for ADCx

3.78.5.58 SIM_HAL_SetDmaClockGate()

```
static void SIM_HAL_SetDmaClockGate (
             SIM_Type * base,
             bool enable ) [inline], [static]
```

Set the DMA Clock Gate from the Platform Clock Gating Control Register.

This function configures the DMA Clock Gate in the Platform Clock Gating Control Register.

Parameters

in	*base*	Base address for current SIM instance.
in	*enable*	DMA clock gate enable setting Implements SIM_HAL_SetDmaClockGate_Activity

3.78.5.59 SIM_HAL_SetEimClockGate()

```
static void SIM_HAL_SetEimClockGate (
             SIM_Type * base,
             bool enable ) [inline], [static]
```

Set the EIM Clock Gate from the Platform Clock Gating Control Register.

This function configures the EIM Clock Gate in the Platform Clock Gating Control Register.

Parameters

in	*base*	Base address for current SIM instance.
in	*EIM*	clock gate enable setting Implements SIM_HAL_SetEimClockGate_Activity

3.78.5.60 SIM_HAL_SetErmClockGate()

```
static void SIM_HAL_SetErmClockGate (
             SIM_Type * base,
             bool enable )  [inline], [static]
```

Set the ERM Clock Gate from the Platform Clock Gating Control Register.

This function configures the ERM Clock Gate in the Platform Clock Gating Control Register.

Parameters

in	*base*	Base address for current SIM instance.
in	*enable*	ERM clock gate enable setting Implements SIM_HAL_SetErmClockGate_Activity

3.78.5.61 SIM_HAL_SetFtmChOutSrcMode()

```
void SIM_HAL_SetFtmChOutSrcMode (
             SIM_Type * base,
             uint32_t instance,
             uint8_t channel,
             sim_ftm_ch_out_src_t select )
```

Sets the FlexTimer x channel y output source select setting.

This function selects the FlexTimer x channel y output source.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	device instance.
in	*channel*	FlexTimer channel y
in	*select*	FlexTimer x channel y output source

3.78.5.62 SIM_HAL_SetFtmChSrcMode()

```
void SIM_HAL_SetFtmChSrcMode (
             SIM_Type * base,
             uint32_t instance,
             uint8_t channel,
             sim_ftm_ch_src_t select )
```

Sets the FlexTimer x channel y input source select setting.

This function selects the FlexTimer x channel y input source.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.
in	*channel*	FlexTimer channel y
in	*select*	FlexTimer x channel y input source

3.78.5.63 SIM_HAL_SetFtmExternalClkPinMode()

```
void SIM_HAL_SetFtmExternalClkPinMode (
            SIM_Type * base,
            uint32_t instance,
            sim_ftm_clk_sel_t select )
```

Sets the FlexTimer x external clock pin select setting.

This function selects the source of FTMx external clock pin select.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.
in	*select*	FTMx external clock pin select

3.78.5.64 SIM_HAL_SetFtmFaultSelMode()

```
void SIM_HAL_SetFtmFaultSelMode (
            SIM_Type * base,
            uint32_t instance,
            uint8_t select )
```

Sets the FlexTimer x faults select settings.

This function sets the FlexTimer x faults select settings.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.
in	*select*	FlexTimer x faults select settings.

3.78.5.65 SIM_HAL_SetFtmGlobalLoad()

```
static void SIM_HAL_SetFtmGlobalLoad (
            SIM_Type * base,
            bool enable ) [inline], [static]
```

Configure the FTM Global Load from the FTM Option Register 1.

This function configures the FTM Global Load in the FTM Option Register 1.

Parameters

in	*base*	Base address for current SIM instance.
in	*enable*	FTM global load setting Implements SIM_HAL_SetFtmGlobalLoad_Activity

3.78.5.66 SIM_HAL_SetFtmSyncCmd()

```
void SIM_HAL_SetFtmSyncCmd (
            SIM_Type * base,
            uint32_t instance,
            bool sync )
```

Set FlexTimer x hardware trigger 0 software synchronization.

This function sets FlexTimer x hardware trigger 0 software synchronization. FTMxSYNCBIT.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	device instance.
in	*sync*	Synchronize or not.

3.78.5.67 SIM_HAL_SetLpoClocks()

```
static void SIM_HAL_SetLpoClocks (
            SIM_Type * base,
            sim_lpo_clock_config_t setting )  [inline], [static]
```

Set the clock selection of LPOCLKSEL.

This function sets the clock selection of LPOCLKSEL.

Parameters

in	*base*	Base address for current SIM instance.
in	*setting*	The value to set.

Note

This function ignores initialize member Implements SIM_HAL_SetLpoClocks_Activity

3.78.5.68 SIM_HAL_SetMpuClockGate()

```
static void SIM_HAL_SetMpuClockGate (
            SIM_Type * base,
            bool enable )  [inline], [static]
```

Configure the MPU Clock Gating from the Platform Clock Gating Control Register.

This function configures the MPU Clock Gating in the Platform Clock Gating Control Register.

Parameters

in	*base*	Base address for current SIM instance.
in	*enable*	MPU clock gate enable setting Implements SIM_HAL_SetMpuClockGate_Activity

3.78.5.69 SIM_HAL_SetMscmClockGate()

```
static void SIM_HAL_SetMscmClockGate (
            SIM_Type * base,
            bool enable )  [inline], [static]
```

Configure the MSCM Clock Gating from the Platform Clock Gating Control Register.

This function configures the MSCM Clock Gating in the Platform Clock Gating Control Register.

Parameters

in	*base*	Base address for current SIM instance.
in	*enable*	MPU clock gate enable setting Implements SIM_HAL_SetMscmClockGate_Activity

3.78.5.70 SIM_HAL_SetObeCtrlCmd()

```
static void SIM_HAL_SetObeCtrlCmd (
            SIM_Type * base,
            uint32_t instance,
            bool setting )  [inline], [static]
```

Sets FTM channel state.

This function sets FTM channel state.

Parameters

in	*base*	Base address for current SIM instance.
in	*instance*	Device instance.
in	*setting*	The value to set. Implements SIM_HAL_SetObeCtrlCmd_Activity

3.78.5.71 SIM_HAL_SetPdbBackToBackSrc()

```
static void SIM_HAL_SetPdbBackToBackSrc (
            SIM_Type * base,
            sim_pdb_bb_src_t setting )  [inline], [static]
```

Set PDB back-to-back selection.

This function sets PDB back-to-back selection.

Parameters

in	*base*	Base address for current SIM instance.
in	*setting*	The value to set. Implements SIM_HAL_SetPdbBackToBackSrc_Activity

3.78.5.72 SIM_HAL_SetSramLRetentionCmd()

```
static void SIM_HAL_SetSramLRetentionCmd (
            SIM_Type * base,
            bool setting )  [inline], [static]
```

Set SRAM L Retention setting.

This function sets SRAM L Retention setting.

Parameters

in	*base*	Base address for current SIM instance.
in	*setting*	The value to set. Implements SIM_HAL_SetSramLRetentionCmd_Activity

3.78.5.73 SIM_HAL_SetSramURetentionCmd()

```
static void SIM_HAL_SetSramURetentionCmd (
            SIM_Type * base,
            bool setting )  [inline], [static]
```

Set SRAM U Retention setting.

This function sets SRAM U Retention setting.

Parameters

in	*base*	Base address for current SIM instance.
in	*setting*	The value to set. Implements SIM_HAL_SetSramURetentionCmd_Activity

3.78.5.74 SIM_HAL_SetSwTriggerTrgmux()

```
static void SIM_HAL_SetSwTriggerTrgmux (
            SIM_Type * base,
            bool disable )  [inline], [static]
```

Sets the Software Trigger bit to TRGMUX setting.

This function sets the Software Trigger bit to TRGMUX in Miscellaneous Control register.

Parameters

in	*base*	Base address for current SIM instance.
in	*disable*	Software Trigger bit Implements SIM_HAL_SetSwTriggerTrgmux_Activity

3.78.5.75 SIM_HAL_SetTClkFreq()

```
static void SIM_HAL_SetTClkFreq (
            SIM_Type * base,
            uint8_t index,
            uint32_t frequency )  [inline], [static]
```

Sets the TClk Frequency.

This function sets the TClk Frequency.

Parameters

in	*base*	Base address for current SIM instance.
in	*index*	Index of the TClk.
in	*frequency*	The frequency of the specified TClk Implements SIM_HAL_SetTClkFreq_Activity

3.78.6 Variable Documentation

3.78.6.1 g_TClkFreq

```
uint32_t g_TClkFreq[NUMBER_OF_TCLK_INPUTS]
```

3.79 System Clock Generator (SCG)

3.79.1 Detailed Description

This module covers System Clock Generator module functionality.

System Clock Generator Hardware Abstraction Layer.

The system clock generator (SCG) module provides the system clocks of the MCU. The SCG contains a phase-locked loop (PLL), a slow internal reference clock (SIRC), a fast internal reference clock (FIRC), and the system oscillator clock (SOSC).

The S32 SDK provides a HAL driver for the System Clock Generator (SCG) module. This section describes the programming interface of the SCG HAL driver. The SCG HAL driver configures the SCG (System Clock Generator).

SCG driver provides three kinds of APIs:

1. APIs for MCU system clock
2. APIs for clock source configuration

APIs for MCU system clock

MCU system clock configuration includes clock source and dividers. There are dedicated control registers for RUN mode, VLPR mode and HSRUN mode. When MCU switches to new power mode, the configuration for corresponding mode will be used. SCG driver provides the APIs to set and get the values of this registers.

Here is an example on how to configure the clock source and dividers to RUN mode:

```
//Example code for how to setup system clock source and dividers for RUN mode:

 scg_system_clock_config_t config;

 config.src     = SCG_SYSTEM_CLOCK_SRC_SIRC;
 config.divBus  = 2;
 config.divCore = 2;
 config.divSlow = 2;

 SCG_HAL_SetSystemClockConfig(SCG,
      SCG_SYSTEM_CLOCK_MODE_RUN, &config);
```

APIs for clock source configuration

SCG has four clock sources, system OSC, system PLL, fast IRC and slow IRC. For each clock source, SCG driver provides such APIs:

- API to get the default configuration structure. This configuration structure could make the clock work, for special use case, please modify this structure.
- API to initialize the clock source based on configuration structure.
- API to get the frequency of this clock source.
- API to get the asynchronous frequency of this clock source.
- API to de-initialize the clock source.

Here is an example on how to setup the clock source base on SCG HAL APIs.

```
 //Example code for how to setup system OSC.

scg_sys_osc_config_t config;

// Get default configuration.
SCG_HAL_GetSysOscDefaultConfig(&config);

// Modify according to board setting.
// ...

// Setup system OSC.
SCG_HAL_InitSysOsc(SCG, &config);

// De-init system OSC.
SCG_HAL_DeinitSysOsc(SCG);
```

Important Notes

Some configurations of the clock source can be done only when the clock source is disabled, so the function `CLOCK_HAL_InitXxx` will disable the clock source, then re-configure it and enable it. As a result, before `CL↩ OCK_HAL_InitXxx`, please make sure the clock source is not used:

- as the system clock.
- as clock for on-chip peripherals.
- as clock source for other clock source.

Data Structures

- struct scg_system_clock_config_t

  SCG system clock configuration. Implements scg_system_clock_config_t_Class. More...
- struct scg_sosc_config_t

  SCG system OSC configuration. Implements scg_sosc_config_t_Class. More...
- struct scg_sirc_config_t

  SCG slow IRC clock configuration. Implements scg_sirc_config_t_Class. More...
- struct scg_firc_config_t

  SCG fast IRC clock configuration. Implements scg_firc_config_t_Class. More...
- struct scg_spll_config_t

  SCG system PLL configuration. Implements scg_spll_config_t_Class. More...
- struct scg_rtc_config_t

  SCG RTC configuration. Implements scg_rtc_config_t_Class. More...
- struct scg_clock_mode_config_t

  SCG Clock Mode Configuration structure. Implements scg_clock_mode_config_t_Class. More...
- struct scg_clockout_config_t

  SCG ClockOut Configuration structure. Implements scg_clockout_config_t_Class. More...
- struct scg_config_t

  SCG configure structure. Implements scg_config_t_Class. More...

Enumerations

- enum scg_async_clock_type_t { SCG_ASYNC_CLOCK_DIV1, SCG_ASYNC_CLOCK_DIV2, SCG_ASYN↩ C_CLOCK_MAX }

  SCG asynchronous clock type. Implements scg_async_clock_type_t_Class.
- enum scg_async_clock_div_t {
  SCG_ASYNC_CLOCK_DISABLE = 0U, SCG_ASYNC_CLOCK_DIV_BY_1 = 1U, SCG_ASYNC_CLOCK_↩ DIV_BY_2 = 2U, SCG_ASYNC_CLOCK_DIV_BY_4 = 3U,
  SCG_ASYNC_CLOCK_DIV_BY_8 = 4U, SCG_ASYNC_CLOCK_DIV_BY_16 = 5U, SCG_ASYNC_CLOC↩ K_DIV_BY_32 = 6U, SCG_ASYNC_CLOCK_DIV_BY_64 = 7U }

  SCG asynchronous clock divider value. Implements scg_async_clock_div_t_Class.
- enum scg_sosc_monitor_mode_t { SCG_SOSC_MONITOR_DISABLE = 0U, SCG_SOSC_MONITOR_INT = SCG_SOSCCSR_SOSCCM_MASK, SCG_SOSC_MONITOR_RESET }

  SCG system OSC monitor mode. Implements scg_sosc_monitor_mode_t_Class.
- enum scg_sosc_range_t { SCG_SOSC_RANGE_LOW = 1U, SCG_SOSC_RANGE_MID = 2U, SCG_SOS↩ C_RANGE_HIGH = 3U }

  SCG OSC frequency range select Implements scg_sosc_range_t_Class.
- enum scg_sosc_gain_t { SCG_SOSC_GAIN_LOW, SCG_SOSC_GAIN_HIGH }

  SCG OSC high gain oscillator select. Implements scg_sosc_gain_t_Class.
- enum scg_sosc_ext_ref_t { SCG_SOSC_REF_EXT, SCG_SOSC_REF_OSC }

  SCG OSC external reference clock select. Implements scg_sosc_ext_ref_t_Class.
- enum scg_sirc_range_t { SCG_SIRC_RANGE_LOW, SCG_SIRC_RANGE_HIGH }

  SCG slow IRC clock frequency range. Implements scg_sirc_range_t_Class.
- enum scg_firc_range_t { SCG_FIRC_RANGE_48M, SCG_FIRC_RANGE_52M, SCG_FIRC_RANGE_56M, SCG_FIRC_RANGE_60M }

  SCG fast IRC clock frequency range. Implements scg_firc_range_t_Class.
- enum scg_spll_monitor_mode_t { SCG_SPLL_MONITOR_DISABLE = 0U, SCG_SPLL_MONITOR_INT = SCG_SPLLCSR_SPLLCM_MASK, SCG_SPLL_MONITOR_RESET }

  SCG system PLL monitor mode. Implements scg_spll_monitor_mode_t_Class.

Variables

- uint32_t g_xtal0ClkFreq
- uint32_t g_RtcClkInFreq

System Clock.

- enum scg_system_clock_type_t { SCG_SYSTEM_CLOCK_CORE, SCG_SYSTEM_CLOCK_BUS, SCG_↩ SYSTEM_CLOCK_SLOW, SCG_SYSTEM_CLOCK_MAX }

  SCG system clock type. Implements scg_system_clock_type_t_Class.
- enum scg_system_clock_src_t {
  SCG_SYSTEM_CLOCK_SRC_SYS_OSC = 1U, SCG_SYSTEM_CLOCK_SRC_SIRC = 2U, SCG_SYST↩ EM_CLOCK_SRC_FIRC = 3U, SCG_SYSTEM_CLOCK_SRC_SYS_PLL = 6U,
  SCG_SYSTEM_CLOCK_SRC_NONE }

  SCG system clock source. Implements scg_system_clock_src_t_Class.
- enum scg_system_clock_mode_t {
  SCG_SYSTEM_CLOCK_MODE_CURRENT = 0U, SCG_SYSTEM_CLOCK_MODE_RUN = 1U, SCG_SY↩ STEM_CLOCK_MODE_VLPR = 2U, SCG_SYSTEM_CLOCK_MODE_HSRUN = 3U,
  SCG_SYSTEM_CLOCK_MODE_NONE }

  SCG system clock modes. Implements scg_system_clock_mode_t_Class.

- enum scg_system_clock_div_t {
SCG_SYSTEM_CLOCK_DIV_BY_1 = 0U, SCG_SYSTEM_CLOCK_DIV_BY_2 = 1U, SCG_SYSTEM_CL↩
OCK_DIV_BY_3 = 2U, SCG_SYSTEM_CLOCK_DIV_BY_4 = 3U,
SCG_SYSTEM_CLOCK_DIV_BY_5 = 4U, SCG_SYSTEM_CLOCK_DIV_BY_6 = 5U, SCG_SYSTEM_CL↩
OCK_DIV_BY_7 = 6U, SCG_SYSTEM_CLOCK_DIV_BY_8 = 7U,
SCG_SYSTEM_CLOCK_DIV_BY_9 = 8U, SCG_SYSTEM_CLOCK_DIV_BY_10 = 9U, SCG_SYSTEM_C↩
LOCK_DIV_BY_11 = 10U, SCG_SYSTEM_CLOCK_DIV_BY_12 = 11U,
SCG_SYSTEM_CLOCK_DIV_BY_13 = 12U, SCG_SYSTEM_CLOCK_DIV_BY_14 = 13U, SCG_SYSTE↩
M_CLOCK_DIV_BY_15 = 14U, SCG_SYSTEM_CLOCK_DIV_BY_16 = 15U }

  SCG system clock divider value. Implements scg_system_clock_div_t_Class.
- status_t SCG_HAL_SetSystemClockConfig (SCG_Type ∗base, scg_system_clock_mode_t mode, scg_↩
system_clock_config_t const ∗config)

  Set the system clock configuration in specified mode.
- void SCG_HAL_GetSystemClockConfig (const SCG_Type ∗base, scg_system_clock_mode_t mode, scg_↩
system_clock_config_t ∗config)

  Get the system clock configuration for specified mode.

SCG Clockout.

- enum scg_clockout_src_t {
SCG_CLOCKOUT_SRC_SCG_SLOW = 0U, SCG_CLOCKOUT_SRC_SOSC = 1U, SCG_CLOCKOUT_S↩
RC_SIRC = 2U, SCG_CLOCKOUT_SRC_FIRC = 3U,
SCG_CLOCKOUT_SRC_SPLL = 6U }

  SCG ClockOut type. Implements scg_clockout_src_t_Class.

SCG Clockout Configuration

- static scg_clockout_src_t SCG_HAL_GetClockoutSourceSel (const SCG_Type ∗base)

  Get SCG ClockOut source select.
- static void SCG_HAL_SetClockoutSourceSel (SCG_Type ∗base, scg_clockout_src_t source)

  Set SCG ClockOut source select.

System OSC Clock.

- void SCG_HAL_GetSysOscDefaultConfig (scg_sosc_config_t ∗config)

  Get the default system OSC configuration.
- status_t SCG_HAL_InitSysOsc (SCG_Type ∗base, scg_sosc_config_t const ∗config)

  Initialize SCG system OSC.
- status_t SCG_HAL_DeinitSysOsc (SCG_Type ∗base)

  De-initialize SCG system OSC.
- uint32_t SCG_HAL_GetSysOscFreq (const SCG_Type ∗base)

  Get SCG system OSC clock frequency (SYSOSC).
- uint32_t SCG_HAL_GetSysOscAsyncFreq (const SCG_Type ∗base, scg_async_clock_type_t type)

  Get SCG asynchronous clock frequency from system OSC.

Slow IRC Clock.

- void SCG_HAL_GetSircDefaultConfig (scg_sirc_config_t ∗config)

  Get the default slow IRC clock configuration.
- status_t SCG_HAL_InitSirc (SCG_Type ∗base, const scg_sirc_config_t ∗config)

  Initialize SCG slow IRC clock.
- status_t SCG_HAL_DeinitSirc (SCG_Type ∗base)

  De-initialize SCG slow IRC.
- uint32_t SCG_HAL_GetSircFreq (const SCG_Type ∗base)

  Get SCG SIRC clock frequency.
- uint32_t SCG_HAL_GetSircAsyncFreq (const SCG_Type ∗base, scg_async_clock_type_t type)

  Get SCG asynchronous clock frequency from SIRC.

Fast IRC Clock.

- void SCG_HAL_GetFircDefaultConfig (scg_firc_config_t ∗config)

  Get the default fast IRC clock configuration.
- status_t SCG_HAL_InitFirc (SCG_Type ∗base, const scg_firc_config_t ∗config)

  Initialize SCG fast IRC clock.
- status_t SCG_HAL_DeinitFirc (SCG_Type ∗base)

  De-initialize SCG fast IRC.
- uint32_t SCG_HAL_GetFircFreq (const SCG_Type ∗base)

  Get SCG FIRC clock frequency.
- uint32_t SCG_HAL_GetFircAsyncFreq (const SCG_Type ∗base, scg_async_clock_type_t type)

  Get SCG asynchronous clock frequency from FIRC.
- static scg_system_clock_src_t SCG_HAL_GetSystemClockSrc (const SCG_Type ∗base)

  Get SCG system clock source.
- uint32_t SCG_HAL_GetSystemClockFreq (const SCG_Type ∗base, scg_system_clock_type_t type)

  Get SCG system clock frequency.

System PLL Clock.

- void SCG_HAL_GetSysPllDefaultConfig (scg_spll_config_t ∗config)

  Get the default system PLL configuration.
- status_t SCG_HAL_InitSysPll (SCG_Type ∗base, scg_spll_config_t const ∗config)

  Initialize SCG system PLL.
- status_t SCG_HAL_DeinitSysPll (SCG_Type ∗base)

  De-initialize SCG system PLL.
- uint32_t SCG_HAL_GetSysPllFreq (const SCG_Type ∗base)

  Get SCG system PLL clock frequency.
- uint32_t SCG_HAL_GetSysPllAsyncFreq (const SCG_Type ∗base, scg_async_clock_type_t type)

  Get SCG asynchronous clock frequency from system PLL.

RTC Clock.

- void SCG_HAL_SetRtcClkInFreq (SCG_Type ∗base, uint32_t frequency)

  Set SCG RTC CLKIN clock frequency.
- uint32_t SCG_HAL_GetRtcClkInFreq (SCG_Type ∗base)

  Get SCG RTC CLKIN clock frequency.

3.79.2 Data Structure Documentation

3.79.2.1 struct scg_system_clock_config_t

SCG system clock configuration. Implements scg_system_clock_config_t_Class.

Data Fields

- scg_system_clock_div_t divSlow
- scg_system_clock_div_t divBus
- scg_system_clock_div_t divCore
- scg_system_clock_src_t src

Field Documentation

3.79.2.1.1 divBus

```
scg_system_clock_div_t divBus
```

BUS clock divider.

3.79.2.1.2 divCore

```
scg_system_clock_div_t divCore
```

Core clock divider.

3.79.2.1.3 divSlow

```
scg_system_clock_div_t divSlow
```

Slow clock divider.

3.79.2.1.4 src

```
scg_system_clock_src_t src
```

System clock source.

3.79.2.2 struct scg_sosc_config_t

SCG system OSC configuration. Implements scg_sosc_config_t_Class.

Data Fields

- uint32_t freq
- scg_sosc_monitor_mode_t monitorMode
- scg_sosc_ext_ref_t extRef
- scg_sosc_gain_t gain
- scg_sosc_range_t range
- scg_async_clock_div_t div1
- scg_async_clock_div_t div2
- bool enableInStop
- bool enableInLowPower
- bool locked
- bool initialize

Field Documentation

3.79.2.2.1 div1

`scg_async_clock_div_t div1`

Divider for platform asynchronous clock.

3.79.2.2.2 div2

`scg_async_clock_div_t div2`

Divider for bus asynchronous clock.

3.79.2.2.3 enableInLowPower

`bool enableInLowPower`

System OSC is enable or not in low power mode.

3.79.2.2.4 enableInStop

`bool enableInStop`

System OSC is enable or not in stop mode.

3.79.2.2.5 extRef

`scg_sosc_ext_ref_t extRef`

System OSC External Reference Select.

3.79.2.2.6 freq

`uint32_t freq`

System OSC frequency.

3.79.2.2.7 gain

`scg_sosc_gain_t gain`

System OSC high-gain operation.

3.79.2.2.8 initialize

`bool initialize`

Initialize or not the System OSC module.

3.79.2.2.9 locked

```
bool locked
```

System OSC Control Register can be written.

3.79.2.2.10 monitorMode

```
scg_sosc_monitor_mode_t monitorMode
```

System OSC Clock monitor mode.

3.79.2.2.11 range

```
scg_sosc_range_t range
```

System OSC frequency range.

3.79.2.3 struct scg_sirc_config_t

SCG slow IRC clock configuration. Implements scg_sirc_config_t_Class.

Data Fields

- scg_sirc_range_t range
- scg_async_clock_div_t div1
- scg_async_clock_div_t div2
- bool initialize
- bool enableInStop
- bool enableInLowPower
- bool locked

Field Documentation

3.79.2.3.1 div1

```
scg_async_clock_div_t div1
```

Divider for platform asynchronous clock.

3.79.2.3.2 div2

```
scg_async_clock_div_t div2
```

Divider for bus asynchronous clock.

3.79.2.3.3 enableInLowPower

```
bool enableInLowPower
```

SIRC is enable or not in low power mode.

3.79.2.3.4 enableInStop

```
bool enableInStop
```

SIRC is enable or not in stop mode.

3.79.2.3.5 initialize

```
bool initialize
```

Initialize or not the SIRC module.

3.79.2.3.6 locked

```
bool locked
```

SIRC Control Register can be written.

3.79.2.3.7 range

```
scg_sirc_range_t range
```

Slow IRC frequency range.

3.79.2.4 struct scg_firc_config_t

SCG fast IRC clock configuration. Implements scg_firc_config_t_Class.

Data Fields

- scg_firc_range_t range
- scg_async_clock_div_t div1
- scg_async_clock_div_t div2
- bool enableInStop
- bool enableInLowPower
- bool regulator
- bool locked
- bool initialize

Field Documentation

3.79.2.4.1 div1

```
scg_async_clock_div_t div1
```

Divider for platform asynchronous clock.

3.79.2.4.2 div2

```
scg_async_clock_div_t div2
```

Divider for bus asynchronous clock.

3.79.2.4.3 enableInLowPower

```
bool enableInLowPower
```

FIRC is enable or not in lowpower mode.

3.79.2.4.4 enableInStop

```
bool enableInStop
```

FIRC is enable or not in stop mode.

3.79.2.4.5 initialize

```
bool initialize
```

Initialize or not the FIRC module.

3.79.2.4.6 locked

```
bool locked
```

FIRC Control Register can be written.

3.79.2.4.7 range

```
scg_firc_range_t range
```

Fast IRC frequency range.

3.79.2.4.8 regulator

```
bool regulator
```

FIRC regulator is enable or not.

3.79.2.5 struct scg_spll_config_t

SCG system PLL configuration. Implements scg_spll_config_t_Class.

Data Fields

- scg_spll_monitor_mode_t monitorMode
- uint8_t prediv
- uint8_t mult
- uint8_t src
- scg_async_clock_div_t div1
- scg_async_clock_div_t div2
- bool enableInStop
- bool locked
- bool initialize

Field Documentation

3.79.2.5.1 div1

```
scg_async_clock_div_t div1
```

Divider for platform asynchronous clock.

3.79.2.5.2 div2

```
scg_async_clock_div_t div2
```

Divider for bus asynchronous clock.

3.79.2.5.3 enableInStop

```
bool enableInStop
```

System PLL clock is enable or not in stop mode.

3.79.2.5.4 initialize

```
bool initialize
```

Initialize or not the System PLL module.

3.79.2.5.5 locked

```
bool locked
```

System PLL Control Register can be written.

3.79.2.5.6 monitorMode

```
scg_spll_monitor_mode_t monitorMode
```

Clock monitor mode selected.

3.79.2.5.7 mult

```
uint8_t mult
```

System PLL multiplier.

3.79.2.5.8 prediv

```
uint8_t prediv
```

PLL reference clock divider.

3.79.2.5.9 src

```
uint8_t src
```

System PLL source.

3.79.2.6 struct scg_rtc_config_t

SCG RTC configuration. Implements scg_rtc_config_t_Class.

Data Fields

- uint32_t rtcClkInFreq
- bool initialize

Field Documentation

3.79.2.6.1 initialize

```
bool initialize
```

Initialize or not the RTC.

3.79.2.6.2 rtcClkInFreq

```
uint32_t rtcClkInFreq
```

RTC_CLKIN frequency.

3.79.2.7 struct scg_clock_mode_config_t

SCG Clock Mode Configuration structure. Implements scg_clock_mode_config_t_Class.

Data Fields

- scg_system_clock_config_t rccrConfig
- scg_system_clock_config_t vccrConfig
- scg_system_clock_config_t hccrConfig
- scg_system_clock_src_t alternateClock
- bool initialize

Field Documentation

3.79.2.7.1 alternateClock

```
scg_system_clock_src_t alternateClock
```

Alternate clock used during initialization

3.79.2.7.2 hccrConfig

```
scg_system_clock_config_t hccrConfig
```

HSRUN Clock Control configuration.

3.79.2.7.3 initialize

```
bool initialize
```

Initialize or not the Clock Mode Configuration.

3.79.2.7.4 rccrConfig

```
scg_system_clock_config_t rccrConfig
```

Run Clock Control configuration.

3.79.2.7.5 vccrConfig

```
scg_system_clock_config_t vccrConfig
```

VLPR Clock Control configuration.

3.79.2.8 struct scg_clockout_config_t

SCG ClockOut Configuration structure. Implements scg_clockout_config_t_Class.

Data Fields

- scg_clockout_src_t source
- bool initialize

Field Documentation

3.79.2.8.1 initialize

```
bool initialize
```

Initialize or not the ClockOut.

3.79.2.8.2 source

```
scg_clockout_src_t source
```

ClockOut source select.

3.79.2.9 struct scg_config_t

SCG configure structure. Implements scg_config_t_Class.

Data Fields

- scg_sirc_config_t sircConfig
- scg_firc_config_t fircConfig
- scg_sosc_config_t soscConfig
- scg_spll_config_t spllConfig
- scg_rtc_config_t rtcConfig
- scg_clockout_config_t clockOutConfig
- scg_clock_mode_config_t clockModeConfig

Field Documentation

3.79.2.9.1 clockModeConfig

`scg_clock_mode_config_t clockModeConfig`

SCG Clock Mode Configuration.

3.79.2.9.2 clockOutConfig

`scg_clockout_config_t clockOutConfig`

SCG ClockOut Configuration.

3.79.2.9.3 fircConfig

`scg_firc_config_t fircConfig`

Fast internal reference clock configuration.

3.79.2.9.4 rtcConfig

`scg_rtc_config_t rtcConfig`

Real Time Clock configuration.

3.79.2.9.5 sircConfig

`scg_sirc_config_t sircConfig`

Slow internal reference clock configuration.

3.79.2.9.6 soscConfig

`scg_sosc_config_t soscConfig`

System oscillator configuration.

3.79.2.9.7 spllConfig

`scg_spll_config_t spllConfig`

System Phase locked loop configuration.

3.79.3 Enumeration Type Documentation

3.79.3.1 scg_async_clock_div_t

`enum scg_async_clock_div_t`

SCG asynchronous clock divider value. Implements scg_async_clock_div_t_Class.

Enumerator

SCG_ASYNC_CLOCK_DISABLE	Clock output is disabled.
SCG_ASYNC_CLOCK_DIV_BY_1	Divided by 1.
SCG_ASYNC_CLOCK_DIV_BY_2	Divided by 2.
SCG_ASYNC_CLOCK_DIV_BY_4	Divided by 4.
SCG_ASYNC_CLOCK_DIV_BY_8	Divided by 8.
SCG_ASYNC_CLOCK_DIV_BY_16	Divided by 16.
SCG_ASYNC_CLOCK_DIV_BY_32	Divided by 32.
SCG_ASYNC_CLOCK_DIV_BY_64	Divided by 64.

3.79.3.2 scg_async_clock_type_t

```
enum scg_async_clock_type_t
```

SCG asynchronous clock type. Implements scg_async_clock_type_t_Class.

Enumerator

SCG_ASYNC_CLOCK_DIV1	Clock divider 1
SCG_ASYNC_CLOCK_DIV2	Clock divider 2
SCG_ASYNC_CLOCK_MAX	Max value.

3.79.3.3 scg_clockout_src_t

```
enum scg_clockout_src_t
```

SCG ClockOut type. Implements scg_clockout_src_t_Class.

Enumerator

SCG_CLOCKOUT_SRC_SCG_SLOW	SCG SLOW.
SCG_CLOCKOUT_SRC_SOSC	System OSC.
SCG_CLOCKOUT_SRC_SIRC	Slow IRC.
SCG_CLOCKOUT_SRC_FIRC	Fast IRC.
SCG_CLOCKOUT_SRC_SPLL	System PLL.

3.79.3.4 scg_firc_range_t

```
enum scg_firc_range_t
```

SCG fast IRC clock frequency range. Implements scg_firc_range_t_Class.

Enumerator

SCG_FIRC_RANGE_48M	Fast IRC is trimmed to 48MHz.
SCG_FIRC_RANGE_52M	Fast IRC is trimmed to 52MHz.
SCG_FIRC_RANGE_56M	Fast IRC is trimmed to 56MHz.
SCG_FIRC_RANGE_60M	Fast IRC is trimmed to 60MHz.

3.79.3.5 scg_sirc_range_t

`enum scg_sirc_range_t`

SCG slow IRC clock frequency range. Implements scg_sirc_range_t_Class.

Enumerator

SCG_SIRC_RANGE_LOW	Slow IRC low range clock (2 MHz).
SCG_SIRC_RANGE_HIGH	Slow IRC high range clock (8 MHz).

3.79.3.6 scg_sosc_ext_ref_t

`enum scg_sosc_ext_ref_t`

SCG OSC external reference clock select. Implements scg_sosc_ext_ref_t_Class.

Enumerator

SCG_SOSC_REF_EXT	
SCG_SOSC_REF_OSC	

3.79.3.7 scg_sosc_gain_t

`enum scg_sosc_gain_t`

SCG OSC high gain oscillator select. Implements scg_sosc_gain_t_Class.

Enumerator

SCG_SOSC_GAIN_LOW	
SCG_SOSC_GAIN_HIGH	

3.79.3.8 scg_sosc_monitor_mode_t

`enum scg_sosc_monitor_mode_t`

SCG system OSC monitor mode. Implements scg_sosc_monitor_mode_t_Class.

Enumerator

SCG_SOSC_MONITOR_DISABLE	Monitor disable.
SCG_SOSC_MONITOR_INT	Interrupt when system OSC error detected.
SCG_SOSC_MONITOR_RESET	Reset when system OSC error detected.

3.79.3.9 scg_sosc_range_t

`enum scg_sosc_range_t`

SCG OSC frequency range select Implements scg_sosc_range_t_Class.

Enumerator

SCG_SOSC_RANGE_LOW	Low frequency range selected for the crystal OSC (32 kHz to 40 kHz).
SCG_SOSC_RANGE_MID	Medium frequency range selected for the crystal OSC (1 Mhz to 8 Mhz).
SCG_SOSC_RANGE_HIGH	High frequency range selected for the crystal OSC (8 Mhz to 32 Mhz).

3.79.3.10 scg_spll_monitor_mode_t

```
enum scg_spll_monitor_mode_t
```

SCG system PLL monitor mode. Implements scg_spll_monitor_mode_t_Class.

Enumerator

SCG_SPLL_MONITOR_DISABLE	Monitor disable.
SCG_SPLL_MONITOR_INT	Interrupt when system PLL error detected.
SCG_SPLL_MONITOR_RESET	Reset when system PLL error detected.

3.79.3.11 scg_system_clock_div_t

```
enum scg_system_clock_div_t
```

SCG system clock divider value. Implements scg_system_clock_div_t_Class.

Enumerator

SCG_SYSTEM_CLOCK_DIV_BY_1	Divided by 1.
SCG_SYSTEM_CLOCK_DIV_BY_2	Divided by 2.
SCG_SYSTEM_CLOCK_DIV_BY_3	Divided by 3.
SCG_SYSTEM_CLOCK_DIV_BY_4	Divided by 4.
SCG_SYSTEM_CLOCK_DIV_BY_5	Divided by 5.
SCG_SYSTEM_CLOCK_DIV_BY_6	Divided by 6.
SCG_SYSTEM_CLOCK_DIV_BY_7	Divided by 7.
SCG_SYSTEM_CLOCK_DIV_BY_8	Divided by 8.
SCG_SYSTEM_CLOCK_DIV_BY_9	Divided by 9.
SCG_SYSTEM_CLOCK_DIV_BY_10	Divided by 10.
SCG_SYSTEM_CLOCK_DIV_BY_11	Divided by 11.
SCG_SYSTEM_CLOCK_DIV_BY_12	Divided by 12.
SCG_SYSTEM_CLOCK_DIV_BY_13	Divided by 13.
SCG_SYSTEM_CLOCK_DIV_BY_14	Divided by 14.
SCG_SYSTEM_CLOCK_DIV_BY_15	Divided by 15.
SCG_SYSTEM_CLOCK_DIV_BY_16	Divided by 16.

3.79.3.12 scg_system_clock_mode_t

```
enum scg_system_clock_mode_t
```

SCG system clock modes. Implements scg_system_clock_mode_t_Class.

Enumerator

SCG_SYSTEM_CLOCK_MODE_CURRENT	Current mode.
SCG_SYSTEM_CLOCK_MODE_RUN	Run mode.
SCG_SYSTEM_CLOCK_MODE_VLPR	Very Low Power Run mode.
SCG_SYSTEM_CLOCK_MODE_HSRUN	High Speed Run mode.
SCG_SYSTEM_CLOCK_MODE_NONE	MAX value.

3.79.3.13 scg_system_clock_src_t

```
enum scg_system_clock_src_t
```

SCG system clock source. Implements scg_system_clock_src_t_Class.

Enumerator

SCG_SYSTEM_CLOCK_SRC_SYS_OSC	System OSC.
SCG_SYSTEM_CLOCK_SRC_SIRC	Slow IRC.
SCG_SYSTEM_CLOCK_SRC_FIRC	Fast IRC.
SCG_SYSTEM_CLOCK_SRC_SYS_PLL	System PLL.
SCG_SYSTEM_CLOCK_SRC_NONE	MAX value.

3.79.3.14 scg_system_clock_type_t

```
enum scg_system_clock_type_t
```

SCG system clock type. Implements scg_system_clock_type_t_Class.

Enumerator

SCG_SYSTEM_CLOCK_CORE	Core clock.
SCG_SYSTEM_CLOCK_BUS	BUS clock.
SCG_SYSTEM_CLOCK_SLOW	System slow clock.
SCG_SYSTEM_CLOCK_MAX	Max value.

3.79.4 Function Documentation

3.79.4.1 SCG_HAL_DeinitFirc()

```
status_t SCG_HAL_DeinitFirc (
             SCG_Type * base )
```

De-initialize SCG fast IRC.

This function disables the SCG fast IRC.

Parameters

in	*base*	Register base address for the SCG instance.

Returns

Status of module de-initialization

Return values

STATUS_SUCCESS	FIRC is deinitialized.
STATUS_BUSY	FIRC is used by system clock.

Note

This function can not detect whether FIRC is used by some IPs.

3.79.4.2 SCG_HAL_DeinitSirc()

```
status_t SCG_HAL_DeinitSirc (
            SCG_Type * base )
```

De-initialize SCG slow IRC.

This function disables the SCG slow IRC.

Parameters

in	*base*	Register base address for the SCG instance.

Returns

Status of module de-initialization

Return values

STATUS_SUCCESS	SIRC is deinitialized.
STATUS_BUSY	SIRC is used by system clock.

Note

This function can not detect whether SIRC is used by some IPs.

3.79.4.3 SCG_HAL_DeinitSysOsc()

```
status_t SCG_HAL_DeinitSysOsc (
            SCG_Type * base )
```

De-initialize SCG system OSC.

This function disables the SCG system OSC clock.

Parameters

in	*base*	Register base address for the SCG instance.

Returns

Status of module de-initialization

Return values

STATUS_SUCCESS	System OSC is deinitialized.
STATUS_BUSY	System OSC is used by system clock.

Note

This function can not detect whether system OSC is used by some IPs.

3.79.4.4 SCG_HAL_DeinitSysPll()

```
status_t SCG_HAL_DeinitSysPll (
            SCG_Type * base )
```

De-initialize SCG system PLL.

This function disables the SCG system PLL.

Parameters

in	*base*	Register base address for the SCG instance.

Returns

Status of module de-initialization

Return values

STATUS_SUCCESS	system PLL is deinitialized.
STATUS_BUSY	system PLL is used by system clock.

Note

This function can not detect whether system PLL is used by some IPs.

3.79.4.5 SCG_HAL_GetClockoutSourceSel()

```
static scg_clockout_src_t SCG_HAL_GetClockoutSourceSel (
            const SCG_Type * base )  [inline], [static]
```

Get SCG ClockOut source select.

This function gets the SCG clockOut source

Parameters

in	*base*	Register base address for the SCG instance.

Returns

ClockOut source. Implements SCG_HAL_GetClockoutSourceSel_Activity

3.79.4.6 SCG_HAL_GetFircAsyncFreq()

```
uint32_t SCG_HAL_GetFircAsyncFreq (
             const SCG_Type * base,
             scg_async_clock_type_t type )
```

Get SCG asynchronous clock frequency from FIRC.

Parameters

in	*base*	Register base address for the SCG instance.
in	*type*	The asynchronous clock type.

Returns

Clock frequency, if clock is invalid, return 0.

3.79.4.7 SCG_HAL_GetFircDefaultConfig()

```
void SCG_HAL_GetFircDefaultConfig (
             scg_firc_config_t * config )
```

Get the default fast IRC clock configuration.

This function gets the default fast IRC clock configuration (FIRC). The default trim coarse value and trim fine value are all 0. If updateTrim is false, then trimCoar and trimFine must be set. If updateTrim is ture, then it is not necessary to set trimCoar and trimFine.

Parameters

out	*config*	Pointer to the configuration structure.

3.79.4.8 SCG_HAL_GetFircFreq()

```
uint32_t SCG_HAL_GetFircFreq (
             const SCG_Type * base )
```

Get SCG FIRC clock frequency.

Parameters

in	*base*	Register base address for the SCG instance.

Returns

Clock frequency, if clock is invalid, return 0.

3.79.4.9 SCG_HAL_GetRtcClkInFreq()

```
uint32_t SCG_HAL_GetRtcClkInFreq (
             SCG_Type * base )
```

Get SCG RTC CLKIN clock frequency.

Parameters

in	*base*	Register base address for the SCG instance.

Returns

Clock frequency

3.79.4.10 SCG_HAL_GetSircAsyncFreq()

```
uint32_t SCG_HAL_GetSircAsyncFreq (
             const SCG_Type * base,
             scg_async_clock_type_t type )
```

Get SCG asynchronous clock frequency from SIRC.

Parameters

in	*base*	Register base address for the SCG instance.
in	*type*	The asynchronous clock type.

Returns

Clock frequency, if clock is invalid, return 0.

3.79.4.11 SCG_HAL_GetSircDefaultConfig()

```
void SCG_HAL_GetSircDefaultConfig (
             scg_sirc_config_t * config )
```

Get the default slow IRC clock configuration.

This function gets the default slow IRC clock configuration (SIRC).

Parameters

out	*config*	Pointer to the configuration structure.

3.79.4.12 SCG_HAL_GetSircFreq()

```
uint32_t SCG_HAL_GetSircFreq (
             const SCG_Type * base )
```

Get SCG SIRC clock frequency.

Parameters

in	*base*	Register base address for the SCG instance.

Returns

Clock frequency, if clock is invalid, return 0.

3.79.4.13 SCG_HAL_GetSysOscAsyncFreq()

```
uint32_t SCG_HAL_GetSysOscAsyncFreq (
             const SCG_Type * base,
             scg_async_clock_type_t type )
```

Get SCG asynchronous clock frequency from system OSC.

Parameters

in	*base*	Register base address for the SCG instance.
in	*type*	The asynchronous clock type.

Returns

Clock frequency, if clock is invalid, return 0.

3.79.4.14 SCG_HAL_GetSysOscDefaultConfig()

```
void SCG_HAL_GetSysOscDefaultConfig (
             scg_sosc_config_t * config )
```

Get the default system OSC configuration.

This function gets the default SCG system OSC configuration.

Parameters

out	*config*	Pointer to the configuration structure.

3.79.4.15 SCG_HAL_GetSysOscFreq()

```
uint32_t SCG_HAL_GetSysOscFreq (
             const SCG_Type * base )
```

Get SCG system OSC clock frequency (SYSOSC).

Parameters

in	*base*	Register base address for the SCG instance.

Returns

Clock frequency, if clock is invalid, return 0.

3.79.4.16 SCG_HAL_GetSysPllAsyncFreq()

```
uint32_t SCG_HAL_GetSysPllAsyncFreq (
             const SCG_Type * base,
             scg_async_clock_type_t type )
```

Get SCG asynchronous clock frequency from system PLL.

Parameters

in	*base*	Register base address for the SCG instance.
in	*type*	The asynchronous clock type.

Returns

Clock frequency, if clock is invalid, return 0.

3.79.4.17 SCG_HAL_GetSysPllDefaultConfig()

```
void SCG_HAL_GetSysPllDefaultConfig (
             scg_spll_config_t * config )
```

Get the default system PLL configuration.

This function gets the default SCG system PLL configuration.

Parameters

out	*config*	Pointer to the configuration structure.

3.79.4.18 SCG_HAL_GetSysPllFreq()

```
uint32_t SCG_HAL_GetSysPllFreq (
             const SCG_Type * base )
```

Get SCG system PLL clock frequency.

Parameters

in	*base*	Register base address for the SCG instance.

Returns

Clock frequency, if clock is invalid, return 0.

3.79.4.19 SCG_HAL_GetSystemClockConfig()

```
void SCG_HAL_GetSystemClockConfig (
            const SCG_Type * base,
            scg_system_clock_mode_t mode,
            scg_system_clock_config_t * config )
```

Get the system clock configuration for specified mode.

This function gets the system configuration for specified mode.

Parameters

in	*base*	Register base address for the SCG instance.
in	*mode*	specifies the mode.
out	*config*	Pointer to the configuration.

3.79.4.20 SCG_HAL_GetSystemClockFreq()

```
uint32_t SCG_HAL_GetSystemClockFreq (
            const SCG_Type * base,
            scg_system_clock_type_t type )
```

Get SCG system clock frequency.

This function gets the SCG system clock frequency, these clocks are used for core, platform, external and bus clock domains.

Parameters

in	*base*	Register base address for the SCG instance.
in	*type*	Which type of clock to get, core clock or slow clock.

Returns

Clock frequency.

3.79.4.21 SCG_HAL_GetSystemClockSrc()

```
static scg_system_clock_src_t SCG_HAL_GetSystemClockSrc (
            const SCG_Type * base ) [inline], [static]
```

Get SCG system clock source.

This function gets the SCG system clock source, these clocks are used for core, platform, external and bus clock domains.

Parameters

in	*base*	Register base address for the SCG instance.

Returns

Clock source. Implements SCG_HAL_GetSystemClockSrc_Activity

3.79.4.22 SCG_HAL_InitFirc()

```
status_t SCG_HAL_InitFirc (
             SCG_Type * base,
             const scg_firc_config_t * config )
```

Initialize SCG fast IRC clock.

This function enables the SCG fast IRC clock according to the configuration.

Parameters

in	*base*	Register base address for the SCG instance.
in	*config*	Pointer to the configuration structure.

Returns

Status of module initialization

Return values

STATUS_SUCCESS	FIRC is initialized.
STATUS_BUSY	FIRC has been enabled and used by system clock.
STATUS_ERROR	FIRC initialization routine detected an error

Note

This function can not detect whether system OSC has been enabled and used by some IPs.

3.79.4.23 SCG_HAL_InitSirc()

```
status_t SCG_HAL_InitSirc (
             SCG_Type * base,
             const scg_sirc_config_t * config )
```

Initialize SCG slow IRC clock.

This function enables the SCG slow IRC clock according to the configuration.

Parameters

in	*base*	Register base address for the SCG instance.
in	*config*	Pointer to the configuration structure.

Returns

Status of module initialization

Return values

STATUS_SUCCESS	SIRC is initialized.
STATUS_BUSY	SIRC has been enabled and used by system clock.

Note

This function can not detect whether SIRC is used by some IPs.

3.79.4.24 SCG_HAL_InitSysOsc()

```
status_t SCG_HAL_InitSysOsc (
            SCG_Type * base,
            scg_sosc_config_t const * config )
```

Initialize SCG system OSC.

This function enables the SCG system OSC clock according to the configuration.

Parameters

in	*base*	Register base address for the SCG instance.
in	*config*	Pointer to the configuration structure.

Returns

Status of module initialization

Return values

STATUS_SUCCESS	System OSC is initialized.
STATUS_BUSY	System OSC has been enabled and used by system clock.

Note

This function can not detect whether system OSC has been enabled and used by some IPs.

3.79.4.25 SCG_HAL_InitSysPll()

```
status_t SCG_HAL_InitSysPll (
            SCG_Type * base,
            scg_spll_config_t const * config )
```

Initialize SCG system PLL.

This function enables the SCG system PLL clock according to the configuration. The system PLL could use system OSC or FIRC as the clock source, please make sure the source clock is valid before this function.

Parameters

in	*base*	Register base address for the SCG instance.
in	*config*	Pointer to the configuration structure.

Returns

Status of module initialization

Return values

STATUS_SUCCESS	System PLL is initialized.
STATUS_BUSY	System PLL has been enabled and used by system clock.

Note

This function can not detect whether system PLL has been enabled and used by some IPs.

3.79.4.26 SCG_HAL_SetClockoutSourceSel()

```
static void SCG_HAL_SetClockoutSourceSel (
             SCG_Type * base,
             scg_clockout_src_t source )  [inline], [static]
```

Set SCG ClockOut source select.

This function sets the SCG ClockOut source

Parameters

in	*base*	Register base address for the SCG instance.
in	*source*	used for ClockOut Implements SCG_HAL_SetClockoutSourceSel_Activity

3.79.4.27 SCG_HAL_SetRtcClkInFreq()

```
void SCG_HAL_SetRtcClkInFreq (
             SCG_Type * base,
             uint32_t frequency )
```

Set SCG RTC CLKIN clock frequency.

Parameters

in	*base*	Register base address for the SCG instance.
in	*frequency*	The frequency of the RTC_CLKIN

3.79.4.28 SCG_HAL_SetSystemClockConfig()

```
status_t SCG_HAL_SetSystemClockConfig (
            SCG_Type * base,
            scg_system_clock_mode_t mode,
            scg_system_clock_config_t const * config )
```

Set the system clock configuration in specified mode.

This function sets the system configuration in specified mode.

Parameters

in	*base*	Register base address for the SCG instance.
in	*mode*	specifies the mode.
in	*config*	Pointer to the configuration.

Returns

Status of the system clock source setting

Return values

STATUS_SUCCESS	the new system clock source has been set
STATUS_ERROR	configuration for the new system clock source is not valid

3.79.5 Variable Documentation

3.79.5.1 g_RtcClkInFreq

```
uint32_t g_RtcClkInFreq
```

3.79.5.2 g_xtal0ClkFreq

```
uint32_t g_xtal0ClkFreq
```

3.80 System Integration Module (SIM)

3.80.1 Detailed Description

This module cover System Integration Module functionality.

Common SIM_HAL API.

The section describes the programming interface of the SIM HAL driver. The System Integration Module (SIM) provides the system control and device configuration registers. The sim_hal provides a set of API functions used to access the SIM registers including clock gate control and other configuration settings.

To configure or read a bit field call the appropriate method. The methods associated with the clock configuration are used by the Clock_Manager driver. Clock related properties can also be configured trough clock_manager configuration.

System Integration Module Hardware Abstraction Layer

Modules

- Sim_hal_s32k144
  This module cover SIM module functionality implemented for S32K144.

3.81 System Mode Controller (SMC)

3.81.1 Detailed Description

This module covers the functionality of the System Mode Controller (SMC) peripheral.

SMC HAL provides the API for reading and writing register bit-fields belonging to the SMC module.

For higher-level functionality, use the Power Manager driver.

Data Structures

- struct power_manager_user_config_t

  Power mode user configuration structure. More...
- struct smc_power_mode_protection_config_t

  Power mode protection configuration Implements smc_power_mode_protection_config_t_Class. More...
- struct smc_power_mode_config_t

  Power mode control configuration used for calling the SMC_SYS_SetPowerMode API Implements smc_power_↩ mode_config_t_Class. More...
- struct smc_version_info_t

  SMC module version number Implements smc_version_info_t_Class. More...

Enumerations

- enum power_manager_modes_t {
  POWER_MANAGER_HSRUN, POWER_MANAGER_RUN, POWER_MANAGER_VLPR, POWER_MAN↩ AGER_STOP,
  POWER_MANAGER_VLPS, POWER_MANAGER_STOP1, POWER_MANAGER_STOP2, POWER_MA↩ NAGER_MAX }

  Power modes enumeration.
- enum power_mode_stat_t {
  STAT_RUN = 0x01, STAT_STOP = 0x02, STAT_VLPR = 0x04, STAT_VLPW = 0x08,
  STAT_VLPS = 0x10, STAT_HSRUN = 0x80, STAT_INVALID = 0xFF }

  Power Modes in PMSTAT Implements power_mode_stat_t_Class.
- enum power_modes_protect_t { ALLOW_HSRUN, ALLOW_VLP, ALLOW_MAX }

  Power Modes Protection Implements power_modes_protect_t_Class.
- enum smc_run_mode_t { SMC_RUN, SMC_RESERVED_RUN, SMC_VLPR, SMC_HSRUN }

  Run mode definition Implements smc_run_mode_t_Class.
- enum smc_stop_mode_t { SMC_STOP = 0U, SMC_RESERVED_STOP1 = 1U, SMC_VLPS = 2U }

  Stop mode definition Implements smc_stop_mode_t_Class.
- enum smc_stop_option_t { SMC_STOP_RESERVED = 0x00, SMC_STOP1 = 0x01, SMC_STOP2 = 0x02 }

  STOP option Implements smc_stop_option_t_Class.

System mode controller APIs

- void SMC_HAL_GetVersion (const SMC_Type ∗const baseAddr, smc_version_info_t ∗const versionInfo)

  Get the version of the SMC module.
- status_t SMC_HAL_SetPowerMode (SMC_Type ∗const baseAddr, const smc_power_mode_config_t ∗const powerModeConfig)

  Configures the power mode.
- void SMC_HAL_SetProtectionMode (SMC_Type ∗const baseAddr, const smc_power_mode_protection_←config_t ∗const protectConfig)

  Configures all power mode protection settings.
- bool SMC_HAL_GetProtectionMode (const SMC_Type ∗const baseAddr, const power_modes_protect_←t protect)

  Gets the the current power mode protection setting.
- static void SMC_HAL_SetRunModeControl (SMC_Type ∗const baseAddr, const smc_run_mode_t runMode)

  Configures the the RUN mode control setting.
- static smc_run_mode_t SMC_HAL_GetRunModeControl (const SMC_Type ∗const baseAddr)

  Gets the current RUN mode configuration setting.
- static void SMC_HAL_SetStopModeControl (SMC_Type ∗const baseAddr, const smc_stop_mode_t stop←Mode)

  Configures the STOP mode control setting.
- static smc_stop_mode_t SMC_HAL_GetVlpsaModeControl (const SMC_Type ∗const baseAddr)

  Checks whether the last very low power stop sequence has been aborted.
- static smc_stop_mode_t SMC_HAL_GetStopModeControl (const SMC_Type ∗const baseAddr)

  Gets the current STOP mode control settings.
- static void SMC_HAL_SetStopOption (SMC_Type ∗const baseAddr, const smc_stop_option_t option)

  Configures the STOPO (Stop Option).
- static smc_stop_option_t SMC_HAL_GetStopOption (const SMC_Type ∗const baseAddr)

  Gets the configuration of the STOPO option.
- static power_mode_stat_t SMC_HAL_GetPowerModeStatus (const SMC_Type ∗const baseAddr)

  Gets the current power mode stat.

3.81.2 Data Structure Documentation

3.81.2.1 struct power_manager_user_config_t

Power mode user configuration structure.

List of power mode configuration structure members depends on power options available for the specific chip. Complete list contains: mode - S32K power mode. List of available modes is chip-specific. See power_manager←_modes_t list of modes. sleepOnExitOption - Controls whether the sleep-on-exit option value is used (when set to true) or ignored (when set to false). See sleepOnExitValue. sleepOnExitValue - When set to true, ARM core returns to sleep (S32K wait modes) or deep sleep state (S32K stop modes) after interrupt service finishes. When set to false, core stays woken-up. Implements power_manager_user_config_t_Class

Data Fields

- power_manager_modes_t powerMode
- bool sleepOnExitOption
- bool sleepOnExitValue

Field Documentation

3.81.2.1.1 powerMode

```
power_manager_modes_t powerMode
```

3.81.2.1.2 sleepOnExitOption

```
bool sleepOnExitOption
```

3.81.2.1.3 sleepOnExitValue

```
bool sleepOnExitValue
```

3.81.2.2 struct smc_power_mode_protection_config_t

Power mode protection configuration Implements smc_power_mode_protection_config_t_Class.

Data Fields

- bool vlpProt
- bool hsrunProt

Field Documentation

3.81.2.2.1 hsrunProt

```
bool hsrunProt
```

HSRUN protect

3.81.2.2.2 vlpProt

```
bool vlpProt
```

VLP protect

3.81.2.3 struct smc_power_mode_config_t

Power mode control configuration used for calling the SMC_SYS_SetPowerMode API Implements smc_power_↩
mode_config_t_Class.

Data Fields

- power_manager_modes_t powerModeName
- bool stopOption
- smc_stop_option_t stopOptionValue

Field Documentation

3.81.2.3.1 powerModeName

```
power_manager_modes_t powerModeName
```

Power mode(enum), see power_manager_modes_t

3.81.2.3.2 stopOption

```
bool stopOption
```

If STOPO option is needed

3.81.2.3.3 stopOptionValue

```
smc_stop_option_t stopOptionValue
```

STOPO option(enum), see smc_stop_option_t

3.81.2.4 struct smc_version_info_t

SMC module version number Implements smc_version_info_t_Class.

Data Fields

- uint32_t majorNumber
- uint32_t minorNumber
- uint32_t featureNumber

Field Documentation

3.81.2.4.1 featureNumber

```
uint32_t featureNumber
```

Feature Specification Number

3.81.2.4.2 majorNumber

```
uint32_t majorNumber
```

Major Version Number

3.81.2.4.3 minorNumber

```
uint32_t minorNumber
```

Minor Version Number

3.81.3 Enumeration Type Documentation

3.81.3.1 power_manager_modes_t

```
enum power_manager_modes_t
```

Power modes enumeration.

Defines power modes. Used in the power mode configuration structure (power_manager_user_config_t). From ARM core perspective, Power modes can be generally divided into run modes (High speed run, Run and Very low power run), sleep (Wait and Very low power wait) and deep sleep modes (all Stop modes). List of power modes supported by specific chip along with requirements for entering and exiting of these modes can be found in chip documentation. List of all supported power modes:

- POWER_MANAGER_HSRUN - High speed run mode.
- POWER_MANAGER_RUN - Run mode.
- POWER_MANAGER_VLPR - Very low power run mode.
- POWER_MANAGER_WAIT - Wait mode.
- POWER_MANAGER_VLPW - Very low power wait mode.
- POWER_MANAGER_STOP - Stop mode.
- POWER_MANAGER_VLPS - Very low power stop mode.
- POWER_MANAGER_PSTOP1 - Partial stop 1 mode.
- POWER_MANAGER_PSTOP2 - Partial stop 2 mode. Implements power_manager_modes_t_Class

Enumerator

POWER_MANAGER_HSRUN	High speed run mode.
POWER_MANAGER_RUN	Run mode.
POWER_MANAGER_VLPR	Very low power run mode.
POWER_MANAGER_STOP	Stop mode.
POWER_MANAGER_VLPS	Very low power stop mode.
POWER_MANAGER_STOP1	Stop 1 mode.
POWER_MANAGER_STOP2	Stop 2 mode.
POWER_MANAGER_MAX	

3.81.3.2 power_mode_stat_t

```
enum power_mode_stat_t
```

Power Modes in PMSTAT Implements power_mode_stat_t_Class.

Enumerator

STAT_RUN	0000_0001 - Current power mode is RUN
STAT_STOP	0000_0010 - Current power mode is STOP
STAT_VLPR	0000_0100 - Current power mode is VLPR

Enumerator

STAT_VLPW	0000_1000 - Current power mode is VLPW
STAT_VLPS	0001_0000 - Current power mode is VLPS
STAT_HSRUN	1000_0000 - Current power mode is HSRUN
STAT_INVALID	1111_1111 - Non-existing power mode

3.81.3.3 power_modes_protect_t

```
enum power_modes_protect_t
```

Power Modes Protection Implements power_modes_protect_t_Class.

Enumerator

ALLOW_HSRUN	Allow High Speed Run mode
ALLOW_VLP	Allow Very-Low-Power Modes
ALLOW_MAX	

3.81.3.4 smc_run_mode_t

```
enum smc_run_mode_t
```

Run mode definition Implements smc_run_mode_t_Class.

Enumerator

SMC_RUN	normal RUN mode
SMC_RESERVED_RUN	
SMC_VLPR	Very-Low-Power RUN mode
SMC_HSRUN	High Speed Run mode (HSRUN)

3.81.3.5 smc_stop_mode_t

```
enum smc_stop_mode_t
```

Stop mode definition Implements smc_stop_mode_t_Class.

Enumerator

SMC_STOP	Normal STOP mode
SMC_RESERVED_STOP1	Reserved
SMC_VLPS	Very-Low-Power STOP mode

3.81.3.6 smc_stop_option_t

```
enum smc_stop_option_t
```

STOP option Implements smc_stop_option_t_Class.

Enumerator

SMC_STOP_RESERVED	Reserved stop mode
SMC_STOP1	Stop with both system and bus clocks disabled
SMC_STOP2	Stop with system clock disabled and bus clock enabled

3.81.4 Function Documentation

3.81.4.1 SMC_HAL_GetPowerModeStatus()

```
static power_mode_stat_t SMC_HAL_GetPowerModeStatus (
            const SMC_Type *const baseAddr )  [inline], [static]
```

Gets the current power mode stat.

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.

Parameters

in	*baseAddr*	Base address for current SMC instance.

Returns

stat Current power mode stat Implements SMC_HAL_GetPowerModeStatus_Activity

3.81.4.2 SMC_HAL_GetProtectionMode()

```
bool SMC_HAL_GetProtectionMode (
            const SMC_Type *const baseAddr,
            const power_modes_protect_t protect )
```

Gets the the current power mode protection setting.

This function gets the current power mode protection settings for a specified power mode.

Parameters

baseAddr[in]	Base address for current SMC instance.
protect[in]	Power mode to set for protection

Returns

state Status of the protection setting

- true: Allowed
- false: Not allowed

3.81.4.3 SMC_HAL_GetRunModeControl()

```
static smc_run_mode_t SMC_HAL_GetRunModeControl (
            const SMC_Type *const baseAddr )  [inline], [static]
```

Gets the current RUN mode configuration setting.

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.

Parameters

in	*baseAddr*	Base address for current SMC instance.

Returns

setting Run mode configuration setting Implements SMC_HAL_GetRunModeControl_Activity

3.81.4.4 SMC_HAL_GetStopModeControl()

```
static smc_stop_mode_t SMC_HAL_GetStopModeControl (
            const SMC_Type *const baseAddr )  [inline], [static]
```

Gets the current STOP mode control settings.

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.

Parameters

in	*baseAddr*	Base address for current SMC instance.

Returns

setting Current stop mode configuration setting Implements SMC_HAL_GetStopModeControl_Activity

3.81.4.5 SMC_HAL_GetStopOption()

```
static smc_stop_option_t SMC_HAL_GetStopOption (
            const SMC_Type *const baseAddr )  [inline], [static]
```

Gets the configuration of the STOPO option.

This function gets the current STOPO option setting. See the configuration function for more details.

Parameters

in	*baseAddr*	Base address for current SMC instance.

Returns

option Current STOPO option setting Implements SMC_HAL_GetStopOption_Activity

3.81.4.6 SMC_HAL_GetVersion()

```
void SMC_HAL_GetVersion (
            const SMC_Type *const baseAddr,
            smc_version_info_t *const versionInfo )
```

Get the version of the SMC module.

Parameters

in	*baseAddr*	base address of the SMC module
out	*versionInfo*	Device Version Number

3.81.4.7 SMC_HAL_GetVlpsaModeControl()

```
static smc_stop_mode_t SMC_HAL_GetVlpsaModeControl (
            const SMC_Type *const baseAddr ) [inline], [static]
```

Checks whether the last very low power stop sequence has been aborted.

This function checks whether the last very low power stop sequence has been aborted.

Parameters

in	*baseAddr*	Base address for current SMC instance.

Returns

aborted Aborted or not Implements SMC_HAL_GetVlpsaModeControl_Activity

3.81.4.8 SMC_HAL_SetPowerMode()

```
status_t SMC_HAL_SetPowerMode (
            SMC_Type *const baseAddr,
            const smc_power_mode_config_t *const powerModeConfig )
```

Configures the power mode.

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

baseAddr	Base address for current SMC instance.
powerModeConfig	Power mode configuration structure smc_power_mode_config_t

Returns

errorCode SMC error code

Power mode transition table Specifies valid power modes for transitioning to the next mode (in comment)

3.81.4.9 SMC_HAL_SetProtectionMode()

```
void SMC_HAL_SetProtectionMode (
             SMC_Type *const baseAddr,
             const smc_power_mode_protection_config_t *const protectConfig )
```

Configures all power mode protection settings.

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

in	*baseAddr*	Base address for current SMC instance.
in	*protectConfig*	Configurations for the supported power mode protect settings • See smc_power_mode_protection_config_t for details.

3.81.4.10 SMC_HAL_SetRunModeControl()

```
static void SMC_HAL_SetRunModeControl (
             SMC_Type *const baseAddr,
             const smc_run_mode_t runMode )  [inline], [static]
```

Configures the the RUN mode control setting.

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

in	*baseAddr*	Base address for current SMC instance.
in	*runMode*	Run mode setting defined in smc_run_mode_t Implements SMC_HAL_SetRunModeControl_Activity

3.81.4.11 SMC_HAL_SetStopModeControl()

```
static void SMC_HAL_SetStopModeControl (
             SMC_Type *const baseAddr,
             const smc_stop_mode_t stopMode )  [inline], [static]
```

Configures the STOP mode control setting.

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

in	*baseAddr*	Base address for current SMC instance.
in	*stopMode*	Stop mode defined in smc_stop_mode_t Implements SMC_HAL_SetStopModeControl_Activity

3.81.4.12 SMC_HAL_SetStopOption()

```
static void SMC_HAL_SetStopOption (
            SMC_Type *const baseAddr,
            const smc_stop_option_t option )  [inline], [static]
```

Configures the STOPO (Stop Option).

It controls the type of the stop operation when STOPM=STOP. When entering Stop mode from RUN mode, the PMC, SCG and flash remain fully powered, allowing the device to wakeup almost instantaneously at the expense of higher power consumption. In STOP2, only system clocks are gated allowing peripherals running on bus clock to remain fully functional. In STOP1, both system and bus clocks are gated.

Parameters

in	*baseAddr*	Base address for current SMC instance.
in	*option*	STOPO option setting defined in smc_stop_option_t Implements SMC_HAL_SetStopOption_Activity

3.82 TRGMUX Driver

3.82.1 Detailed Description

Trigger MUX Control Peripheral Driver.

Overview

This section describes the programing interface of the TRGMUX driver. The TRGMUX driver configures the TR↩ GMUX (Trigger Mux Control). The Trigger MUX module allows software to configure the trigger inputs for various peripherals.

TRGMUX Driver model building

TRGMUX can be seen as a collection of muxes, each mux allowing to select one output from a list of input signals that are common to all muxes. The TRGMUX registers are identical as structure and all bitfields can be read/written using the TRGMUX driver API.

TRGMUX Initialization

The **TRGMUX_DRV_Init()** function is used to initialize the TRGMUX IP. The function receives as parameter a pointer to the trgmux_user_config_t structure. This structure contains a variable number of mappings between a trgmux trigger source and a trgmux target modules.

TRGMUX API

After initialization, the driver allows the reconfiguration of the source trigger for a given target module using **TR↩ GMUX_DRV_SetTrigSourceForTargetModule()**. Also, by using **TRGMUX_DRV_SetLockForTargetModule()**, a given target module can be locked, such that it cannot be updated until a reset.

Data Structures

- struct trgmux_inout_mapping_config_t

  Configuration structure for pairing source triggers with target modules. More...
- struct trgmux_user_config_t

  User configuration structure for the TRGMUX driver. More...

Functions

- status_t TRGMUX_DRV_Init (const uint32_t instance, const trgmux_user_config_t ∗const trgmuxUserConfig)

  Initialize a TRGMUX instance for operation.
- status_t TRGMUX_DRV_Deinit (const uint32_t instance)

  Reset to default values the source triggers corresponding to all target modules, if none of the target modules is locked.
- status_t TRGMUX_DRV_SetTrigSourceForTargetModule (const uint32_t instance, const trgmux_trigger_↩ source_t triggerSource, const trgmux_target_module_t targetModule)

  Configure a source trigger for a selected target module.
- trgmux_trigger_source_t TRGMUX_DRV_GetTrigSourceForTargetModule (const uint32_t instance, const trgmux_target_module_t targetModule)

  Get the source trigger configured for a target module.
- void TRGMUX_DRV_SetLockForTargetModule (const uint32_t instance, const trgmux_target_module_↩ t targetModule)

  Locks the TRGMUX register of a target module.
- bool TRGMUX_DRV_GetLockForTargetModule (const uint32_t instance, const trgmux_target_module_↩ t targetModule)

  Get the Lock bit status of the TRGMUX register of a target module.

3.82.2 Data Structure Documentation

3.82.2.1 struct trgmux_inout_mapping_config_t

Configuration structure for pairing source triggers with target modules.

Use an instance of this structure to define a TRGMUX link between a trigger source and a target module. This structure is used by the user configuration structure.

Implements : trgmux_inout_mapping_config_t_Class

Data Fields

- trgmux_trigger_source_t triggerSource
- trgmux_target_module_t targetModule
- bool lockTargetModuleReg

Field Documentation

3.82.2.1.1 lockTargetModuleReg

```
bool lockTargetModuleReg
```

if true, the LOCK bit of the target module register will be set by TRGMUX_DRV_INIT(), after the current mapping is configured

3.82.2.1.2 targetModule

```
trgmux_target_module_t targetModule
```

selects one of the TRGMUX target modules

3.82.2.1.3 triggerSource

```
trgmux_trigger_source_t triggerSource
```

selects one of the TRGMUX trigger sources

3.82.2.2 struct trgmux_user_config_t

User configuration structure for the TRGMUX driver.

Use an instance of this structure with the TRGMUX_DRV_Init() function. This enables configuration of TRGMUX with the user defined mappings between inputs (source triggers) and outputs (target modules), via a single function call.

Implements : trgmux_user_config_t_Class

Data Fields

- uint8_t numInOutMappingConfigs
- const trgmux_inout_mapping_config_t $*$ inOutMappingConfig

Field Documentation

3.82.2.2.1 inOutMappingConfig

```
const trgmux_inout_mapping_config_t* inOutMappingConfig
```

pointer to array of in-out mapping structures

3.82.2.2.2 numInOutMappingConfigs

```
uint8_t numInOutMappingConfigs
```

number of in-out mappings defined in TRGMUX configuration

3.82.3 Function Documentation

3.82.3.1 TRGMUX_DRV_Deinit()

```
status_t TRGMUX_DRV_Deinit (
            const uint32_t instance )
```

Reset to default values the source triggers corresponding to all target modules, if none of the target modules is locked.

Parameters

in	*instance*	The TRGMUX instance number.

Returns

Execution status:
STATUS_SUCCESS
STATUS_ERROR - if at least one of the target module register is locked.

3.82.3.2 TRGMUX_DRV_GetLockForTargetModule()

```
bool TRGMUX_DRV_GetLockForTargetModule (
            const uint32_t instance,
            const trgmux_target_module_t targetModule )
```

Get the Lock bit status of the TRGMUX register of a target module.

This function gets the value of the LK bit from the TRGMUX register corresponding to the selected target module.

Parameters

in	*instance*	The TRGMUX instance number.
in	*targetModule*	One of the values in the trgmux_target_module_t enumeration

Returns

true - if the selected targetModule register is locked
false - if the selected targetModule register is not locked

3.82.3.3 TRGMUX_DRV_GetTrigSourceForTargetModule()

```
trgmux_trigger_source_t TRGMUX_DRV_GetTrigSourceForTargetModule (
            const uint32_t instance,
            const trgmux_target_module_t targetModule )
```

Get the source trigger configured for a target module.

This function returns the TRGMUX source trigger linked to a selected target module.

Parameters

in	*instance*	The TRGMUX instance number.
in	*targetModule*	One of the values in the trgmux_target_module_t enumeration.

Returns

Enum value corresponding to the trigger source configured for the selected target module.

3.82.3.4 TRGMUX_DRV_Init()

```
status_t TRGMUX_DRV_Init (
            const uint32_t instance,
            const trgmux_user_config_t *const trgmuxUserConfig )
```

Initialize a TRGMUX instance for operation.

This function first resets the source triggers of all TRGMUX target modules to their default values, then configures the TRGMUX with all the user defined in-out mappings. If at least one of the target modules is locked, the function will not change any of the TRGMUX target modules and return error code. This example shows how to set up the trgmux_user_config_t parameters and how to call the TRGMUX_DRV_Init() function with the required parameters:

```
trgmux_user_config_t              trgmuxConfig;
trgmux_inout_mapping_config_t     trgmuxInoutMappingConfig[] =
{
    {TRGMUX_TRIG_SOURCE_TRGMUX_IN9,
      TRGMUX_TARGET_MODULE_DMA_CH0,      false},
    {TRGMUX_TRIG_SOURCE_FTM1_EXT_TRIG,
      TRGMUX_TARGET_MODULE_TRGMUX_OUT4, true}
};

trgmuxConfig.numInOutMappingConfigs = 2;
trgmuxConfig.inOutMappingConfig     = trgmuxInoutMappingConfig;

TRGMUX_DRV_Init(instance, &trgmuxConfig);
```

Parameters

in	*instance*	The TRGMUX instance number.
in	*trgmuxUserConfig*	Pointer to the user configuration structure.

Returns

Execution status:
STATUS_SUCCESS
STATUS_ERROR - if at least one of the target module register is locked.

3.82.3.5 TRGMUX_DRV_SetLockForTargetModule()

```
void TRGMUX_DRV_SetLockForTargetModule (
             const uint32_t instance,
             const trgmux_target_module_t targetModule )
```

Locks the TRGMUX register of a target module.

This function sets the LK bit of the TRGMUX register corresponding to the selected target module. Please note that some TRGMUX registers can contain up to 4 SEL bitfields, meaning that these registers can be used to configure up to 4 target modules independently. Because the LK bit is only one per register, the configuration of all target modules referred from that register will be locked.

Parameters

in	*instance*	The TRGMUX instance number.
in	*targetModule*	One of the values in the trgmux_target_module_t enumeration

3.82.3.6 TRGMUX_DRV_SetTrigSourceForTargetModule()

```
status_t TRGMUX_DRV_SetTrigSourceForTargetModule (
             const uint32_t instance,
             const trgmux_trigger_source_t triggerSource,
             const trgmux_target_module_t targetModule )
```

Configure a source trigger for a selected target module.

This function configures a TRGMUX link between a source trigger and a target module, if the requested target module is not locked.

Parameters

in	*instance*	The TRGMUX instance number.
in	*triggerSource*	One of the values in the trgmux_trigger_source_t enumeration
in	*targetModule*	One of the values in the trgmux_target_module_t enumeration

Returns

Execution status:
STATUS_SUCCESS
STATUS_ERROR - if requested target module is locked

3.83 TRGMUX HAL

3.83.1 Detailed Description

Trigger MUX Control Hardware Abstraction Layer.

This HAL provides low-level access to all hardware features of the TRGMUX.

Enumerations

- enum trgmux_trigger_source_t {
  TRGMUX_TRIG_SOURCE_DISABLED = 0x0U, TRGMUX_TRIG_SOURCE_VDD = 0x1U, TRGMUX_TRI↩
  G_SOURCE_TRGMUX_IN0 = 0x2U, TRGMUX_TRIG_SOURCE_TRGMUX_IN1 = 0x3U,
  TRGMUX_TRIG_SOURCE_TRGMUX_IN2 = 0x4U, TRGMUX_TRIG_SOURCE_TRGMUX_IN3 = 0x5U, T↩
  RGMUX_TRIG_SOURCE_TRGMUX_IN4 = 0x6U, TRGMUX_TRIG_SOURCE_TRGMUX_IN5 = 0x7U,
  TRGMUX_TRIG_SOURCE_TRGMUX_IN6 = 0x8U, TRGMUX_TRIG_SOURCE_TRGMUX_IN7 = 0x9U, T↩
  RGMUX_TRIG_SOURCE_TRGMUX_IN8 = 0xAU, TRGMUX_TRIG_SOURCE_TRGMUX_IN9 = 0xBU,
  TRGMUX_TRIG_SOURCE_TRGMUX_IN10 = 0xCU, TRGMUX_TRIG_SOURCE_TRGMUX_IN11 = 0xDU,
  TRGMUX_TRIG_SOURCE_CMP0_OUT = 0xEU, TRGMUX_TRIG_SOURCE_LPIT_CH0 = 0x11U,
  TRGMUX_TRIG_SOURCE_LPIT_CH1 = 0x12U, TRGMUX_TRIG_SOURCE_LPIT_CH2 = 0x13U, TRGM↩
  UX_TRIG_SOURCE_LPIT_CH3 = 0x14U, TRGMUX_TRIG_SOURCE_LPTMR0 = 0x15U,
  TRGMUX_TRIG_SOURCE_FTM0_INIT_TRIG = 0x16U, TRGMUX_TRIG_SOURCE_FTM0_EXT_TRIG =
  0x17U, TRGMUX_TRIG_SOURCE_FTM1_INIT_TRIG = 0x18U, TRGMUX_TRIG_SOURCE_FTM1_EXT↩
  _TRIG = 0x19U,
  TRGMUX_TRIG_SOURCE_FTM2_INIT_TRIG = 0x1AU, TRGMUX_TRIG_SOURCE_FTM2_EXT_TRIG =
  0x1BU, TRGMUX_TRIG_SOURCE_FTM3_INIT_TRIG = 0x1CU, TRGMUX_TRIG_SOURCE_FTM3_EXT↩
  _TRIG = 0x1DU,
  TRGMUX_TRIG_SOURCE_ADC0_SC1A_COCO = 0x1EU, TRGMUX_TRIG_SOURCE_ADC0_SC1B_C↩
  OCO = 0x1FU, TRGMUX_TRIG_SOURCE_ADC1_SC1A_COCO = 0x20U, TRGMUX_TRIG_SOURCE_A↩
  DC1_SC1B_COCO = 0x21U,
  TRGMUX_TRIG_SOURCE_PDB0_CH0_TRIG = 0x22U, TRGMUX_TRIG_SOURCE_PDB0_PULSE_OUT
  = 0x24U, TRGMUX_TRIG_SOURCE_PDB1_CH0_TRIG = 0x25U, TRGMUX_TRIG_SOURCE_PDB1_PU↩
  LSE_OUT = 0x27U,
  TRGMUX_TRIG_SOURCE_RTC_ALARM = 0x2BU, TRGMUX_TRIG_SOURCE_RTC_SECOND = 0x2CU,
  TRGMUX_TRIG_SOURCE_FLEXIO_TRIG0 = 0x2DU, TRGMUX_TRIG_SOURCE_FLEXIO_TRIG1 = 0x2↩
  EU,
  TRGMUX_TRIG_SOURCE_FLEXIO_TRIG2 = 0x2FU, TRGMUX_TRIG_SOURCE_FLEXIO_TRIG3 =
  0x30U, TRGMUX_TRIG_SOURCE_LPUART0_RX_DATA = 0x31U, TRGMUX_TRIG_SOURCE_LPUA↩
  RT0_TX_DATA = 0x32U,
  TRGMUX_TRIG_SOURCE_LPUART0_RX_IDLE = 0x33U, TRGMUX_TRIG_SOURCE_LPUART1_RX_D↩
  ATA = 0x34U, TRGMUX_TRIG_SOURCE_LPUART1_TX_DATA = 0x35U, TRGMUX_TRIG_SOURCE_L↩
  PUART1_RX_IDLE = 0x36U,
  TRGMUX_TRIG_SOURCE_LPI2C0_MASTER_TRIGGER = 0x37U, TRGMUX_TRIG_SOURCE_LPI2C0↩
  _SLAVE_TRIGGER = 0x38U, TRGMUX_TRIG_SOURCE_LPSPI0_FRAME = 0x3BU, TRGMUX_TRIG_S↩
  OURCE_LPSPI0_RX_DATA = 0x3CU,
  TRGMUX_TRIG_SOURCE_LPSPI1_FRAME = 0x3DU, TRGMUX_TRIG_SOURCE_LPSPI1_RX_DATA =
  0x3EU, TRGMUX_TRIG_SOURCE_SIM_SW_TRIG = 0x3FU }

  Describes all possible inputs (trigger sources) of the TRGMUX IP.
- enum trgmux_target_module_t {
  TRGMUX_TARGET_MODULE_DMA_CH0 = 0U, TRGMUX_TARGET_MODULE_DMA_CH1 = 1U, TRG↩
  MUX_TARGET_MODULE_DMA_CH2 = 2U, TRGMUX_TARGET_MODULE_DMA_CH3 = 3U,
  TRGMUX_TARGET_MODULE_TRGMUX_OUT0 = 4U, TRGMUX_TARGET_MODULE_TRGMUX_OUT1 =
  5U, TRGMUX_TARGET_MODULE_TRGMUX_OUT2 = 6U, TRGMUX_TARGET_MODULE_TRGMUX_O↩
  UT3 = 7U,
  TRGMUX_TARGET_MODULE_TRGMUX_OUT4 = 8U, TRGMUX_TARGET_MODULE_TRGMUX_OUT5 =

9U, TRGMUX_TARGET_MODULE_TRGMUX_OUT6 = 10U, TRGMUX_TARGET_MODULE_TRGMUX_↩
OUT7 = 11U,
TRGMUX_TARGET_MODULE_ADC0_ADHWT_TLA0 = 12U, TRGMUX_TARGET_MODULE_ADC0_AD↩
HWT_TLA1 = 13U, TRGMUX_TARGET_MODULE_ADC0_ADHWT_TLA2 = 14U, TRGMUX_TARGET_M↩
ODULE_ADC0_ADHWT_TLA3 = 15U,
TRGMUX_TARGET_MODULE_ADC1_ADHWT_TLA0 = 16U, TRGMUX_TARGET_MODULE_ADC1_AD↩
HWT_TLA1 = 17U, TRGMUX_TARGET_MODULE_ADC1_ADHWT_TLA2 = 18U, TRGMUX_TARGET_M↩
ODULE_ADC1_ADHWT_TLA3 = 19U,
TRGMUX_TARGET_MODULE_CMP0_SAMPLE_INPUT = 28U, TRGMUX_TARGET_MODULE_FTM0_↩
HWTRIG0 = 40U, TRGMUX_TARGET_MODULE_FTM0_FAULT0 = 41U, TRGMUX_TARGET_MODULE↩
_FTM0_FAULT1 = 42U,
TRGMUX_TARGET_MODULE_FTM0_FAULT2 = 43U, TRGMUX_TARGET_MODULE_FTM1_HWTRIG0 =
44U, TRGMUX_TARGET_MODULE_FTM1_FAULT0 = 45U, TRGMUX_TARGET_MODULE_FTM1_FAU↩
LT1 = 46U,
TRGMUX_TARGET_MODULE_FTM1_FAULT2 = 47U, TRGMUX_TARGET_MODULE_FTM2_HWTRIG0 =
48U, TRGMUX_TARGET_MODULE_FTM2_FAULT0 = 49U, TRGMUX_TARGET_MODULE_FTM2_FAU↩
LT1 = 50U,
TRGMUX_TARGET_MODULE_FTM2_FAULT2 = 51U, TRGMUX_TARGET_MODULE_FTM3_HWTRIG0 =
52U, TRGMUX_TARGET_MODULE_FTM3_FAULT0 = 53U, TRGMUX_TARGET_MODULE_FTM3_FAU↩
LT1 = 54U,
TRGMUX_TARGET_MODULE_FTM3_FAULT2 = 55U, TRGMUX_TARGET_MODULE_PDB0_TRG_IN =
56U, TRGMUX_TARGET_MODULE_PDB1_TRG_IN = 60U, TRGMUX_TARGET_MODULE_FLEXIO_TR↩
G_TIM0 = 68U,
TRGMUX_TARGET_MODULE_FLEXIO_TRG_TIM1 = 69U, TRGMUX_TARGET_MODULE_FLEXIO_TR↩
G_TIM2 = 70U, TRGMUX_TARGET_MODULE_FLEXIO_TRG_TIM3 = 71U, TRGMUX_TARGET_MODU↩
LE_LPIT_TRG_CH0 = 72U,
TRGMUX_TARGET_MODULE_LPIT_TRG_CH1 = 73U, TRGMUX_TARGET_MODULE_LPIT_TRG_CH2 =
74U, TRGMUX_TARGET_MODULE_LPIT_TRG_CH3 = 75U, TRGMUX_TARGET_MODULE_LPUART0↩
_TRG = 76U,
TRGMUX_TARGET_MODULE_LPUART1_TRG = 80U, TRGMUX_TARGET_MODULE_LPI2C0_TRG =
84U, TRGMUX_TARGET_MODULE_LPSPI0_TRG = 92U, TRGMUX_TARGET_MODULE_LPSPI1_TRG =
96U,
TRGMUX_TARGET_MODULE_LPTMR0_ALT0 = 100U }

Describes all possible outputs (target modules) of the TRGMUX IP.

Functions

- status_t TRGMUX_HAL_Init (TRGMUX_Type ∗const base)

  Restore the TRGMUX module to reset value.
- void TRGMUX_HAL_SetTrigSourceForTargetModule (TRGMUX_Type ∗const base, const trgmux_trigger_↩
source_t triggerSource, const trgmux_target_module_t targetModule)

  Configures a source trigger for a target module.
- trgmux_trigger_source_t TRGMUX_HAL_GetTrigSourceForTargetModule (const TRGMUX_Type ∗const base, const trgmux_target_module_t targetModule)

  Get the source trigger configured for a target module.
- void TRGMUX_HAL_SetLockForTargetModule (TRGMUX_Type ∗const base, const trgmux_target_module↩
_t targetModule)

  Lock the TRGMUX register of a target module.
- bool TRGMUX_HAL_GetLockForTargetModule (const TRGMUX_Type ∗const base, const trgmux_target_↩
module_t targetModule)

  Get the Lock bit status of the TRGMUX register of a target module.

3.83.2 Enumeration Type Documentation

3.83.2.1 trgmux_target_module_t

```
enum trgmux_target_module_t
```

Describes all possible outputs (target modules) of the TRGMUX IP.

Implements : trgmux_target_module_t_Class

Enumerator

TRGMUX_TARGET_MODULE_DMA_CH0	
TRGMUX_TARGET_MODULE_DMA_CH1	
TRGMUX_TARGET_MODULE_DMA_CH2	
TRGMUX_TARGET_MODULE_DMA_CH3	
TRGMUX_TARGET_MODULE_TRGMUX_OUT0	
TRGMUX_TARGET_MODULE_TRGMUX_OUT1	
TRGMUX_TARGET_MODULE_TRGMUX_OUT2	
TRGMUX_TARGET_MODULE_TRGMUX_OUT3	
TRGMUX_TARGET_MODULE_TRGMUX_OUT4	
TRGMUX_TARGET_MODULE_TRGMUX_OUT5	
TRGMUX_TARGET_MODULE_TRGMUX_OUT6	
TRGMUX_TARGET_MODULE_TRGMUX_OUT7	
TRGMUX_TARGET_MODULE_ADC0_ADHWT_TLA0	
TRGMUX_TARGET_MODULE_ADC0_ADHWT_TLA1	
TRGMUX_TARGET_MODULE_ADC0_ADHWT_TLA2	
TRGMUX_TARGET_MODULE_ADC0_ADHWT_TLA3	
TRGMUX_TARGET_MODULE_ADC1_ADHWT_TLA0	
TRGMUX_TARGET_MODULE_ADC1_ADHWT_TLA1	
TRGMUX_TARGET_MODULE_ADC1_ADHWT_TLA2	
TRGMUX_TARGET_MODULE_ADC1_ADHWT_TLA3	
TRGMUX_TARGET_MODULE_CMP0_SAMPLE_INPUT	
TRGMUX_TARGET_MODULE_FTM0_HWTRIG0	
TRGMUX_TARGET_MODULE_FTM0_FAULT0	
TRGMUX_TARGET_MODULE_FTM0_FAULT1	
TRGMUX_TARGET_MODULE_FTM0_FAULT2	
TRGMUX_TARGET_MODULE_FTM1_HWTRIG0	
TRGMUX_TARGET_MODULE_FTM1_FAULT0	
TRGMUX_TARGET_MODULE_FTM1_FAULT1	
TRGMUX_TARGET_MODULE_FTM1_FAULT2	
TRGMUX_TARGET_MODULE_FTM2_HWTRIG0	
TRGMUX_TARGET_MODULE_FTM2_FAULT0	
TRGMUX_TARGET_MODULE_FTM2_FAULT1	
TRGMUX_TARGET_MODULE_FTM2_FAULT2	
TRGMUX_TARGET_MODULE_FTM3_HWTRIG0	
TRGMUX_TARGET_MODULE_FTM3_FAULT0	
TRGMUX_TARGET_MODULE_FTM3_FAULT1	
TRGMUX_TARGET_MODULE_FTM3_FAULT2	
TRGMUX_TARGET_MODULE_PDB0_TRG_IN	
TRGMUX_TARGET_MODULE_PDB1_TRG_IN	
TRGMUX_TARGET_MODULE_FLEXIO_TRG_TIM0	

Enumerator

TRGMUX_TARGET_MODULE_FLEXIO_TRG_TIM1	
TRGMUX_TARGET_MODULE_FLEXIO_TRG_TIM2	
TRGMUX_TARGET_MODULE_FLEXIO_TRG_TIM3	
TRGMUX_TARGET_MODULE_LPIT_TRG_CH0	
TRGMUX_TARGET_MODULE_LPIT_TRG_CH1	
TRGMUX_TARGET_MODULE_LPIT_TRG_CH2	
TRGMUX_TARGET_MODULE_LPIT_TRG_CH3	
TRGMUX_TARGET_MODULE_LPUART0_TRG	
TRGMUX_TARGET_MODULE_LPUART1_TRG	
TRGMUX_TARGET_MODULE_LPI2C0_TRG	
TRGMUX_TARGET_MODULE_LPSPI0_TRG	
TRGMUX_TARGET_MODULE_LPSPI1_TRG	
TRGMUX_TARGET_MODULE_LPTMR0_ALT0	

3.83.2.2 trgmux_trigger_source_t

```
enum trgmux_trigger_source_t
```

Describes all possible inputs (trigger sources) of the TRGMUX IP.

Implements : trgmux_trigger_source_t_Class

Enumerator

TRGMUX_TRIG_SOURCE_DISABLED	
TRGMUX_TRIG_SOURCE_VDD	
TRGMUX_TRIG_SOURCE_TRGMUX_IN0	
TRGMUX_TRIG_SOURCE_TRGMUX_IN1	
TRGMUX_TRIG_SOURCE_TRGMUX_IN2	
TRGMUX_TRIG_SOURCE_TRGMUX_IN3	
TRGMUX_TRIG_SOURCE_TRGMUX_IN4	
TRGMUX_TRIG_SOURCE_TRGMUX_IN5	
TRGMUX_TRIG_SOURCE_TRGMUX_IN6	
TRGMUX_TRIG_SOURCE_TRGMUX_IN7	
TRGMUX_TRIG_SOURCE_TRGMUX_IN8	
TRGMUX_TRIG_SOURCE_TRGMUX_IN9	
TRGMUX_TRIG_SOURCE_TRGMUX_IN10	
TRGMUX_TRIG_SOURCE_TRGMUX_IN11	
TRGMUX_TRIG_SOURCE_CMP0_OUT	
TRGMUX_TRIG_SOURCE_LPIT_CH0	
TRGMUX_TRIG_SOURCE_LPIT_CH1	
TRGMUX_TRIG_SOURCE_LPIT_CH2	
TRGMUX_TRIG_SOURCE_LPIT_CH3	
TRGMUX_TRIG_SOURCE_LPTMR0	
TRGMUX_TRIG_SOURCE_FTM0_INIT_TRIG	
TRGMUX_TRIG_SOURCE_FTM0_EXT_TRIG	
TRGMUX_TRIG_SOURCE_FTM1_INIT_TRIG	
TRGMUX_TRIG_SOURCE_FTM1_EXT_TRIG	
TRGMUX_TRIG_SOURCE_FTM2_INIT_TRIG	

Enumerator

TRGMUX_TRIG_SOURCE_FTM2_EXT_TRIG	
TRGMUX_TRIG_SOURCE_FTM3_INIT_TRIG	
TRGMUX_TRIG_SOURCE_FTM3_EXT_TRIG	
TRGMUX_TRIG_SOURCE_ADC0_SC1A_COCO	
TRGMUX_TRIG_SOURCE_ADC0_SC1B_COCO	
TRGMUX_TRIG_SOURCE_ADC1_SC1A_COCO	
TRGMUX_TRIG_SOURCE_ADC1_SC1B_COCO	
TRGMUX_TRIG_SOURCE_PDB0_CH0_TRIG	
TRGMUX_TRIG_SOURCE_PDB0_PULSE_OUT	
TRGMUX_TRIG_SOURCE_PDB1_CH0_TRIG	
TRGMUX_TRIG_SOURCE_PDB1_PULSE_OUT	
TRGMUX_TRIG_SOURCE_RTC_ALARM	
TRGMUX_TRIG_SOURCE_RTC_SECOND	
TRGMUX_TRIG_SOURCE_FLEXIO_TRIG0	
TRGMUX_TRIG_SOURCE_FLEXIO_TRIG1	
TRGMUX_TRIG_SOURCE_FLEXIO_TRIG2	
TRGMUX_TRIG_SOURCE_FLEXIO_TRIG3	
TRGMUX_TRIG_SOURCE_LPUART0_RX_DATA	
TRGMUX_TRIG_SOURCE_LPUART0_TX_DATA	
TRGMUX_TRIG_SOURCE_LPUART0_RX_IDLE	
TRGMUX_TRIG_SOURCE_LPUART1_RX_DATA	
TRGMUX_TRIG_SOURCE_LPUART1_TX_DATA	
TRGMUX_TRIG_SOURCE_LPUART1_RX_IDLE	
TRGMUX_TRIG_SOURCE_LPI2C0_MASTER_TRIGGER	
TRGMUX_TRIG_SOURCE_LPI2C0_SLAVE_TRIGGER	
TRGMUX_TRIG_SOURCE_LPSPI0_FRAME	
TRGMUX_TRIG_SOURCE_LPSPI0_RX_DATA	
TRGMUX_TRIG_SOURCE_LPSPI1_FRAME	
TRGMUX_TRIG_SOURCE_LPSPI1_RX_DATA	
TRGMUX_TRIG_SOURCE_SIM_SW_TRIG	

3.83.3 Function Documentation

3.83.3.1 TRGMUX_HAL_GetLockForTargetModule()

```
bool TRGMUX_HAL_GetLockForTargetModule (
            const TRGMUX_Type *const base,
            const trgmux_target_module_t targetModule )
```

Get the Lock bit status of the TRGMUX register of a target module.

This function gets the value of the LK bit from the TRGMUX register corresponding to the selected target module.

Parameters

in	*base*	The TRGMUX peripheral base address
in	*targetModule*	One of the values in the trgmux_target_module_t enumeration

Returns

true or false depending on the state of the LK bit

3.83.3.2 TRGMUX_HAL_GetTrigSourceForTargetModule()

```
trgmux_trigger_source_t TRGMUX_HAL_GetTrigSourceForTargetModule (
             const TRGMUX_Type *const base,
             const trgmux_target_module_t targetModule )
```

Get the source trigger configured for a target module.

This function returns the TRGMUX source trigger linked to a selected target module.

Parameters

in	*base*	The TRGMUX peripheral base address
in	*targetModule*	One of the values in the trgmux_target_module_t enumeration

Returns

Enum value corresponding to the trigger source configured for the given target module

3.83.3.3 TRGMUX_HAL_Init()

```
status_t TRGMUX_HAL_Init (
             TRGMUX_Type *const base )
```

Restore the TRGMUX module to reset value.

This function restores the TRGMUX module to reset value.

Parameters

in	*base*	The TRGMUX peripheral base address

Returns

Execution status:
STATUS_SUCCESS
STATUS_ERROR - if at least one of the target module register is locked.

3.83.3.4 TRGMUX_HAL_SetLockForTargetModule()

```
void TRGMUX_HAL_SetLockForTargetModule (
             TRGMUX_Type *const base,
             const trgmux_target_module_t targetModule )
```

Lock the TRGMUX register of a target module.

This function sets the LK bit of the TRGMUX register corresponding to the selected target module. Please note that some TRGMUX registers can contain up to 4 SEL bitfields, meaning that these registers can be used to configure up to 4 target modules independently. Because the LK bit is only one per register, the configuration of all target modules referred from that register will be locked.

Parameters

in	*base*	The TRGMUX peripheral base address
in	*targetModule*	One of the values in the trgmux_target_module_t enumeration

3.83.3.5 TRGMUX_HAL_SetTrigSourceForTargetModule()

```
void TRGMUX_HAL_SetTrigSourceForTargetModule (
            TRGMUX_Type *const base,
            const trgmux_trigger_source_t triggerSource,
            const trgmux_target_module_t targetModule )
```

Configures a source trigger for a target module.

This function configures a TRGMUX link between a source trigger and a target module, if the requested target module is not locked.

Parameters

in	*base*	The TRGMUX peripheral base address
in	*triggerSource*	One of the values in the trgmux_trigger_source_t enumeration
in	*targetModule*	One of the values in the trgmux_target_module_t enumeration

3.84 Trigger MUX Control (TRGMUX)

3.84.1 Detailed Description

The TRGMUX introduces an extremely flexible methodology for connecting various trigger sources to multiple pins/peripherals.

The S32 SDK provides both HAL and Peripheral Drivers for the Trigger MUX Control (TRGMUX) module of S32 SDK devices.

Modules

- TRGMUX Driver

  Trigger MUX Control Peripheral Driver.
- TRGMUX HAL

  Trigger MUX Control Hardware Abstraction Layer.

3.85 WDOG Driver

3.85.1 Detailed Description

Watchdog Timer Peripheral Driver.

How to use the WDOG driver in your application

In order to be able to use the Watchdog in your application, the first thing to do is initializing it with the desired configuration. This is done by calling the **WDOG_DRV_Init** function. One of the arguments passed to this function is the configuration which will be used for the Watchdog, specified by the **wdog_user_config_t** structure.

The **wdog_user_config_t** structure allows you to configure the following:

- the clock source of the Watchdog;
- the prescaler (a fixed 256 pre-scaling of the Watchdog counter reference clock may be enabled);
- the operation modes in which the Watchdog is functional (by default, the Watchdog is not functional in Debug mode, Wait mode or Stop mode);
- the timeout value to which the Watchdog counter is compared;
- the window mode option for the refresh mechanism (by default, the window mode is disabled, but it may be enabled and a window value may be set);
- the Watchdog timeout interrupt (if enabled, after a reset-triggering event, the Watchdog first generates an interrupt request; next, the Watchdog delays 128 bus clocks before forcing a reset, to allow the interrupt service routine to perform tasks (like analyzing the stack to debug code));
- the update mechanism (by default, the Watchdog reconfiguration is enabled, but updates can be desabled)

Please note that if the updates are disabled the Watchdog cannot be later modified without forcing a reset (this implies that further calls of the **WDOG_DRV_Init**, **WDOG_DRV_Deinit** or **WDOG_DRV_SetInt** functions will lead to a reset).

As mentioned before, a timeout interrupt may be enabled by specifying it at the module initialization. The **WDOG_DRV_Init** only allows enabling/disabling the interrupt, and it does not set up the ISR to be used for the interrupt request. In order to set up a function to be called after a reset-triggering event (and also enable/disable the interrupt), the **WDOG_DRV_SetInt** function may be used. **Please note** that, due to the 128 bus clocks delay before the reset, a limited amount of job can be done in the ISR.

Example:

```
wdog_user_config_t userConfigPtr = {
   WDOG_LPO_CLOCK,           /* Use the LPO clock as source */
   { false, false, false }, /* WDOG not functional in Wait/Debug/Stop mode */
   true,                     /* Enable further updates of the WDOG configuration */
   false,                    /* Timeout interrupt disabled */
   true,                     /* Window mode enabled */
   0x100,                    /* Window value */
   0x400,                    /* Timeout value */
   false                     /* Prescaler disabled */
};

 WDOG_DRV_Init(0, &userConfigPtr);

 /* Enable the timeout interrupt and set the ISR */
 WDOG_DRV_SetInt(0, true, wdogIntHandler);

 while (1) {

     /* Do something that takes between 0x100 and 0x400 clock cycles */

     /* Refresh the counter */
     WDOG_DRV_Trigger(0);
 }

 WDOG_DRV_Deinit(0);
```

Variables

- WDOG_Type ∗const g_wdogBase [WDOG_INSTANCE_COUNT]

  Table of base addresses for WDOG instances.
- const IRQn_Type g_wdogIrqId [WDOG_INSTANCE_COUNT]

  Table to save WDOG IRQ enum numbers defined in CMSIS header file.

WDOG Driver API

- status_t WDOG_DRV_Init (uint32_t instance, const wdog_user_config_t ∗userConfigPtr)

  Initializes the WDOG driver.
- void WDOG_DRV_Deinit (uint32_t instance)

  De-initializes the WDOG driver.
- void WDOG_DRV_GetConfig (uint32_t instance, wdog_user_config_t ∗config)

  Gets the current configuration of the WDOG.
- status_t WDOG_DRV_SetInt (uint32_t instance, bool enable, void(∗handler)(void))

  Enables/Disables the WDOG timeout interrupt and sets a function to be called when a timeout interrupt is received, before reset.
- void WDOG_DRV_Trigger (uint32_t instance)

  Refreshes the WDOG counter.

3.85.2 Function Documentation

3.85.2.1 WDOG_DRV_Deinit()

```
void WDOG_DRV_Deinit (
            uint32_t instance )
```

De-initializes the WDOG driver.

Parameters

in	*instance*	WDOG peripheral instance number

3.85.2.2 WDOG_DRV_GetConfig()

```
void WDOG_DRV_GetConfig (
            uint32_t instance,
            wdog_user_config_t * config )
```

Gets the current configuration of the WDOG.

Parameters

in	*instance*	WDOG peripheral instance number
out	*config*	the current configuration

3.85.2.3 WDOG_DRV_Init()

```
status_t WDOG_DRV_Init (
            uint32_t instance,
            const wdog_user_config_t * userConfigPtr )
```

Initializes the WDOG driver.

Parameters

in	*instance*	WDOG peripheral instance number
in	*userConfigPtr*	pointer to the WDOG user configuration structure

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failed. Possible causes: previous clock source or the one specified in the configuration structure is disabled; WDOG configuration updates are not allowed.

3.85.2.4 WDOG_DRV_SetInt()

```
status_t WDOG_DRV_SetInt (
            uint32_t instance,
            bool enable,
            void(*)(void) handler )
```

Enables/Disables the WDOG timeout interrupt and sets a function to be called when a timeout interrupt is received, before reset.

Parameters

in	*instance*	WDOG peripheral instance number
in	*enable*	enable/disable interrupt
in	*handler*	timeout interrupt handler

Returns

operation status

- STATUS_SUCCESS: Operation was successful.
- STATUS_ERROR: Operation failed. Possible causes: failed to install handler; WDOG configuration updates not allowed.

3.85.2.5 WDOG_DRV_Trigger()

```
void WDOG_DRV_Trigger (
            uint32_t instance )
```

Refreshes the WDOG counter.

Parameters

in	*instance*	WDOG peripheral instance number

3.85.3 Variable Documentation

3.85.3.1 g_wdogBase

```
WDOG_Type* const g_wdogBase[WDOG_INSTANCE_COUNT]
```

Table of base addresses for WDOG instances.

3.85.3.2 g_wdogIrqId

```
const IRQn_Type g_wdogIrqId[WDOG_INSTANCE_COUNT]
```

Table to save WDOG IRQ enum numbers defined in CMSIS header file.

3.86 WDOG HAL

3.86.1 Detailed Description

Watchdog Timer Hardware Abstraction Level.

This HAL provides low-level access to all hardware features of the WDOG.

Data Structures

- struct wdog_op_mode_t

  WDOG configuration structure Implements : wdog_op_mode_t_Class. More...
- struct wdog_user_config_t

  WDOG configuration structure Implements : wdog_user_config_t_Class. More...

Macros

- #define WDOG_UNLOCK32(base)
- #define WDOG_UNLOCK16(base)
- #define WDOG_UNLOCK(base)

Enumerations

- enum wdog_clk_source_t { WDOG_BUS_CLOCK = 0x00U, WDOG_LPO_CLOCK = 0x01U, WDOG_SOS←C_CLOCK = 0x02U, WDOG_SIRC_CLOCK = 0x03U }

  Clock sources for the WDOG. Implements : wdog_clk_source_t_Class.
- enum wdog_test_mode_t { WDOG_TST_DISABLED = 0x00U, WDOG_TST_USER = 0x01U, WDOG_TS←T_LOW = 0x02U, WDOG_TST_HIGH = 0x03U }

  Test modes for the WDOG. Implements : wdog_test_mode_t_Class.

WDOG Common Configurations

- void WDOG_HAL_Init (WDOG_Type ∗base)

  Initializes the WDOG.
- static bool WDOG_HAL_IsEnabled (const WDOG_Type ∗base)

  Verifies if the WDOG is enabled.
- static void WDOG_HAL_Enable (WDOG_Type ∗base)

  Enables the WDOG.
- static void WDOG_HAL_Disable (WDOG_Type ∗base)

  Disables the WDOG.
- static void WDOG_HAL_Trigger (WDOG_Type ∗base)

  Refreshes the WDOG counter.
- void WDOG_HAL_Config (WDOG_Type ∗base, const wdog_user_config_t ∗config)

  Configures all WDOG registers.
- wdog_user_config_t WDOG_HAL_GetConfig (const WDOG_Type ∗base)

  Gets the current WDOG configuration.
- static void WDOG_HAL_SetWindow (WDOG_Type ∗base, bool enable)

  Enables/Disables window mode.
- static bool WDOG_HAL_GetInt (const WDOG_Type ∗base)

Gets Interrupt Flag (FLG) status.
- static void WDOG_HAL_ClearInt (WDOG_Type ∗base)

  Clears the Interrupt Flag.
- static void WDOG_HAL_SetPrescaler (WDOG_Type ∗base, bool enable)

  Enables/Disables prescaler.
- static void WDOG_HAL_SetClockSource (WDOG_Type ∗base, wdog_clk_source_t clkSource)

  Selects the clock source used by the WDOG.
- static void WDOG_HAL_SetInt (WDOG_Type ∗base, bool enable)

  Enables/Disables WDOG interrupt.
- static bool WDOG_HAL_IsUpdateEnabled (const WDOG_Type ∗base)

  Verifies if the WDOG updates are allowed.
- static void WDOG_HAL_SetUpdate (WDOG_Type ∗base, bool enable)

  Enables/Disables WDOG updates.
- static wdog_test_mode_t WDOG_HAL_GetTestMode (const WDOG_Type ∗base)

  Gets the WDOG test mode.
- static void WDOG_HAL_SetTestMode (WDOG_Type ∗base, wdog_test_mode_t testMode)

  Changes the WDOG test mode.
- static void WDOG_HAL_SetDebug (WDOG_Type ∗base, bool enable)

  Enables/Disables WDOG in debug mode.
- static void WDOG_HAL_SetWait (WDOG_Type ∗base, bool enable)

  Enables/Disables WDOG in wait mode.
- static void WDOG_HAL_SetStop (WDOG_Type ∗base, bool enable)

  Enables/Disables WDOG in stop mode.
- static uint16_t WDOG_HAL_GetCounter (const WDOG_Type ∗base)

  Gets the value of the WDOG counter.
- static void WDOG_HAL_SetCounter (WDOG_Type ∗base, uint32_t counter)

  Sets the value of the WDOG counter.
- static uint16_t WDOG_HAL_GetTimeout (const WDOG_Type ∗base)

  Gets the value of the WDOG timeout.
- static void WDOG_HAL_SetTimeout (WDOG_Type ∗base, uint16_t timeout)

  Sets the value of the WDOG timeout.
- static void WDOG_HAL_SetWindowValue (WDOG_Type ∗base, uint16_t window)

  Sets the value of the WDOG window.
- static bool WDOG_HAL_IsUnlocked (const WDOG_Type ∗base)

  Checks if the WDOG is unlocked.
- static bool WDOG_HAL_IsReconfigurationComplete (const WDOG_Type ∗base)

  Checks if the last configuration of the WDOG was successful.
- static void WDOG_HAL_SetCmd32 (WDOG_Type ∗base, bool enable)

  Enables/Disables support for 32-bit refresh/unlock command write words.

3.86.2 Data Structure Documentation

3.86.2.1 struct wdog_op_mode_t

WDOG configuration structure Implements : wdog_op_mode_t_Class.

Data Fields

- bool wait
- bool stop
- bool debug

Field Documentation

3.86.2.1.1 debug

```
bool debug
```

Debug mode

3.86.2.1.2 stop

```
bool stop
```

Stop mode

3.86.2.1.3 wait

```
bool wait
```

Wait mode

3.86.2.2 struct wdog_user_config_t

WDOG configuration structure Implements : wdog_user_config_t_Class.

Data Fields

- wdog_clk_source_t clkSource
- wdog_op_mode_t opMode
- bool updateEnable
- bool intEnable
- bool winEnable
- uint32_t windowValue
- uint32_t timeoutValue
- bool prescalerEnable

Field Documentation

3.86.2.2.1 clkSource

```
wdog_clk_source_t clkSource
```

The clock source of the WDOG

3.86.2.2.2 intEnable

```
bool intEnable
```

If true, an interrupt request is generated before reset

3.86.2.2.3 opMode

```
wdog_op_mode_t opMode
```

The modes in which the WDOG is functional

3.86.2.2.4 prescalerEnable

```
bool prescalerEnable
```

If true, a fixed 256 prescaling of the counter reference clock is enabled

3.86.2.2.5 timeoutValue

```
uint32_t timeoutValue
```

The timeout value

3.86.2.2.6 updateEnable

```
bool updateEnable
```

If true, further updates of the WDOG are enabled

3.86.2.2.7 windowValue

```
uint32_t windowValue
```

The window value

3.86.2.2.8 winEnable

```
bool winEnable
```

If true, window mode is enabled

3.86.3 Macro Definition Documentation

3.86.3.1 WDOG_UNLOCK

```
#define WDOG_UNLOCK(
                base )
```

Value:

```
{ \
        if (((base)->CS & WDOG_CS_CMD32EN_MASK) != 0U) \
        { \
            WDOG_UNLOCK32(base); \
        } \
        else \
        { \
            WDOG_UNLOCK16(base); \
        } \
    }
```

3.86.3.2 WDOG_UNLOCK16

```
#define WDOG_UNLOCK16(
              base )
```

Value:

```
{ \
        (base)->CNT = FEATURE_WDOG_UNLOCK16_FIRST_VALUE; \
        (void)(base)->CNT; \
        (base)->CNT = FEATURE_WDOG_UNLOCK16_SECOND_VALUE; \
        (void)(base)->CNT; \
    }
```

3.86.3.3 WDOG_UNLOCK32

```
#define WDOG_UNLOCK32(
              base )
```

Value:

```
{ \
        (base)->CNT = FEATURE_WDOG_UNLOCK_VALUE; \
        (void)(base)->CNT; \
    }
```

3.86.4 Enumeration Type Documentation

3.86.4.1 wdog_clk_source_t

```
enum wdog_clk_source_t
```

Clock sources for the WDOG. Implements : wdog_clk_source_t_Class.

Enumerator

WDOG_BUS_CLOCK	Bus clock
WDOG_LPO_CLOCK	LPO clock
WDOG_SOSC_CLOCK	SOSC clock
WDOG_SIRC_CLOCK	SIRC clock

3.86.4.2 wdog_test_mode_t

```
enum wdog_test_mode_t
```

Test modes for the WDOG. Implements : wdog_test_mode_t_Class.

Enumerator

WDOG_TST_DISABLED	Test mode disabled
WDOG_TST_USER	User mode enabled. (Test mode disabled.)
WDOG_TST_LOW	Test mode enabled, only the low byte is used.
WDOG_TST_HIGH	Test mode enabled, only the high byte is used.

3.86.5 Function Documentation

3.86.5.1 WDOG_HAL_ClearInt()

```
static void WDOG_HAL_ClearInt (
             WDOG_Type * base )  [inline], [static]
```

Clears the Interrupt Flag.

This function clears the Interrupt Flag (FLG).

Parameters

in	*base*	WDOG base pointer. Implements : WDOG_HAL_ClearInt_Activity

3.86.5.2 WDOG_HAL_Config()

```
void WDOG_HAL_Config (
             WDOG_Type * base,
             const wdog_user_config_t * config )
```

Configures all WDOG registers.

This function configures all WDOG registers.

Parameters

in	*base*	WDOG base pointer.
in	*config*	the new configuration to be used.

3.86.5.3 WDOG_HAL_Disable()

```
static void WDOG_HAL_Disable (
             WDOG_Type * base )  [inline], [static]
```

Disables the WDOG.

This function diables the WDOG.

Parameters

in	*base*	WDOG base pointer. Implements : WDOG_HAL_Disable_Activity

3.86.5.4 WDOG_HAL_Enable()

```
static void WDOG_HAL_Enable (
             WDOG_Type * base )  [inline], [static]
```

Enables the WDOG.

This function enables the WDOG.

Parameters

in	*base*	WDOG base pointer. Implements : WDOG_HAL_Enable_Activity

3.86.5.5 WDOG_HAL_GetConfig()

```
wdog_user_config_t WDOG_HAL_GetConfig (
            const WDOG_Type * base )
```

Gets the current WDOG configuration.

This function gets the current WDOG configuration

Parameters

in	*base*	WDOG base pointer.

Returns

the current WDOG configuration

3.86.5.6 WDOG_HAL_GetCounter()

```
static uint16_t WDOG_HAL_GetCounter (
            const WDOG_Type * base ) [inline], [static]
```

Gets the value of the WDOG counter.

This function gets the value of the WDOG counter.

Parameters

in	*base*	WDOG base pointer.

Returns

the value of the WDOG counter. Implements : WDOG_HAL_GetCounter_Activity

3.86.5.7 WDOG_HAL_GetInt()

```
static bool WDOG_HAL_GetInt (
            const WDOG_Type * base ) [inline], [static]
```

Gets Interrupt Flag (FLG) status.

This function verifies if an interrupt occurred.

Parameters

in	*base*	WDOG base pointer.

Returns

the state of the Interrupt Flag:

- false: no interrupt occurred
- true: an interrupt occurred Implements : WDOG_HAL_GetInt_Activity

3.86.5.8 WDOG_HAL_GetTestMode()

```
static wdog_test_mode_t WDOG_HAL_GetTestMode (
            const WDOG_Type * base )  [inline], [static]
```

Gets the WDOG test mode.

This function verifies the test mode of the WDOG.

Parameters

in	*base*	WDOG base pointer.

Returns

the test mode:

- 0x00U: Test mode disabled
- 0x01U: User mode enabled (test mode disabled)
- 0x02U: Test mode enabled (for the low byte)
- 0x03U: Test mode enabled (for the high byte) Implements : WDOG_HAL_GetTestMode_Activity

3.86.5.9 WDOG_HAL_GetTimeout()

```
static uint16_t WDOG_HAL_GetTimeout (
            const WDOG_Type * base )  [inline], [static]
```

Gets the value of the WDOG timeout.

This function gets the value of the WDOG timeout.

Parameters

in	*base*	WDOG base pointer.

Returns

the value of the WDOG timeout. Implements : WDOG_HAL_GetTimeout_Activity

3.86.5.10 WDOG_HAL_Init()

```
void WDOG_HAL_Init (
            WDOG_Type * base )
```

Initializes the WDOG.

This function initializes the WDOG to known state.

Parameters

in	*base*	WDOG base pointer.

3.86.5.11 WDOG_HAL_IsEnabled()

```
static bool WDOG_HAL_IsEnabled (
            const WDOG_Type * base )  [inline], [static]
```

Verifies if the WDOG is enabled.

This function verifies the state of the WDOG.

Parameters

in	*base*	WDOG base pointer.

Returns

the state of the WDOG:

- false: WDOG is disabled
- true: WDOG is enabled Implements : WDOG_HAL_IsEnabled_Activity

3.86.5.12 WDOG_HAL_IsReconfigurationComplete()

```
static bool WDOG_HAL_IsReconfigurationComplete (
            const WDOG_Type * base )  [inline], [static]
```

Checks if the last configuration of the WDOG was successful.

This function checks if the last configuration of the WDOG was successful.

Parameters

in	*base*	WDOG base pointer.

Returns

true if the configuration was successful, false if not. Implements : WDOG_HAL_IsReconfigurationComplete↩
_Activity

3.86.5.13 WDOG_HAL_IsUnlocked()

```
static bool WDOG_HAL_IsUnlocked (
            const WDOG_Type * base )  [inline], [static]
```

Checks if the WDOG is unlocked.

This function checks if the WDOG is unlocked. If the module is unlocked, reconfiguration of the registers is allowed.

Parameters

in	*base*	WDOG base pointer.

Returns

true if the module is unlocked, false if the module is locked. Implements : WDOG_HAL_IsUnlocked_Activity

3.86.5.14 WDOG_HAL_IsUpdateEnabled()

```
static bool WDOG_HAL_IsUpdateEnabled (
             const WDOG_Type * base )  [inline], [static]
```

Verifies if the WDOG updates are allowed.

This function verifies if software is allowed to reconfigure the WDOG without a reset.

Parameters

in	*base*	WDOG base pointer.

Returns

the state of the WDOG updates:

- false: updates are not allowed
- true: updates are allowed Implements : WDOG_HAL_IsUpdateEnabled_Activity

3.86.5.15 WDOG_HAL_SetClockSource()

```
static void WDOG_HAL_SetClockSource (
             WDOG_Type * base,
             wdog_clk_source_t clkSource )  [inline], [static]
```

Selects the clock source used by the WDOG.

This function selects the clock source used by the WDOG.

Parameters

in	*base*	WDOG base pointer.
in	*clkSource*	the clock source to be used by the WDOG: • 0x00U: Bus clock • 0x01U: LPO clock • 0x02U: System oscillator clock (SOSC, from SCG) • 0x03U: Slow internal reference clock (SIRC, from SCG) Implements : WDOG_HAL_SetClockSource_Activity

3.86.5.16 WDOG_HAL_SetCmd32()

```
static void WDOG_HAL_SetCmd32 (
            WDOG_Type * base,
            bool enable )  [inline], [static]
```

Enables/Disables support for 32-bit refresh/unlock command write words.

This function enables/disables support for 32-bit refresh/unlock command write words.

Parameters

in	*base*	WDOG base pointer.
in	*enable*	enable/disable 32-bit refresh/unlock command write words. Implements : WDOG_HAL_SetCmd32_Activity

3.86.5.17 WDOG_HAL_SetCounter()

```
static void WDOG_HAL_SetCounter (
            WDOG_Type * base,
            uint32_t counter )  [inline], [static]
```

Sets the value of the WDOG counter.

This sets the value of the WDOG counter.

Parameters

in	*base*	WDOG base pointer.
in	*counter*	the value of the WDOG counter. Implements : WDOG_HAL_SetCounter_Activity

3.86.5.18 WDOG_HAL_SetDebug()

```
static void WDOG_HAL_SetDebug (
            WDOG_Type * base,
            bool enable )  [inline], [static]
```

Enables/Disables WDOG in debug mode.

This function enables/disables the WDOG in debug mode.

Parameters

in	*base*	WDOG base pointer.
in	*enable*	enable/disable WDOG in debug mode Implements : WDOG_HAL_SetDebug_Activity

3.86.5.19 WDOG_HAL_SetInt()

```
static void WDOG_HAL_SetInt (
            WDOG_Type * base,
            bool enable )  [inline], [static]
```

Enables/Disables WDOG interrupt.

This function enables/disables the interrupts from the WDOG.

Parameters

in	*base*	WDOG base pointer.
in	*enable*	enable/disable interrupts. Implements : WDOG_HAL_SetInt_Activity

3.86.5.20 WDOG_HAL_SetPrescaler()

```
static void WDOG_HAL_SetPrescaler (
             WDOG_Type * base,
             bool enable )  [inline], [static]
```

Enables/Disables prescaler.

This function enables/disables the prescaler for the WDOG.

Parameters

in	*base*	WDOG base pointer.
in	*enable*	enable/disable prescaler. Implements : WDOG_HAL_SetPrescaler_Activity

3.86.5.21 WDOG_HAL_SetStop()

```
static void WDOG_HAL_SetStop (
             WDOG_Type * base,
             bool enable )  [inline], [static]
```

Enables/Disables WDOG in stop mode.

This function enables/disables the WDOG in stop mode.

Parameters

in	*base*	WDOG base pointer.
in	*enable*	enable/disable WDOG in stop mode. Implements : WDOG_HAL_SetStop_Activity

3.86.5.22 WDOG_HAL_SetTestMode()

```
static void WDOG_HAL_SetTestMode (
             WDOG_Type * base,
             wdog_test_mode_t testMode )  [inline], [static]
```

Changes the WDOG test mode.

This function changes the test mode of the WDOG. If the WDOG is tested in mode, software should set this field to 0x01U in order to indicate that the WDOG is functioning normally.

Parameters

in	*base*	WDOG base pointer.
	testMode	the test mode: • 0x00U: Test mode disabled • 0x01U: User mode enabled (test mode disabled) • 0x02U: Test mode enabled (for the low byte) • 0x03U: Test mode enabled (for the high byte) Implements : WDOG_HAL_SetTestMode_Activity

3.86.5.23 WDOG_HAL_SetTimeout()

```
static void WDOG_HAL_SetTimeout (
            WDOG_Type * base,
            uint16_t timeout )  [inline], [static]
```

Sets the value of the WDOG timeout.

This sets the value of the WDOG timeout.

Parameters

in	*base*	WDOG base pointer.
in	*timeout*	the value of the WDOG timeout. Implements : WDOG_HAL_SetTimeout_Activity

3.86.5.24 WDOG_HAL_SetUpdate()

```
static void WDOG_HAL_SetUpdate (
            WDOG_Type * base,
            bool enable )  [inline], [static]
```

Enables/Disables WDOG updates.

This function enables/disables the possibility to update the WDOG config.

Parameters

in	*base*	WDOG base pointer.
in	*enable*	enable/disable updates. Implements : WDOG_HAL_SetUpdate_Activity

3.86.5.25 WDOG_HAL_SetWait()

```
static void WDOG_HAL_SetWait (
            WDOG_Type * base,
            bool enable )  [inline], [static]
```

Enables/Disables WDOG in wait mode.

This function enables/disables the WDOG in wait mode.

Parameters

in	*base*	WDOG base pointer.
in	*enable*	enable/disable WDOG in wait mode. Implements : WDOG_HAL_SetWait_Activity

3.86.5.26 WDOG_HAL_SetWindow()

```
static void WDOG_HAL_SetWindow (
            WDOG_Type * base,
            bool enable )  [inline], [static]
```

Enables/Disables window mode.

This function enables/disables window mode for the WDOG.

Parameters

in	*base*	WDOG base pointer.
in	*enable*	enable/disable window mode Implements : WDOG_HAL_SetWindow_Activity

3.86.5.27 WDOG_HAL_SetWindowValue()

```
static void WDOG_HAL_SetWindowValue (
            WDOG_Type * base,
            uint16_t window )  [inline], [static]
```

Sets the value of the WDOG window.

This sets the value of the WDOG window.

Parameters

in	*base*	WDOG base pointer.
in	*window*	the value of the WDOG window. Implements : WDOG_HAL_SetWindowValue_Activity

3.86.5.28 WDOG_HAL_Trigger()

```
static void WDOG_HAL_Trigger (
            WDOG_Type * base )  [inline], [static]
```

Refreshes the WDOG counter.

Parameters

in	*base*	WDOG base pointer. Implements : WDOG_HAL_Trigger_Activity

3.87 Watchdog timer (WDOG)

3.87.1 Detailed Description

The S32 SDK provides both HAL and Peripheral Driver for the Watchdog timer (WDOG) module of S32 SDK devices .

Hardware background

The Watchdog Timer (WDOG) module is an independent timer that is available for system use. It provides a safety feature to ensure that software is executing as planned and that the CPU is not stuck in an infinite loop or executing unintended code. If the WDOG module is not serviced (refreshed) within a certain period, it resets the MCU.

Features of the WDOG module include:

- Configurable clock source inputs independent from the bus clock
- Programmable timeout period
  - Programmable 16-bit timeout value
  - Optional fixed 256 clock prescaler when longer timeout periods are needed
- Window mode option for the refresh mechanism
  - Programmable 16-bit window value
  - Provides robust check that program flow is faster than expected
  - Early refresh attempts trigger a reset
- Optional timeout interrupt to allow post-processing diagnostics
  - Interrupt request to CPU with interrupt vector for an interrupt service routine (ISR)
  - Forced reset occurs 128 bus clocks after the interrupt vector fetch
- Configuration bits are write-once-after-reset to ensure watchdog configuration cannot be mistakenly altered
- Robust write sequence for unlocking write-once configuration bits

Modules

- WDOG Driver

  Watchdog Timer Peripheral Driver.
- WDOG HAL

  Watchdog Timer Hardware Abstraction Level.

Index