JBLopen

Embedded Software Insight

BASEplatform STM32 F4 Reference Manual

PM0003

July 9, 2019

Contents

Chapter

1

Overview

Welcome to the BASEplatform™ platform reference manual for the STMicroelectronics STM32® F4 Series. This reference manual covers the platform-specific API relevant to the STM32 F4. This document includes important configuration information as well as the complete API reference for the STM32 F4. The core API of the BASEplatform can be found in the BASEplatform API reference available on the documentation section of the JBLopen website. Similarly to the core API, the platform-specific API is written in ISO/IEC 9899:1999 (C99) compliant C and designed to be portable across the toolchains supporting the STM32 family.

For convenience during development, all the information related to each individual API element is also reproduced within the relevant header source files in human readable format.

About the BASEplatform

The BASEplatform is a collection of low-level interface modules, drivers and board support packages (BSPs) designed to provide the foundation for an embedded software application. The BASEplatform can support a variety of free or commercial RTOSes as well as bare-metal applications, both in multi-core and single core configurations. BASEplatform packages are created specifically for an application's needs, and usually include support for an RTOS or bare-metal, low level I/Os, such as UART, I2C, GPIO etc. as well as communication and storage stacks, as selected by the application developer, alongside the necessary drivers, integration and IDE files to get everything working out of the box.

Missing Module or API?

The BASEplatform's set of modules is developed prioritizing our customer's need. If a needed module or API function is missing do not hesitate to inform us. In addition make sure to consult both the BASEplatform API reference manual as well as the platform-specific reference manual for a complete list of API and modules.

Elements of the API Reference

Each documented API element, be it a function, data type or preprocessor definition is presented using a similar layout which are described below. This section briefly describes the various elements of the API reference.

Functions

The most numerous and important API elements documented are functions. Below is an example of API reference for a hypothetical function named `bp_example_func()`:

Function

`bp_example_func()`

<example/bp_example.h>

Example function description.

Prototype

```
int  bp_example_func  ( uint32_t  arg1,
                        uint32_t  arg2 );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`arg1`	First argument's description.
`arg2`	Second argument's description.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Example

```
uint32_t a = 0u;
uint32_t b = 1u;
int result;

result = bp_example_func(a, b);
if(result != RTNC_SUCCESS) \{
    // Handle error.
\}
```

Function Name

At the top of each API is the name of the function or object as it appears in the source code. BASEplatform functions are always prefixed with `bp_` followed by the module name and then the function's specific name.

Header

Following the name is the header file where the declaration of the documented API can be found. It is recommended to use the displayed path relative to the root of the source directory of the BASEplatform when including BASEplatform's headers.

For example, to include the UART module header file `bp_uart.h` the following include directive is recommended.

```
#include <uart/bp_uart.h>
```

The root of the BASEplatform source directory should be added to the include path of the compiler.

Description

A description of the API element including basic usage information.

Prototype

For functions, the full signature of the API along with parameter names, types, and function return type.

Attributes

For functions only, this section lists the relevant function attributes. See the function attributes section of this manual for a detailed description of each attribute.

Parameters

Function parameters list along with a short description of each parameter.

Returned Errors or Return Values

For functions that return a BASEplatform standard error code, this section is named Returned Errors and lists the relevant errors that can be returned. See the error handling convention section of this manual for more information on the BASEplatform error handling.

For other functions that do not return a standard error code, this section lists the possible output values of the function. In this case the section is named "Returned Values".

Example

Some API functions may include a small code example to illustrate usage. Note that these examples are for documentation purpose and may not include error handling and checking to keep the examples concise.

Data Types

Data types include structure definitions, enumerated types as well as scalar type definitions. They all follow a similar documentation layout, below is an example of API reference for a hypothetical structure definition named `bp_example_struct_t`:

Data Type

bp_example_struct_t

<example/bp_example.h>

Example structure description.

Members

`member1`	`uint32_t`	First member's description.
`member2`	`uint32_t`	Second member's description.

Data Type Name

At the top of each API is the name of the data type as it appears in the source code. In the case of structures and enumerated types, this is always the `typedef`'d data type. BASEplatform types always prefixed with `bp_` followed by the module name and then the type's specific name. Types are also always suffixed with `_t` to differentiate them from other definitions.

Header

Following the name is the header file where the declaration of the documented API can be found. It is recommended to use the displayed path relative to the root of the source directory of the BASEplatform when including BASEplatform's headers.

Description

A description of the data type including basic usage information.

Members/Enumeration Values

If documented, the API reference will list the structure members or the list of enumeration constants along with a short description of each member. The list of members for opaque types with no publicly accessible members are omitted from the list of members in the API documentation.

Macros

Relevant and preprocessor macros that are part of the public API are documented in the API reference. This includes function-like macros as well as object-like macros. The latter is often referred to as preprocessor definitions or simply defines. Below is an example of function-like macro named `BP_EXAMPLE_MACRO()`:

Macro

BP_EXAMPLE_MACRO()

<example/bp_example.h>

Example macro description.

Prototype

```
BP_EXAMPLE_MACRO ( arg1,
                   arg2 );
```

Parameters

`arg1`	First argument's description.
`arg2`	First argument's description.

Expansion Macro expansion's description.

Macro Name

At the top of each API is the name of the macro as it appears in the source code. BASEplatform preprocessor definitions are always in capital letters and prefixed with `BP_` followed by the module name and then the macro's specific name.

Header

Following the name is the header file where the declaration of the documented API can be found. It is recommended to use the displayed path relative to the root of the source directory of the BASEplatform when including BASEplatform's headers.

Description

A description of the macro including basic usage information.

Parameters

Macro parameters list along with a short description of each parameter.

Expansion

For function-like macros an expansion section describes the macro's expansion including the type if applicable.

Function Attributes

The API reference documentation for API functions includes a set of attributes that clarifies in which context it is safe to call a specific API function. The attributes are as follows:

- Blocking
- ISR-safe
- Critical-safe
- Thread-safe

Blocking

The function is potentially blocking, which means it can wait or pend on a kernel object such as a semaphore or mutex, in order to wait for a resource to be available or for an operation to complete. Some functions may be optionally blocking depending on the function's arguments. Those functions are always marked as blocking in the API reference regardless.

In a bare-metal environment, any function marked as blocking can potentially suspend the background task while waiting for a specific interrupt. Many of those functions take a timeout parameter that can be set to 0 to make them non-blocking (polling) if suspension of the background task is undesired.

As a general rule, blocking functions should not be called from an interrupt service routine, also known as interrupt handler or while the CPU interrupts are disabled. In addition, some RTOSes allow suspending or locking the scheduler, when this is the case, blocking functions should not be called while the scheduler is suspended or locked.

ISR-Safe

An ISR-safe function can be called from within an interrupt service routine. This also includes callback functions that are called from an interrupt context. Note that while an ISR-safe function is usually critical-safe this is not always the case. Also an ISR-safe function may not necessarily be thread-safe.

Critical Safe

Critical safe functions can be called when the CPU interrupts are disabled, this is also called a critical context or sometimes a critical section. Critical sections are usually entered by calling a spinlock acquire or critical section enter function. Calling a non-critical-safe function from within a critical section can corrupt the state of the CPU's interrupt disable flags and cause runtime faults or data corruption.

Thread-Safe

A thread safe function guarantees correct operations between multiple threads or tasks when running under a multitasking kernel. In the context of the BASEplatform API, thread-safe also implies thread safety on an SMP system, which means it is safe to use the API function from different threads in parallel. Due to the design of the BASEplatform, thread-safe functions are also re-entrant assuming that the other function attributes, such as ISR safety, are respected.

Function Attributes in Header Files

Function attributes are documented slightly differently in the source header files in order to be more concise and easier to maintain. The attributes are documented under an "Attributes" section and are named as follows:

- non-blocking
- non-thread-safe
- ISR-safe
- critical-section-safe

Absence of an attribute implies that the opposite attribute applies to the function. For example, in the absence of any explicit function attribute in the header documentation, a function is assumed to be blocking, thread-safe and not safe to call from ISRs and critical sections.

API Conventions

The BASEplatform API adheres to a few conventions with respect to the naming, error handling and timeouts that are useful for the application developers.

Naming

The BASEplatform API function names are all written in lower case, except preprocessor macros which are in upper case. Words within an object name are separated by underscores and the whole name is prefixed with bp_ followed by the module name and finally the function specific part of the name.

For example, the time module function to get the current time is written as follows:

```
bp_time_get()
```

And the memory barrier macro from the architecture module, ”arch” for short, is named as follows:

```
BP_ARCH_MB()
```

Error Handling

Most API functions return a status in the form of a plain int as the function’s return value. As a general exception, some functions that cannot fail are allowed to return nothing (void) or another value.

In general, the BASEplatform attempts to minimize the number of different error codes to simplify the application’s error handling and improve performance. The list of possible error codes is included within every function’s documentation. The meaning of each error code is also documented in a function’s description. See the Error Codes chapter for a list of defined error codes.

As with other preprocessor macros and enumeration constants, the application should never rely on the exact numerical value of any specific error code. However, two guarantees are made with respect to the error code numerical values. The first is that `RTNC_SUCCESS` will always expand to 0. The second is that all other error codes are negative. Positive values are not used for any valid error code. Any undefined or unexpected error code returned by a function should be treated as a fatal error.

Two error codes have the exact same meaning for all the functions, namely `RTNC_SUCCESS` and `RTNC_FATAL`.

`RTNC_SUCCESS` is returned when a function completed successfully without issue.

`RTNC_FATAL` is returned if and only if an unexpected situation that should not happen at runtime is detected. This includes invalid function arguments, internal data corruption and assertion failures within the code. In addition, any unexpected error code returned from a function should be treated as a fatal error. It is up to the application to decide on the proper action to perform upon receiving a fatal error. As a general rule, the application should not perform any other calls to that module instance. Safety critical applications should consider an `RTNC_FATAL` error code as a severe assertion failure and act accordingly.

Some modules, especially IO modules such as UART and I2C, provides a reset API call that can be used to reset the internal state of a module as well as the underlying peripheral. This can be used to attempt to recover from a fatal error in case the error condition is temporary.

Timeouts

Most of the blocking functions have a timeout argument that takes a timeout value in milliseconds. The timeout period is guaranteed to be at least the requested value rounded up to the next multiple of the kernel’s tick rate if necessary. Internally, the BASEplatform modules and drivers will attempt to respect the timeout value as closely as possible while guaranteeing the minimum timeout value. However, RTOS

scheduling, higher priority tasks and interrupt response time may increase the amount of time taken to return from a timeout condition.

For all functions that take a timeout value, specifying a timeout value of 0 means that the function will return immediately instead of blocking when having to wait on a mutex or an interrupt. A value of `TIMEOUT_INF` or `-1` will result in an infinite timeout.

Numerical Values of Macros and Enumeration Constants

To ease maintainability and ensure compatibility with future versions, the application should never rely on enumeration constants and macros numerical value.

Driver API

Many of the BASEplatform modules, especially the IO modules, use drivers to perform hardware access. In those situations the top-level module provides lifecycle management as well as thread-safety. However, it may be desirable in some circumstances to access the driver API directly. The various driver function signatures are gathered at the end of this manual but additional details may be available from each platform's reference manual.

Advanced Driver API

Each driver is allowed to implement additional, driver specific, functionalities not available from the top level module API. These functions are usually meant to control advanced features of the underlying peripherals. Each I/O module provides an API to retrieve the driver's handle which can be used to access those advanced functions directly. There is also an optional locking mechanism that can be used to ensure thread safety while performing direct operations on the drivers.

Accessing the Drivers Directly

It is also possible to access the drivers standard operation directly at the driver level. This reduces the overhead associated the kernel mutexes and driver dereference at the cost of thread safety. As such, direct driver access should be done with care. As with the case of the advanced driver features, there is an optional exclusive lock mechanism that can be used to ensure thread safety.

STM32 F4 Specific Configurations

SoC Definition

The generic definitions for the STM32 F4 SoC peripherals can be found in `soc/stmicro/stm32f4/bp_stm32f4_soc_def.h`. These can be used to write a custom board definition.

External Clocks Frequency

Since there is no way for the BASEplatform to derive the input frequency of external clock inputs and external oscillators, they must be specified in the board definition file. For example:

```
#define BP_STM32F4_BOARD_DEF_LSE_CLK_FREQ (32768U)

#define BP_STM32F4_BOARD_DEF_HSE_CLK_FREQ (8000000U)
```

Chapter 2

STM32F4 Definitions

STM32F4 global definitions. This module contains various definitions pertaining to the STM32 F4 series including interrupt, reset and clock lists as well as the base addresses of the various peripherals. These definitions are used in the SoC definition files for the STM32F4 but can also be used as input values for the clock, reset and interrupt management modules.

Data Type

bp_stm32f4_clk_gate_t

<soc/stmicro/stm32f4/bp_stm32f4_def.h>

STM32F4 clock gate list. These enumeration constants can be used with the clock management API such as `bp_clock_en()` and `bp_clock_dis()`.

Values

`BP_STM32F4_CLK_GATE_OTGHS_ULPI`	USB OTG HS ULPI.
`BP_STM32F4_CLK_GATE_OTGHS`	USB OTG HS.
`BP_STM32F4_CLK_GATE_ETH_MAC_PTP`	Ethernet MAC PTP.
`BP_STM32F4_CLK_GATE_ETH_MAC_RX`	Ethernet MAC RX.
`BP_STM32F4_CLK_GATE_ETH_MAC_TX`	Ethernet MAC TX.
`BP_STM32F4_CLK_GATE_ETH_MAC`	Ethernet MAC.
`BP_STM32F4_CLK_GATE_DMA2D`	DMA2D.
`BP_STM32F4_CLK_GATE_DMA1`	DMA1.
`BP_STM32F4_CLK_GATE_DMA2`	DMA2.
`BP_STM32F4_CLK_GATE_CCM_DATA`	CCM Data RAM.
`BP_STM32F4_CLK_GATE_BKP`	Backup RAM.

BP_STM32F4_CLK_GATE_CRC	CRC.
BP_STM32F4_CLK_GATE_GPIOA	GPIOA.
BP_STM32F4_CLK_GATE_GPIOB	GPIOB.
BP_STM32F4_CLK_GATE_GPIOC	GPIOC.
BP_STM32F4_CLK_GATE_GPIOD	GPIOD.
BP_STM32F4_CLK_GATE_GPIOE	GPIOE.
BP_STM32F4_CLK_GATE_GPIOF	GPIOF.
BP_STM32F4_CLK_GATE_GPIOG	GPIOG.
BP_STM32F4_CLK_GATE_GPIOH	GPIOH.
BP_STM32F4_CLK_GATE_GPIOI	GPIOI.
BP_STM32F4_CLK_GATE_GPIOJ	GPIOJ.
BP_STM32F4_CLK_GATE_GPIOK	GPIOK.
BP_STM32F4_CLK_GATE_OTGFS	USB OTG FS.
BP_STM32F4_CLK_GATE_RNG	RNG.
BP_STM32F4_CLK_GATE_HASH	HASH.
BP_STM32F4_CLK_GATE_CRYP	CRYP.
BP_STM32F4_CLK_GATE_DCMI	DCMI.
BP_STM32F4_CLK_GATE_QSPI	QSPI.
BP_STM32F4_CLK_GATE_FMC	FMC.
BP_STM32F4_CLK_GATE_UART1	UART1.
BP_STM32F4_CLK_GATE_UART2	UART2.
BP_STM32F4_CLK_GATE_UART3	UART3.
BP_STM32F4_CLK_GATE_UART4	UART4.
BP_STM32F4_CLK_GATE_UART5	UART5.
BP_STM32F4_CLK_GATE_UART6	UART6.
BP_STM32F4_CLK_GATE_UART7	UART7.
BP_STM32F4_CLK_GATE_UART8	UART8.
BP_STM32F4_CLK_GATE_DAC	DAC.
BP_STM32F4_CLK_GATE_PWR	PWR.
BP_STM32F4_CLK_GATE_CAN1	CAN1.
BP_STM32F4_CLK_GATE_CAN2	CAN2.
BP_STM32F4_CLK_GATE_I2C1	I2C1.

BP_STM32F4_CLK_GATE_I2C2	I2C2.
BP_STM32F4_CLK_GATE_I2C3	I2C3.
BP_STM32F4_CLK_GATE_WWDOG	WWDOG.
BP_STM32F4_CLK_GATE_TIM1	TIM1.
BP_STM32F4_CLK_GATE_TIM2	TIM2.
BP_STM32F4_CLK_GATE_TIM3	TIM3.
BP_STM32F4_CLK_GATE_TIM4	TIM4.
BP_STM32F4_CLK_GATE_TIM5	TIM5.
BP_STM32F4_CLK_GATE_TIM6	TIM6.
BP_STM32F4_CLK_GATE_TIM7	TIM7.
BP_STM32F4_CLK_GATE_TIM8	TIM8.
BP_STM32F4_CLK_GATE_TIM9	TIM9.
BP_STM32F4_CLK_GATE_TIM10	TIM10.
BP_STM32F4_CLK_GATE_TIM11	TIM11.
BP_STM32F4_CLK_GATE_TIM12	TIM12.
BP_STM32F4_CLK_GATE_TIM13	TIM13.
BP_STM32F4_CLK_GATE_TIM14	TIM14.
BP_STM32F4_CLK_GATE_DSI	DSI.
BP_STM32F4_CLK_GATE_LTDC	LTDC.
BP_STM32F4_CLK_GATE_SAI1	SAI1.
BP_STM32F4_CLK_GATE_SAI2	SAI2.
BP_STM32F4_CLK_GATE_SPI1	SPI1.
BP_STM32F4_CLK_GATE_SPI2	SPI2.
BP_STM32F4_CLK_GATE_SPI3	SPI3.
BP_STM32F4_CLK_GATE_SPI4	SPI4.
BP_STM32F4_CLK_GATE_SPI5	SPI5.
BP_STM32F4_CLK_GATE_SPI6	SPI6.
BP_STM32F4_CLK_GATE_SYSCFG	SYSCFG.
BP_STM32F4_CLK_GATE_SDIO	SDIO.
BP_STM32F4_CLK_GATE_ADC1	ADC1.
BP_STM32F4_CLK_GATE_ADC2	ADC2.
BP_STM32F4_CLK_GATE_ADC3	ADC3.
BP_STM32F4_CLK_GATE_NONE	Special invalid value.

Data Type

bp_stm32f4_clk_t

<soc/stmicro/stm32f4/bp_stm32f4_def.h>

STM32F4 clock list. These enumeration constants can be used with the clock management API such as `bp_clk_freq_get()`.

Values

`BP_STM32F4_CLK_LSI`	Low speed internal.
`BP_STM32F4_CLK_LSE`	Low speed external.
`BP_STM32F4_CLK_HSI`	High speed internal.
`BP_STM32F4_CLK_HSE`	High speed external.
`BP_STM32F4_CLK_EXT`	External I2D clock input.
`BP_STM32F4_CLK_HSE_RTC`	High speed RTC input.
`BP_STM32F4_CLK_PLL`	Main PLL output.
`BP_STM32F4_CLK_PLL_48`	Main PLL 48MHz output.
`BP_STM32F4_CLK_PLL_DSI`	Main PLL DSI output.
`BP_STM32F4_CLK_PLL_I2S`	I2S PLL I2S output.
`BP_STM32F4_CLK_PLL_I2S_SAI`	I2S PLL SAI output.
`BP_STM32F4_CLK_PLL_SAI`	SAI PLL SAI output.
`BP_STM32F4_CLK_PLL_SAI_48`	SAI PLL 48MHz output.
`BP_STM32F4_CLK_PLL_SAI_LCD`	SAI PLL LCD output.
`BP_STM32F4_CLK_MCO1`	MCO1 clock output.
`BP_STM32F4_CLK_MCO2`	MCO2 clock output.
`BP_STM32F4_CLK_PTP`	Ethernet PTP.
`BP_STM32F4_CLK_HCLK`	AHB, Core and DMA clock.
`BP_STM32F4_CLK_SYS_CLK`	System clock.
`BP_STM32F4_CLK_CORTEX_TIM`	Cortex system timer clock.
`BP_STM32F4_CLK_CORTEX_FCLK`	Cortex free running clock.
`BP_STM32F4_CLK_APB1`	APB1 peripheral clock.
`BP_STM32F4_CLK_APB1_TIM`	APB1 timer clock.
`BP_STM32F4_CLK_APB2`	APB1 peripheral clock.
`BP_STM32F4_CLK_APB2_TIM`	APB1 timer clock.
`BP_STM32F4_CLK_48`	48MHz clock.
`BP_STM32F4_CLK_SDIO`	SDIO clock.

`BP_STM32F4_CLK_DSI`	DSI host clock.
`BP_STM32F4_CLK_I2S`	I2S clock.
`BP_STM32F4_CLK_SAI1_A`	SAI1_A clock.
`BP_STM32F4_CLK_SAI1_B`	SAI1_B clock.
`BP_STM32F4_CLK_LCD_TFT`	LCD TFT clock.
`BP_STM32F4_CLK_NONE`	Special invalid value.

Data Type

bp_stm32f4_int_t

<soc/stmicro/stm32f4/bp_stm32f4_def.h>

STM32F4 interrupt list. These enumeration constants can be used with the interrupt management API such as `bp_int_reg()` and `bp_int_src_en()` for convenience.

Values

`BP_STM32F4_INT_RSVD0`	Reserved.
`BP_STM32F4_INT_RSVD1`	Reserved.
`BP_STM32F4_INT_NMI`	ARMv7M NMI.
`BP_STM32F4_INT_HARD_FAULT`	ARMv7M Hard fault.
`BP_STM32F4_INT_MM_FAULT`	ARMv7M Memory management fault.
`BP_STM32F4_INT_BUS_FAULT`	ARMv7M bus fault.
`BP_STM32F4_INT_USAGE_FAULT`	ARMv7M usage fault.
`BP_STM32F4_INT_SEC_FAULT`	ARMv7M secure fault.
`BP_STM32F4_INT_RSVD2`	Reserved.
`BP_STM32F4_INT_RSVD3`	Reserved.
`BP_STM32F4_INT_RSVD4`	Reserved.
`BP_STM32F4_INT_SVC`	ARMv7M SVC.
`BP_STM32F4_INT_DEBUG`	ARMv7M Debug monitor.
`BP_STM32F4_INT_RSVD5`	Reserved.
`BP_STM32F4_INT_PENDSV`	ARMv7M PendSV.
`BP_STM32F4_INT_SYSTICK`	ARMv7M SysTick.
`BP_STM32F4_INT_WWDOG`	Window watchdog.
`BP_STM32F4_INT_PVD`	PVD.
`BP_STM32F4_INT_TAMP`	Tamper and TimeStamp.

BP_STM32F4_INT_RTC_WKUP	RTC Wakeup.
BP_STM32F4_INT_FLASH	Flash.
BP_STM32F4_INT_RCC	RCC.
BP_STM32F4_INT_EXTI0	EXTI0.
BP_STM32F4_INT_EXTI1	EXTI1.
BP_STM32F4_INT_EXTI2	EXTI2.
BP_STM32F4_INT_EXTI3	EXTI3.
BP_STM32F4_INT_EXTI4	EXTI4.
BP_STM32F4_INT_DMA1S0	DMA1 STREAM0.
BP_STM32F4_INT_DMA1S1	DMA1 STREAM1.
BP_STM32F4_INT_DMA1S2	DMA1 STREAM2.
BP_STM32F4_INT_DMA1S3	DMA1 STREAM3.
BP_STM32F4_INT_DMA1S4	DMA1 STREAM4.
BP_STM32F4_INT_DMA1S5	DMA1 STREAM5.
BP_STM32F4_INT_DMA1S6	DMA1 STREAM6.
BP_STM32F4_INT_ADC	ADC.
BP_STM32F4_INT_CAN1_TX	CAN1 Transmit.
BP_STM32F4_INT_CAN1_RX0	CAN1 Receive 0.
BP_STM32F4_INT_CAN1_RX1	CAN1 Receive 1.
BP_STM32F4_INT_CAN1_SCE	CAN1 SCE.
BP_STM32F4_INT_EXTI9_5	EXTI 5 to 9.
BP_STM32F4_INT_TIM1_BRK_TIM9	Timer1 break and timer 9 global.
BP_STM32F4_INT_TIM1_UP_TIM10	Timer1 update and timer 10 global.
BP_STM32F4_INT_TIM1_TRIG_TIM11	Timer1 trigger and timer 11 global.
BP_STM32F4_INT_TIM1_CC	Timer1 compare.
BP_STM32F4_INT_TIM2	Timer2 global.
BP_STM32F4_INT_TIM3	Timer3 global.
BP_STM32F4_INT_TIM4	Timer4 global.
BP_STM32F4_INT_I2C1_EV	I2C1 event.
BP_STM32F4_INT_I2C1_ER	I2C1 error.
BP_STM32F4_INT_I2C2_EV	I2C2 event.
BP_STM32F4_INT_I2C2_ER	I2C2 error.

BP_STM32F4_INT_SPI1	SPI1.
BP_STM32F4_INT_SPI2	SPI2.
BP_STM32F4_INT_UART1	USART1.
BP_STM32F4_INT_UART2	USART2.
BP_STM32F4_INT_UART3	USART3.
BP_STM32F4_INT_EXTI10_15	EXTI 10 to 15.
BP_STM32F4_INT_RTC_ALARM	RTC alarm.
BP_STM32F4_INT_OTG_FS_WKUP	USB OTG FS wakeup.
BP_STM32F4_INT_TIM8_BRK_TIM12	Timer 8 break and timer 12 global.
BP_STM32F4_INT_TIM8_UP_TIM13	Timer 8 update and timer 13 global.
BP_STM32F4_INT_TIM8_TRIG_TIM14	Timer 8 trigger and timer 14 global.
BP_STM32F4_INT_TIM8_CC	Timer 8 compare.
BP_STM32F4_INT_DMA1S7	DAM1 STREAM7.
BP_STM32F4_INT_FMC	FMC.
BP_STM32F4_INT_SDIO	SDIO.
BP_STM32F4_INT_TIM5	TIM5.
BP_STM32F4_INT_SPI3	SPI3.
BP_STM32F4_INT_UART4	USART4.
BP_STM32F4_INT_UART5	USART5.
BP_STM32F4_INT_TIM6_DAC	Timer 6 global and DAC1/2 underrun.
BP_STM32F4_INT_TIM7	Timer 7.
BP_STM32F4_INT_DMA2S0	DMA2 STREAM0.
BP_STM32F4_INT_DMA2S1	DMA2 STREAM1.
BP_STM32F4_INT_DMA2S2	DMA2 STREAM2.
BP_STM32F4_INT_DMA2S3	DMA2 STREAM3.
BP_STM32F4_INT_DMA2S4	DMA2 STREAM4.
BP_STM32F4_INT_ETH	Ethernet.
BP_STM32F4_INT_ETH_WKUP	Ethernet wakeup.
BP_STM32F4_INT_CAN2_TX	CAN2 Transmit.
BP_STM32F4_INT_CAN2_RX0	CAN2 Receive 0.
BP_STM32F4_INT_CAN2_RX1	CAN2 Receive 1.
BP_STM32F4_INT_CAN2_SCE	CAN2 SCE.

BP_STM32F4_INT_OTG_FS	OTG_FS.
BP_STM32F4_INT_DMA2S5	DMA2 STREAM5.
BP_STM32F4_INT_DMA2S6	DMA2 STREAM6.
BP_STM32F4_INT_DMA2S7	DMA2 STREAM7.
BP_STM32F4_INT_UART6	USART6.
BP_STM32F4_INT_I2C3_EV	I2C3 event.
BP_STM32F4_INT_I2C3_ER	I2C3 error.
BP_STM32F4_INT_OTG_HS_EP1_OUT	USB OTG HS endpoint 1 out.
BP_STM32F4_INT_OTG_HS_EP1_IN	USB OTG HS endpoint 1 in.
BP_STM32F4_INT_OTG_HS_WKUP	USB OTG HS wakeup.
BP_STM32F4_INT_OTG_HS	USB OTG HS.
BP_STM32F4_INT_DCMI	DCMI.
BP_STM32F4_INT_CRYP	CRYP.
BP_STM32F4_INT_HASH_RNG	HASH and RNG global.
BP_STM32F4_INT_FPU	FPU.
BP_STM32F4_INT_UART7	UART7.
BP_STM32F4_INT_UART8	UART8.
BP_STM32F4_INT_SPI4	SPI4.
BP_STM32F4_INT_SPI5	SPI5.
BP_STM32F4_INT_SPI6	SPI6.
BP_STM32F4_INT_SAI1	SAI1.
BP_STM32F4_INT_LCD	LCD global.
BP_STM32F4_INT_LCD_ER	LCD error.
BP_STM32F4_INT_DMA2D	DMA2D.
BP_STM32F4_INT_QSPI	QSPI.
BP_STM32F4_INT_DSI	DSI.
BP_STM32F4_INT_NONE	Special invalid value.

Data Type

bp_stm32f4_reset_t

<soc/stmicro/stm32f4/bp_stm32f4_def.h>

STM32F4 reset list. These enumeration constants can be used with the reset management API such as `bp_periph_reset_assert()` and `bp_periph_reset_deassert()`.

Values

BP_STM32F4_RESET_OTGHS	USB OTG HS.
BP_STM32F4_RESET_ETH	Ethernet MAC.
BP_STM32F4_RESET_DMA2D	DMA2D.
BP_STM32F4_RESET_DMA1	DMA1.
BP_STM32F4_RESET_DMA2	DMA2.
BP_STM32F4_RESET_CRC	CRC.
BP_STM32F4_RESET_GPIOA	GPIOA.
BP_STM32F4_RESET_GPIOB	GPIOB.
BP_STM32F4_RESET_GPIOC	GPIOC.
BP_STM32F4_RESET_GPIOD	GPIOD.
BP_STM32F4_RESET_GPIOE	GPIOE.
BP_STM32F4_RESET_GPIOF	GPIOF.
BP_STM32F4_RESET_GPIOG	GPIOG.
BP_STM32F4_RESET_GPIOH	GPIOH.
BP_STM32F4_RESET_GPIOI	GPIOI.
BP_STM32F4_RESET_GPIOJ	GPIOJ.
BP_STM32F4_RESET_GPIOK	GPIOK.
BP_STM32F4_RESET_OTGFS	USB OTG FS.
BP_STM32F4_RESET_RNG	RNG.
BP_STM32F4_RESET_HASH	HASH.
BP_STM32F4_RESET_CRYP	CRYP.
BP_STM32F4_RESET_DCMI	DCMI.
BP_STM32F4_RESET_QSPI	QSPI.
BP_STM32F4_RESET_FMC	FMC.
BP_STM32F4_RESET_UART1	UART1.
BP_STM32F4_RESET_UART2	UART2.
BP_STM32F4_RESET_UART3	UART3.
BP_STM32F4_RESET_UART4	UART4.
BP_STM32F4_RESET_UART5	UART5.
BP_STM32F4_RESET_UART6	UART6.
BP_STM32F4_RESET_UART7	UART7.

BP_STM32F4_RESET_UART8	UART8.
BP_STM32F4_RESET_DAC	DAC.
BP_STM32F4_RESET_PWR	PWR.
BP_STM32F4_RESET_CAN1	CAN1.
BP_STM32F4_RESET_CAN2	CAN2.
BP_STM32F4_RESET_I2C1	I2C1.
BP_STM32F4_RESET_I2C2	I2C2.
BP_STM32F4_RESET_I2C3	I2C3.
BP_STM32F4_RESET_SPI1	SPI1.
BP_STM32F4_RESET_SPI2	SPI2.
BP_STM32F4_RESET_SPI3	SPI3.
BP_STM32F4_RESET_SPI4	SPI4.
BP_STM32F4_RESET_SPI5	SPI5.
BP_STM32F4_RESET_SPI6	SPI6.
BP_STM32F4_RESET_WWDG	WWDG.
BP_STM32F4_RESET_TIM1	TIM1.
BP_STM32F4_RESET_TIM2	TIM2.
BP_STM32F4_RESET_TIM3	TIM3.
BP_STM32F4_RESET_TIM4	TIM4.
BP_STM32F4_RESET_TIM5	TIM5.
BP_STM32F4_RESET_TIM6	TIM6.
BP_STM32F4_RESET_TIM7	TIM7.
BP_STM32F4_RESET_TIM8	TIM8.
BP_STM32F4_RESET_TIM9	TIM9.
BP_STM32F4_RESET_TIM10	TIM10.
BP_STM32F4_RESET_TIM11	TIM11.
BP_STM32F4_RESET_TIM12	TIM12.
BP_STM32F4_RESET_TIM13	TIM13.
BP_STM32F4_RESET_TIM14	TIM14.
BP_STM32F4_RESET_DSI	DSI.
BP_STM32F4_RESET_LTDC	LTDC.
BP_STM32F4_RESET_SAI1	SAI1.

BP_STM32F4_RESET_SAI2	SAI2.
BP_STM32F4_RESET_SYSCFG	SYSCFG.
BP_STM32F4_RESET_SDIO	SDIO.
BP_STM32F4_RESET_ADC	ADC.
BP_STM32F4_RESET_NONE	Special invalid value.

Macro

BP_NVIC_SOC_DEF_INT_CNT

<soc/stmicro/stm32f4/bp_stm32f4_def.h>

Number of NVIC interrupts.

Macro

BP_STM32F4_GPIO_MAX_BANK_CNT

<soc/stmicro/stm32f4/bp_stm32f4_def.h>

Maximum possible bank count for the STM32 F4 series.

Macro

BP_STM32F4_BASE_*

<soc/stmicro/stm32f4/bp_stm32f4_def.h>

Description STM32F4 peripheral base addresses.

BP_STM32F4_BASE_BKPSRAM_MEM_START	BACKUP SRAM START
BP_STM32F4_BASE_FMC	FMC
BP_STM32F4_BASE_QSPI	QSPI
BP_STM32F4_BASE_RNG	RNG
BP_STM32F4_BASE_HASH	HASH
BP_STM32F4_BASE_CRYP	CRYP
BP_STM32F4_BASE_DCMI	DCMI
BP_STM32F4_BASE_USBFS	USB OTG FS
BP_STM32F4_BASE_USBHS	USB OTG HS
BP_STM32F4_BASE_DMA2D	DMA2D
BP_STM32F4_BASE_ETH	ETHERNET
BP_STM32F4_BASE_DMA2	DMA2

BP_STM32F4_BASE_DMA1	DMA1
BP_STM32F4_BASE_FLASH	FLASH INTERFACE
BP_STM32F4_BASE_RCC	RCC
BP_STM32F4_BASE_CRC	CRC
BP_STM32F4_BASE_GPIOK	GPIOK
BP_STM32F4_BASE_GPIOJ	GPIOJ
BP_STM32F4_BASE_GPIOI	GPIOI
BP_STM32F4_BASE_GPIOH	GPIOH
BP_STM32F4_BASE_GPIOG	GPIOG
BP_STM32F4_BASE_GPIOF	GPIOF
BP_STM32F4_BASE_GPIOE	GPIOE
BP_STM32F4_BASE_GPIOD	GPIOD
BP_STM32F4_BASE_GPIOC	GPIOC
BP_STM32F4_BASE_GPIOB	GPIOB
BP_STM32F4_BASE_GPIOA	GPIOA
BP_STM32F4_BASE_DSI	DSI HOST
BP_STM32F4_BASE_LCD	LCD TFT
BP_STM32F4_BASE_SAI1	SAI1
BP_STM32F4_BASE_SPI6	SPI6
BP_STM32F4_BASE_SPI5	SPI5
BP_STM32F4_BASE_TIM11	TIM11
BP_STM32F4_BASE_TIM10	TIM10
BP_STM32F4_BASE_TIM9	TIM9
BP_STM32F4_BASE_EXTI	EXTI
BP_STM32F4_BASE_SYSCFG	SYSCFG
BP_STM32F4_BASE_SPI4	SPI4
BP_STM32F4_BASE_SPI1	SPI1
BP_STM32F4_BASE_SDIO	SDIO
BP_STM32F4_BASE_ADC	ADC
BP_STM32F4_BASE_UART6	USART6
BP_STM32F4_BASE_UART1	USART1
BP_STM32F4_BASE_TIM8	TIM8

BP_STM32F4_BASE_TIM1	TIM1
BP_STM32F4_BASE_USART8	USART8
BP_STM32F4_BASE_USART7	USART7
BP_STM32F4_BASE_DAC	DAC
BP_STM32F4_BASE_PWR	PWR
BP_STM32F4_BASE_CAN2	CAN2
BP_STM32F4_BASE_CAN1	CAN1
BP_STM32F4_BASE_I2C3	I2C3
BP_STM32F4_BASE_I2C2	I2C2
BP_STM32F4_BASE_I2C1	I2C1
BP_STM32F4_BASE_UART5	USART5
BP_STM32F4_BASE_UART4	USART4
BP_STM32F4_BASE_UART3	USART3
BP_STM32F4_BASE_UART2	USART2
BP_STM32F4_BASE_I2S3	I2S3
BP_STM32F4_BASE_SPI3	SPI3
BP_STM32F4_BASE_SPI2	SPI2
BP_STM32F4_BASE_I2S2	I2S2
BP_STM32F4_BASE_IWDG	IWDG
BP_STM32F4_BASE_WWDG	WWDG
BP_STM32F4_BASE_RTC	RTC
BP_STM32F4_BASE_TIM14	TIM14
BP_STM32F4_BASE_TIM13	TIM13
BP_STM32F4_BASE_TIM12	TIM12
BP_STM32F4_BASE_TIM7	TIM7
BP_STM32F4_BASE_TIM6	TIM6
BP_STM32F4_BASE_TIM5	TIM5
BP_STM32F4_BASE_TIM4	TIM4
BP_STM32F4_BASE_TIM3	TIM3
BP_STM32F4_BASE_TIM2	TIM2

Chapter

3 STM32F4 Reset and Clock Control

Interface module to the STM32 F4 Reset and Clock Control (RCC). This module also provides the implementation of the BASEplatform clock and reset modules. Note that the generic clock and reset modules can be used to manipulate the clock and reset lines at a higher level using a portable API.

Function

bp_stm32f4_rcc_bd_reset()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Resets the backup domain.

Prototype

```
void bp_stm32f4_rcc_bd_reset ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_clk_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables a specific clock.

Prototype

```
int bp_stm32f4_rcc_clk_dis ( bp_stm32f4_clk_gate_t clk );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `clk` Clock to disable.

Returned Errors `RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_clk_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables a specific clock.

Prototype
```
int bp_stm32f4_rcc_clk_en ( bp_stm32f4_clk_gate_t clk );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `clk` Clock to enable.

Returned Errors `RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_clk_freq_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the frequency of a specific clock in Hertz.

Prototype
```
int bp_stm32f4_rcc_clk_freq_get ( bp_stm32f4_clk_t clk,
                                  uint32_t *       p_clk_freq );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters
`clk` Clock to query.
`p_clk_freq` Pointer to the returned clock frequency in Hertz.

Returned Errors `RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_clk_id_is_valid()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Prototype

```
bool bp_stm32f4_rcc_clk_id_is_valid ( bp_stm32f4_clk_t clk );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`clk`	Clock id to check.

Returned Values

`true` if the clock id is valid, `false` otherwise.

Function

bp_stm32f4_rcc_clk_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returned the state of a specific clock. Returns `true` through `p_is_en` if the clock `clk` is enabled, `false` otherwise.

Prototype

```
int bp_stm32f4_rcc_clk_is_en ( bp_stm32f4_clk_gate_t clk,
                               bool *                p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`clk`	Clock to query.
`p_is_en`	Pointer to the returned state.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_css_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the clock security systems.

Prototype

```
void bp_stm32f4_rcc_css_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_css_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the clock security system.

Prototype

```
void bp_stm32f4_rcc_css_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_css_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the clock security system.

Prototype

```
bool bp_stm32f4_rcc_css_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_gate_id_is_valid()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Prototype

```
bool bp_stm32f4_rcc_gate_id_is_valid ( bp_stm32f4_clk_gate_t clk_gate );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`clk_gate` Clock gate id to check.

Returned Values `true` if the clock id is valid, `false` otherwise.

Function

bp_stm32f4_rcc_hse_bypass_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the High-Speed External(HSE) oscillator bypass.

Prototype

```
void  bp_stm32f4_rcc_hse_bypass_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hse_bypass_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the High-Speed External(HSE) oscillator bypass.

Prototype

```
void  bp_stm32f4_rcc_hse_bypass_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hse_bypass_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the High-Speed External(HSE) oscillator bypass.

Prototype

```
bool  bp_stm32f4_rcc_hse_bypass_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hse_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the High-Speed External(HSE) oscillator.

Prototype

```
void bp_stm32f4_rcc_hse_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hse_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the High-Speed External(HSE) oscillator. If `wait` is true `bp_stm32f4_rcc_hse_en()` will busy wait until the oscillator is stable. If the oscillator does not become ready within approximately 5 milliseconds `RTNC_TIMEOUT` is returned;

Prototype

```
int bp_stm32f4_rcc_hse_en ( bool wait );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`wait`	Set to `true` to busy-wait for the oscillator to be ready, `false` to return immediately.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_hse_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the High-Speed External(HSE) oscillator.

Note that this will just return the state of the HSEON bit and not the ready flag.
`bp_stm32f4_rcc_hse_ready_wait()` should be used to wait for the HSE oscillator to be ready.

Prototype

```
bool bp_stm32f4_rcc_hse_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hse_is_rdy()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the ready flag state of the High-Speed External(HSE) oscillator.

Prototype

```
bool  bp_stm32f4_rcc_hse_is_rdy  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hsi_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the High-Speed Internal(HSI) oscillator.

Prototype

```
void  bp_stm32f4_rcc_hsi_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hsi_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the High-Speed Internal(HSI) oscillator. If `wait` is true `bp_stm32f4_rcc_hsi_en()` will busy wait until the oscillator is stable. If the oscillator does not become ready within approximately 5 milliseconds `RTNC_TIMEOUT` is returned;

Prototype

```
int  bp_stm32f4_rcc_hsi_en  ( bool  wait );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `wait` Set to `true` to busy-wait for the oscillator to be ready, `false` to return immediately.

Returned Errors `RTNC_SUCCESS` `RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_hsi_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the High-Speed Internal(HSI) oscillator.

Note that this will just return the state of the HSION bit and not the ready flag.
`bp_stm32f4_rcc_hsi_ready_wait()` should be used to wait for the HSI oscillator to be ready.

Prototype

```
bool bp_stm32f4_rcc_hsi_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hsi_is_rdy()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the ready flag state of the High-Speed Internal(HSI) oscillator.

Prototype

```
bool bp_stm32f4_rcc_hsi_is_rdy ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hsical_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the value of the High-Speed Internal(HSI) oscillator calibration value.

Prototype

```
uint32_t bp_stm32f4_rcc_hsical_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hsitrim_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the value of the High-Speed Internal(HSI) oscillator trim value.

Prototype

```
uint32_t  bp_stm32f4_rcc_hsitrim_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_hsitrim_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Sets the value of the High-Speed Internal(HSI) oscillator trim value.

Prototype

```
int  bp_stm32f4_rcc_hsitrim_set  ( uint32_t  trim );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`trim` Trim value to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_i2s_pll_cfg_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Retrieves the I2S PLL configuration. See `bp_stm32f4_rcc_i2s_pll_cfg_t` for details of the configuration parameters.

Prototype

```
int  bp_stm32f4_rcc_i2s_pll_cfg_get  ( bp_stm32f4_rcc_i2s_pll_cfg_t *  p_cfg );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `p_cfg` Pointer to the returned configuration.

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_i2s_pll_cfg_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Configures the I2S PLL. See `bp_stm32f4_rcc_i2s_pll_cfg_t` for details of the configuration parameters.

Prototype

```
int bp_stm32f4_rcc_i2s_pll_cfg_set ( bp_stm32f4_rcc_i2s_pll_cfg_t * p_cfg );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `p_cfg` Configuration to apply.

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_i2s_pll_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the I2S PLL.

Prototype

```
void bp_stm32f4_rcc_i2s_pll_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_i2s_pll_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the I2S PLL. If `wait` is true `bp_stm32f4_rcc_main_pll_en()` will busy wait until the PLL is ready. If the PLL does not become ready within approximately 100 microseconds `RTNC_TIMEOUT` is returned;

Prototype

```
int  bp_stm32f4_rcc_i2s_pll_en  ( bool  wait );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`wait`	Set to `true` to busy-wait for the PLL to be ready, `false` to return immediately.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_i2s_pll_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the I2S PLL.

Note that this will just return the state of the PLLON bit and not the ready flag. `bp_stm32f4_rcc_i2s_pll_is_rdy()` should be used to check if the main PLL is ready.

Prototype

```
bool  bp_stm32f4_rcc_i2s_pll_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_i2s_pll_is_rdy()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the ready flag state of the I2S PLL.

Prototype

```
bool  bp_stm32f4_rcc_i2s_pll_is_rdy  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_lpclk_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables a specific clock in low-power mode.

Prototype

```
int bp_stm32f4_rcc_lpclk_dis ( bp_stm32f4_clk_gate_t clk );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`clk` Clock to disable.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_lpclk_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables a specific clock in low-power mode.

Prototype

```
int bp_stm32f4_rcc_lpclk_en ( bp_stm32f4_clk_gate_t clk );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`clk` Clock to enable.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_lse_bypass_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the Low-Speed External(LSE) oscillator bypass.

Prototype

```
void bp_stm32f4_rcc_lse_bypass_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_lse_bypass_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the Low-Speed External(LSE) oscillator bypass.

Prototype

```
void bp_stm32f4_rcc_lse_bypass_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_lse_bypass_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the Low-Speed External(LSE) oscillator bypass.

Prototype

```
bool bp_stm32f4_rcc_lse_bypass_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_lse_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the Low-Speed External(LSE) oscillator.

Prototype `void bp_stm32f4_rcc_lse_dis ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_lse_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the Low-Speed External(LSE) oscillator. If `wait` is true `bp_stm32f4_rcc_lse_en()` will busy wait until the oscillator is stable. If the oscillator does not become ready within approximately 1 second `RTNC_TIMEOUT` is returned;

Prototype `int bp_stm32f4_rcc_lse_en ( bool wait );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`wait`	Set to `true` to busy-wait for the oscillator to be ready, `false` to return immediately.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_lse_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the Low-Speed External(LSE) oscillator.

Note that this will just return the state of the LSEON bit and not the ready flag. `bp_stm32f4_rcc_lse_is_rdy()` to check if the LSE oscillator is ready.

Prototype `bool bp_stm32f4_rcc_lse_is_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_lse_is_rdy()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the ready flag state of the Low-Speed Internal(LSE) oscillator.

Prototype

```
bool bp_stm32f4_rcc_lse_is_rdy ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_lse_mode_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the configured mode of the Low-Speed External(LSE) oscillator.

Prototype

```
bp_stm32f4_rcc_lse_mode_t bp_stm32f4_rcc_lse_mode_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_lse_mode_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Sets the mode of the Low-Speed External(LSE) oscillator. See `bp_stm32f4_rcc_lse_mode_t` for a list of modes.

Prototype

```
int bp_stm32f4_rcc_lse_mode_set ( bp_stm32f4_rcc_lse_mode_t mode );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`mode` Mode to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_lsi_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the Low-Speed Internal(LSI) oscillator.

Prototype

```
void bp_stm32f4_rcc_lsi_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_lsi_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the Low-Speed Internal(LSI) oscillator. If `wait` is true `bp_stm32f4_rcc_lsi_en()` will busy wait until the oscillator is stable. If the oscillator does not become ready within approximately 5 milliseconds `RTNC_TIMEOUT` is returned;

Prototype

```
int bp_stm32f4_rcc_lsi_en ( bool wait );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`wait`	Set to `true` to busy-wait for the oscillator to be ready, `false` to return immediately.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_lsi_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the Low-Speed Internal(LSI) oscillator.

Note that this will just return the state of the LSION bit and not the ready flag. `bp_stm32f4_rcc_lse_is_rdy()` to check if the LSI oscillator is ready.

Prototype

```
bool bp_stm32f4_rcc_lsi_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_lsi_is_rdy()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the ready flag state of the Low-Speed Internal(LSI) oscillator.

Prototype

```
bool  bp_stm32f4_rcc_lsi_is_rdy  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_main_clk_cfg_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Retrieves the main clocks (AHB, APB1, APB2) configuration. See `bp_stm32f4_rcc_main_clk_cfg_t` for a list of configuration parameters.

Prototype

```
int  bp_stm32f4_rcc_main_clk_cfg_get  ( bp_stm32f4_rcc_main_clk_cfg_t *  p_cfg );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`p_cfg` Configuration to apply.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_main_clk_cfg_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Configures the main clocks (AHB, APB1, APB2). See `bp_stm32f4_rcc_main_clk_cfg_t` for a list of configuration parameters.

Prototype

```
int  bp_stm32f4_rcc_main_clk_cfg_set  ( bp_stm32f4_rcc_main_clk_cfg_t *  p_cfg );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `p_cfg` Configuration to apply.

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_main_pll_cfg_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Retrieves the main PLL configuration. See `bp_stm32f4_rcc_pll_cfg_t` for details of the configuration parameters.

Prototype

```
int bp_stm32f4_rcc_main_pll_cfg_get ( bp_stm32f4_rcc_pll_cfg_t * p_cfg );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `p_cfg` Pointer to the returned configuration.

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_main_pll_cfg_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Configures the main PLL. See `bp_stm32f4_rcc_pll_cfg_t` for details of the configuration parameters.

Prototype

```
int bp_stm32f4_rcc_main_pll_cfg_set ( bp_stm32f4_rcc_pll_cfg_t * p_cfg );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `p_cfg` Configuration to apply.

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_main_pll_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the main PLL.

Prototype

```
void bp_stm32f4_rcc_main_pll_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_main_pll_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the main PLL. If `wait` is true `bp_stm32f4_rcc_main_pll_en()` will busy wait until the PLL is ready. If the PLL does not become ready within approximately 100 microseconds `RTNC_TIMEOUT` is returned;

Prototype

```
int bp_stm32f4_rcc_main_pll_en ( bool wait );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`wait` Set to `true` to busy-wait for the PLL to be ready, `false` to return immediately.

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_main_pll_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the main PLL.

Note that this will just return the state of the PLLON bit and not the ready flag.
`bp_stm32f4_rcc_main_pll_is_rdy()` should be used to check if the main PLL is ready.

Prototype `bool bp_stm32f4_rcc_main_pll_is_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_main_pll_is_rdy()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the ready flag state of the main PLL.

Prototype `bool bp_stm32f4_rcc_main_pll_is_rdy ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_mco1_pre_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the configured MCO1 clock output prescaler.

Prototype `uint32_t bp_stm32f4_rcc_mco1_pre_get ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values Configured prescaler.

Function

bp_stm32f4_rcc_mco1_pre_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Sets the MCO1 prescaler.

Prototype `int bp_stm32f4_rcc_mco1_pre_set ( uint32_t pre );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `pre` Clock prescaler to set.

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_mco1_src_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Retrieves the configured MCO1 clock source. See `bp_stm32f4_rcc_mco1_clk_sel_t` for a list of available sources.

Prototype

```
bp_stm32f4_rcc_mco1_clk_sel_t bp_stm32f4_rcc_mco1_src_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values Configured MCO1 clock source.

Function

bp_stm32f4_rcc_mco1_src_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Sets the source of the MCO1 clock output. See `bp_stm32f4_rcc_mco1_clk_sel_t` for a list of available sources.

Prototype

```
int bp_stm32f4_rcc_mco1_src_set ( bp_stm32f4_rcc_mco1_clk_sel_t src );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `src` Clock source to set.

Returned Errors `RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_mco2_pre_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the configured MCO2 clock output prescaler.

Prototype

```
uint32_t  bp_stm32f4_rcc_mco2_pre_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

Configured prescaler.

Function

bp_stm32f4_rcc_mco2_pre_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Sets the MCO2 prescaler.

Prototype

```
int  bp_stm32f4_rcc_mco2_pre_set  ( uint32_t  pre );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`pre` Clock prescaler to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_mco2_src_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Retrieves the configured MCO2 clock source. See `bp_stm32f4_rcc_mco2_clk_sel_t` for a list of available sources.

Prototype

```
bp_stm32f4_rcc_mco2_clk_sel_t  bp_stm32f4_rcc_mco2_src_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values Configured MCO2 clock source.

Function

bp_stm32f4_rcc_mco2_src_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Sets the source of the MCO2 clock output. See `bp_stm32f4_rcc_mco2_clk_sel_t` for a list of available sources.

Prototype

```
int bp_stm32f4_rcc_mco2_src_set ( bp_stm32f4_rcc_mco2_clk_sel_t src );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`src` Clock source to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_reset_assert()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Asserts a peripheral reset line.

Prototype

```
int bp_stm32f4_rcc_reset_assert ( bp_stm32f4_reset_t reset );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`reset` Reset line to assert.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_reset_deassert()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Deasserts a peripheral reset line.

Prototype

```
int bp_stm32f4_rcc_reset_deassert ( bp_stm32f4_reset_t reset );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`reset`	Reset line to deassert.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_reset_is_assert()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the asserted/deasserted state of a reset line.

Prototype

```
int bp_stm32f4_rcc_reset_is_assert ( bp_stm32f4_reset_t reset,
                                     bool *              p_is_asserted );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`reset`	Reset line to query.
`p_is_asserted`	Pointer to the result, `true` if the reset is asserted, `false` otherwise.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_rcc_reset_is_valid()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Checks if a peripheral reset id is valid.

Prototype `bool bp_stm32f4_rcc_reset_is_valid ( periph_reset_id );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `periph_reset_id` Peripheral reset id to check.

Returned Values `true` if the peripheral reset id is valid, `false` otherwise.

Function

bp_stm32f4_rcc_rtc_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the real-time clock.

Prototype `void bp_stm32f4_rcc_rtc_dis ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_rtc_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the real-time clock.

Prototype `void bp_stm32f4_rcc_rtc_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_rtc_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the real-time clock.

Prototype `bool bp_stm32f4_rcc_rtc_is_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_rtc_pre_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the configured RTC clock prescaler.

Prototype `uint32_t bp_stm32f4_rcc_rtc_pre_get ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values Configured prescaler.

Function

bp_stm32f4_rcc_rtc_pre_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Sets the RTC prescaler. This only applied when the RTC clock source is set to the High Speed External oscillator.

Prototype `int bp_stm32f4_rcc_rtc_pre_set ( uint32_t rtc_pre );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`rtc_pre`	Clock prescaler to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_rtc_src_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the configured source of the real-time clock (RTC).

Prototype

```
bp_stm32f4_rcc_rtc_src_sel_t  bp_stm32f4_rcc_rtc_src_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_rtc_src_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Sets the source of the real-time clock (RTC). See bp_stm32f4_rcc_rtc_src_sel_t for a list of available sources.

Prototype

```
int  bp_stm32f4_rcc_rtc_src_set  ( bp_stm32f4_rcc_rtc_src_sel_t  src );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

src	Clock source to set.

Returned Errors

RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_rcc_sai_pll_cfg_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Retrieves the SAI PLL configuration. See bp_stm32f4_rcc_sai_pll_cfg_t for details of the configuration parameters.

Prototype

```
int  bp_stm32f4_rcc_sai_pll_cfg_get  ( bp_stm32f4_rcc_sai_pll_cfg_t *  p_cfg );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `p_cfg` Pointer to the returned configuration.

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_sai_pll_cfg_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Configures the SAI PLL. See `bp_stm32f4_rcc_sai_pll_cfg_t` for details of the configuration parameters.

Prototype
```
int bp_stm32f4_rcc_sai_pll_cfg_set ( bp_stm32f4_rcc_sai_pll_cfg_t * p_cfg );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `p_cfg` Configuration to apply.

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_sai_pll_dis()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Disables the SAI PLL.

Prototype
```
void bp_stm32f4_rcc_sai_pll_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_sai_pll_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Enables the SAI PLL. If `wait` is true `bp_stm32f4_rcc_sai_pll_en()` will busy wait until the PLL is ready. If the PLL does not become ready within approximately 100 microseconds `RTNC_TIMEOUT` is returned;

Prototype `int bp_stm32f4_rcc_sai_pll_en ( bool wait );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`wait`	Set to `true` to busy-wait for the PLL to be ready, `false` to return immediately.

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_rcc_sai_pll_is_en()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the enabled/disabled state of the SAI PLL.

Note that this will just return the state of the PLLON bit and not the ready flag.
`bp_stm32f4_rcc_sai_pll_is_rdy()` should be used to check if the main PLL is ready.

Prototype `bool bp_stm32f4_rcc_sai_pll_is_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_sai_pll_is_rdy()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the ready flag state of the SAI PLL.

Prototype `bool bp_stm32f4_rcc_sai_pll_is_rdy ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_rcc_sys_clk_status_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the system clock source as reported by the hardware. See `bp_stm32f4_rcc_sys_clk_sel_t` for a list of sources.

Prototype

```
bp_stm32f4_rcc_sys_clk_sel_t bp_stm32f4_rcc_sys_clk_status_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

Configured clock source.

Function

bp_stm32f4_rcc_sys_clk_switch_get()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Returns the configured system clock source. See `bp_stm32f4_rcc_sys_clk_sel_t` for a list of sources. Note that this returns the value configured, use `bp_stm32f4_rcc_sys_clk_status_get()` to get the current clock source used by the hardware.

Prototype

```
bp_stm32f4_rcc_sys_clk_sel_t bp_stm32f4_rcc_sys_clk_switch_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

Configured clock source.

Function

bp_stm32f4_rcc_sys_clk_switch_set()

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Configures the system clock source. See `bp_stm32f4_rcc_sys_clk_sel_t` for a list of available sources.

Prototype

```
int bp_stm32f4_rcc_sys_clk_switch_set ( bp_stm32f4_rcc_sys_clk_sel_t clk_sel )
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`clk_sel` Clock source to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Data Type

bp_stm32f4_rcc_lse_mode_t

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Low-Speed External(LSE) oscillator drive mode. Used with `bp_stm32f4_rcc_lse_mode_set()` and `bp_stm32f4_rcc_lse_mode_get()`.

Values

`BP_STM32F4_RCC_LSE_MODE_LP`	Low-power drive mode.
`BP_STM32F4_RCC_LSE_MODE_HP`	High-power drive mode.

Data Type

bp_stm32f4_rcc_mco1_clk_sel_t

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

MCO1 clock source selection.

Values

`BP_STM32F4_MCO1_CLK_HSI`	High-speed internal oscillator selected.
`BP_STM32F4_MCO1_CLK_LSE`	Low-speed external oscillator selected.
`BP_STM32F4_MCO1_CLK_HSE`	High-speed external oscillator selected.
`BP_STM32F4_MCO1_CLK_PLL`	Main PLL output P selected.

Data Type

bp_stm32f4_rcc_mco2_clk_sel_t

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

MCO2 clock source selection.

Values

BP_STM32F4_MCO2_CLK_SYS	System clock selected.
BP_STM32F4_MCO2_CLK_PLLI2S	I2S PLL output selected.
BP_STM32F4_MCO2_CLK_HSE	High-speed external oscillator selected.
BP_STM32F4_MCO2_CLK_PLL	Main PLL output P selected.

Data Type

bp_stm32f4_rcc_pll_src_sel_t

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Main and I2S PLL source clock. Used within the bp_stm32f4_rcc_pll_cfg_t PLL configuration structure.

Values

BP_STM32F4_RCC_PLL_SRC_HSI	High-Speed Internal (HSI) oscillator selected.
BP_STM32F4_RCC_PLL_SRC_HSE	High-Speed External (HSE) oscillator selected.

Data Type

bp_stm32f4_rcc_rtc_src_sel_t

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Low-Speed External(LSE) oscillator drive mode. Used with bp_stm32f4_rcc_lse_mode_set() and bp_stm32f4_rcc_lse_mode_get().

Values

BP_STM32F4_RCC_RTC_SRC_NONE	No clock source selected.
BP_STM32F4_RCC_RTC_SRC_LSE	RTC clock derived from Low-Speed External (LSE) oscillator.
BP_STM32F4_RCC_RTC_SRC_LSI	RTC clock derived from Low-Speed Internal (LSI) oscillator.
BP_STM32F4_RCC_RTC_SRC_HSE	RTC clock derived from High-Speed External (HSE) oscillator.

Data Type

bp_stm32f4_rcc_sys_clk_sel_t

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

System clock switch source selection.

Values

BP_STM32F4_SYS_CLK_HSI	High-speed internal oscillator selected.

`BP_STM32F4_SYS_CLK_HSE`	High-speed external oscillator selected.
`BP_STM32F4_SYS_CLK_PLLP`	Main PLL P output clock selected.
`BP_STM32F4_SYS_CLK_PLLR`	Main PLL R output clock selected.

Data Type

bp_stm32f4_rcc_i2s_pll_cfg_t

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

I2S PLL configuration structure.

Members

`pllsr`	`uint32_t`	I2S PLL division factor for I2S clocks.
`pllsq`	`uint32_t`	I2S PLL division factor for SAI clocks.
`pllsp`	`uint32_t`	I2S PLL division factor for SPDIF clocks.
`pllsn`	`uint32_t`	I2S PLL multiplication factor for VCO.
`pllsm`	`uint32_t`	I2S PLL division factor for audio PLL input clock.

Data Type

bp_stm32f4_rcc_main_clk_cfg_t

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Main bus clocks (AHB, APB1, APB2) configuration structure.

Members

`hpre`	`uint32_t`	AHB prescaler.
`ppre1`	`uint32_t`	APB1 prescaler.
`ppre2`	`uint32_t`	APB2 prescaler.

Data Type

bp_stm32f4_rcc_pll_cfg_t

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

Main PLL configuration structure.

Members

clk_src	bp_stm32f4_rcc_pll_src_sel_t	PLL clock source.
pllr	uint32_t	Main PLL division factor for I2S, SAI, SYSTEM and SPDIF.
pllq	uint32_t	Main PLL division factor for USB OTG FS and SDIO.
pllp	uint32_t	Main PLL division factor for main system clock.
plln	uint32_t	Main PLL multiplication factor for VCO.
pllm	uint32_t	Main PLL division factor for the main PLL input clock.

Data Type

bp_stm32f4_rcc_sai_pll_cfg_t

<soc_comp/stmicro/stm32f4_rcc/bp_stm32f4_rcc.h>

SAI PLL configuration structure.

Members

pllsaiq	uint32_t	SAI PLL division factor for SAI clocks.
pllsaip	uint32_t	SAI PLL division factor for 48 MHz clocks.
pllsain	uint32_t	SAI PLL multiplication factor for VCO.
pllsaim	uint32_t	SAI PLL division factor for Audio input clocks.

Chapter

4

STM32F4 Embedded Flash Interface

Interface module to the STM32 F4 embedded Flash memory.

Function

bp_stm32f4_flash_dcache_dis()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Disables the flash data cache.

Prototype

```
void  bp_stm32f4_flash_dcache_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_flash_dcache_en()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Enables the flash data cache.

Prototype

```
void  bp_stm32f4_flash_dcache_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_flash_dcache_is_en()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Returned the enabled/disabled state of the flash data cache. Returns `true` if the cache is enabled, `false` otherwise.

Prototype

```
bool bp_stm32f4_flash_dcache_is_en ( p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`p_is_en`	Pointer to the returned state.

Returned Values

`true` if the flash data cache is enabled, `false` otherwise.

Function

bp_stm32f4_flash_dcache_reset()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Resets the data cache.

Prototype

```
void bp_stm32f4_flash_dcache_reset ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_flash_icache_dis()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Disables the flash instruction cache.

Prototype

```
void bp_stm32f4_flash_icache_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_flash_icache_en()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Enables the flash instruction cache.

Prototype

```
void bp_stm32f4_flash_icache_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_flash_icache_is_en()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Returned the enabled/disabled state of the flash instruction cache. Returns `true` if the cache is enabled, `false` otherwise.

Prototype

```
bool bp_stm32f4_flash_icache_is_en ( p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`p_is_en`	Pointer to the returned state.

Returned Values

`true` if the flash instruction cache is enabled, `false` otherwise.

Function

bp_stm32f4_flash_icache_reset()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Resets the instruction cache.

Prototype

```
void bp_stm32f4_flash_icache_reset ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_flash_latency_get()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Returns the currently configured flash access latency.

Prototype

```
uint32_t bp_stm32f4_flash_latency_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

Flash access latency in cycles.

Function

bp_stm32f4_flash_latency_set()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Sets the flash access latency.

Prototype

```
int bp_stm32f4_flash_latency_set ( uint32_t flash_latency );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`flash_latency`	Flash access latency to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_flash_prefetch_dis()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Disables the flash prefetcher.

Prototype

```
void bp_stm32f4_flash_prefetch_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_flash_prefetch_en()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Enables the flash prefetcher.

Prototype

```
void bp_stm32f4_flash_prefetch_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_flash_prefetch_is_en()

<soc_comp/stmicro/stm32f4_flash/bp_stm32f4_flash.h>

Returned the enabled/disabled state of the flash prefetch feature. Returns `true` if the prefetcher is enabled, `false` otherwise.

Prototype

```
bool bp_stm32f4_flash_prefetch_is_en ( p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`p_is_en`	Pointer to the returned state.

Returned Values

`true` if the flash prefetcher is enabled, `false` otherwise.

Chapter

5 STM32F4 Power Control

Interface module to the STM32F4 power control.

Function

bp_stm32f4_over_drive_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables the flash memory overdrive.

Prototype

```
void bp_stm32f4_over_drive_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_adcdc1_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables ADCDC1.

Prototype

```
void bp_stm32f4_pwr_adcdc1_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_adcdc1_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables ADCDC1.

Prototype

```
void bp_stm32f4_pwr_adcdc1_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_adcdc1_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the ADCDC1 bit.

Prototype

```
bool bp_stm32f4_pwr_adcdc1_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the ADCDC1 is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_bre_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables low power deep sleep.

Prototype

```
void bp_stm32f4_pwr_bre_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_bre_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables low power deep-sleep.

Prototype

```
void bp_stm32f4_pwr_bre_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_bre_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the low power deep sleep.

Prototype

```
bool bp_stm32f4_pwr_bre_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if low power deep sleep is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_dbp_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables backup domain write protection.

Prototype

```
void bp_stm32f4_pwr_dbp_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_dpb_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables backup domain write protection.

Prototype

```
void bp_stm32f4_pwr_dpb_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_dpb_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the backup domain write protection.

Prototype

```
bool bp_stm32f4_pwr_dpb_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if backup domain write protection is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_flash_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables the flash memory while the system is running. This function will set the FMSSR bit.

Prototype

```
void bp_stm32f4_pwr_flash_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_flash_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables the flash memory while the system is running. This function will clear the FMSSR bit.

Prototype

```
void bp_stm32f4_pwr_flash_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

RTNC_SUCCESS
RTNC_TIMEOUT

Function

bp_stm32f4_pwr_flash_if_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables the flash interface while the system is running. This function will set the FISSR bit.

Prototype

```
void bp_stm32f4_pwr_flash_if_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_flash_if_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables the flash interface while the system is running. This function will clear the FISSR bit.

Prototype

```
void bp_stm32f4_pwr_flash_if_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

RTNC_SUCCESS
RTNC_TIMEOUT

Function

bp_stm32f4_pwr_flash_if_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the flash interface in run mode.

Prototype

```
bool bp_stm32f4_pwr_flash_if_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the flash interface is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_flash_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the flash memory in run mode.

Prototype

```
bool bp_stm32f4_pwr_flash_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the flash memory is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_flash_ods_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the flash memory overdrive switching.

Prototype

```
bool bp_stm32f4_pwr_flash_ods_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if the flash memory overdrive switching is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_flash_over_drive_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the flash memory overdrive.

Prototype

```
bool bp_stm32f4_pwr_flash_over_drive_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if the flash memory overdrive is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_flash_under_drive_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the flash memory under-drive.

Prototype

```
bool bp_stm32f4_pwr_flash_under_drive_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if the flash memory under-drive is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_fpds_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables flash power down in stop mode.

Prototype

```
void bp_stm32f4_pwr_fpds_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_fpds_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables flash power down in stop mode.

Prototype

```
void bp_stm32f4_pwr_fpds_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_fpds_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the flash power down in stop mode feature.

Prototype

```
bool bp_stm32f4_pwr_fpds_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if flash power down in stop mode is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_lpds_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables low power deep sleep.

Prototype

```
void bp_stm32f4_pwr_lpds_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_lpds_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables low power deep-sleep.

Prototype

```
void  bp_stm32f4_pwr_lpds_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_lpds_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the low power deep sleep.

Prototype

```
bool  bp_stm32f4_pwr_lpds_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if low power deep sleep is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_lpuds_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables low-power regulator in deepsleep under-drive mode.

Prototype

```
void  bp_stm32f4_pwr_lpuds_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_lpuds_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables low-power regulator in deepsleep under-drive mode.

Prototype

```
void  bp_stm32f4_pwr_lpuds_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_lpuds_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the low-power regulator in deepsleep under-drive mode.

Prototype

```
bool  bp_stm32f4_pwr_lpuds_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the low power regulator in deepsleep mode is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_mruds_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables main regulator power down in deep sleep mode.

Prototype

```
void  bp_stm32f4_pwr_mruds_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_mruds_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables main regulator power down in deep sleep mode.

Prototype

```
void  bp_stm32f4_pwr_mruds_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_mruds_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the main regulator power down in deep-sleep mode feature.

Prototype

```
bool  bp_stm32f4_pwr_mruds_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the main regulator power down in deep-sleep is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_od_is_rdy()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the state of the overdrive ready flag.

Prototype

```
bool  bp_stm32f4_pwr_od_is_rdy  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if overdrive is ready, `false` otherwise.

Function

bp_stm32f4_pwr_ods_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables the flash memory overdrive switching.

Prototype

```
void bp_stm32f4_pwr_ods_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_ods_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables the flash memory overdrive switching.

Prototype

```
void bp_stm32f4_pwr_ods_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_ods_is_rdy()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the state of the overdrive switching ready flag.

Prototype

```
bool bp_stm32f4_pwr_ods_is_rdy ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if overdrive switching is ready, `false` otherwise.

Function

bp_stm32f4_pwr_over_drive_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables the flash memory overdrive.

Prototype

```
void bp_stm32f4_pwr_over_drive_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors `RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_pdds_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables power-down deep sleep.

Prototype

```
void bp_stm32f4_pwr_pdds_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_pdds_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables power down deep-sleep.

Prototype

```
void bp_stm32f4_pwr_pdds_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_pdds_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of power down deep sleep.

Prototype

```
bool  bp_stm32f4_pwr_pdds_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if power down deep sleep is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_pvd_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables the power voltage detector.

Prototype

```
void  bp_stm32f4_pwr_pvd_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_pvd_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables the power voltage detector.

Prototype

```
void  bp_stm32f4_pwr_pvd_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_pvd_get()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns currently configured PVD voltage level. See `bp_stm32f4_pwr_pvd_t` for a list of supported levels.

Prototype

```
bp_stm32f4_pwr_pvd_t  bp_stm32f4_pwr_pvd_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

Configured PVD voltage level.

Function

bp_stm32f4_pwr_pvd_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the power voltage detector.

Prototype

```
bool  bp_stm32f4_pwr_pvd_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the power voltage detector is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_pvd_set()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Sets the PVD voltage level. See bp_stm32f4_pwr_pvd_t for a list of supported levels.

Prototype

```
int bp_stm32f4_pwr_pvd_set ( bp_stm32f4_pwr_pvd_t pvd );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

pvd	Voltage level to set.

Returned Errors

RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_pwr_sbf_is_set()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the state of the standby flag.

Prototype

```
bool bp_stm32f4_pwr_sbf_is_set ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the standby flag is set, `false` otherwise.

Function

bp_stm32f4_pwr_standby_flg_clr()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Clears the standby flag.

Prototype

```
void bp_stm32f4_pwr_standby_flg_clr ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_ud_is_rdy()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the state of the under-drive ready flag. The flag is reset after reading.

Prototype

```
bool bp_stm32f4_pwr_ud_is_rdy ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if under-drive is ready, `false` otherwise.

Function

bp_stm32f4_pwr_under_drive_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables the flash memory under-drive.

Prototype

```
void bp_stm32f4_pwr_under_drive_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_under_driver_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables the flash memory under-drive.

Prototype

```
void bp_stm32f4_pwr_under_driver_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_vos_is_rdy()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the state of the voltage scaling ready flag.

Prototype

```
bool bp_stm32f4_pwr_vos_is_rdy ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if voltage scaling is ready, `false` otherwise.

Function

bp_stm32f4_pwr_vos_scale_get()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the configured regulator voltage scaling mode. See `bp_stm32f4_pwr_vos_t` for a list of scaling mode.

Prototype

```
bp_stm32f4_pwr_vos_t bp_stm32f4_pwr_vos_scale_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

Current regulator voltage scaling mode.

Function

bp_stm32f4_pwr_vos_scale_set()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Sets the regulator voltage scaling mode. See `bp_stm32f4_pwr_vos_t` for a list of scaling mode.

Prototype

```
int bp_stm32f4_pwr_vos_scale_set ( bp_stm32f4_pwr_vos_t scale );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `scale` Scaling mode to apply.

Returned Errors `RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_pwr_wakeup_flg_clr()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Clears the wakeup flag.

Prototype

```
void bp_stm32f4_pwr_wakeup_flg_clr ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_wuf_is_set()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the state of the wakeup flag.

Prototype

```
bool bp_stm32f4_pwr_wuf_is_set ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if the wakeup flag is set, `false` otherwise.

Function

bp_stm32f4_pwr_wup1_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables WKUP1 pin.

Prototype

```
void bp_stm32f4_pwr_wup1_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_wup1_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables WKUP1 pin.

Prototype

```
void bp_stm32f4_pwr_wup1_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_wup1_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the WKUP1 pin.

Prototype

```
bool bp_stm32f4_pwr_wup1_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the WKUP1 pin is enabled, `false` otherwise.

Function

bp_stm32f4_pwr_wup2_dis()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Disables WKUP2 pin.

Prototype

```
void bp_stm32f4_pwr_wup2_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_stm32f4_pwr_wup2_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Enables WKUP2 pin.

Prototype

```
void  bp_stm32f4_pwr_wup2_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`

Function

bp_stm32f4_pwr_wup2_is_en()

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

Returns the enabled/disabled state of the WKUP2 pin.

Prototype

```
bool  bp_stm32f4_pwr_wup2_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the WKUP2 pin is enabled, `false` otherwise.

Data Type

bp_stm32f4_pwr_pvd_t

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

STM32F4 PVD level selection.

Values

BP_STM32F4_PWR_PVD_2_0V	2.0V.
BP_STM32F4_PWR_PVD_2_1V	2.1V.
BP_STM32F4_PWR_PVD_2_3V	2.3V.
BP_STM32F4_PWR_PVD_2_5V	2.5V.
BP_STM32F4_PWR_PVD_2_6V	2.6V.
BP_STM32F4_PWR_PVD_2_7V	2.7V.
BP_STM32F4_PWR_PVD_2_8V	2.8V.
BP_STM32F4_PWR_PVD_2_9V	2.9V.

Data Type

bp_stm32f4_pwr_vos_t

<soc_comp/stmicro/stm32f4_pwr/bp_stm32f4_pwr.h>

STM32F4 regulator voltage scaling.

Values

BP_STM32F4_PWR_VOS_SCALE3	Scale 3 mode.
BP_STM32F4_PWR_VOS_SCALE2	Scale 2 mode.
BP_STM32F4_PWR_VOS_SCALE1	Scale 1 mode.

Chapter

6 STM32F4 GPIO Multiplexer

GPIO MUX interface module for the STMicroelectronics STM32 F4 series. This module can be used to configure the STM32 F4 pin mode and alternate function.

Note that since the GPIO pin direction cannot be set independently from the alternate or analog mode changing the mode of a pin will override any direction set using the GPIO module and driver. The opposite is also true, setting the direction of a GPIO pin from the GPIO module will override the current mode unconditionally.

Pin and bank numbering follow the manufacturer's documentation with GPIO bank letter name converted to a bank number. For example, bank A is 0, bank B is 1 and bank K is 10.

Function

bp_stm32f4_gpio_mux_altf_get()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

Retrieves the alternate function of a pin.

Prototype

```
int  bp_stm32f4_gpio_mux_altf_get  ( uint32_t   bank,
                                     uint32_t   pin,
                                     uint32_t *  p_altf );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`bank`	Bank number of the pin to set.
`pin`	Pin number of the pin to set.
`p_altf`	Pointer to the returned alternate function.

Returned Errors RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_gpio_mux_altf_set()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

Sets the alternate function of a pin. Note that for the alternate function to be usable the proper mode should be set using bp_stm32f4_gpio_mux_mode_set().

Prototype

```
int  bp_stm32f4_gpio_mux_altf_set  ( uint32_t  bank,
                                     uint32_t  pin,
                                     uint32_t  altf );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

bank	Bank number of the pin to set.
pin	Pin number of the pin to set.
altf	Alternate function to set.

Returned Errors RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_gpio_mux_mode_get()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

Retrieves the mode of a pin. See bp_stm32f4_gpio_mode_t for a list of possible modes.

Prototype

```
int  bp_stm32f4_gpio_mux_mode_get  ( uint32_t                  bank,
                                     uint32_t                  pin,
                                     bp_stm32f4_gpio_mode_t *  p_mode );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

bank	Bank number of the pin to set.
pin	Pin number of the pin to set.
p_mode	Pointer to the returned mode.

Returned Errors
RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_gpio_mux_mode_set()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

Sets the mode of a pin. See bp_stm32f4_gpio_mode_t for a list of available modes.

Prototype

```
int  bp_stm32f4_gpio_mux_mode_set  ( uint32_t               bank,
                                     uint32_t               pin,
                                     bp_stm32f4_gpio_mode_t mode );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

bank	Bank number of the pin to set.
pin	Pin number of the pin to set.
mode	Mode to set.

Returned Errors
RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_gpio_mux_pupd_get()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

Retrieves the pull-up or pull-down configuration of a pin. See bp_stm32f4_gpio_pupd_t for a list of possible types.

Prototype

```
int  bp_stm32f4_gpio_mux_pupd_get  ( uint32_t                bank,
                                     uint32_t                pin,
                                     bp_stm32f4_gpio_pupd_t* p_pupd );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

bank	Bank number of the pin to set.
pin	Pin number of the pin to set.
p_pupd	Pointer to the returned configuration.

Returned Errors RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_gpio_mux_pupd_set()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

Enables/disables the pull-up or pull-down of a pin. See bp_stm32f4_gpio_pupd_t for a list of available types.

Prototype

```
int  bp_stm32f4_gpio_mux_pupd_set  ( uint32_t               bank,
                                     uint32_t               pin,
                                     bp_stm32f4_gpio_pupd_t pupd );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

bank	Bank number of the pin to set.
pin	Pin number of the pin to set.
pupd	Configuration to set.

Returned Errors RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_gpio_mux_speed_get()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

Retrieves the output speed of a pin. See bp_stm32f4_gpio_speed_t for a list of possible types.

Prototype

```
int  bp_stm32f4_gpio_mux_speed_get  ( uint32_t                bank,
                                      uint32_t                pin,
                                      bp_stm32f4_gpio_speed_t * p_speed );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

bank	Bank number of the pin to set.
pin	Pin number of the pin to set.
p_speed	Pointer to the returned speed.

Returned Errors
RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_gpio_mux_speed_set()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

Sets the output speed of a pin. See bp_stm32f4_gpio_speed_t for a list of available speeds.

Prototype
```
int  bp_stm32f4_gpio_mux_speed_set  ( uint32_t  bank,
                                      uint32_t  pin,
                                                type );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

bank	Bank number of the pin to set.
pin	Pin number of the pin to set.
type	Type to set.

Returned Errors
RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_gpio_mux_type_get()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

Retrieves the output type of a pin. See bp_stm32f4_gpio_type_t for a list of possible types.

Prototype
```
int  bp_stm32f4_gpio_mux_type_get  ( uint32_t                  bank,
                                     uint32_t                  pin,
                                     bp_stm32f4_gpio_type_t *  p_type );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

bank	Bank number of the pin to set.
pin	Pin number of the pin to set.
p_type	Pointer to the returned type.

Returned Errors RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32f4_gpio_mux_type_set()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

Sets the output type of a pin. See bp_stm32f4_gpio_type_t for a list of available types.

Prototype

```
int  bp_stm32f4_gpio_mux_type_set  ( uint32_t                bank,
                                     uint32_t                pin,
                                     bp_stm32f4_gpio_type_t  type );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

bank	Bank number of the pin to set.
pin	Pin number of the pin to set.
type	Type to set.

Returned Errors RTNC_SUCCESS
RTNC_FATAL

Data Type

bp_stm32f4_gpio_mode_t

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

STM32F4 GPIO modes. See bp_stm32f4_gpio_mux_mode_set() and bp_stm32f4_gpio_mux_mode_get() for usage details.

Values

BP_STM32F4_GPIO_MODE_INPUT	GPIO input.
BP_STM32F4_GPIO_MODE_OUTPUT	GPIO output.
BP_STM32F4_GPIO_MODE_ALTF	Alternate function.
BP_STM32F4_GPIO_MODE_ANALOG	Analog function.

Data Type

bp_stm32f4_gpio_pupd_t

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

STM32F4 GPIO pull-up/pull-down. See `bp_stm32f4_gpio_mux_pupd_set()` and `bp_stm32f4_gpio_mux_pupd_get()` for usage details.

Values

`BP_STM32F4_GPIO_PUPD_NONE`	Pull-up/Pull-down disabled.
`BP_STM32F4_GPIO_PUPD_PU`	Pull-up enabled.
`BP_STM32F4_GPIO_PUPD_PD`	Pull-down enabled.

Data Type

bp_stm32f4_gpio_speed_t

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

STM32F4 GPIO speed. See `bp_stm32f4_gpio_mux_speed_set()` and `bp_stm32f4_gpio_mux_speed_get()` for usage details.

Values

`BP_STM32F4_GPIO_SPEED_LOW`	Low speed.
`BP_STM32F4_GPIO_SPEED_MED`	Medium speed.
`BP_STM32F4_GPIO_SPEED_HIGH`	High speed.
`BP_STM32F4_GPIO_SPEED_FAST`	Fast speed.

Data Type

bp_stm32f4_gpio_type_t

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_mux.h>

STM32F4 GPIO output types. See `bp_stm32f4_gpio_mux_type_set()` and `bp_stm32f4_gpio_mux_type_get()` for usage details.

Values

`BP_STM32F4_GPIO_TYPE_PP`	GPIO output type push-pull.
`BP_STM32F4_GPIO_TYPE_OD`	GPIO output type open drain.

Chapter

7 ARM v7-M System Control

ARM v7-M System Control Block. Note that the NVIC, SysTick and DWT functionalities are implemented in separate modules.

Function

bp_arch_v7m_afsr_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the auxiliary fault address.

Prototype

```
uint32_t  bp_arch_v7m_afsr_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

AFSR register content.

Function

bp_arch_v7m_bfar_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the BusFault fault address.

Prototype

```
uint32_t  bp_arch_v7m_bfar_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values BFAR register content.

Function

bp_arch_v7m_bfhfnmign_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables high priority handler precise fault ignore.

Prototype

```
void  bp_arch_v7m_bfhfnmign_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_bfhfnmign_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables high priority handler precise fault ignore.

Prototype

```
void  bp_arch_v7m_bfhfnmign_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_bfhfnmign_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the state of the BFHNMIGN bit.

Prototype

```
bool  bp_arch_v7m_bfhfnmign_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if high priority handlers ignore precise faults, `false` otherwise.

Function

bp_arch_v7m_bfsr_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the bus fault status.

Prototype

```
uint32_t  bp_arch_v7m_bfsr_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values BFSR register content.

Function

bp_arch_v7m_busfault_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables BusFault exceptions.

Prototype

```
void  bp_arch_v7m_busfault_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_busfault_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables BusFault exceptions.

Prototype

```
void  bp_arch_v7m_busfault_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_busfault_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns `true` if BusFault exceptions are enabled.

Prototype

```
bool  bp_arch_v7m_busfault_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if BusFault exceptions are enabled, `false` otherwise.

Function

bp_arch_v7m_cfsr_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the complete configurable fault status register value.

Prototype

```
uint32_t  bp_arch_v7m_cfsr_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

CFSR register content.

Function

bp_arch_v7m_cpacr_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the coprocessor access control register content.

Prototype

```
uint32_t  bp_arch_v7m_cpacr_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values CPACR register content.

Function

bp_arch_v7m_cpacr_set()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Sets the CPACR register.

Prototype

```
void bp_arch_v7m_cpacr_set ( uint32_t cpacr );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `cpacr` CPACR value to set.

Function

bp_arch_v7m_cpuid_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns CPUID register value.

Prototype

```
uint32_t bp_arch_v7m_cpuid_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values CPUID register value.

Function

bp_arch_v7m_cpuid_impl_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the implementer portion of the CPUID register value.

Prototype

```
uint32_t bp_arch_v7m_cpuid_impl_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values CPUID implementer value.

Function

bp_arch_v7m_cpuid_pn_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the part number portion of the CPUID register value.

Prototype

```
uint32_t  bp_arch_v7m_cpuid_pn_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values CPUID part number value.

Function

bp_arch_v7m_cpuid_rev_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the revision portion of the CPUID register value.

Prototype

```
uint32_t  bp_arch_v7m_cpuid_rev_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values CPUID revision value.

Function

bp_arch_v7m_cpuid_var_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the variant portion of the CPUID register value.

Prototype

```
uint32_t  bp_arch_v7m_cpuid_var_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

CPUID variant value.

Function

bp_arch_v7m_div0_trp_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables trap on divide by zero.

Prototype

```
void  bp_arch_v7m_div0_trp_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_div0_trp_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables trap on divide by zero.

Prototype

```
void  bp_arch_v7m_div0_trp_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_div0_trp_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns if divisions by zero are trapped.

Prototype

```
bool bp_arch_v7m_div0_trp_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if divisions by zero are trapped, `false` otherwise.

Function

bp_arch_v7m_endian_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the endianness of the CPU.

Prototype

```
bool bp_arch_v7m_endian_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the CPU is big-endian, false otherwise.

Function

bp_arch_v7m_hfsr_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the hard fault status.

Prototype

```
uint32_t bp_arch_v7m_hfsr_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values HFSR register content.

Function

bp_arch_v7m_icsr_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the value of the Interrupt Control and State Register (ICSR).

Prototype

```
uint32_t  bp_arch_v7m_icsr_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values ICSR value.

Function

bp_arch_v7m_ictr_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the maximum number of supported interrupt lines.

Prototype

```
uint32_t  bp_arch_v7m_ictr_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values Maximum number of supported interrupt lines.

Function

bp_arch_v7m_memfault_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables MemManage exceptions.

Prototype

```
void  bp_arch_v7m_memfault_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_memfault_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables MemManage exceptions.

Prototype

```
void  bp_arch_v7m_memfault_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_memfault_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns `true` if MemManage exceptions are enabled.

Prototype

```
bool  bp_arch_v7m_memfault_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if MemManage exceptions are enabled, `false` otherwise.

Function

bp_arch_v7m_mmfar_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the MemManage fault address.

Prototype

```
uint32_t  bp_arch_v7m_mmfar_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values MMFAR register content.

Function

bp_arch_v7m_mmfsr_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the memory management fault status.

Prototype

```
uint32_t bp_arch_v7m_mmfsr_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values MMFSR register content.

Function

bp_arch_v7m_nonbasethrd_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables switching to thread mode from an active exception.

Prototype

```
void bp_arch_v7m_nonbasethrd_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_nonbasethrd_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables switching to thread mode from an active exception.

Prototype

```
void bp_arch_v7m_nonbasethrd_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_nonbasethrd_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns true if switching to thread mode from an active exception is enabled,

Prototype

```
bool  bp_arch_v7m_nonbasethrd_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if switching to thread mode from an exception is permitted, `false` otherwise.

Function

bp_arch_v7m_prigroup_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the currently configured priority grouping binary set point.

Prototype

```
uint32_t  bp_arch_v7m_prigroup_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

Priority grouping binary set point.

Function

bp_arch_v7m_prigroup_set()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Sets the priority group binary set point.

Prototype

```
int  bp_arch_v7m_prigroup_set  (  p_vtor );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`p_vtor` Binary point to set.

Returned Errors `RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_arch_v7m_sevonpend_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables send event on interrupt pending.

Prototype `void bp_arch_v7m_sevonpend_dis ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_sevonpend_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables send event on interrupt pending.

Prototype `void bp_arch_v7m_sevonpend_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_sevonpend_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the enabled/disabled state of the send event on interrupt pending bit.

Prototype `bool bp_arch_v7m_sevonpend_is_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if SEVONPEND is set, `false` otherwise.

Function

bp_arch_v7m_sh_prio_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the configured priority of a system handler. Returns the priority of system handler `id` through `p_prio`.

Prototype

```
int  bp_arch_v7m_sh_prio_get  ( uint32_t   id,
                                uint32_t *  p_prio );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`id`	System handler id.
`p_prio`	Pointer to the returned priority.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_arch_v7m_sh_prio_set()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Sets the priority of a system handler. Sets the priority of system handler `id` to priority `prio`.

Prototype

```
int  bp_arch_v7m_sh_prio_set  ( uint32_t  id,
                                uint32_t  prio );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`id`	System handler id.
`prio`	Priority to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_arch_v7m_sh_status_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the system handlers status.

Prototype `uint32_t bp_arch_v7m_sh_status_get ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values System handlers status.

Function

bp_arch_v7m_sleepdeep_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables the deep sleep hint. Note that this is a hint and the exact effect of the SLEEPDEEP bit is implementation dependent.

Prototype `void bp_arch_v7m_sleepdeep_dis ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_sleepdeep_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables the deep sleep hint. Note that this is a hint and the exact effect of the SLEEPDEEP bit is implementation dependent.

Prototype `void bp_arch_v7m_sleepdeep_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_sleepdeep_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the enabled/disabled state of the deep sleep hint bit.

Prototype `bool bp_arch_v7m_sleepdeep_is_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if SLEEPDEEP is set, `false` otherwise.

Function

bp_arch_v7m_sleeponexit_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables sleep on interrupt exit.

Prototype `void bp_arch_v7m_sleeponexit_dis ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_sleeponexit_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables sleep on interrupt exit.

Prototype `void bp_arch_v7m_sleeponexit_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_sleeponexit_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the enabled/disabled of the sleep on interrupt exit bit.

Prototype `bool bp_arch_v7m_sleeponexit_is_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if SLEEPONEXIT is set, `false` otherwise.

Function

bp_arch_v7m_stkalign_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the stack alignment on exception entry bit.

Prototype

```
bool  bp_arch_v7m_stkalign_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if stack is aligned tp 8 bytes on exception entry, `false` if stack is aligned on 4 bytes.

Function

bp_arch_v7m_sysreset_req()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Sets the system reset request bit.

Prototype

```
void  bp_arch_v7m_sysreset_req  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_ufsr_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the usage fault status.

Prototype

```
uint32_t  bp_arch_v7m_ufsr_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values UFSR register content.

Function

bp_arch_v7m_unalign_trp_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables trap on unaligned access.

Prototype

```
void  bp_arch_v7m_unalign_trp_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_unalign_trp_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables trap on unaligned access.

Prototype

```
void  bp_arch_v7m_unalign_trp_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_unalign_trp_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns if unaligned accesses are trapped.

Prototype

```
bool  bp_arch_v7m_unalign_trp_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if unaligned accesses are trapped, `false` otherwise.

Function

bp_arch_v7m_usagefault_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables UsageFault exceptions.

Prototype

```
void bp_arch_v7m_usagefault_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_usagefault_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables UsageFault exceptions.

Prototype

```
void bp_arch_v7m_usagefault_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_usagefault_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns `true` if UsageFault exceptions are enabled.

Prototype

```
bool bp_arch_v7m_usagefault_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if UsageFault exceptions are enabled, `false` otherwise.

Function

bp_arch_v7m_user_stir_dis()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Disables software interrupt trigger from non-privileged mode.

Prototype

```
void bp_arch_v7m_user_stir_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_user_stir_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Enables software interrupt trigger from non-privileged mode.

Prototype

```
void bp_arch_v7m_user_stir_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_v7m_user_stir_is_en()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns true if software interrupt trigger from non-privileged mode is enabled.

Prototype

```
bool bp_arch_v7m_user_stir_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if STIR access is permitted from a non-privileged mode, `false` otherwise.

Function

bp_arch_v7m_vectactive_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the currently active vector id.

Prototype

```
uint32_t  bp_arch_v7m_vectactive_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

Active vector id.

Function

bp_arch_v7m_vectpending_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the currently pending vector id.

Prototype

```
uint32_t  bp_arch_v7m_vectpending_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

Pending vector id.

Function

bp_arch_v7m_vtor_get()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Returns the currently configured vector table offset.

Prototype

```
uint32_t  bp_arch_v7m_vtor_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values Vector table offset.

Function

bp_arch_v7m_vtor_set()

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Sets the vector table offset.

Prototype

```
void bp_arch_v7m_vtor_set ( void * p_vtor );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

p_vtor	Pointer to the vector table.

Macro

BP_V7M_SH_STAT_*

<arch/port/arm_v7m/bp_arm_v7m_scb.h>

Description System handler state flags.

BP_V7M_SH_STAT_SVCALL	SVCall pending.
BP_V7M_SH_STAT_BUSFAULT	BusFault pending.
BP_V7M_SH_STAT_MEMFAULT	MemManage pending.
BP_V7M_SH_STAT_USAGEFAULT	UsageFault pending.
BP_V7M_SH_STAT_SYSTICKACT	SYSTICK active.
BP_V7M_SH_STAT_PENDSVACT	PENDSV active.
BP_V7M_SH_STAT_MONITORACT	Monitor active.
BP_V7M_SH_STAT_SVCALLACT	SVCall active.
BP_V7M_SH_STAT_USAGEFAULTACT	UsageFault active.
BP_V7M_SH_STAT_BUSFAULTACT	BusFault active.
BP_V7M_SH_STAT_MEMFAULTACT	MemManage active.

Chapter

8 ARM v7-M SysTick

Interface module to the ARMv7M SYStick countdown timer. This modules API is designed to directly interact with the SYStick timer. Care should be taken not to adversely affect the SysTick timer when it is the primary time source or used as a system or kernel timer.

Function

bp_v7m_systick_calib_get()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Returns the current TENMS calibration value.

Prototype

```
uint32_t  bp_v7m_systick_calib_get  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

SysTick calibration value.

Function

bp_v7m_systick_calib_set()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Sets the SysTick TENMS calibration value.

Prototype

```
int  bp_v7m_systick_calib_set  ( uint32_t  calib );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `calib` Calibration value to set.

Returned Errors `RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_v7m_systick_clk_src_get()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Returns the currently configured clock source.

Prototype

```
uint32_t bp_v7m_systick_clk_src_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values SysTick clock source.

Function

bp_v7m_systick_clk_src_set()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Sets the SysTick clock source. Set to 1 to use the processor clock or 0 to use the external clock source.

Prototype

```
int bp_v7m_systick_clk_src_set ( uint32_t clk_src );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters `clk_src` Clock source to set, should be 1 or 0.

Returned Errors `RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_v7m_systick_count_flag_get()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Returns `true` if the counter overflowed since it was last read.

Prototype

```
bool bp_v7m_systick_count_flag_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if the counter flag is set `false` otherwise.

Function

bp_v7m_systick_counter_clr()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Clears the SysTick counter.

Prototype

```
void bp_v7m_systick_counter_clr ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_v7m_systick_counter_get()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Returns the current counter value.

Prototype

```
uint32_t bp_v7m_systick_counter_get ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

SysTick counter value.

Function

bp_v7m_systick_dis()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Disables the SysTick counter.

Prototype `void bp_v7m_systick_dis ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_v7m_systick_en()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Enables the SysTick counter.

Prototype `void bp_v7m_systick_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_v7m_systick_int_dis()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Disables the SysTick interrupt.

Prototype `void bp_v7m_systick_int_dis ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_v7m_systick_int_en()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Enables the SysTick interrupt.

Prototype `void bp_v7m_systick_int_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_v7m_systick_int_is_en()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Returns `true` if the SysTick interrupt is enabled.

Prototype `bool bp_v7m_systick_int_is_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if the SysTick interrupt is enabled `false` otherwise.

Function

bp_v7m_systick_is_en()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Returns `true` if the SysTick counter is enabled.

Prototype `bool bp_v7m_systick_is_en ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values `true` if the SysTick counter is enabled `false` otherwise.

Function

bp_v7m_systick_reload_get()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Returns the currently configured counter reload value.

Prototype `uint32_t bp_v7m_systick_reload_get ( );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values SysTick counter reload value.

Function

bp_v7m_systick_reload_set()

<soc_comp/arm/arm_v7m_systick/bp_arm_v7m_systick.h>

Sets the SysTick counter reload value.

Prototype `int bp_v7m_systick_reload_set ( uint32_t reload );`

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`reload`	Counter reload value.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Chapter

9 ARM Cortex-M4 Control

ARM Cortex-M4 specific control functions.

Function

bp_arch_cm4_fold_dis()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Disables IT folding.

Prototype

```
void bp_arch_cm4_fold_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_cm4_fold_en()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Enables IT folding.

Prototype

```
void bp_arch_cm4_fold_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_cm4_fold_is_en()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Returns `true` IT folding is enabled.

Prototype

```
bool bp_arch_cm4_fold_is_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if IT folding is enabled, `false` otherwise.

Function

bp_arch_cm4_fpca_dis()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Disables automatic update of the FPCA register.

Prototype

```
void bp_arch_cm4_fpca_dis ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_cm4_fpca_en()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Enables automatic update of the FPCA register.

Prototype

```
void bp_arch_cm4_fpca_en ( );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_cm4_fpca_is_en()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Returns `true` if automatic update of the FPCA register is enabled.

Prototype

```
bool  bp_arch_cm4_fpca_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if automatic update is enabled, `false` otherwise.

Function

bp_arch_cm4_mcycint_dis()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Disables interruption of multi-cycle instructions.

Prototype

```
void  bp_arch_cm4_mcycint_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_cm4_mcycint_en()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Enables interruption of multi-cycle instructions.

Prototype

```
void  bp_arch_cm4_mcycint_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_cm4_mcycint_is_en()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Returns `true` interruption of multi-cycle instructions is enabled.

Prototype

```
bool  bp_arch_cm4_mcycint_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if interruption of multi-cycle instructions is enabled, `false` otherwise.

Function

bp_arch_cm4_oofp_dis()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Disables out of order completion of floating point instructions.

Prototype

```
void  bp_arch_cm4_oofp_dis  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_cm4_oofp_en()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Enables out of order completion of floating point instructions.

Prototype

```
void  bp_arch_cm4_oofp_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Function

bp_arch_cm4_oofp_is_en()

<arch/port/arm_v7m/cortex_m4/bp_cortex_m4_ctrl.h>

Returns `true` if out of order completion of floating point instructions is enabled.

Prototype

```
bool  bp_arch_cm4_oofp_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Values

`true` if out of order completion is enabled, `false` otherwise.

Chapter

10

STM32F4 GPIO Driver

GPIO driver for the STMicroelectronics STM32 F4 series. Note that for most applications it is recommended to use the GPIO module API instead of the driver interface. This module contains the GPIO driver interface to be used by the GPIO module as well as additional driver specific functions. The driver interface as well as the driver specific functions can be called by the application using the driver handle which can be retrieved using the `bp_gpio_drv_hndl_get()` function.

Note that since the GPIO pin direction cannot be set independently from the alternate or analog mode changing the mode of a pin will override any direction set using the GPIO module and driver. The opposite is also true, setting the direction of a GPIO pin from the GPIO module will override the current mode unconditionally.

Pin and bank numbering follow the manufacturer's documentation with GPIO bank letter name converted to a bank number. For example, bank A is 0, bank B is 1 and bank K is 10.

See the BASEplatform manual for additional information on calling the driver interface directly.

Function

bp_stm32f4_gpio_create()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

Creates a GPIO driver instance.

See `bp_gpio_drv_create_t` for usage details.

Prototype

```
int  bp_stm32f4_gpio_create  ( const bp_gpio_board_def_t *  p_def,
                               bp_gpio_drv_hndl_t *        p_hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✗	✗	✓

Parameters

`p_def`	Board definition of the GPIO peripheral to initialize.
`p_hndl`	Handle to the created GPIO driver instance.

Returned Errors

`RTNC_SUCCESS`
`RTNC_NO_RESOURCE`
`RTNC_FATAL`

Function

bp_stm32f4_gpio_data_get()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

Gets the state of a GPIO pin.

See `bp_gpio_drv_data_get_t` for usage details.

Prototype

```
int  bp_stm32f4_gpio_data_get  ( bp_gpio_drv_hndl_t  hndl,
                                 uint32_t            bank,
                                 uint32_t            pin,
                                 uint32_t *          p_data );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the driver to query.
`bank`	Bank number of the pin to query.
`pin`	Pin number of the pin to query.
`p_data`	Pointer to the variable that will receive the data.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_gpio_data_set()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

Sets the state of a GPIO pin.

See `bp_gpio_drv_data_set_t` for usage details.

Prototype

```
int  bp_stm32f4_gpio_data_set  ( bp_gpio_drv_hndl_t  hndl,
                                 uint32_t            bank,
                                 uint32_t            pin,
                                 data );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the driver to set.
`bank`	Bank number of the pin to set.
`pin`	Pin number of the pin to set.
`data`	State of the pin to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_gpio_data_tog()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

Toggle the state of a GPIO pin.

See `bp_gpio_drv_data_tog_t` for usage details.

Prototype

```
int  bp_stm32f4_gpio_data_tog  ( bp_gpio_drv_hndl_t  hndl,
                                 uint32_t            bank,
                                 uint32_t            pin );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the driver to toggle.
`bank`	Bank number of the pin to toggle.
`pin`	Pin number of the pin to toggle.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_gpio_destroy()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

Destroys a GPIO driver instance.

See `bp_gpio_drv_destroy_t` for usage details.

Prototype

```
int  bp_stm32f4_gpio_destroy  ( bp_gpio_drv_hndl_t  hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the GPIO driver instance to destroy.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_gpio_dir_get()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

Gets the direction of a GPIO pin.

See `bp_gpio_drv_dir_get_t` for usage details.

Prototype

```
int  bp_stm32f4_gpio_dir_get  ( bp_gpio_drv_hndl_t  hndl,
                                uint32_t            bank,
                                uint32_t            pin,
                                bp_gpio_dir_t *     p_dir );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the driver to query.
`bank`	Bank number of the pin to query.
`pin`	Pin number of the pin to query.
`p_dir`	Pointer to the variable that will receive the direction.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_gpio_dir_set()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

Sets the direction of a GPIO pin.

See `bp_gpio_drv_dir_set_t` for usage details.

Prototype

```
int  bp_stm32f4_gpio_dir_set  ( bp_gpio_drv_hndl_t  hndl,
                                uint32_t            bank,
                                uint32_t            pin,
                                                    dir );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the interface to set.
`bank`	Bank number of the pin to set.
`pin`	Pin number of the pin to set.
`dir`	Direction of the pin to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_gpio_dis()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

Disables a GPIO interface.

See `bp_gpio_drv_dis_t` for usage details.

Prototype

```
int  bp_stm32f4_gpio_dis  ( bp_gpio_drv_hndl_t  hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the GPIO driver to disable.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_gpio_en()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

Enables a GPIO interface.

See `bp_gpio_drv_en_t` for usage details.

Prototype

```
int  bp_stm32f4_gpio_en  ( bp_gpio_drv_hndl_t  hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the GPIO driver to enable.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32f4_gpio_is_en()

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

Returns the enabled/disabled state of a GPIO interface.

See `bp_gpio_drv_is_en_t` for usage details.

Prototype

```
int  bp_stm32f4_gpio_is_en  ( bp_gpio_drv_hndl_t  hndl,
                              bool *              p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the GPIO driver to query.
`p_is_en`	Interface state, true if enabled false otherwise.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_stm32f4_gpio_drv_def_t

<soc_comp/stmicro/stm32f4_gpio/bp_stm32f4_gpio_drv.h>

STM32 F4 GPIO driver hardware definition structure. Those parameters are required by the GPIO driver and are configured through a `bp_gpio_soc_def_t` structure.

Members

`base_addr`	`void *`	Peripheral base address.

Chapter

11

STM32 UART Driver

UART driver for the STMicroelectronics STM32 microcontrollers. Note that for most applications it is recommended to use the UART module API instead of the driver interface. This module contains the UART driver interface to be used by the UART module as well as additional driver specific functions. The driver interface as well as the driver specific functions can be called by the application using the driver handle which can be retrieved using the `bp_uart_drv_hndl_get()` function.

See the BASEplatform manual for additional information on calling the driver interface directly.

Function

`bp_stm32_uart_cfg_get()`

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Retrieves the current configuration of a UART interface.

See `bp_uart_drv_cfg_get_t` for usage details.

Prototype

```
int  bp_stm32_uart_cfg_get  ( bp_uart_drv_hndl_t  hndl,
                              bp_uart_cfg_t *     p_cfg,
                                                  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✗

Parameters

`hndl`	Handle of the UART peripheral to query.
`p_cfg`	Pointer to the UART configuration.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32_uart_cfg_set()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Configures a UART peripheral.

See bp_uart_drv_cfg_set_t for usage details.

Prototype

```
int  bp_stm32_uart_cfg_set  (                       hndl,
                              const bp_uart_cfg_t *  p_cfg,
                              uint32_t               timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the UART peripheral to configure.
p_cfg	UART configuration.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_NOT_SUPPORTED
RTNC_FATAL

Function

bp_stm32_uart_create()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Creates a UART driver instance.

See bp_uart_drv_create_t for usage details.

Prototype

```
int  bp_stm32_uart_create  ( const bp_uart_board_def_t *  p_def,
                               bp_uart_drv_hndl_t *       p_hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✗	✗	✗

Parameters

`p_def`	Board definition of the UART peripheral to initialize.
`p_hndl`	Pointer to the newly created UART driver instance.

Returned Errors

`RTNC_SUCCESS`
`RTNC_NO_RESOURCE`
`RTNC_FATAL`

Function

bp_stm32_uart_destroy()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Destroys a UART driver instance.

See `bp_uart_drv_destroy_t` for usage details.

Prototype

```
int  bp_stm32_uart_destroy  ( bp_uart_drv_hndl_t  hndl,
                              uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the UART driver instance to destroy.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_uart_dis()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Disables a UART peripheral.

See `bp_uart_drv_dis_t` for usage details.

Prototype

```
int  bp_stm32_uart_dis  ( bp_uart_drv_hndl_t  hndl,
                          uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the UART peripheral to disable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_uart_en()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Enables a UART peripheral.

See `bp_uart_drv_en_t` for usage details.

Prototype

```
int  bp_stm32_uart_en  ( bp_uart_drv_hndl_t  hndl,
                         uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the UART peripheral to enable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_uart_is_en()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Returns the enabled/disabled state of a UART peripheral.

See `bp_uart_drv_is_en_t` for usage details.

Prototype

```
int  bp_stm32_uart_is_en  ( bp_uart_drv_hndl_t  hndl,
                            bool *              p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the UART peripheral to check.
`p_is_en`	Interface state, true if enabled false otherwise.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32_uart_reset()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Resets a UART peripheral.

See `bp_uart_drv_reset_t` for usage details.

Prototype

```
int  bp_stm32_uart_reset  ( bp_uart_drv_hndl_t  hndl,
                            uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✗

Parameters

`hndl`	Handle of the UART peripheral to reset.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_uart_rx()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Receives data.

See `bp_uart_drv_rx_t` for usage details.

Prototype

```
int  bp_stm32_uart_rx  (          hndl,
                          void *  p_buf,
                          size_t  len,
                          size_t *  p_rx_len,
                          uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the interface to use for reception.
`p_buf`	Pointer to the buffer that will receive the data.
`len`	Length of the data to receive in bytes.
`p_rx_len`	Return pointer of the actual number of bytes read, can be NULL.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_IO_ERR`
`RTNC_FATAL`

Function

bp_stm32_uart_rx_async()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Receive data asynchronously.

See `bp_uart_drv_rx_async_t` for usage details.

Prototype

```
int  bp_stm32_uart_rx_async  (                bp_uart_tf_t *  p_tf,
                              bp_uart_tf_t *  p_tf,
                              uint32_t        timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the interface to use for reception.
`p_tf`	Transfer parameters.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_uart_rx_async_abort()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Aborts an asynchronous transfer.

See `bp_uart_drv_rx_async_abort_t` for usage details.

Prototype

```
int  bp_stm32_uart_rx_async_abort  (          hndl,
                                    size_t *  p_rx_len,
                                    uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✗	✗	✗

Parameters

`hndl`	Handle of the interface to abort.
`p_rx_len`	Pointer to the number of bytes received, can be NULL.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_uart_rx_flush()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Flush the transmit path.

See `bp_uart_drv_rx_flush_t` for usage details.

Prototype

```
int  bp_stm32_uart_rx_flush  (          hndl,
                               uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the interface to flush.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_uart_rx_idle_wait()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Waits for a UART interface to be idle.

See bp_uart_drv_rx_idle_wait_t for usage details.

Prototype

```
int  bp_stm32_uart_rx_idle_wait  (          hndl,
                                   uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the interface to wait.
timeout_ms	Timeout in milliseconds.

Returned Errors

RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Function

bp_stm32_uart_tx()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Transmits data.

See bp_uart_drv_tx_t for usage details.

Prototype

```
int  bp_stm32_uart_tx  (             hndl,
                         const void *  p_buf,
                         size_t        len,
                         uint32_t      timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the interface to use for transmission.
p_buf	Pointer to the buffer to transmit.
len	Length of the data to transmit in bytes.
timeout_ms	Timeout value in milliseconds.

Returned Errors

RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_IO_ERR
RTNC_FATAL

Function

bp_stm32_uart_tx_async()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Transmits data asynchronously.

See bp_uart_drv_tx_async_t for usage details.

Prototype

```
int  bp_stm32_uart_tx_async  (                hndl,
                              bp_uart_tf_t *  p_tf,
                              uint32_t        timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the interface to use for reception.
p_tf	Transfer parameters.
timeout_ms	Timeout value in milliseconds.

Returned Errors

RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Function

bp_stm32_uart_tx_async_abort()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Aborts an asynchronous transfer.

See bp_uart_drv_tx_async_abort_t for usage details.

Prototype

```
int  bp_stm32_uart_tx_async_abort  (           hndl,
                                    size_t *   p_tx_len,
                                    uint32_t   timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✗	✗	✗

Parameters

hndl	Handle of the interface to abort.
p_tx_len	Pointer to the number of bytes transmitted, can be NULL.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Function

bp_stm32_uart_tx_flush()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Flush the receive path.

See bp_uart_drv_tx_flush_t for usage details.

Prototype
```
int  bp_stm32_uart_tx_flush  (          hndl,
                               uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the interface to flush.
timeout_ms	Timeout in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Function

bp_stm32_uart_tx_idle_wait()

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

Waits for a UART interface to be idle.

See bp_uart_drv_tx_idle_wait_t for usage details.

Prototype
```
int  bp_stm32_uart_tx_idle_wait  (          hndl,
                                   uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the interface to wait.
timeout_ms	Timeout in milliseconds.

Returned Errors
`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_stm32_uart_drv_def_t

<soc_comp/stmicro/stm32_uart/bp_stm32_uart_drv.h>

STM32 UART driver hardware definition structure. Those parameters are required by the UART driver and are configured through a `bp_uart_soc_def_t` structure.

Members

`base_addr`	`void *`	Peripheral base address.
`int_id`	`uint32_t`	Peripheral interrupt id.
`clk_id`	`int`	Peripheral clock id.
`clk_gate_id`	`int`	Peripheral clock gate id.
`reset_id`	`int`	Peripheral reset id.

Chapter

12

STM32 I2C Driver

I2C driver for the STMicroelectronics I2C peripheral. Note that for most applications it is recommended to use the I2C module API instead of the driver interface. This module contains the I2C driver interface to be used by the I2C module as well as additional driver specific functions. The driver interface as well as the driver specific functions can be called by the application using the driver handle which can be retrieved using the `bp_i2c_drv_hndl_get()` function.

See the BASEplatform manual for additional information on calling the driver interface directly.

Function

bp_stm32_i2c_cfg_get()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Retrieves the current configuration of an I2C interface.

See `bp_i2c_drv_cfg_get_t` for usage details.

Prototype

```
int  bp_stm32_i2c_cfg_get  ( bp_i2c_drv_hndl_t  hndl,
                             bp_i2c_cfg_t *     p_cfg,
                             uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the I2C driver to query.
`p_cfg`	Pointer to the I2C configuration.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Function

bp_stm32_i2c_cfg_set()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Configures an I2C interface.

See bp_i2c_drv_cfg_set_t for usage details.

Prototype

```
int  bp_stm32_i2c_cfg_set  ( bp_i2c_drv_hndl_t    hndl,
                             const bp_i2c_cfg_t *  p_cfg,
                             uint32_t              timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the I2C driver to configure.
p_cfg	I2C configuration.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_NOT_SUPPORTED
RTNC_FATAL

Function

bp_stm32_i2c_create()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Creates an I2C driver instance.

See bp_i2c_drv_create_t for usage details.

Prototype

```
int  bp_stm32_i2c_create  ( const bp_i2c_board_def_t *  p_def,
                            bp_i2c_drv_hndl_t *        p_hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✗	✗	✓

Parameters

`p_def`	Board definition of the I2C peripheral to create.
`p_hndl`	Pointer to the newly created I2C driver interface.

Returned Errors

`RTNC_SUCCESS`
`RTNC_NO_RESOURCE`
`RTNC_FATAL`

Function

bp_stm32_i2c_destroy()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Destroys an I2C interface.

See `bp_i2c_drv_destroy_t` for usage details.

Prototype

```
int  bp_stm32_i2c_destroy  ( bp_i2c_drv_hndl_t  hndl,
                             uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the I2C driver to destroy.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_i2c_dis()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Disables an I2C interface.

See `bp_i2c_drv_dis_t` for usage details.

Prototype

```
int  bp_stm32_i2c_dis  ( bp_i2c_drv_hndl_t  hndl,
                         uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the I2C interface to disable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_i2c_en()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Enables an I2C interface.

See `bp_i2c_drv_en_t` for usage details.

Prototype

```
int  bp_stm32_i2c_en  ( bp_i2c_drv_hndl_t  hndl,
                        uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the I2C driver to enable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_i2c_flush()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Flush the transmit and receive paths.

See `bp_i2c_drv_flush_t` for usage details.

Prototype

```
int  bp_stm32_i2c_flush  ( bp_i2c_drv_hndl_t  hndl,
                           uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the driver to flush.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_i2c_idle_wait()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Waits for an I2C interface to be idle.

See `bp_i2c_drv_idle_wait_t` for usage details.

Prototype

```
int  bp_stm32_i2c_idle_wait  ( bp_i2c_drv_hndl_t  hndl,
                               uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the driver to wait on.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_i2c_is_en()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Returns the enabled/disabled state of an I2C interface.

See `bp_i2c_drv_is_en_t` for usage details.

Prototype

```
int  bp_stm32_i2c_is_en  ( bp_i2c_drv_hndl_t  hndl,
                           bool *             p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

`hndl`	Handle of the I2C driver to query.
`p_is_en`	Interface state, true if enabled false otherwise.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Function

bp_stm32_i2c_reset()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Resets an I2C interface.

See `bp_i2c_drv_reset_t` for usage details.

Prototype

```
int  bp_stm32_i2c_reset  ( bp_i2c_drv_hndl_t  hndl,
                           uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the I2C driver to reset.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_i2c_xfer()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Performs an I2C operation.

See `bp_i2c_drv_xfer_t` for usage details.

Prototype

```
int  bp_stm32_i2c_xfer  ( bp_i2c_drv_hndl_t  hndl,
                          bp_i2c_tf_t *      p_tf,
                                             p_recv_len,
                          uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the driver to use.
`p_tf`	Pointer to an `bp_i2c_tf_t` structure describing the transfer to perform.
`p_recv_len`	Amount of data actually received.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_IO_ERR`
`RTNC_FATAL`

Function

bp_stm32_i2c_xfer_async()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Transfers data asynchronously.

See `bp_i2c_drv_xfer_async_t` for usage details.

Prototype

```
int  bp_stm32_i2c_xfer_async  ( bp_i2c_drv_hndl_t  hndl,
                                bp_i2c_tf_t *      p_tf,
                                uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the driver to use for transferring.
`p_tf`	Transfer parameters.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_i2c_xfer_async_abort()

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Aborts an asynchronous transfer.

See `bp_i2c_drv_xfer_async_abort_t` for usage details.

Prototype

```
int  bp_stm32_i2c_xfer_async_abort  ( bp_i2c_drv_hndl_t  hndl,
                                      size_t *          p_tf_len,
                                      uint32_t          timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the driver to abort.
`p_tf_len`	Amount of data transferred.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_stm32_i2c_drv_def_t

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

STM32 I2C driver hardware definition structure. Those parameters are required by the I2C driver and are configured through a `bp_i2c_soc_def_t` structure.

Members

`base_addr`	`void *`	Peripheral base address.
`int_id`	`uint32_t`	Peripheral interrupt id.
`clk_id`	`int`	Peripheral clock id.
`clk_gate_id`	`int`	Peripheral clock gate id.
`reset_id`	`int`	Peripheral reset id.

Data Type

bp_stm32_i2c_param_t

<soc_comp/stmicro/stm32_i2c/bp_stm32_i2c_drv.h>

Chapter

13

STM32 SPI Driver

SPI driver for the STMicroelectronics STM32 SPI peripheral. Note that for most applications it is recommended to use the SPI module API instead of the driver interface. This module contains the SPI driver interface to be used by the SPI module as well as additional driver specific functions. The driver interface as well as the driver specific functions can be called by the application using the driver handle which can be retrieved using the `bp_spi_drv_hndl_get()` function.

See the BASEplatform manual for additional information on calling the driver interface directly.

Function

`bp_stm32_spi_cfg_get()`

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Retrieves the current configuration of an SPI peripheral.

See `bp_spi_drv_cfg_get_t` for usage details.

The configuration get procedure for this driver is non-blocking, as such the timeout_ms argument is ignored.

Prototype

```
int  bp_stm32_spi_cfg_get  (                hndl,
                            bp_spi_cfg_t *  p_cfg,
                            uint32_t        timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✗	✗	✓

Parameters

`hndl`	Handle of the SPI driver to query.
`p_cfg`	Pointer to the SPI configuration.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32_spi_cfg_set()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Configures an SPI peripheral.

See bp_spi_drv_cfg_set_t for usage details.

Prototype

```
int  bp_stm32_spi_cfg_set  (                      hndl,
                            const bp_spi_cfg_t *  p_cfg,
                            uint32_t              timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the SPI driver to configure.
p_cfg	SPI configuration.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_NOT_SUPPORTED
RTNC_TIMEOUT
RTNC_FATAL

Function

bp_stm32_spi_create()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Creates an SPI driver instance.

See bp_spi_drv_create_t for usage details.

Prototype

```
int  bp_stm32_spi_create  ( const bp_spi_board_def_t *  p_def,
                            bp_spi_drv_hndl_t *         p_hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✗	✗	✗

Parameters

`p_def`	Board definition of the SPI driver to create.
`p_hndl`	Pointer to the newly created SPI interface.

Returned Errors

`RTNC_SUCCESS`
`RTNC_NO_RESOURCE`
`RTNC_FATAL`

Function

bp_stm32_spi_destroy()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Destroys an SPI driver instance.

See `bp_spi_drv_destroy_t` for usage details.

Prototype

```
int  bp_stm32_spi_destroy  (          hndl,
                            uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the SPI driver to destroy.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_spi_dis()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Disables an SPI peripheral.

See `bp_spi_drv_dis_t` for usage details.

Prototype

```
int  bp_stm32_spi_dis  (          hndl,
                        uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the SPI driver to disable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_spi_en()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Enables an SPI peripheral

See `bp_spi_drv_en_t` for usage details.

Prototype

```
int  bp_stm32_spi_en  (          hndl,
                        uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the SPI driver to enable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_spi_flush()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Flush the transmit and receive paths.

See `bp_spi_drv_flush_t` for usage details.

Prototype

```
int  bp_stm32_spi_flush  (          hndl,
                           uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the peripheral to flush.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_spi_idle_wait()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Wait for an SPI peripheral to be idle.

See `bp_spi_drv_idle_wait_t` for usage details.

Prototype

```
int  bp_stm32_spi_idle_wait  (          hndl,
                               uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the driver to wait on.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_spi_is_en()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Returns the enabled/disabled state of an SPI peripheral.

See `bp_spi_drv_is_en_t` for usage details.

Prototype

```
int  bp_stm32_spi_is_en  (  );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Returned Errors RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32_spi_last_err_get()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Returns the last error specific to the SPI peripheral.

See bp_spi_drv_last_err_get_t for usage details.

Prototype

```
int  bp_stm32_spi_last_err_get  (  hndl,
                                   p_err );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

hndl	Handle of the SPI peripheral to query.
p_err	Pointer to an int that will receive the last SPI error.

Returned Errors RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32_spi_reset()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Resets an SPI peripheral.

See bp_spi_drv_reset_t for usage details.

The reset procedure for this driver is non-blocking, as such the timeout_md argument is ignored.

Prototype

```
int  bp_stm32_spi_reset  (           hndl,
                           uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✗	✓	✓	✓

Parameters

hndl	Handle of the SPI driver to reset.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_FATAL

Function

bp_stm32_spi_slave_desel()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Deselect a selected SPI slave.

See bp_spi_drv_slave_desel_t for usage details.

Prototype

```
int  bp_stm32_spi_slave_desel  (          hndl,
                                uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the SPI driver.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Function

bp_stm32_spi_slave_sel()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Select a specific SPI slave.

See bp_spi_drv_slave_sel_t for usage details.

Prototype

```
int  bp_stm32_spi_slave_sel  (          hndl,
                               uint32_t  ss_id,
                               uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the SPI peripheral.
ss_id	Numeric id of the slave select line to assert.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Function

bp_stm32_spi_xfer()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Performs an SPI operation.

See bp_spi_drv_xfer_t for usage details.

Prototype

```
int  bp_stm32_spi_xfer  (               hndl,
                         bp_spi_tf_t *  p_tf,
                                        p_recv_len,
                         uint32_t       timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

hndl	Handle of the driver to use.
p_tf	Pointer to an bp_spi_tf_t structure describing the transfer to perform.
p_recv_len	Amount of data actually received.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_IO_ERR
RTNC_FATAL

Function

bp_stm32_spi_xfer_async()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Transfer data asynchronously.

See bp_spi_drv_xfer_async_t for usage details.

Prototype

```
int  bp_stm32_spi_xfer_async  (               hndl,
                               bp_spi_tf_t *  p_tf,
                               uint32_t       timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the driver to use for transferring.
`p_tf`	Transfer parameters.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Function

bp_stm32_spi_xfer_async_abort()

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

Aborts an asynchronous transfer.

See `bp_spi_drv_xfer_async_abort_t` for usage details.

Prototype

```
int  bp_stm32_spi_xfer_async_abort  (          hndl,
                                     size_t *  p_tx_len,
                                     size_t *  p_rx_len,
                                     uint32_t  timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✗

Parameters

`hndl`	Handle of the driver to abort.
`p_tx_len`	Pointer to the amount of data already transferred.
`p_rx_len`	Pointer to the amount of data already received.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_stm32_spi_drv_def_t

<soc_comp/stmicro/stm32_spi/bp_stm32_spi_drv.h>

STM32 SPI driver hardware definition structure. Those parameters are required by the SPI driver and are configured through a `bp_spi_soc_def_t` structure.

Members

`base_addr`	`void *`	Peripheral base address.
`int_id`	`uint32_t`	Peripheral interrupt id.
`clk_id`	`int`	Peripheral clock id.
`clk_gate_id`	`int`	Peripheral clock gate id.
`reset_id`	`int`	Peripheral reset id.

Chapter

14 Error Codes

Generic return code definitions. The descriptions below are a general guideline to the meaning of each return code. Consult the API documentation for a detailed list and description of errors that can be returned by each API.

Unexpected error codes returned by any functions, including error codes outside of the range of defined error codes should be treated as a fatal error.

Macro

RTNC_*

<util/rtnc.h>

Description Return codes.

RTNC_SUCCESS	Function completed successfully.
RTNC_FATAL	Fatal error occurred.
RTNC_NO_RESOURCE	Resource allocation failure.
RTNC_IO_ERR	Transfer or peripheral operation failed.
RTNC_TIMEOUT	Function timed out.
RTNC_NOT_SUPPORTED	API, feature or configuration is not supported.
RTNC_NOT_FOUND	Requested object not found.
RTNC_ALREADY_EXIST	Object already created or allocated.
RTNC_ABORT	Operation aborted by software.
RTNC_INVALID_OP	Invalid operation.
RTNC_WANT_READ	Read operation requested.
RTNC_WANT_WRITE	Write operation requested.

Chapter

15

GPIO Driver Reference

The GPIO driver declarations found in this module serves as the basis of GPIO drivers usually used in combination with the GPIO module to access GPIO peripherals. All GPIO drivers are composed of a standard set of API expected by the GPIO module in addition to any number of implementation-specific functions. The driver specific functions can be used by the application to access advanced features of a GPIO peripheral not exposed through the standard API. Note that usage of those extended functionalities is non-portable contrary to the standard API. The GPIO module API function `bp_gpio_drv_hndl_get()` function can be used to retrieve the driver handle associated with a GPIO module instance, and can subsequently be used to call the driver directly. See the individual driver's documentation for details of the extended functions.

In addition to accessing extended functionalities, an application can access the driver standard API directly bypassing the GPIO module. This reduces the call overhead. Contrary to most types of drivers, the GPIO drivers are usually thread-safe by design while other drivers usually require the top-level modules mutexes to be thread-safe.

Finally, as yet another feature of the GPIO driver API, it can be invoked in a standalone fashion without a GPIO module instance. This reduces the RAM overhead of using a GPIO peripheral. In this case the driver create function is called directly by the application in a matter similar to `bp_gpio_create()` to instantiate the driver.

Data Type

bp_gpio_drv_create_t

<gpio/bp_gpio_drv.h>

GPIO driver's create function.

Prototype

```
int  bp_gpio_drv_create_t  ( const bp_gpio_board_def_t *  p_def,
                             bp_gpio_drv_hndl_t *        p_hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`p_def`	Board definition of the GPIO peripheral to create.
`p_hndl`	Handle to the created GPIO driver instance.

Returned Errors

`RTNC_SUCCESS`
`RTNC_ALREADY_EXIST`
`RTNC_NO_RESOURCE`
`RTNC_FATAL`

Data Type

bp_gpio_drv_data_get_t

<gpio/bp_gpio_drv.h>

GPIO driver's data_get function. Returns the data state of pin number `pin` of bank `bank`.

Prototype

```
int  bp_gpio_drv_data_get_t  ( bp_gpio_drv_hndl_t  hndl,
                               uint32_t            bank,
                               uint32_t            pin,
                               uint32_t *          data );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to query.
`bank`	Bank number of the pin to query.
`pin`	Pin number of the pin to query.
`data`	Pointer to the variable that will receive the data.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_gpio_drv_data_set_t

<gpio/bp_gpio_drv.h>

GPIO driver's data_set function. Set the state of pin number `pin` of bank `bank` to the data specified by `data`.

Prototype

```
int  bp_gpio_drv_data_set_t  ( bp_gpio_drv_hndl_t  hndl,
                               uint32_t            bank,
                               uint32_t            pin,
                               uint32_t            data );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the interface to set.
`bank`	Bank number of the pin to set.
`pin`	Pin number of the pin to set.
`data`	State of the pin to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_gpio_drv_data_tog_t

<gpio/bp_gpio_drv.h>

Toggle the state of a GPIO pin. Toggle the data of pin number `pin` of bank `bank`.

Prototype

```
int  bp_gpio_drv_data_tog_t  ( bp_gpio_drv_hndl_t  hndl,
                               uint32_t            bank,
                               uint32_t            pin );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the interface to toggle.
`bank`	Bank number of the pin to toggle.
`pin`	Pin number of the pin to toggle.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_gpio_drv_destroy_t

<gpio/bp_gpio_drv.h>

GPIO driver's destroy function.

Prototype

```
int  bp_gpio_drv_destroy_t  ( bp_gpio_drv_hndl_t  hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the GPIO driver instance to destroy.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_gpio_drv_dir_get_t

<gpio/bp_gpio_drv.h>

GPIO driver'd dir_get function. Returns the direction of pin number `pin` of bank `bank`.

Prototype

```
int  bp_gpio_drv_dir_get_t  ( bp_gpio_drv_hndl_t  hndl,
                              uint32_t            bank,
                              uint32_t            pin,
                              bp_gpio_            dir );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to query.
`bank`	Bank number of the pin to query.
`pin`	Pin number of the pin to query.
`dir`	Pointer to the variable that will receive the direction.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_gpio_drv_dir_set_t

<gpio/bp_gpio_drv.h>

GPIO driver's dir_set function. Sets the direction of pin number `pin` of bank `bank` to the direction specified by `dir`.

Prototype

```
int  bp_gpio_drv_dir_set_t  ( bp_gpio_drv_hndl_t  hndl,
                              uint32_t            bank,
                              uint32_t            pin,
                              bp_gpio_dir_t       dir );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to set.
`bank`	Bank number of the pin to set.
`pin`	Pin number of the pin to set.
`dir`	Direction of the pin to set.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_gpio_drv_dis_t

<gpio/bp_gpio_drv.h>

GPIO driver's disable function.

Prototype

```
int  bp_gpio_drv_dis_t  ( bp_gpio_drv_hndl_t  hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the GPIO driver instance to disable.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_gpio_drv_en_t

<gpio/bp_gpio_drv.h>

GPIO driver's enable function.

Prototype

```
int  bp_gpio_drv_en_t  ( bp_gpio_drv_hndl_t  hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl` Handle of the GPIO driver instance to enable.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_gpio_drv_is_en_t

<gpio/bp_gpio_drv.h>

GPIO driver's is_en function.

Prototype

```
int  bp_gpio_drv_is_en_t  ( bp_gpio_drv_hndl_t  hndl,
                            bool *              p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl` Handle of the GPIO driver instance to query.
`p_is_en` Driver state, true if enabled false otherwise.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_gpio_drv_reset_t

<gpio/bp_gpio_drv.h>

GPIO driver's reset function.

Prototype

```
int  bp_gpio_drv_reset_t  ( bp_gpio_drv_hndl_t  hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl` Handle of the GPIO driver to reset.

Returned Errors `RTNC_SUCCESS`
`RTNC_FATAL`

Macro

BP_GPIO_DRV_HNDL_IS_NULL()

<gpio/bp_gpio_drv.h>

Evaluates if a GPIO driver handle is NULL.

Prototype `BP_GPIO_DRV_HNDL_IS_NULL ( hndl );`

Parameters `hndl` Handle to be checked.

Expansion `true` if the handle is NULL, `false` otherwise.

Macro

BP_GPIO_DRV_NULL_HNDL

<gpio/bp_gpio_drv.h>

NULL GPIO driver handle.

Chapter

16

I2C Driver Reference

The I2C driver declarations found in this module serves as the basis of I2C drivers usually used in combination with the I2C module to access I2C peripherals. All I2C drivers are composed of a standard set of API expected by the I2C module in addition to any number of implementation specific functions. The driver specific functions can be used by the application to access advanced features of a I2C peripheral not exposed through the standard API. Note that usage of those extended functionalities is non-portable contrary to the standard API. The I2C module API function `bp_i2c_drv_hndl_get()` function can be used to retrieve the driver handle associated with a I2C module instance, and can subsequently be used to call the driver directly. See the individual driver's documentation for details of the extended functions.

In addition to accessing extended functionalities, an application can access the driver standard API directly bypassing the I2C module. This reduces the call overhead at the cost of thread-safety as bare driver functions are usually not thread-safe when called directly. If thread-safety is required while calling driver functions directly, it is possible to use `bp_i2c_acquire()` and `bp_i2c_release()` to lock the I2C module preventing it from being accessed by other threads.

Finally, as yet another feature of the I2C driver API, it can be invoked in a standalone fashion without a UART module instance. This reduces the RAM overhead of using an I2C peripheral by dropping the I2C module mutexes and internal data structures. In this case the driver create function is called directly by the application in a matter similar to `bp_i2c_create()` to instantiate the driver. In this case thread safety has to be managed by the application, either using external mutexes or by ensuring that only one thread accesses the I2C peripheral.

Data Type

bp_i2c_drv_cfg_get_t

<i2c/bp_i2c_drv.h>

I2C driver's configuration get function.

Prototype

```
int  bp_i2c_drv_cfg_get_t  ( bp_i2c_drv_hndl_t  hndl,
                             bp_i2c_cfg_t *     p_cfg,
                             uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the I2C driver to query.
`p_cfg`	Pointer to the I2C configuration.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_i2c_drv_cfg_set_t

<i2c/bp_i2c_drv.h>

Prototype

```
int  bp_i2c_drv_cfg_set_t  ( bp_i2c_drv_hndl_t    hndl,
                             const bp_i2c_cfg_t *  p_cfg,
                             uint32_t              timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the I2C driver to configure.
`p_cfg`	I2C configuration.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_NOT_SUPPORTED`
`RTNC_FATAL`

Data Type

bp_i2c_drv_create_t

<i2c/bp_i2c_drv.h>

I2C driver's open function.

Prototype

```
int  bp_i2c_drv_create_t  ( const bp_i2c_board_def_t *  p_def,
                            bp_i2c_drv_hndl_t *        p_hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`p_def`	Board definition of the I2C driver to initialize.
`p_hndl`	Pointer to the newly created I2C interface.

Returned Errors

`RTNC_SUCCESS`
`RTNC_ALREADY_EXIST`
`RTNC_NO_RESOURCE`
`RTNC_FATAL`

Data Type

bp_i2c_drv_destroy_t

<i2c/bp_i2c_drv.h>

I2C driver's destroy function.

Prototype

```
int  bp_i2c_drv_destroy_t  ( bp_i2c_drv_hndl_t  hndl,
                             uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the I2C driver to enable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_i2c_drv_dis_t

<i2c/bp_i2c_drv.h>

I2C driver's disable function.

Prototype

```
int  bp_i2c_drv_dis_t  ( bp_i2c_drv_hndl_t  hndl,
                         uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the I2C driver to disable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_i2c_drv_en_t

<i2c/bp_i2c_drv.h>

I2C driver's enable function.

Prototype

```
int  bp_i2c_drv_en_t  ( bp_i2c_drv_hndl_t  hndl,
                        uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the I2C driver to enable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_i2c_drv_flush_t

<i2c/bp_i2c_drv.h>

I2C driver's flush function.

Prototype

```
int  bp_i2c_drv_flush_t  ( bp_i2c_drv_hndl_t  hndl,
                           uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the interface to flush.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_i2c_drv_idle_wait_t

<i2c/bp_i2c_drv.h>

I2C driver's idle wait function.

Prototype

```
int  bp_i2c_drv_idle_wait_t  ( bp_i2c_drv_hndl_t  hndl,
                               uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to wait.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_i2c_drv_is_en_t

<i2c/bp_i2c_drv.h>

I2C driver is_en function.

Prototype

```
int  bp_i2c_drv_is_en_t  ( bp_i2c_drv_hndl_t  hndl,
                           bool *             p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the I2C driver to query.
`p_is_en`	Interface state, true if enabled false otherwise.

Returned Errors
RTNC_SUCCESS
RTNC_FATAL

Data Type

bp_i2c_drv_reset_t

<i2c/bp_i2c_drv.h>

I2C drivers's reset function.

Prototype

```
int  bp_i2c_drv_reset_t  ( bp_i2c_drv_hndl_t  hndl,
                           uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

hndl	Handle of the I2C driver to reset.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Data Type

bp_i2c_drv_xfer_async_abort_t

<i2c/bp_i2c_drv.h>

I2C driver's asynchronous transfer abort function.

Prototype

```
int  bp_i2c_drv_xfer_async_abort_t  ( bp_i2c_drv_hndl_t  hndl,
                                      size_t *           p_tf_len,
                                      uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

hndl	Handle of the interface to abort.
p_tf_len	Amount of data transferred.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Data Type

bp_i2c_drv_xfer_async_t

<i2c/bp_i2c_drv.h>

I2C driver asynchronous transfer function.

Prototype

```
int  bp_i2c_drv_xfer_async_t  ( bp_i2c_drv_hndl_t  hndl,
                                bp_i2c_tf_t *      p_tf,
                                uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

hndl	Handle of the driver to use for the transfer.
p_tf	Transfer parameters.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Data Type

bp_i2c_drv_xfer_t

<i2c/bp_i2c_drv.h>

I2C driver's transfer function.

Prototype

```
int  bp_i2c_drv_xfer_t  ( bp_i2c_drv_hndl_t  hndl,
                          bp_i2c_tf_t *      p_tf,
                          size_t *           p_tf_len,
                          uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the interface to use.
`p_tf`	Pointer to an `bp_i2c_tf_t` structure describing the transfer to perform.
`p_tf_len`	
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_IO_ERR`
`RTNC_FATAL`

Macro

BP_I2C_DRV_HNDL_IS_NULL()

<i2c/bp_i2c_drv.h>

Evaluates if an I2C driver handle is NULL.

Prototype

```
BP_I2C_DRV_HNDL_IS_NULL ( hndl );
```

Parameters

`hndl`	Handle to be checked.

Expansion

`true` if the handle is NULL, `false` otherwise.

Macro

BP_I2C_DRV_NULL_HNDL

<i2c/bp_i2c_drv.h>

NULL I2C driver handle.

Chapter

17

SPI Driver Reference

The SPI driver declarations found in this module serves as the basis of SPI drivers usually used in combination with the SPI module to access SPI peripherals. All SPI drivers are composed of a standard set of API expected by the SPI module in addition to any number of implementation-specific functions. The driver specific functions can be used by the application to access advanced features of a SPI peripheral not exposed through the standard API. Note that usage of those extended functionalities is non-portable contrary to the standard API. The SPI module API function `bp_spi_drv_hndl_get()` function can be used to retrieve the driver handle associated with a SPI module instance, and can subsequently be used to call the driver directly. See the individual driver's documentation for details of the extended functions.

In addition to accessing extended functionalities, an application can access the driver standard API directly bypassing the SPI module. This reduces the call overhead at the cost of thread-safety as bare driver functions are usually not thread-safe when called directly. If thread-safety is required while calling driver functions directly, it is possible to use `bp_spi_slave_sel()` and `bp_spi_slave_desel()` to lock the SPI module preventing it from being accessed by other threads.

Finally, as yet another feature of the SPI driver API, it can be invoked in a standalone fashion without a SPI module instance. This reduces the RAM overhead of using an SPI peripheral by dropping the SPI module mutexes and internal data structures. In this case the driver create function is called directly by the application in a matter similar to `bp_spi_create()` to instantiate the driver. In this case thread safety has to be managed by the application, either using external mutexes or by ensuring that only one thread accesses the SPI peripheral.

Data Type

bp_spi_drv_cfg_get_t

<spi/bp_spi_drv.h>

SPI driver's cfg_get function.

Prototype

```
int  bp_spi_drv_cfg_get_t  ( bp_spi_drv_hndl_t  hndl,
                             bp_spi_cfg_t *     p_cfg,
                             uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the SPI driver to query.
`p_cfg`	Pointer to the SPI configuration.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_cfg_set_t

<spi/bp_spi_drv.h>

SPI driver's cfg_set function.

Prototype

```
int  bp_spi_drv_cfg_set_t  ( bp_spi_drv_hndl_t    hndl,
                             const bp_spi_cfg_t *  p_cfg,
                             uint32_t              timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the SPI driver to configure.
`p_cfg`	SPI configuration.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_NOT_SUPPORTED`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_create_t

<spi/bp_spi_drv.h>

SPI driver's create function.

Prototype

```
int  bp_spi_drv_create_t  ( const bp_spi_board_def_t *  p_def,
                            bp_spi_drv_hndl_t *        p_hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

p_def	Board definition of the SPI peripheral to initialize.
p_hndl	Pointer to the newly created SPI driver instance.

Returned Errors

RTNC_SUCCESS
RTNC_ALREADY_EXIST
RTNC_NO_RESOURCE
RTNC_FATAL

Data Type

bp_spi_drv_destroy_t

<spi/bp_spi_drv.h>

SPI driver's destroy function.

Prototype

```
int  bp_spi_drv_destroy_t  ( bp_spi_drv_hndl_t  hndl,
                             uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

hndl	Handle of the SPI driver to destroy.
timeout_ms	Timeout value in milliseconds.

Returned Errors

RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Data Type

bp_spi_drv_dis_t

<spi/bp_spi_drv.h>

SPI driver's disable function.

Prototype

```
int  bp_spi_drv_dis_t  ( bp_spi_drv_hndl_t  hndl,
                         uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the SPI driver to disable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_en_t

<spi/bp_spi_drv.h>

SPI driver's enable function.

Prototype

```
int  bp_spi_drv_en_t  ( bp_spi_drv_hndl_t  hndl,
                        uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the SPI driver to enable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_flush_t

<spi/bp_spi_drv.h>

SPI driver's flush function.

Prototype

```
int  bp_spi_drv_flush_t  ( bp_spi_drv_hndl_t  hndl,
                           uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to flush.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_idle_wait_t

<spi/bp_spi_drv.h>

SPI driver's idle wait function.

Prototype

```
int  bp_spi_drv_idle_wait_t  ( bp_spi_drv_hndl_t  hndl,
                               uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to wait.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_is_en_t

<spi/bp_spi_drv.h>

SPI driver's is_en function.

Prototype

```
int  bp_spi_drv_is_en_t  ( bp_spi_drv_hndl_t  hndl,
                           bool *             p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the SPI interface to check.
`p_is_en`	Interface state, true if enabled false otherwise.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_spi_drv_reset_t

<spi/bp_spi_drv.h>

SPI driver's reset function.

Prototype

```
int  bp_spi_drv_reset_t  ( bp_spi_drv_hndl_t  hndl,
                           uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the SPI interface to reset.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_slave_desel_t

<spi/bp_spi_drv.h>

SPI driver's slave deselect function.

Prototype

```
int  bp_spi_drv_slave_desel_t  ( bp_spi_drv_hndl_t  hndl,
                                 uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the SPI driver to wait on.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_slave_sel_t

<spi/bp_spi_drv.h>

SPI driver'd slave select function.

Prototype

```
int  bp_spi_drv_slave_sel_t  ( bp_spi_drv_hndl_t  hndl,
                               uint32_t           ss_id,
                               uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the SPI driver to wait on.
`ss_id`	Numeric id of the slave select line to assert.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_xfer_async_abort_t

<spi/bp_spi_drv.h>

SPI driver's asynchronous transfer abort function.

Prototype

```
int  bp_spi_drv_xfer_async_abort_t  ( bp_spi_drv_hndl_t  hndl,
                                      size_t *          p_tx_len,
                                      size_t *          p_rx_len,
                                      uint32_t          timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to abort.
`p_tx_len`	Pointer to the amount of data already transferred.
`p_rx_len`	Pointer to the amount of data already received.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_xfer_async_t

<spi/bp_spi_drv.h>

SPI driver's asynchronous transfer function.

Prototype

```
int  bp_spi_drv_xfer_async_t  ( bp_spi_drv_hndl_t  hndl,
                                bp_spi_tf_t *      p_tf,
                                uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to use for the transfer.
`p_tf`	Transfer parameters.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_spi_drv_xfer_t

<spi/bp_spi_drv.h>

SPI driver's xfer function.

Prototype

```
int  bp_spi_drv_xfer_t  ( bp_spi_drv_hndl_t  hndl,
                          bp_spi_tf_t *      p_tf,
                          size_t *           p_tf_len,
                          uint32_t           timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the interface to use for the transfer.
`p_tf`	Pointer to an bp_spi_tf_t structure describing the transfer to perform.
`p_tf_len`	Amount of data actually transferred.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_IO_ERR`
`RTNC_FATAL`

Macro

BP_SPI_DRV_HNDL_IS_NULL()

<spi/bp_spi_drv.h>

Evaluates if an SPI driver handle is NULL.

Prototype

```
BP_SPI_DRV_HNDL_IS_NULL  (  hndl );
```

Parameters

`hndl` Handle to be checked.

Expansion

`true` if the handle is NULL, `false` otherwise.

Macro

BP_SPI_DRV_NULL_HNDL

<spi/bp_spi_drv.h>

NULL SPI driver handle.

Chapter

18

UART Driver Reference

The UART driver declarations found in this module serves as the basis of UART drivers usually used in combination with the UART module to access UART peripherals. All UART drivers are composed of a standard set of API expected by the UART module in addition to any number of implementation-specific functions. The driver specific functions can be used by the application to access advanced features of a UART peripheral not exposed through the standard API. Note that usage of those extended functionalities is non-portable contrary to the standard API. The UART module API function `bp_uart_drv_hndl_get()` function can be used to retrieve the driver handle associated with a UART module instance, and can subsequently be used to call the driver directly. See the individual driver's documentation for details of the extended functions.

In addition to accessing extended functionalities, an application can access the driver standard API directly bypassing the UART module. This reduces the call overhead at the cost of thread-safety as bare driver functions are usually not thread-safe when called directly. If thread-safety is required while calling driver functions directly, it is possible to use `bp_uart_acquire()` and `bp_uart_release()` to lock the UART module preventing its access by other threads.

Finally, as yet another feature of the UART driver API, it can be invoked in a standalone fashion without a UART module instance. This reduces the RAM overhead of using a UART peripheral by dropping the UART module mutexes and internal data structures. In this case the driver create function is called directly by the application in a matter similar to `bp_uart_create()` to instantiate the driver. In this case thread safety has to be managed by the application, either using external mutexes or by ensuring that only one thread accesses the UART peripheral.

Data Type

bp_uart_cfg_get_t

<uart/bp_uart_drv.h>

UART driver's cfg_get function.

Prototype

```
int  bp_uart_cfg_get_t  ( bp_uart_drv_hndl_t  hndl,
                          bp_uart_cfg_t *     p_cfg );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the UART driver to query.
`p_cfg`	Pointer to the UART configuration.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_uart_drv_cfg_set_t

<uart/bp_uart_drv.h>

UART driver's cfg_set function.

Prototype

```
int  bp_uart_drv_cfg_set_t  ( bp_uart_drv_hndl_t     hndl,
                              const bp_uart_cfg_t *  p_cfg,
                              uint32_t               timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the UART drover to configure.
`p_cfg`	UART configuration.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_NOT_SUPPORTED`
`RTNC_FATAL`

Data Type

bp_uart_drv_create_t

<uart/bp_uart_drv.h>

UART driver's create function.

Prototype

```
int  bp_uart_drv_create_t  ( const bp_uart_board_def_t *  p_def,
                             bp_uart_drv_hndl_t *         p_hndl );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`p_def`	Board definition of the UART peripheral to initialize.
`p_hndl`	Pointer to the newly created UART driver instance.

Returned Errors

`RTNC_SUCCESS`
`RTNC_ALREADY_EXIST`
`RTNC_NO_RESOURCE`
`RTNC_FATAL`

Data Type

bp_uart_drv_destroy_t

<uart/bp_uart_drv.h>

UART driver's destroy function.

Prototype

```
int  bp_uart_drv_destroy_t  ( bp_uart_drv_hndl_t  hndl,
                              uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the UART driver instance to enable.
`timeout_ms`	

Returned Errors

`RTNC_SUCCESS`
`RTNC_NOT_SUPPORTED`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_uart_drv_dis_t

<uart/bp_uart_drv.h>

UART driver'd disable function.

Prototype

```
int  bp_uart_drv_dis_t  ( bp_uart_drv_hndl_t  hndl,
                          uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the UART driver to disable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_uart_drv_en_t

<uart/bp_uart_drv.h>

UART driver's enable function.

Prototype

```
int  bp_uart_drv_en_t  ( bp_uart_drv_hndl_t  hndl,
                         uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the UART driver to enable.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_uart_drv_is_en_t

<uart/bp_uart_drv.h>

UART driver's is_en function.

Prototype

```
int  bp_uart_drv_is_en_t  ( bp_uart_drv_hndl_t  hndl,
                            bool *              p_is_en );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the UART driver to query.
`p_is_en`	Interface state, true if enabled false otherwise.

Returned Errors

`RTNC_SUCCESS`
`RTNC_FATAL`

Data Type

bp_uart_drv_reset_t

<uart/bp_uart_drv.h>

UART driver's reset function.

Prototype

```
int  bp_uart_drv_reset_t  ( bp_uart_drv_hndl_t  hndl,
                            uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the UART driver to reset.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_uart_drv_rx_async_abort_t

<uart/bp_uart_drv.h>

UART driver's asynchronous receive abort function.

Prototype

```
int  bp_uart_drv_rx_async_abort_t  ( bp_uart_drv_hndl_t  hndl,
                                     size_t *            p_rx_len,
                                     uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to abort.
`p_rx_len`	Pointer to the number of bytes received, can be NULL.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Data Type

bp_uart_drv_rx_async_t

<uart/bp_uart_drv.h>

UART driver's asynchronous receive function.

Prototype

```
int  bp_uart_drv_rx_async_t  ( bp_uart_drv_hndl_t  hndl,
                               bp_uart_tf_t *      p_tf,
                               uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

hndl	Handle of the driver to use for reception.
p_tf	Transfer parameters.
timeout_ms	Timeout value in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Data Type

bp_uart_drv_rx_flush_t

<uart/bp_uart_drv.h>

UART driver's receive flush function.

Prototype

```
int  bp_uart_drv_rx_flush_t  ( bp_uart_drv_hndl_t  hndl,
                               uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

hndl	Handle of the driver to flush.
timeout_ms	Timeout in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Data Type

bp_uart_drv_rx_idle_wait_t

<uart/bp_uart_drv.h>

UART driver's receive idle wait function.

Prototype

```
int  bp_uart_drv_rx_idle_wait_t  ( bp_uart_drv_hndl_t  hndl,
                                   uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

hndl	Handle of the driver to wait.
timeout_ms	Timeout in milliseconds.

Returned Errors
RTNC_SUCCESS
RTNC_TIMEOUT
RTNC_FATAL

Data Type

bp_uart_drv_rx_t

<uart/bp_uart_drv.h>

UART driver's receive function.

Prototype

```
int  bp_uart_drv_rx_t  ( bp_uart_drv_hndl_t  hndl,
                         void *              p_buf,
                         size_t              len,
                         size_t *            p_rx_len,
                         uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to use for reception.
`p_buf`	Pointer to the buffer that will receive the data.
`len`	Length of the data to receive in bytes.
`p_rx_len`	
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_IO_ERR`
`RTNC_FATAL`

Data Type

bp_uart_drv_tx_async_abort_t

<uart/bp_uart_drv.h>

UART driver's asynchronous transmit abort function.

Prototype

```
int  bp_uart_drv_tx_async_abort_t  ( bp_uart_drv_hndl_t  hndl,
                                     size_t *            p_tx_len,
                                     uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to abort.
`p_tx_len`	Pointer to the number of bytes transmitted, can be NULL.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_uart_drv_tx_async_t

<uart/bp_uart_drv.h>

UART driver's asynchronous transmit function.

Prototype

```
int  bp_uart_drv_tx_async_t  ( bp_uart_drv_hndl_t  hndl,
                               bp_uart_tf_t *      p_tf,
                               uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to use for reception.
`p_tf`	Transfer parameters.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_uart_drv_tx_flush_t

<uart/bp_uart_drv.h>

UART driver's transmit flush function.

Prototype

```
int  bp_uart_drv_tx_flush_t  ( bp_uart_drv_hndl_t  hndl,
                               uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to flush.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_uart_drv_tx_idle_wait_t

<uart/bp_uart_drv.h>

UART driver's transmit idle wait function.

Prototype

```
int  bp_uart_drv_tx_idle_wait_t  ( bp_uart_drv_hndl_t  hndl,
                                   uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to wait.
`timeout_ms`	Timeout in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_FATAL`

Data Type

bp_uart_drv_tx_t

<uart/bp_uart_drv.h>

UART driver's transmit function.

Prototype

```
int  bp_uart_drv_tx_t  ( bp_uart_drv_hndl_t  hndl,
                         const void *        p_buf,
                         size_t              len,
                         uint32_t            timeout_ms );
```

Attributes

Blocking	ISR-safe	Critical safe	Thread-safe
✓	✗	✗	✓

Parameters

`hndl`	Handle of the driver to use for transmission.
`p_buf`	Pointer to the buffer to transmit.
`len`	Length of the data to transmit in bytes.
`timeout_ms`	Timeout value in milliseconds.

Returned Errors

`RTNC_SUCCESS`
`RTNC_TIMEOUT`
`RTNC_IO_ERR`
`RTNC_FATAL`

Macro

BP_UART_DRV_HNDL_IS_NULL()

<uart/bp_uart_drv.h>

Evaluates if a UART driver handle is NULL.

Prototype

```
BP_UART_DRV_HNDL_IS_NULL  (  hndl );
```

Parameters

`hndl`	Handle to be checked.

Expansion `true` if the handle is NULL, `false` otherwise.

Macro

BP_UART_DRV_NULL_HNDL

<uart/bp_uart_drv.h>

NULL UART driver handle.

Chapter

19

Document Revision History

The revision history of the BASEplatform user manual and reference manuals can be found within the BASEplatform source package.