Bootloader

Generated by Doxygen 1.8.17

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 dataBuffer_t Struct Reference

The Uart Data Buffer.

Data Fields

- uint8_t ∗ **ptr**
- uint32_t **pos**
- uint32_t **size**
- uint8_t **state**

3.1.1 Detailed Description

The Uart Data Buffer.

The documentation for this struct was generated from the following file:

- Uart.c

3.2 Fpec_t Struct Reference

Data Fields

- uint32_t **ACR**
- uint32_t **KEYR**
- uint32_t **OPTKEYR**
- uint32_t **SR**
- uint32_t **CR**
- uint32_t **AR**
- uint32_t **DUMMY**
- uint32_t **OBR**
- uint32_t **WRPR**

The documentation for this struct was generated from the following file:

- Fpec.c

3.3 gpio_t Struct Reference

The GPIO Type contains GPIO configurations.

```
#include <Gpio.h>
```

Data Fields

- uint32_t **pins**
- uint32_t **speed**
- uint32_t **mode**
- uint32_t **port**

3.3.1 Detailed Description

The GPIO Type contains GPIO configurations.

The documentation for this struct was generated from the following file:

- Gpio.h

3.4 gpioReg_t Struct Reference

Data Fields

- uint64_t **CR**
- uint32_t **IDR**
- uint32_t **ODR**
- uint32_t **BSR**
- uint32_t **BRR**
- uint32_t **LCK**

The documentation for this struct was generated from the following file:

- Gpio.c

3.5 header_t Struct Reference

Data Fields

- uint32_t **key**
- uint16_t **type**
- uint16_t **length**

The documentation for this struct was generated from the following file:

- Protocol.c

3.6 nvic_t Struct Reference

The NVIC Registers.

Data Fields

- uint32_t **SETEN** [8]
- uint32_t **_RESERVED0** [24]
- uint32_t **CLREN** [8]
- uint32_t **_RSERVED1** [24]
- uint32_t **SETPND** [8]
- uint32_t **_RESERVED2** [24]
- uint32_t **CLRPND** [8]
- uint32_t **_RESERVED3** [24]
- uint32_t **AB** [8]
- uint32_t **_RESERVED4** [56]
- uint8_t **PRI** [240]
- uint32_t **_RESERVED5** [644]
- uint32_t **STIR**

3.6.1 Detailed Description

The NVIC Registers.

The documentation for this struct was generated from the following file:

- Nvic.c

3.7 packet_t Struct Reference

Data Fields

- union {
  header_t **header**
  uint8_t **headerData** [PROTOCOL_HEADER_SIZE]
  };

The documentation for this struct was generated from the following file:

- Protocol.c

3.8 switch_t Struct Reference

The Switch pin layout.

```
#include <Switch.h>
```

Data Fields

- uint32_t **pin**
- uint32_t **port**
- uint8_t **activeState**

3.8.1 Detailed Description

The Switch pin layout.

The documentation for this struct was generated from the following file:

- Switch.h

3.9 Uart_cfg_t Struct Reference

Data Fields

- uint32_t **baudRate**
- uint32_t **stopBits**
- uint32_t **parity**
- uint32_t **flowControl**
- uint32_t **sysClk**
- uint32_t **linEn**
- uint8_t **interrupts**
- uint8_t **uartModule**

The documentation for this struct was generated from the following file:

- Uart.h

3.10 uart_t Struct Reference

The UART Registers.

Data Fields

- uint32_t **SR**
- uint32_t **DR**
- uint32_t **BRR**
- uint32_t **CR1**
- uint32_t **CR2**
- uint32_t **CR3**
- uint32_t **GTPR**

3.10.1 Detailed Description

The UART Registers.

The documentation for this struct was generated from the following file:

- Uart.c

Chapter 4

File Documentation

4.1 Bootloader.c File Reference

A Simple Bootloader Application Over Uart Using A Designed Software Protocol.

```
#include "Std_Types.h"
#include "Uart.h"
#include "Fpec.h"
#include "Rcc.h"
#include "HRcc.h"
#include "Nvic.h"
#include "Gpio.h"
#include "Switch.h"
#include "Protocol.h"
```

Macros

- #define **MAX_DATA_TO_BUFFER** 1024
- #define **APP_EXIST** 0xAAAA
- #define **APP_NOT_EXIST** 0xBBBB

Typedefs

- typedef void(∗ **app_t**) (void)

Functions

- int **main** (void)

Variables

- const uint16_t ∗ **appExistMarker** = (uint16_t∗)0x0800FFE0
- const uint16_t ∗ **addressMarker** = (uint16_t∗)0x0800FFD0

4.1.1 Detailed Description

A Simple Bootloader Application Over Uart Using A Designed Software Protocol.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-05-25

Copyright

Copyright (c) 2020

4.2 Fpec.h File Reference

This is the user interface for the FPEC Driver.

Functions

- Std_ReturnType Fpec_Lock (void)
  Locks The FPEC.
- Std_ReturnType Fpec_Unlock (void)
  Unlocks The FPEC.
- Std_ReturnType Fpec_WriteHalfWord (uint16_t ∗address, uint16_t data)
  Writes A Half Word To The Flash.
- Std_ReturnType Fpec_WriteBlock (uint16_t ∗flashAddress, uint16_t ∗srcAddress, uint16_t blockSize)
  Writes A Block To The Flash.
- Std_ReturnType Fpec_ErasePage (uint32_t ∗pageAddress)
  Erases A Page In The Flash.
- Std_ReturnType Fpec_MassErase (void)
  Erases The Flash Completely !!

4.2.1 Detailed Description

This is the user interface for the FPEC Driver.

This is the implementation for the FPEC Driver.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-05-09

Copyright

Copyright (c) 2020

4.2.2 Function Documentation

4.2.2.1 Fpec_ErasePage()

```
Std_ReturnType Fpec_ErasePage (
            uint32_t * pageAddress )
```

Erases A Page In The Flash.

Parameters

pageAddress	The Address Of The Page

Returns

Std_ReturnType A Status E_OK : If The Function Executed Successfully E_NOT_OK : If The Function Didn't Execute Successfully

4.2.2.2 Fpec_Lock()

```
Std_ReturnType Fpec_Lock (
            void )
```

Locks The FPEC.

Returns

Std_ReturnType A Status E_OK : If The Function Executed Successfully E_NOT_OK : If The Function Didn't Execute Successfully

4.2.2.3 Fpec_MassErase()

```
Std_ReturnType Fpec_MassErase (
            void )
```

Erases The Flash Completely !!

Returns

Std_ReturnType A Status E_OK : If The Function Executed Successfully E_NOT_OK : If The Function Didn't Execute Successfully

4.2.2.4 Fpec_Unlock()

```
Std_ReturnType Fpec_Unlock (
            void  )
```

Unlocks The FPEC.

Returns

> Std_ReturnType A Status E_OK : If The Function Executed Successfully E_NOT_OK : If The Function Didn't Execute Successfully

4.2.2.5 Fpec_WriteBlock()

```
Std_ReturnType Fpec_WriteBlock (
            uint16_t * flashAddress,
            uint16_t * srcAddress,
            uint16_t blockSize )
```

Writes A Block To The Flash.

Parameters

flashAddress	The Address In Flash
srcAddress	The Source Address To Fetch Data From
blockSize	The Size Of The Block In Half Words

Returns

> Std_ReturnType A Status E_OK : If The Function Executed Successfully E_NOT_OK : If The Function Didn't Execute Successfully

4.2.2.6 Fpec_WriteHalfWord()

```
Std_ReturnType Fpec_WriteHalfWord (
            uint16_t * address,
            uint16_t data )
```

Writes A Half Word To The Flash.

Parameters

address	The Address In Flash
data	The Half Word To Write

Returns

Std_ReturnType A Status E_OK : If The Function Executed Successfully E_NOT_OK : If The Function Didn't Execute Successfully

4.3 Gpio.c File Reference

This file is to be used as an implementation of the GPIO driver.

```
#include "Std_Types.h"
#include "Gpio.h"
```

Data Structures

- struct gpioReg_t

Macros

- #define **GPIO_MODE_INPUT_MASK** 0xF0
- #define **GPIO_MODE_MASK** 0x0C

Functions

- Std_ReturnType Gpio_InitPins (gpio_t ∗gpio)

  Initializes pins mode and speed for a specific port.
- Std_ReturnType Gpio_WritePin (uint32_t port, uint32_t pin, uint32_t pinStatus)

  Write a value to a pin(0/1)
- Std_ReturnType Gpio_ReadPin (uint32_t port, uint32_t pin, uint8_t ∗state)

  Reads a value to a pin(0/1)

4.3.1 Detailed Description

This file is to be used as an implementation of the GPIO driver.

Author

Mark Attia

Date

February 6, 2020

4.3.2 Function Documentation

4.3.2.1 Gpio_InitPins()

```
Std_ReturnType Gpio_InitPins (
            gpio_t ∗ gpio )
```

Initializes pins mode and speed for a specific port.

4.3.2.2 Function: Gpio_InitPins

Parameters

gpio	An object of type gpio_t to set pins for

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.3.2.3 Gpio_ReadPin()

```
Std_ReturnType Gpio_ReadPin (
             uint32_t port,
             uint32_t pin,
             uint8_t * state )
```

Reads a value to a pin(0/1)

4.3.2.4 Function: Gpio_ReadPin

Parameters

port	The port you want to read from • GPIO_PORTX : The pin number you want to read from
pin	The pin you want to read • GPIO_PIN_X : The pin number you want to read //You can OR more than one pin\
state	To return a status in • GPIO_PIN_SET : The pin is set to 1 • GPIO_PIN_RESET : The pin is set to 0

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.3.2.5 Gpio_WritePin()

```
Std_ReturnType Gpio_WritePin (
             uint32_t port,
             uint32_t pin,
             uint32_t pinStatus )
```

Write a value to a pin(0/1)

4.3.2.6 Function: Gpio_WritePin

Parameters

port	The port you want to configure • GPIO_PORTX : The pin number you want to configure
pin	The pin you want to configure • GPIO_PIN_X : The pin number you want to configure //You can OR more than one pin\
pinStatus	The status of the pins (GPIO_PIN_SET/GPIO_PIN_RESET) • GPIO_PIN_SET : Sets the pin value to 1 • GPIO_PIN_RESET : Resets the pin value to 0

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.4 Gpio.h File Reference

This file is to be used as an interface for the user of GPIO driver.

Data Structures

- struct gpio_t
  The GPIO Type contains GPIO configurations.

Macros

- #define **GPIO_PIN_SET** 0
- #define **GPIO_PIN_RESET** !GPIO_PIN_SET
- #define **GPIO_PIN_0** 0x0001
- #define **GPIO_PIN_1** 0x0002
- #define **GPIO_PIN_2** 0x0004
- #define **GPIO_PIN_3** 0x0008
- #define **GPIO_PIN_4** 0x0010
- #define **GPIO_PIN_5** 0x0020
- #define **GPIO_PIN_6** 0x0040
- #define **GPIO_PIN_7** 0x0080
- #define **GPIO_PIN_8** 0x0100
- #define **GPIO_PIN_9** 0x0200
- #define **GPIO_PIN_10** 0x0400
- #define **GPIO_PIN_11** 0x0800
- #define **GPIO_PIN_12** 0x1000
- #define **GPIO_PIN_13** 0x2000
- #define **GPIO_PIN_14** 0x4000
- #define **GPIO_PIN_15** 0x8000
- #define **GPIO_PIN_ALL** 0xFFFF
- #define **GPIO_SPEED_10_MHZ** 0x01
- #define **GPIO_SPEED_02_MHZ** 0x02

- #define **GPIO_SPEED_50_MHZ** 0x03
- #define **GPIO_MODE_GP_OUTPUT_PP** 0x00
- #define **GPIO_MODE_GP_OUTPUT_OD** 0x04
- #define **GPIO_MODE_AF_OUTPUT_PP** 0x08
- #define **GPIO_MODE_AF_OUTPUT_OD** 0x0C
- #define **GPIO_MODE_INPUT_ANALOG** 0x10
- #define **GPIO_MODE_INPUT_FLOATING** 0x14
- #define **GPIO_MODE_INPUT_PULL_DOWN** 0x18
- #define **GPIO_MODE_INPUT_PULL_UP** 0x28
- #define **GPIO_PORTA** (uint32_t)0x40010800
- #define **GPIO_PORTB** (uint32_t)0x40010C00
- #define **GPIO_PORTC** (uint32_t)0x40011000
- #define **GPIO_PORTD** (uint32_t)0x40011400
- #define **GPIO_PORTE** (uint32_t)0x40011800
- #define **GPIO_PORTF** (uint32_t)0x40011C00
- #define **GPIO_PORTG** (uint32_t)0x40012000

Functions

- Std_ReturnType Gpio_InitPins (gpio_t ∗gpio)

  Initializes pins mode and speed for a specific port.
- Std_ReturnType Gpio_WritePin (uint32_t port, uint32_t pin, uint32_t pinStatus)

  Write a value to a pin(0/1)
- Std_ReturnType Gpio_ReadPin (uint32_t port, uint32_t pin, uint8_t ∗state)

  Reads a value to a pin(0/1)

4.4.1 Detailed Description

This file is to be used as an interface for the user of GPIO driver.

Author

Mark Attia

Date

February 6, 2020

4.4.2 Function Documentation

4.4.2.1 Gpio_InitPins()

```
Std_ReturnType Gpio_InitPins (
            gpio_t * gpio )
```

Initializes pins mode and speed for a specific port.

4.4.2.2 Function: Gpio_InitPins

Parameters

gpio	An object of type gpio_t to set pins for

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.4.2.3 Function: Gpio_InitPins

Parameters

gpio	An object of type gpio_t to set pins for

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.4.2.4 Gpio_ReadPin()

```
Std_ReturnType Gpio_ReadPin (
             uint32_t port,
             uint32_t pin,
             uint8_t * state )
```

Reads a value to a pin(0/1)

4.4.2.5 Function: Gpio_ReadPin

Parameters

port	The port you want to read from • GPIO_PORTX : The pin number you want to read from
pin	The pin you want to read • GPIO_PIN_X : The pin number you want to read //You can OR more than one pin\
state	To return a status in • GPIO_PIN_SET : The pin is set to 1 • GPIO_PIN_RESET : The pin is set to 0

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.4.2.6 Function: Gpio_ReadPin

Parameters

port	The port you want to read from • GPIO_PORTX : The pin number you want to read from
pin	The pin you want to read • GPIO_PIN_X : The pin number you want to read //You can OR more than one pin\
state	To return a status in • GPIO_PIN_SET : The pin is set to 1 • GPIO_PIN_RESET : The pin is set to 0

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.4.2.7 Gpio_WritePin()

```
Std_ReturnType Gpio_WritePin (
            uint32_t port,
            uint32_t pin,
            uint32_t pinStatus )
```

Write a value to a pin(0/1)

4.4.2.8 Function: Gpio_WritePin

Parameters

port	The port you want to configure • GPIO_PORTX : The pin number you want to configure
pin	The pin you want to configure • GPIO_PIN_X : The pin number you want to configure //You can OR more than one pin\
pinStatus	The status of the pins (GPIO_PIN_SET/GPIO_PIN_RESET) • GPIO_PIN_SET : Sets the pin value to 1 • GPIO_PIN_RESET : Resets the pin value to 0

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.4.2.9 Function: Gpio_WritePin

Parameters

port	The port you want to configure • GPIO_PORTX : The pin number you want to configure
pin	The pin you want to configure • GPIO_PIN_X : The pin number you want to configure //You can OR more than one pin\
pinStatus	The status of the pins (GPIO_PIN_SET/GPIO_PIN_RESET) • GPIO_PIN_SET : Sets the pin value to 1 • GPIO_PIN_RESET : Resets the pin value to 0

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.5 HRcc.h File Reference

This is the user interface for the RCC Handler.

Functions

- Std_ReturnType HRcc_SystemClockInit (void)

  This function initializes the system clock.
- Std_ReturnType HRcc_EnPortClock (uint32_t port)

  This function initializes the clock for a specific GPIO port.

4.5.1 Detailed Description

This is the user interface for the RCC Handler.

This is implementation for the RCC Handler.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-03-24

Copyright

Copyright (c) 2020

4.5.2 Function Documentation

4.5.2.1 HRcc_EnPortClock()

```
Std_ReturnType HRcc_EnPortClock (
            uint32_t port )
```

This function initializes the clock for a specific GPIO port.

Parameters

port	The GPIO port • GPIO_PORTX : The pin number you want to configure

Returns

Std_ReturnType
E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.5.2.2 HRcc_SystemClockInit()

```
Std_ReturnType HRcc_SystemClockInit (
            void )
```

This function initializes the system clock.

Returns

Std_ReturnType E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.6 Nvic.c File Reference

This file is an implementation for the NVIC driver for the arm cortex m3.

```
#include "Std_Types.h"
#include "Nvic.h"
#include "Nvic_Cfg.h"
```

Data Structures

- struct nvic_t

  The NVIC Registers.

Macros

- #define **NVIC_BASE_ADDRESS** 0xE000E100
- #define **AIRC** ∗((volatile uint32_t∗)0xE000ED0C) /∗ Application interrupt and reset control register ∗/
- #define **AIRC_LOCK** 0x05FA0000 /∗ Application interrupt and reset control register Lock ∗/
- #define **AIRC_LOCK_CLR** 0x0000FFFF /∗ Application interrupt and reset control register Lock Clear Mask ∗/
- #define **AIRC_SYS_RST** 0x00000004
- #define **AIRC_GROUP_CLR** 0x0000F8FF
- #define **NVIC_GROUP_CHECK** 0xFFFFFFF8
- #define **NVIC_NON_IMPLEMENTED_PRI** (8 - NVIC_GROUP_SIZE - NVIC_SUBGROUP_SIZE)
- #define **NVIC_0_BIT_MASK** 0b0
- #define **NVIC_1_BIT_MASK** 0b1
- #define **NVIC_2_BIT_MASK** 0b11
- #define **NVIC_3_BIT_MASK** 0b111
- #define **NVIC_4_BIT_MASK** 0b1111
- #define **NVIC_5_BIT_MASK** 0b11111
- #define **NVIC_6_BIT_MASK** 0b111111
- #define **NVIC_7_BIT_MASK** 0b1111111
- #define **NVIC_CONCAT_MASK**(x) NVIC_CONCAT_MASK_HELP(x)
- #define **NVIC_CONCAT_MASK_HELP**(x) NVIC_##x##_BIT_MASK
- #define **NVIC** ((volatile nvic_t∗)(NVIC_BASE_ADDRESS))

Functions

- Std_ReturnType Nvic_EnableInterrupt (uint8_t intNumber)

  Enables a specific Interrupt.
- Std_ReturnType Nvic_DisableInterrupt (uint8_t intNumber)

  Disables a specific Interrupt.
- Std_ReturnType Nvic_SetPending (uint8_t intNumber)

  Sets the pending flag for a specific interrupt.
- Std_ReturnType Nvic_ClearPending (uint8_t intNumber)

  Clears the pending flag for a specific interrupt.
- Std_ReturnType Nvic_IsInterruptActive (uint8_t ∗activeState, uint8_t intNumber)

  Checks if the interrupt is active.
- Std_ReturnType Nvic_SetSubpriority (uint8_t priority, uint8_t intNumber)

  Sets the subpriority for aspecific interrupt.
- Std_ReturnType Nvic_GetSubpriority (uint8_t ∗priority, uint8_t intNumber)

  Gets the subpriority for aspecific interrupt.
- Std_ReturnType Nvic_SetGroupPriority (uint8_t priority, uint8_t intNumber)

  Sets the group priority for aspecific interrupt.
- Std_ReturnType Nvic_GetGroupPriority (uint8_t ∗priority, uint8_t intNumber)

  Gets the group priority for aspecific interrupt.
- Std_ReturnType Nvic_ConfigGroupSize (void)

  Configure the group size.
- Std_ReturnType Nvic_ResetSystem (void)

  Resets the microcontroller.
- Std_ReturnType Nvic_EnablePeripheral (void)

  Enables the prepherals interrupts.
- Std_ReturnType Nvic_DisablePeripheral (void)

  Disable the prepherals interrupts.
- Std_ReturnType Nvic_SetFault (void)

Blocks all interrupts including hard fault.
- Std_ReturnType Nvic_ClearFault (void)

  Returns from fault mode.
- Std_ReturnType Nvic_FilterPriority (uint8_t pri)

  Only allow interrupts over a certain priority.
- Std_ReturnType Nvic_GenerateSoftwareInterrupt (uint8_t intNumber)

  Generates a software interrupt (Atomic function to generate interrupt immediately)

4.6.1 Detailed Description

This file is an implementation for the NVIC driver for the arm cortex m3.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-02-29

Copyright

Copyright (c) 2020

4.6.2 Function Documentation

4.6.2.1 Nvic_ClearFault()

```
Std_ReturnType Nvic_ClearFault (
                void  )
```

Returns from fault mode.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.2 Nvic_ClearPending()

```
Std_ReturnType Nvic_ClearPending (
                uint8_t intNumber )
```

Clears the pending flag for a specific interrupt.

Parameters

intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.3 Nvic_ConfigGroupSize()

```
Std_ReturnType Nvic_ConfigGroupSize (
            void  )
```

Configure the group size.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.4 Nvic_DisableInterrupt()

```
Std_ReturnType Nvic_DisableInterrupt (
            uint8_t intNumber )
```

Disables a specific Interrupt.

Parameters

intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.5 Nvic_DisablePeripheral()

```
Std_ReturnType Nvic_DisablePeripheral (
            void  )
```

Disable the prepherals interrupts.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.6 Nvic_EnableInterrupt()

```
Std_ReturnType Nvic_EnableInterrupt (
            uint8_t intNumber )
```

Enables a specific Interrupt.

Parameters

intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.7 Nvic_EnablePeripheral()

```
Std_ReturnType Nvic_EnablePeripheral (
            void  )
```

Enables the prepherals interrupts.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.8 Nvic_FilterPriority()

```
Std_ReturnType Nvic_FilterPriority (
            uint8_t pri )
```

Only allow interrupts over a certain priority.

Parameters

per	the minimum priority allowed

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.9 Nvic_GenerateSoftwareInterrupt()

```
Std_ReturnType Nvic_GenerateSoftwareInterrupt (
            uint8_t intNumber )
```

Generates a software interrupt (Atomic function to generate interrupt immediately)

Parameters

intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.10 Nvic_GetGroupPriority()

```
Std_ReturnType Nvic_GetGroupPriority (
            uint8_t * priority,
            uint8_t intNumber )
```

Gets the group priority for aspecific interrupt.

Parameters

priority	the priority you want to get
intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.11 Nvic_GetSubpriority()

```
Std_ReturnType Nvic_GetSubpriority (
            uint8_t * priority,
            uint8_t intNumber )
```

Gets the subpriority for aspecific interrupt.

Parameters

priority	the priority you want to get
intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.12 Nvic_IsInterruptActive()

```
Std_ReturnType Nvic_IsInterruptActive (
            uint8_t * activeState,
            uint8_t intNumber )
```

Checks if the interrupt is active.

Parameters

activeState	the active state of the interrupt • NVIC_ACTIVE • NVIC_NOT_ACTIVE
intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.13 Nvic_ResetSystem()

```
Std_ReturnType Nvic_ResetSystem (
            void  )
```

Resets the microcontroller.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.14 Nvic_SetFault()

```
Std_ReturnType Nvic_SetFault (
            void  )
```

Blocks all interrupts including hard fault.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.15 Nvic_SetGroupPriority()

```
Std_ReturnType Nvic_SetGroupPriority (
            uint8_t priority,
            uint8_t intNumber )
```

Sets the group priority for aspecific interrupt.

Parameters

priority	the priority you want to set
intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.16 Nvic_SetPending()

```
Std_ReturnType Nvic_SetPending (
            uint8_t intNumber )
```

Sets the pending flag for a specific interrupt.

Parameters

intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.6.2.17 Nvic_SetSubpriority()

```
Std_ReturnType Nvic_SetSubpriority (
            uint8_t priority,
            uint8_t intNumber )
```

Sets the subpriority for aspecific interrupt.

Parameters

priority	the priority you want to set
intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7 Nvic.h File Reference

This file is a user interface for the NVIC driver for the arm cortex m3.

Macros

- #define **NVIC_ACTIVE** 0
- #define **NVIC_NOT_ACTIVE** !NVIC_ACTIVE
- #define NVIC_IRQNUM_WWDG 0
- #define **NVIC_IRQNUM_PVD** 1
- #define **NVIC_IRQNUM_TAMPER** 2
- #define **NVIC_IRQNUM_RTC** 3
- #define **NVIC_IRQNUM_FLASH** 4
- #define **NVIC_IRQNUM_RCC** 5
- #define **NVIC_IRQNUM_EXTI0** 6
- #define **NVIC_IRQNUM_EXTI1** 7
- #define **NVIC_IRQNUM_EXTI2** 8
- #define **NVIC_IRQNUM_EXTI3** 9
- #define **NVIC_IRQNUM_EXTI4** 10
- #define **NVIC_IRQNUM_DMA1_CHANNEL1** 11
- #define **NVIC_IRQNUM_DMA1_CHANNEL2** 12
- #define **NVIC_IRQNUM_DMA1_CHANNEL3** 13
- #define **NVIC_IRQNUM_DMA1_CHANNEL4** 14
- #define **NVIC_IRQNUM_DMA1_CHANNEL5** 15

- #define **NVIC_IRQNUM_DMA1_CHANNEL6** 16
- #define **NVIC_IRQNUM_DMA1_CHANNEL7** 17
- #define **NVIC_IRQNUM_ADC1_2** 18
- #define **NVIC_IRQNUM_USB_HP_CAN_TX** 19
- #define **NVIC_IRQNUM_USB_HP_CAN_RX0** 20
- #define **NVIC_IRQNUM_CAN_RX1** 21
- #define **NVIC_IRQNUM_CAN_SCE** 22
- #define **NVIC_IRQNUM_EXTI9_5** 23
- #define **NVIC_IRQNUM_TIM1_BRK** 24
- #define **NVIC_IRQNUM_TIM1_UP** 25
- #define **NVIC_IRQNUM_TIM1_TRG_COM** 26
- #define **NVIC_IRQNUM_TIM1_CC** 27
- #define **NVIC_IRQNUM_TIM2** 28
- #define **NVIC_IRQNUM_TIM3** 29
- #define **NVIC_IRQNUM_TIM4** 30
- #define **NVIC_IRQNUM_I2C1_EV** 31
- #define **NVIC_IRQNUM_I2C1_ER** 32
- #define **NVIC_IRQNUM_I2C2_EV** 33
- #define **NVIC_IRQNUM_I2C2_ER** 34
- #define **NVIC_IRQNUM_SPI1** 35
- #define **NVIC_IRQNUM_SPI2** 36
- #define **NVIC_IRQNUM_USART1** 37
- #define **NVIC_IRQNUM_USART2** 38
- #define **NVIC_IRQNUM_USART3** 39
- #define **NVIC_IRQNUM_EXTI15_10** 40
- #define **NVIC_IRQNUM_RTC_ALARM** 41
- #define **NVIC_IRQNUM_USB_WAKE_UP** 42
- #define **NVIC_IRQNUM_TIM8_BRK** 43
- #define **NVIC_IRQNUM_TIM8_UP** 44
- #define **NVIC_IRQNUM_TIM8_TRG_COM** 45
- #define **NVIC_IRQNUM_TIM8_CC** 46
- #define **NVIC_IRQNUM_ADC3** 47
- #define **NVIC_IRQNUM_FSMC** 48
- #define **NVIC_IRQNUM_SDIO** 49
- #define **NVIC_IRQNUM_TIM5** 50
- #define **NVIC_IRQNUM_SPI3** 51
- #define **NVIC_IRQNUM_UART4** 52
- #define **NVIC_IRQNUM_UART5** 53
- #define **NVIC_IRQNUM_TIM6** 54
- #define **NVIC_IRQNUM_TIM7** 55
- #define **NVIC_IRQNUM_DMA2_Channel1** 56
- #define **NVIC_IRQNUM_DMA2_Channel2** 57
- #define **NVIC_IRQNUM_DMA2_Channel3** 58
- #define **NVIC_IRQNUM_DMA2_Channel4_5** 59

Functions

- Std_ReturnType Nvic_EnableInterrupt (uint8_t intNumber)

  Enables a specific Interrupt.
- Std_ReturnType Nvic_DisableInterrupt (uint8_t intNumber)

  Disables a specific Interrupt.
- Std_ReturnType Nvic_SetPending (uint8_t intNumber)

  Sets the pending flag for a specific interrupt.

- Std_ReturnType Nvic_ClearPending (uint8_t intNumber)

  Clears the pending flag for a specific interrupt.
- Std_ReturnType Nvic_IsInterruptActive (uint8_t ∗activeState, uint8_t intNumber)

  Checks if the interrupt is active.
- Std_ReturnType Nvic_SetSubpriority (uint8_t priority, uint8_t intNumber)

  Sets the subpriority for aspecific interrupt.
- Std_ReturnType Nvic_GetSubpriority (uint8_t ∗priority, uint8_t intNumber)

  Gets the subpriority for aspecific interrupt.
- Std_ReturnType Nvic_SetGroupPriority (uint8_t priority, uint8_t intNumber)

  Sets the group priority for aspecific interrupt.
- Std_ReturnType Nvic_GetGroupPriority (uint8_t ∗priority, uint8_t intNumber)

  Gets the group priority for aspecific interrupt.
- Std_ReturnType Nvic_ConfigGroupSize (void)

  Configure the group size.
- Std_ReturnType Nvic_ResetSystem (void)

  Resets the microcontroller.
- Std_ReturnType Nvic_EnablePeripheral (void)

  Enables the prepherals interrupts.
- Std_ReturnType Nvic_DisablePeripheral (void)

  Disable the prepherals interrupts.
- Std_ReturnType Nvic_SetFault (void)

  Blocks all interrupts including hard fault.
- Std_ReturnType Nvic_ClearFault (void)

  Returns from fault mode.
- Std_ReturnType Nvic_FilterPriority (uint8_t pri)

  Only allow interrupts over a certain priority.
- Std_ReturnType Nvic_GenerateSoftwareInterrupt (uint8_t intNumber)

  Generates a software interrupt (Atomic function to generate interrupt immediately)

4.7.1 Detailed Description

This file is a user interface for the NVIC driver for the arm cortex m3.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-02-29

Copyright

Copyright (c) 2020

4.7.2 Macro Definition Documentation

4.7.2.1 NVIC_IRQNUM_WWDG

```
#define NVIC_IRQNUM_WWDG 0
```

Interrupt Number

4.7.3 Function Documentation

4.7.3.1 Nvic_ClearFault()

```
Std_ReturnType Nvic_ClearFault (
             void  )
```

Returns from fault mode.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.2 Nvic_ClearPending()

```
Std_ReturnType Nvic_ClearPending (
             uint8_t intNumber )
```

Clears the pending flag for a specific interrupt.

Parameters

intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.3 Nvic_ConfigGroupSize()

```
Std_ReturnType Nvic_ConfigGroupSize (
            void )
```

Configure the group size.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.4 Nvic_DisableInterrupt()

```
Std_ReturnType Nvic_DisableInterrupt (
            uint8_t intNumber )
```

Disables a specific Interrupt.

Parameters

intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.5 Nvic_DisablePeripheral()

```
Std_ReturnType Nvic_DisablePeripheral (
            void )
```

Disable the prepherals interrupts.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.6 Nvic_EnableInterrupt()

```
Std_ReturnType Nvic_EnableInterrupt (
            uint8_t intNumber )
```

Enables a specific Interrupt.

Parameters

intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.7 Nvic_EnablePeripheral()

```
Std_ReturnType Nvic_EnablePeripheral (
            void  )
```

Enables the prepherals interrupts.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.8 Nvic_FilterPriority()

```
Std_ReturnType Nvic_FilterPriority (
            uint8_t pri )
```

Only allow interrupts over a certain priority.

Parameters

per	the minimum priority allowed

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.9 Nvic_GenerateSoftwareInterrupt()

```
Std_ReturnType Nvic_GenerateSoftwareInterrupt (
            uint8_t intNumber )
```

Generates a software interrupt (Atomic function to generate interrupt immediately)

Parameters

intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.10 Nvic_GetGroupPriority()

```
Std_ReturnType Nvic_GetGroupPriority (
            uint8_t * priority,
            uint8_t intNumber )
```

Gets the group priority for aspecific interrupt.

Parameters

priority	the priority you want to get
intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.11 Nvic_GetSubpriority()

```
Std_ReturnType Nvic_GetSubpriority (
            uint8_t * priority,
            uint8_t intNumber )
```

Gets the subpriority for aspecific interrupt.

Parameters

priority	the priority you want to get
intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.12 Nvic_IsInterruptActive()

```
Std_ReturnType Nvic_IsInterruptActive (
            uint8_t * activeState,
            uint8_t intNumber )
```

Checks if the interrupt is active.

Parameters

activeState	the active state of the interrupt • NVIC_ACTIVE • NVIC_NOT_ACTIVE
intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.13 Nvic_ResetSystem()

```
Std_ReturnType Nvic_ResetSystem (
            void )
```

Resets the microcontroller.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.14 Nvic_SetFault()

```
Std_ReturnType Nvic_SetFault (
            void )
```

Blocks all interrupts including hard fault.

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.15 Nvic_SetGroupPriority()

```
Std_ReturnType Nvic_SetGroupPriority (
            uint8_t priority,
            uint8_t intNumber )
```

Sets the group priority for aspecific interrupt.

Parameters

priority	the priority you want to set
intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.16 Nvic_SetPending()

```
Std_ReturnType Nvic_SetPending (
            uint8_t intNumber )
```

Sets the pending flag for a specific interrupt.

Parameters

intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.7.3.17 Nvic_SetSubpriority()

```
Std_ReturnType Nvic_SetSubpriority (
            uint8_t priority,
            uint8_t intNumber )
```

Sets the subpriority for aspecific interrupt.

Parameters

priority	the priority you want to set
intNumber	the number of the interrupt in the vector table • NVIC_IRQNUM_X

Returns

Std_ReturnType E_OK: If the function executed successfully E_NOT_OK: If the function failed to execute

4.8 Protocol.c File Reference

This is the implementation for a self designed software protocol.

```
#include "Std_Types.h"
#include "Uart.h"
#include "Protocol.h"
```

Data Structures

- struct header_t
- struct packet_t

Macros

- #define **PROTOCOL_HEADER_SIZE** 8
- #define **PROTOCOL_ACK** 0x55
- #define **PROTOCOL_ACK_SIZE** 1
- #define **PROTOCOL_DATA_KEY** 2
- #define **PROTOCOL_ADDRESS_KEY** 4
- #define **PROTOCOL_EOT_KEY** 6

Functions

- Std_ReturnType Protocol_Send (uint16_t msgType, uint16_t length, uint8_t ∗data)

  Sends a packet.
- Std_ReturnType Protocol_Receive (uint16_t ∗msgType, uint16_t ∗length, uint8_t ∗data)

  Sends a packet.
- Std_ReturnType Protocol_SendAck (void)

  Sends Acknowledgement.

4.8.1 Detailed Description

This is the implementation for a self designed software protocol.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-05-15

Copyright

Copyright (c) 2020

4.8.2 Function Documentation

4.8.2.1 Protocol_Receive()

```
Std_ReturnType Protocol_Receive (
            uint16_t * msgType,
            uint16_t * length,
            uint8_t * data )
```

Sends a packet.

Parameters

msgType	The type of the message • PROTOCOL_x
length	the length of the data in the message
data	the data to send

Returns

Std_ReturnType A Status E_OK : If the function was executed successfully E_NOT_OK : If the function didn't execute successfully

4.8.2.2 Protocol_Send()

```
Std_ReturnType Protocol_Send (
             uint16_t msgType,
             uint16_t length,
             uint8_t * data )
```

Sends a packet.

Parameters

msgType	The type of the message • PROTOCOL_x
length	the length of the data in the message
data	the data to send

Returns

> Std_ReturnType A Status E_OK : If the function was executed successfully E_NOT_OK : If the function didn't execute successfully

4.8.2.3 Protocol_SendAck()

```
Std_ReturnType Protocol_SendAck (
             void )
```

Sends Acknowledgement.

Returns

> Std_ReturnType A Status E_OK : If the function was executed successfully E_NOT_OK : If the function didn't execute successfully

4.9 Protocol.h File Reference

This is the user interface for the self designed software protocol.

Macros

- #define **PROTOCOL_DATA** 1
- #define **PROTOCOL_ADDRESS** 3
- #define **PROTOCOL_EOT** 5

Functions

- Std_ReturnType Protocol_Send (uint16_t msgType, uint16_t length, uint8_t ∗data)

  Sends a packet.
- Std_ReturnType Protocol_Receive (uint16_t ∗msgType, uint16_t ∗length, uint8_t ∗data)

  Sends a packet.
- Std_ReturnType Protocol_SendAck (void)

  Sends Acknowledgement.

4.9.1 Detailed Description

This is the user interface for the self designed software protocol.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-05-16

Copyright

Copyright (c) 2020

4.9.2 Function Documentation

4.9.2.1 Protocol_Receive()

```
Std_ReturnType Protocol_Receive (
            uint16_t ∗ msgType,
            uint16_t ∗ length,
            uint8_t ∗ data )
```

Sends a packet.

Parameters

msgType	The type of the message • PROTOCOL_x
length	the length of the data in the message
data	the data to send

Returns

Std_ReturnType A Status E_OK : If the function was executed successfully E_NOT_OK : If the function didn't execute successfully

4.9.2.2 Protocol_Send()

```
Std_ReturnType Protocol_Send (
            uint16_t msgType,
            uint16_t length,
            uint8_t * data )
```

Sends a packet.

Parameters

msgType	The type of the message - PROTOCOL_x
length	the length of the data in the message
data	the data to send

Returns

Std_ReturnType A Status E_OK : If the function was executed successfully E_NOT_OK : If the function didn't execute successfully

4.9.2.3 Protocol_SendAck()

```
Std_ReturnType Protocol_SendAck (
            void )
```

Sends Acknowledgement.

Returns

Std_ReturnType A Status E_OK : If the function was executed successfully E_NOT_OK : If the function didn't execute successfully

4.10 Rcc.c File Reference

This file is to be used as an implementation of the RCC driver.

```
#include "Std_Types.h"
#include "RCC.h"
```

Macros

- #define **RCC_BASE_ADDRESS** 0x40021000
- #define **RCC_CR** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x00))
- #define **RCC_CFGR** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x04))
- #define **RCC_CIR** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x08))
- #define **RCC_APB2RSTR** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x0C))
- #define **RCC_APB1RSTR** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x10))
- #define **RCC_AHBENR** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x14))
- #define **RCC_APB2ENR** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x18))
- #define **RCC_APB1ENR** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x1C))
- #define **RCC_BDCR** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x20))
- #define **RCC_CRS** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x24))
- #define **RCC_AHBRSTR** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x28))
- #define **RCC_CFGR2** ∗((volatile u32∗)(RCC_BASE_ADDRESS + 0x2C))
- #define **RCC_MCO_CLR** 0xF8FFFFFF
- #define **RCC_SYS_CLK_SELECT_CLR** 0xFFFFFFFC
- #define **RCC_PLL_MUL_CLR** 0xFFC3FFFF
- #define **RCC_PLL_SRC_CLR** 0xFFFEFFFF
- #define **RCC_SYS_CLK_STATUS** 0x0000000C

Functions

- Std_ReturnType Rcc_SetClockState (uint32_t clock, uint8_t state)

  Choose a specific clock and changes its state (On / Off)
- Std_ReturnType Rcc_IsClockReady (uint32_t clock, uint8_t ∗ready)

  Checks if a specific clock is ready or not.
- Std_ReturnType Rcc_SelectMcoClock (uint32_t clock)

  Selects the clock on the mco pin.
- Std_ReturnType Rcc_SetPrescaler (uint32_t clock, uint32_t value)

  Sets the prescaler value for a specific clock.
- Std_ReturnType Rcc_SetPllMultiplier (uint32_t pll)

  Sets the PLL Multiplication factor.
- Std_ReturnType Rcc_SetPllSource (uint32_t source)

  Chooses the PLL clock source.
- Std_ReturnType Rcc_GetSystemClockStatus (uint8_t ∗sysClk)

  Which clock is working as system clock.
- Std_ReturnType Rcc_SwitchSystemClock (uint32_t clock)

  Switches the system clock (HSI / HSE / PLL)
- Std_ReturnType Rcc_SetApb2PeriphClockState (uint32_t periph, uint8_t state)

  Choose a specific peripheral on the APB2 bus and changes its state (On / Off)
- Std_ReturnType Rcc_ResetApb2Periph (uint32_t periph)

  Resets a specific peripheral on the APB2 bus.
- Std_ReturnType Rcc_SetApb1PeriphClockState (uint32_t periph, uint8_t state)

  Choose a specific peripheral on the APB1 bus and changes its state (On / Off)
- Std_ReturnType Rcc_ResetApb1Periph (uint32_t periph)

  Resets a specific peripheral on the APB1 bus.
- Std_ReturnType Rcc_SetAhbPeriphClockState (uint32_t periph, uint8_t state)

  Choose a specific peripheral on the AHB bus and changes its state (On / Off)
- Std_ReturnType Rcc_ResetAhbPeriph (uint32_t periph)

  Resets a specific peripheral on the AHB bus.

4.10.1 Detailed Description

This file is to be used as an implementation of the RCC driver.

Author

Mark Attia

Date

January 22, 2020

4.10.2 Function Documentation

4.10.2.1 Rcc_GetSystemClockStatus()

```
Std_ReturnType Rcc_GetSystemClockStatus (
            uint8_t * sysClk )
```

Which clock is working as system clock.

4.10.2.2 Function: Rcc_GetSystemClockStatus

Parameters

sysClk	Saves the clock that is working as system clock in • RCC_HSI_SYS : High speed internal clock is used as system clock • RCC_HSE_SYS : High speed external clock is used as system clock • RCC_PLL_SYS : PLL clock is used as system clock
returns	A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.3 Rcc_IsClockReady()

```
Std_ReturnType Rcc_IsClockReady (
            uint32_t clock,
            uint8_t * ready )
```

Checks if a specific clock is ready or not.

4.10.2.4 Function: Rcc_IsReady

Parameters

clock	The clock you want to check for - RCC_HSI_RDY: for the high speed internal clock - RCC_HSE_RDY: for the high speed external clock - RCC_PLL_RDY: for the PLL clock
ready	Saves the ready state in - RCC_IS_RDY : if the clock is ready - RCC_NOT_RDY : if the clock is not ready

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.5 Rcc_ResetAhbPeriph()

```
Std_ReturnType Rcc_ResetAhbPeriph (
            uint32_t periph )
```

Resets a specific peripheral on the AHB bus.

4.10.2.6 Function: Rcc_ResetAhbPeriph

Parameters

periph	The peripheral you want to reset - RCC_OTGFS_RST: OTGFS reset - RCC_ETHMAC_RST: Ethernet MAC reset

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.7 Rcc_ResetApb1Periph()

```
Std_ReturnType Rcc_ResetApb1Periph (
            uint32_t periph )
```

Resets a specific peripheral on the APB1 bus.

4.10.2.8 Function: Rcc_ResetApb1Periph

Parameters

periph	The peripheral you want to reset • RCC_TIM2_RST: Timer 2 reset • RCC_TIM3_RST: Timer 3 reset • RCC_TIM4_RST: Timer 4 reset • RCC_TIM5_RST: Timer 5 reset • RCC_TIM6_RST: Timer 6 reset • RCC_TIM7_RST: Timer 7 reset • RCC_TIM12_RST: Timer 12 reset • RCC_TIM13_RST: Timer 13 reset • RCC_TIM14_RST: Timer 14 reset • RCC_WWD_GEN_RST: Window watchdog reset • RCC_SPI2_RST: SPI 2 reset • RCC_SPI3_RST: SPI 3 reset • RCC_USART2_RST: USART 2 reset • RCC_USART3_RST: USART 3 reset • RCC_USART4_RST: USART 4 reset • RCC_USART5_RST: USART 5 reset • RCC_I2C1_RST: I2C 1 reset • RCC_I2C2_RST: I2C 2 reset • RCC_USB_RST: USB reset • RCC_CAN_RST: CAN reset • RCC_BKP_RST: Backup interface reset • RCC_PWR_RST: Power interface reset • RCC_DAC_RST: DAC interface reset

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.9 Rcc_ResetApb2Periph()

```
Std_ReturnType Rcc_ResetApb2Periph (
            uint32_t periph )
```

Resets a specific peripheral on the APB2 bus.

4.10.2.10 Function: Rcc_ResetApb2Periph

Parameters

periph	The peripheral you want to reset • RCC_AFIO_RST: Alternate function input output reset • RCC_IOPA_RST: Port A input output reset • RCC_IOPB_RST: Port B input output reset • RCC_IOPC_RST: Port C input output reset • RCC_IOPD_RST: Port D input output reset • RCC_IOPE_RST: Port E input output reset • RCC_IOPF_RST: Port F input output reset • RCC_IOPG_RST: Port G input output reset • RCC_ADC1_RST: ADC 1 reset • RCC_ADC2_RST: ADC 2 reset • RCC_TIM1_RST: Timer 1 reset • RCC_SPI1_RST: SPI 1 reset • RCC_TIM8_RST: Timer 8 reset • RCC_USART1_RST: USART 1 reset • RCC_ADC3_RST: ADC 3 reset • RCC_TIM9_RST: Timer 9 reset • RCC_TIM10_RSTN: Timer 10 reset • RCC_TIM11_RSTN: Timer 11 reset

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.11 Rcc_SelectMcoClock()

```
Std_ReturnType Rcc_SelectMcoClock (
             uint32_t clock )
```

Selects the clock on the mco pin.

4.10.2.12 Function: Rcc_SelectMcoClock

Parameters

clock	The clock you want to configure • RCC_MCO_NO_CLK : No clock will be on MCO • RCC_MCO_SYS_CLK : Select system clock on the MCO • RCC_MCO_HSI_CLK : Select high speed internal clock on the MCO • RCC_MCO_HSE_CLK : Select high speed external clock on the MCO • RCC_MCO_PLL_CLK : Select PLL clock on the MCO

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.13 Rcc_SetAhbPeriphClockState()

```
Std_ReturnType Rcc_SetAhbPeriphClockState (
            uint32_t periph,
            uint8_t state )
```

Choose a specific peripheral on the AHB bus and changes its state (On / Off)

4.10.2.14 Function: Rcc_SetAhbPeriphClockState

Parameters

periph	The peripheral clock you want to configure • RCC_DMA1_CLK_EN: DMA 1 clock enable • RCC_DMA2_CLK_EN: DMA 2 clock enable • RCC_SRAM_CLK_EN: SRAM interface clock enable • RCC_FLITF_CLK_EN: FLITF clock enable • RCC_CRC_CLK_EN: CRC clock enable • RCC_OTGFS_CLK_EN: OTGFS clock enable • RCC_ETHMAC_CLK_EN: Ethernet MAC clock enable • RCC_ETHMACTX_CLK_EN: Ethernet MAC TX clock enable • RCC_ETHMACRX_CLK_EN: Ethernet MAC RX clock enable
state	: The state of the clock (On / Off) • RCC_PERIPH_CLK_ON: To set the clock on • RCC_PERIPH_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.15 Rcc_SetApb1PeriphClockState()

```
Std_ReturnType Rcc_SetApb1PeriphClockState (
            uint32_t periph,
            uint8_t state )
```

Choose a specific peripheral on the APB1 bus and changes its state (On / Off)

4.10.2.16 Function: Rcc_SetApb1PeriphClockState

Parameters

periph	The peripheral clock you want to configure • RCC_TIM2_CLK_EN: Timer 2 clock enable • RCC_TIM3_CLK_EN: Timer 3 clock enable • RCC_TIM4_CLK_EN: Timer 4 clock enable • RCC_TIM5_CLK_EN: Timer 5 clock enable • RCC_TIM6_CLK_EN: Timer 6 clock enable • RCC_TIM7_CLK_EN: Timer 7 clock enable • RCC_TIM12_CLK_EN: Timer 12 clock enable • RCC_TIM13_CLK_EN: Timer 13 clock enable • RCC_TIM14_CLK_EN: Timer 14 clock enable • RCC_WWD_GEN_CLK_EN: Window watchdog clock enable • RCC_SPI2_CLK_EN: SPI 2 clock enable • RCC_SPI3_CLK_EN: SPI 3 clock enable • RCC_USART2_CLK_EN: USART 2 clock enable • RCC_USART3_CLK_EN: USART 3 clock enable • RCC_USART4_CLK_EN: USART 4 clock enable • RCC_USART5_CLK_EN: USART 5 clock enable • RCC_I2C1_CLK_EN: I2C 1 clock enable • RCC_I2C2_CLK_EN: I2C 2 clock enable • RCC_USB_CLK_EN: USB clock enable • RCC_CAN_CLK_EN: CAN clock enable • RCC_BKP_CLK_EN: Backup interface clock enable • RCC_PWR_CLK_EN: Power interface clock enable • RCC_DAC_CLK_EN: DAC interface clock enable
state	The state of the clock (On / Off) • RCC_PERIPH_CLK_ON : To set the clock on • RCC_PERIPH_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.17 Rcc_SetApb2PeriphClockState()

```
Std_ReturnType Rcc_SetApb2PeriphClockState (
            uint32_t periph,
            uint8_t state )
```

Choose a specific peripheral on the APB2 bus and changes its state (On / Off)

4.10.2.18 Function: Rcc_SetApb2PeriphClockState

Parameters

periph	The peripheral clock you want to configure • RCC_AFIO_CLK_EN: Alternate function input output clock enable • RCC_IOPA_CLK_EN: Port A input output clock enable • RCC_IOPB_CLK_EN: Port B input output clock enable • RCC_IOPC_CLK_EN: Port C input output clock enable • RCC_IOPD_CLK_EN: Port D input output clock enable • RCC_IOPE_CLK_EN: Port E input output clock enable • RCC_IOPF_CLK_EN: Port F input output clock enable • RCC_IOPG_CLK_EN: Port G input output clock enable • RCC_ADC1_CLK_EN: ADC 1 clock enable • RCC_ADC2_CLK_EN: ADC 2 clock enable • RCC_TIM1_CLK_EN: Timer 1 clock enable • RCC_SPI1_CLK_EN: SPI 1 clock enable • RCC_TIM8_CLK_EN: Timer 8 clock enable • RCC_USART1_CLK_EN: USART 1 clock enable • RCC_ADC3_CLK_EN: ADC 3 clock enable • RCC_TIM9_CLK_EN: Timer 9 clock enable • RCC_TIM10_CLK_EN: Timer 10 clock enable • RCC_TIM11_CLK_EN: Timer 11 clock enable
state	The state of the clock (On / Off) • RCC_PERIPH_CLK_ON : To set the clock on • RCC_PERIPH_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.19 Rcc_SetClockState()

```
Std_ReturnType Rcc_SetClockState (
            uint32_t clock,
            uint8_t state )
```

Choose a specific clock and changes its state (On / Off)

4.10.2.20 Function: Rcc_SetClockState

Parameters

clock	The clock you want to configure • RCC_HSI_SET: for the high speed internal clock • RCC_HSE_SET: for the high speed external clock • RCC_PLL_SET: for the PLL clock
state	: The state of the clock (On / Off) • RCC_CLK_ON : To set the clock on • RCC_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.21 Rcc_SetPllMultiplier()

```
Std_ReturnType Rcc_SetPllMultiplier (
            uint32_t pll )
```

Sets the PLL Multiplication factor.

4.10.2.22 Function: Rcc_SetPllMultiplier

Parameters

pll	: The PLL multiplication factor • RCC_PLL_MUL_XX : Set PLL multiplication factor

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.23 Rcc_SetPllSource()

```
Std_ReturnType Rcc_SetPllSource (
             uint32_t source )
```

Chooses the PLL clock source.

4.10.2.24 Function: Rcc_SetPllSource

Parameters

source	: The PLL clock source • RCC_PLL_SRC_HSI : Choose high speed internal clock / 2 as a PLL clock source • RCC_PLL_SRC_HSE : Choose high speed external clock as a PLL clock source

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.25 Rcc_SetPrescaler()

```
Std_ReturnType Rcc_SetPrescaler (
             uint32_t clock,
             uint32_t value )
```

Sets the prescaler value for a specific clock.

4.10.2.26 Function: Rcc_SetPrescaler

Parameters

clock	The clock you want to configure • RCC_USB_PRE : For the USB prescaler • RCC_PLL_HSE_PRE : For the Pll prescaler • RCC_ADC_PRE : For the ADC prescaler • RCC_APB2_PRE : For the APB2 prescaler • RCC_APB1_PRE : For the APB1 prescaler • RCC_AHB_PRE : For the AHB prescaler

Parameters

value	: The state of the clock (On / Off) • RCC_USB_PRE_1_5 : No USB prescaler value • RCC_USB_PRE_1_5 : USB prescaler 1.5 • RCC_PLL_HSE_PRE_X : PLL Prescaler value using high speed external clock • RCC_ADC_PRE_X : ADC Prescaler value • RCC_APB2_PRE_XX : APB2 prescater value • RCC_APB1_PRE_XX : APB1 prescater value • RCC_AHB_PRE_XXX : AHB prescater value

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.10.2.27 Rcc_SwitchSystemClock()

```
Std_ReturnType Rcc_SwitchSystemClock (
            uint32_t clock )
```

Switches the system clock (HSI / HSE / PLL)

4.10.2.28 Function: Rcc_SwitchSystemClock

Parameters

clock	: The PLL clock source • RCC_SYS_CLK_SELECT_HSI : Select high speed internal clock as system clock • RCC_SYS_CLK_SELECT_HSE : Select high speed external clock as system clock • RCC_SYS_CLK_SELECT_PLL : Select PLL clock as system clock

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11 Rcc.h File Reference

This file is to be used as an interface for the user of RCC driver.

Macros

- #define **RCC_YES** 4
- #define **RCC_NO** 5
- #define **RCC_CLK_ON** 6
- #define **RCC_CLK_OFF** 7
- #define **RCC_IS_RDY** 0
- #define **RCC_NOT_RDY** !RCC_IS_RDY
- #define **RCC_HSI_SET** 0x00000001
- #define **RCC_HSE_SET** 0x00010000
- #define **RCC_PLL_SET** 0x01000000
- #define **RCC_HSI_RDY** 0xFFFFFFFD
- #define **RCC_HSE_RDY** 0xFFFDFFFF
- #define **RCC_PLL_RDY** 0xFDFFFFFF
- #define **RCC_MCO_NO_CLK** 0x00000000
- #define **RCC_MCO_SYS_CLK** 0x04000000
- #define **RCC_MCO_HSI_CLK** 0x05000000
- #define **RCC_MCO_HSE_CLK** 0x06000000
- #define **RCC_MCO_PLL_CLK** 0x07000000
- #define **RCC_PLL_SRC_HSI** 0x00000000
- #define **RCC_PLL_SRC_HSE** 0x00010000
- #define **RCC_USB_PRE** 0xFFBFFFFF
- #define **RCC_PLL_HSE_PRE** 0xFFFDFFFF
- #define **RCC_ADC_PRE** 0xFFFF3FFF
- #define **RCC_APB2_PRE** 0xFFFFC7FF
- #define **RCC_APB1_PRE** 0xFFFFF8FF
- #define **RCC_AHB_PRE** 0xFFFFFF0F
- #define **RCC_USB_PRE_1_5** 0x00000000
- #define **RCC_USB_PRE_0** 0x00400000
- #define **RCC_PLL_MUL_02** 0x00000000
- #define **RCC_PLL_MUL_03** 0x00040000
- #define **RCC_PLL_MUL_04** 0x00080000
- #define **RCC_PLL_MUL_05** 0x000C0000
- #define **RCC_PLL_MUL_06** 0x00100000
- #define **RCC_PLL_MUL_07** 0x00140000
- #define **RCC_PLL_MUL_08** 0x00180000
- #define **RCC_PLL_MUL_09** 0x001C0000
- #define **RCC_PLL_MUL_10** 0x00200000
- #define **RCC_PLL_MUL_11** 0x00240000
- #define **RCC_PLL_MUL_12** 0x00280000
- #define **RCC_PLL_MUL_13** 0x002C0000
- #define **RCC_PLL_MUL_14** 0x00300000
- #define **RCC_PLL_MUL_15** 0x00340000
- #define **RCC_PLL_MUL_16** 0x00380000
- #define **RCC_PLL_HSE_PRE_0** 0x00000000
- #define **RCC_PLL_HSE_PRE_2** 0x00020000
- #define **RCC_ADC_PRE_2** 0x00000000
- #define **RCC_ADC_PRE_4** 0x00004000
- #define **RCC_ADC_PRE_6** 0x00008000
- #define **RCC_ADC_PRE_8** 0x0000C000
- #define **RCC_APB2_PRE_00** 0x00000000
- #define **RCC_APB2_PRE_02** 0x00002000
- #define **RCC_APB2_PRE_04** 0x00002800
- #define **RCC_APB2_PRE_08** 0x00003000
- #define **RCC_APB2_PRE_16** 0x00003800

- #define **RCC_APB1_PRE_00** 0x00000000
- #define **RCC_APB1_PRE_02** 0x00000400
- #define **RCC_APB1_PRE_04** 0x00000500
- #define **RCC_APB1_PRE_08** 0x00000600
- #define **RCC_APB1_PRE_16** 0x00000700
- #define **RCC_AHB_PRE_000** 0x00000000
- #define **RCC_AHB_PRE_002** 0x00000080
- #define **RCC_AHB_PRE_004** 0x00000090
- #define **RCC_AHB_PRE_008** 0x000000A0
- #define **RCC_AHB_PRE_016** 0x000000B0
- #define **RCC_AHB_PRE_064** 0x000000C0
- #define **RCC_AHB_PRE_128** 0x000000D0
- #define **RCC_AHB_PRE_256** 0x000000E0
- #define **RCC_AHB_PRE_512** 0x000000F0
- #define **RCC_HSI_SYS** 0x00000000
- #define **RCC_HSE_SYS** 0x00000001
- #define **RCC_PLL_SYS** 0x00000002
- #define **RCC_SYS_CLK_SELECT_HSI** 0x00000000
- #define **RCC_SYS_CLK_SELECT_HSE** 0x00000001
- #define **RCC_SYS_CLK_SELECT_PLL** 0x00000002
- #define **RCC_PERIPH_CLK_ON** 0
- #define **RCC_PERIPH_CLK_OFF** 1
- #define **RCC_AFIO_CLK_EN** 0x00000001
- #define **RCC_IOPA_CLK_EN** 0x00000004
- #define **RCC_IOPB_CLK_EN** 0x00000008
- #define **RCC_IOPC_CLK_EN** 0x00000010
- #define **RCC_IOPD_CLK_EN** 0x00000020
- #define **RCC_IOPE_CLK_EN** 0x00000040
- #define **RCC_IOPF_CLK_EN** 0x00000080
- #define **RCC_IOPG_CLK_EN** 0x00000100
- #define **RCC_ADC1_CLK_EN** 0x00000200
- #define **RCC_ADC2_CLK_EN** 0x00000400
- #define **RCC_TIM1_CLK_EN** 0x00000800
- #define **RCC_SPI1_CLK_EN** 0x00001000
- #define **RCC_TIM8_CLK_EN** 0x00002000
- #define **RCC_USART1_CLK_EN** 0x00004000
- #define **RCC_ADC3_CLK_EN** 0x00008000
- #define **RCC_TIM9_CLK_EN** 0x00080000
- #define **RCC_TIM10_CLK_EN** 0x00100000
- #define **RCC_TIM11_CLK_EN** 0x00200000
- #define **RCC_AFIO_RST** 0x00000001
- #define **RCC_IOPA_RST** 0x00000004
- #define **RCC_IOPB_RST** 0x00000008
- #define **RCC_IOPC_RST** 0x00000010
- #define **RCC_IOPD_RST** 0x00000020
- #define **RCC_IOPE_RST** 0x00000040
- #define **RCC_IOPF_RST** 0x00000080
- #define **RCC_IOPG_RST** 0x00000100
- #define **RCC_ADC1_RST** 0x00000200
- #define **RCC_ADC2_RST** 0x00000400
- #define **RCC_TIM1_RST** 0x00000800
- #define **RCC_SPI1_RST** 0x00001000
- #define **RCC_TIM8_RST** 0x00002000
- #define **RCC_USART1_RST** 0x00004000
- #define **RCC_ADC3_RST** 0x00008000

- #define **RCC_TIM9_RST** 0x00080000
- #define **RCC_TIM10_RST** 0x00100000
- #define **RCC_TIM11_RST** 0x00200000
- #define **RCC_TIM2_CLK_EN** 0x00000001
- #define **RCC_TIM3_CLK_EN** 0x00000002
- #define **RCC_TIM4_CLK_EN** 0x00000004
- #define **RCC_TIM5_CLK_EN** 0x00000008
- #define **RCC_TIM6_CLK_EN** 0x00000010
- #define **RCC_TIM7_CLK_EN** 0x00000020
- #define **RCC_TIM12_CLK_EN** 0x00000040
- #define **RCC_TIM13_CLK_EN** 0x00000080
- #define **RCC_TIM14_CLK_EN** 0x00000100
- #define **RCC_WWD_GEN_CLK_EN** 0x00000800
- #define **RCC_SPI2_CLK_EN** 0x00004000
- #define **RCC_SPI3_CLK_EN** 0x00008000
- #define **RCC_USART2_CLK_EN** 0x00020000
- #define **RCC_USART3_CLK_EN** 0x00040000
- #define **RCC_USART4_CLK_EN** 0x00080000
- #define **RCC_USART5_CLK_EN** 0x00100000
- #define **RCC_I2C1_CLK_EN** 0x00200000
- #define **RCC_I2C2_CLK_EN** 0x00400000
- #define **RCC_USB_CLK_EN** 0x00800000
- #define **RCC_CAN_CLK_EN** 0x02000000
- #define **RCC_BKP_CLK_EN** 0x08000000
- #define **RCC_PWR_CLK_EN** 0x10000000
- #define **RCC_DAC_CLK_EN** 0x20000000
- #define **RCC_TIM2_RST** 0x00000001
- #define **RCC_TIM3_RST** 0x00000002
- #define **RCC_TIM4_RST** 0x00000004
- #define **RCC_TIM5_RST** 0x00000008
- #define **RCC_TIM6_RST** 0x00000010
- #define **RCC_TIM7_RST** 0x00000020
- #define **RCC_TIM12_RST** 0x00000040
- #define **RCC_TIM13_RST** 0x00000080
- #define **RCC_TIM14_RST** 0x00000100
- #define **RCC_WWD_GEN_RST** 0x00000800
- #define **RCC_SPI2_RST** 0x00004000
- #define **RCC_SPI3_RST** 0x00008000
- #define **RCC_USART2_RST** 0x00020000
- #define **RCC_USART3_RST** 0x00040000
- #define **RCC_USART4_RST** 0x00080000
- #define **RCC_USART5_RST** 0x00100000
- #define **RCC_I2C1_RST** 0x00200000
- #define **RCC_I2C2_RST** 0x00400000
- #define **RCC_USB_RST** 0x00800000
- #define **RCC_CAN_RST** 0x02000000
- #define **RCC_BKP_RST** 0x08000000
- #define **RCC_PWR_RST** 0x10000000
- #define **RCC_DAC_RST** 0x20000000
- #define **RCC_DMA1_CLK_EN** 0x00000001
- #define **RCC_DMA2_CLK_EN** 0x00000002
- #define **RCC_SRAM_CLK_EN** 0x00000004
- #define **RCC_FLITF_CLK_EN** 0x00000010
- #define **RCC_CRC_CLK_EN** 0x00000040
- #define **RCC_OTGFS_CLK_EN** 0x00001000

- #define **RCC_ETHMAC_CLK_EN** 0x00004000
- #define **RCC_ETHMACTX_CLK_EN** 0x00008000
- #define **RCC_ETHMACRX_CLK_EN** 0x00010000
- #define **RCC_OTGFS_RST** 0x00001000
- #define **RCC_ETHMAC_RST** 0x00004000

Functions

- Std_ReturnType Rcc_SetClockState (uint32_t clock, uint8_t state)

  Choose a specific clock and changes its state (On / Off)
- Std_ReturnType Rcc_IsClockReady (uint32_t clock, uint8_t ∗ready)

  Checks if a specific clock is ready or not.
- Std_ReturnType Rcc_SelectMcoClock (uint32_t clock)

  Selects the clock on the mco pin.
- Std_ReturnType Rcc_SetPrescaler (uint32_t clock, uint32_t value)

  Sets the prescaler value for a specific clock.
- Std_ReturnType Rcc_SetPllMultiplier (uint32_t pll)

  Sets the PLL Multiplication factor.
- Std_ReturnType Rcc_SetPllSource (uint32_t source)

  Chooses the PLL clock source.
- Std_ReturnType Rcc_GetSystemClockStatus (uint8_t ∗sysClk)

  Which clock is working as system clock.
- Std_ReturnType Rcc_SwitchSystemClock (uint32_t clock)

  Switches the system clock (HSI / HSE / PLL)
- Std_ReturnType Rcc_SetApb2PeriphClockState (uint32_t periph, uint8_t state)

  Choose a specific peripheral on the APB2 bus and changes its state (On / Off)
- Std_ReturnType Rcc_ResetApb2Periph (uint32_t periph)

  Resets a specific peripheral on the APB2 bus.
- Std_ReturnType Rcc_SetApb1PeriphClockState (uint32_t periph, uint8_t state)

  Choose a specific peripheral on the APB1 bus and changes its state (On / Off)
- Std_ReturnType Rcc_ResetApb1Periph (uint32_t periph)

  Resets a specific peripheral on the APB1 bus.
- Std_ReturnType Rcc_SetAhbPeriphClockState (uint32_t periph, uint8_t state)

  Choose a specific peripheral on the AHB bus and changes its state (On / Off)
- Std_ReturnType Rcc_ResetAhbPeriph (uint32_t periph)

  Resets a specific peripheral on the AHB bus.

4.11.1 Detailed Description

This file is to be used as an interface for the user of RCC driver.

Author

Mark Attia

Date

January 22, 2020

4.11.2 Function Documentation

4.11.2.1 Rcc_GetSystemClockStatus()

```
Std_ReturnType Rcc_GetSystemClockStatus (
             uint8_t * sysClk )
```

Which clock is working as system clock.

4.11.2.2 Function: Rcc_GetSystemClockStatus

Parameters

sysClk	Saves the clock that is working as system clock in • RCC_HSI_SYS : High speed internal clock is used as system clock • RCC_HSE_SYS : High speed external clock is used as system clock • RCC_PLL_SYS : PLL clock is used as system clock
returns	A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.3 Function: Rcc_GetSystemClockStatus

Parameters

sysClk	Saves the clock that is working as system clock in • RCC_HSI_SYS : High speed internal clock is used as system clock • RCC_HSE_SYS : High speed external clock is used as system clock • RCC_PLL_SYS : PLL clock is used as system clock
returns	A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.4 Rcc_IsClockReady()

```
Std_ReturnType Rcc_IsClockReady (
             uint32_t clock,
             uint8_t * ready )
```

Checks if a specific clock is ready or not.

4.11.2.5 Function: Rcc_IsReady

Parameters

clock	The clock you want to check for • RCC_HSI_RDY: for the high speed internal clock • RCC_HSE_RDY: for the high speed external clock • RCC_PLL_RDY: for the PLL clock
ready	Saves the ready state in • RCC_IS_RDY : if the clock is ready • RCC_NOT_RDY : if the clock is not ready

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.6 Function: Rcc_IsReady

Parameters

clock	The clock you want to check for • RCC_HSI_RDY: for the high speed internal clock • RCC_HSE_RDY: for the high speed external clock • RCC_PLL_RDY: for the PLL clock
ready	Saves the ready state in • RCC_IS_RDY : if the clock is ready • RCC_NOT_RDY : if the clock is not ready

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.7 Rcc_ResetAhbPeriph()

```
Std_ReturnType Rcc_ResetAhbPeriph (
             uint32_t periph )
```

Resets a specific peripheral on the AHB bus.

4.11.2.8 Function: Rcc_ResetAhbPeriph

Parameters

periph	The peripheral you want to reset • RCC_OTGFS_RST: OTGFS reset • RCC_ETHMAC_RST: Ethernet MAC reset

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.9 Function: Rcc_ResetAhbPeriph

Parameters

periph	The peripheral you want to reset • RCC_OTGFS_RST: OTGFS reset • RCC_ETHMAC_RST: Ethernet MAC reset

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.10 Rcc_ResetApb1Periph()

```
Std_ReturnType Rcc_ResetApb1Periph (
             uint32_t periph )
```

Resets a specific peripheral on the APB1 bus.

4.11.2.11 Function: Rcc_ResetApb1Periph

Parameters

periph	The peripheral you want to reset • RCC_TIM2_RST: Timer 2 reset • RCC_TIM3_RST: Timer 3 reset • RCC_TIM4_RST: Timer 4 reset • RCC_TIM5_RST: Timer 5 reset • RCC_TIM6_RST: Timer 6 reset • RCC_TIM7_RST: Timer 7 reset • RCC_TIM12_RST: Timer 12 reset • RCC_TIM13_RST: Timer 13 reset • RCC_TIM14_RST: Timer 14 reset • RCC_WWD_GEN_RST: Window watchdog reset • RCC_SPI2_RST: SPI 2 reset • RCC_SPI3_RST: SPI 3 reset • RCC_USART2_RST: USART 2 reset • RCC_USART3_RST: USART 3 reset • RCC_USART4_RST: USART 4 reset • RCC_USART5_RST: USART 5 reset • RCC_I2C1_RST: I2C 1 reset • RCC_I2C2_RST: I2C 2 reset • RCC_USB_RST: USB reset • RCC_CAN_RST: CAN reset • RCC_BKP_RST: Backup interface reset • RCC_PWR_RST: Power interface reset • RCC_DAC_RST: DAC interface reset

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.12 Function: Rcc_ResetApb1Periph

Parameters

periph	The peripheral you want to reset • RCC_TIM2_RST: Timer 2 reset • RCC_TIM3_RST: Timer 3 reset • RCC_TIM4_RST: Timer 4 reset • RCC_TIM5_RST: Timer 5 reset • RCC_TIM6_RST: Timer 6 reset • RCC_TIM7_RST: Timer 7 reset • RCC_TIM12_RST: Timer 12 reset • RCC_TIM13_RST: Timer 13 reset • RCC_TIM14_RST: Timer 14 reset • RCC_WWD_GEN_RST: Window watchdog reset • RCC_SPI2_RST: SPI 2 reset • RCC_SPI3_RST: SPI 3 reset • RCC_USART2_RST: USART 2 reset • RCC_USART3_RST: USART 3 reset • RCC_USART4_RST: USART 4 reset • RCC_USART5_RST: USART 5 reset • RCC_I2C1_RST: I2C 1 reset • RCC_I2C2_RST: I2C 2 reset • RCC_USB_RST: USB reset • RCC_CAN_RST: CAN reset • RCC_BKP_RST: Backup interface reset • RCC_PWR_RST: Power interface reset • RCC_DAC_RST: DAC interface reset

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.13 Rcc_ResetApb2Periph()

```
Std_ReturnType Rcc_ResetApb2Periph (
            uint32_t periph )
```

Resets a specific peripheral on the APB2 bus.

4.11.2.14 Function: Rcc_ResetApb2Periph

Parameters

periph	The peripheral you want to reset • RCC_AFIO_RST: Alternate function input output reset • RCC_IOPA_RST: Port A input output reset • RCC_IOPB_RST: Port B input output reset • RCC_IOPC_RST: Port C input output reset • RCC_IOPD_RST: Port D input output reset • RCC_IOPE_RST: Port E input output reset • RCC_IOPF_RST: Port F input output reset • RCC_IOPG_RST: Port G input output reset • RCC_ADC1_RST: ADC 1 reset • RCC_ADC2_RST: ADC 2 reset • RCC_TIM1_RST: Timer 1 reset • RCC_SPI1_RST: SPI 1 reset • RCC_TIM8_RST: Timer 8 reset • RCC_USART1_RST: USART 1 reset • RCC_ADC3_RST: ADC 3 reset • RCC_TIM9_RST: Timer 9 reset • RCC_TIM10_RSTN: Timer 10 reset • RCC_TIM11_RSTN: Timer 11 reset

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.15 Function: Rcc_ResetApb2Periph

Parameters

periph	The peripheral you want to reset • RCC_AFIO_RST: Alternate function input output reset • RCC_IOPA_RST: Port A input output reset • RCC_IOPB_RST: Port B input output reset • RCC_IOPC_RST: Port C input output reset • RCC_IOPD_RST: Port D input output reset • RCC_IOPE_RST: Port E input output reset • RCC_IOPF_RST: Port F input output reset • RCC_IOPG_RST: Port G input output reset • RCC_ADC1_RST: ADC 1 reset • RCC_ADC2_RST: ADC 2 reset • RCC_TIM1_RST: Timer 1 reset • RCC_SPI1_RST: SPI 1 reset • RCC_TIM8_RST: Timer 8 reset • RCC_USART1_RST: USART 1 reset • RCC_ADC3_RST: ADC 3 reset • RCC_TIM9_RST: Timer 9 reset • RCC_TIM10_RSTN: Timer 10 reset • RCC_TIM11_RSTN: Timer 11 reset

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.16 Rcc_SelectMcoClock()

```
Std_ReturnType Rcc_SelectMcoClock (
            uint32_t clock )
```

Selects the clock on the mco pin.

4.11.2.17 Function: Rcc_SelectMcoClock

Parameters

clock	The clock you want to configure • RCC_MCO_NO_CLK : No clock will be on MCO • RCC_MCO_SYS_CLK : Select system clock on the MCO • RCC_MCO_HSI_CLK : Select high speed internal clock on the MCO • RCC_MCO_HSE_CLK : Select high speed external clock on the MCO • RCC_MCO_PLL_CLK : Select PLL clock on the MCO

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.18 Function: Rcc_SelectMcoClock

Parameters

clock	The clock you want to configure • RCC_MCO_NO_CLK : No clock will be on MCO • RCC_MCO_SYS_CLK : Select system clock on the MCO • RCC_MCO_HSI_CLK : Select high speed internal clock on the MCO • RCC_MCO_HSE_CLK : Select high speed external clock on the MCO • RCC_MCO_PLL_CLK : Select PLL clock on the MCO

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.19 Rcc_SetAhbPeriphClockState()

```
Std_ReturnType Rcc_SetAhbPeriphClockState (
            uint32_t periph,
            uint8_t state )
```

Choose a specific peripheral on the AHB bus and changes its state (On / Off)

4.11.2.20 Function: Rcc_SetAhbPeriphClockState

Parameters

periph	The peripheral clock you want to configure • RCC_DMA1_CLK_EN: DMA 1 clock enable • RCC_DMA2_CLK_EN: DMA 2 clock enable • RCC_SRAM_CLK_EN: SRAM interface clock enable • RCC_FLITF_CLK_EN: FLITF clock enable • RCC_CRC_CLK_EN: CRC clock enable • RCC_OTGFS_CLK_EN: OTGFS clock enable • RCC_ETHMAC_CLK_EN: Ethernet MAC clock enable • RCC_ETHMACTX_CLK_EN: Ethernet MAC TX clock enable • RCC_ETHMACRX_CLK_EN: Ethernet MAC RX clock enable
state	: The state of the clock (On / Off) • RCC_PERIPH_CLK_ON: To set the clock on • RCC_PERIPH_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.21 Function: Rcc_SetAhbPeriphClockState

Parameters

periph	The peripheral clock you want to configure • RCC_DMA1_CLK_EN: DMA 1 clock enable • RCC_DMA2_CLK_EN: DMA 2 clock enable • RCC_SRAM_CLK_EN: SRAM interface clock enable • RCC_FLITF_CLK_EN: FLITF clock enable • RCC_CRC_CLK_EN: CRC clock enable • RCC_OTGFS_CLK_EN: OTGFS clock enable • RCC_ETHMAC_CLK_EN: Ethernet MAC clock enable • RCC_ETHMACTX_CLK_EN: Ethernet MAC TX clock enable • RCC_ETHMACRX_CLK_EN: Ethernet MAC RX clock enable
state	: The state of the clock (On / Off) • RCC_PERIPH_CLK_ON: To set the clock on • RCC_PERIPH_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.22 Rcc_SetApb1PeriphClockState()

```
Std_ReturnType Rcc_SetApb1PeriphClockState (
            uint32_t periph,
            uint8_t state )
```

Choose a specific peripheral on the APB1 bus and changes its state (On / Off)

4.11.2.23 Function: Rcc_SetApb1PeriphClockState

Parameters

periph	The peripheral clock you want to configure • RCC_TIM2_CLK_EN: Timer 2 clock enable • RCC_TIM3_CLK_EN: Timer 3 clock enable • RCC_TIM4_CLK_EN: Timer 4 clock enable • RCC_TIM5_CLK_EN: Timer 5 clock enable • RCC_TIM6_CLK_EN: Timer 6 clock enable • RCC_TIM7_CLK_EN: Timer 7 clock enable • RCC_TIM12_CLK_EN: Timer 12 clock enable • RCC_TIM13_CLK_EN: Timer 13 clock enable • RCC_TIM14_CLK_EN: Timer 14 clock enable • RCC_WWD_GEN_CLK_EN: Window watchdog clock enable • RCC_SPI2_CLK_EN: SPI 2 clock enable • RCC_SPI3_CLK_EN: SPI 3 clock enable • RCC_USART2_CLK_EN: USART 2 clock enable • RCC_USART3_CLK_EN: USART 3 clock enable • RCC_USART4_CLK_EN: USART 4 clock enable • RCC_USART5_CLK_EN: USART 5 clock enable • RCC_I2C1_CLK_EN: I2C 1 clock enable • RCC_I2C2_CLK_EN: I2C 2 clock enable • RCC_USB_CLK_EN: USB clock enable • RCC_CAN_CLK_EN: CAN clock enable • RCC_BKP_CLK_EN: Backup interface clock enable • RCC_PWR_CLK_EN: Power interface clock enable • RCC_DAC_CLK_EN: DAC interface clock enable
state	The state of the clock (On / Off) • RCC_PERIPH_CLK_ON : To set the clock on • RCC_PERIPH_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.24 Function: Rcc_SetApb1PeriphClockState

Parameters

periph	The peripheral clock you want to configure • RCC_TIM2_CLK_EN: Timer 2 clock enable • RCC_TIM3_CLK_EN: Timer 3 clock enable • RCC_TIM4_CLK_EN: Timer 4 clock enable • RCC_TIM5_CLK_EN: Timer 5 clock enable • RCC_TIM6_CLK_EN: Timer 6 clock enable • RCC_TIM7_CLK_EN: Timer 7 clock enable • RCC_TIM12_CLK_EN: Timer 12 clock enable • RCC_TIM13_CLK_EN: Timer 13 clock enable • RCC_TIM14_CLK_EN: Timer 14 clock enable • RCC_WWD_GEN_CLK_EN: Window watchdog clock enable • RCC_SPI2_CLK_EN: SPI 2 clock enable • RCC_SPI3_CLK_EN: SPI 3 clock enable • RCC_USART2_CLK_EN: USART 2 clock enable • RCC_USART3_CLK_EN: USART 3 clock enable • RCC_USART4_CLK_EN: USART 4 clock enable • RCC_USART5_CLK_EN: USART 5 clock enable • RCC_I2C1_CLK_EN: I2C 1 clock enable • RCC_I2C2_CLK_EN: I2C 2 clock enable • RCC_USB_CLK_EN: USB clock enable • RCC_CAN_CLK_EN: CAN clock enable • RCC_BKP_CLK_EN: Backup interface clock enable • RCC_PWR_CLK_EN: Power interface clock enable • RCC_DAC_CLK_EN: DAC interface clock enable
state	The state of the clock (On / Off) • RCC_PERIPH_CLK_ON : To set the clock on • RCC_PERIPH_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.25 Rcc_SetApb2PeriphClockState()

```
Std_ReturnType Rcc_SetApb2PeriphClockState (
            uint32_t periph,
            uint8_t state )
```

Choose a specific peripheral on the APB2 bus and changes its state (On / Off)

4.11.2.26 Function: Rcc_SetApb2PeriphClockState

Parameters

periph	The peripheral clock you want to configure • RCC_AFIO_CLK_EN: Alternate function input output clock enable • RCC_IOPA_CLK_EN: Port A input output clock enable • RCC_IOPB_CLK_EN: Port B input output clock enable • RCC_IOPC_CLK_EN: Port C input output clock enable • RCC_IOPD_CLK_EN: Port D input output clock enable • RCC_IOPE_CLK_EN: Port E input output clock enable • RCC_IOPF_CLK_EN: Port F input output clock enable • RCC_IOPG_CLK_EN: Port G input output clock enable • RCC_ADC1_CLK_EN: ADC 1 clock enable • RCC_ADC2_CLK_EN: ADC 2 clock enable • RCC_TIM1_CLK_EN: Timer 1 clock enable • RCC_SPI1_CLK_EN: SPI 1 clock enable • RCC_TIM8_CLK_EN: Timer 8 clock enable • RCC_USART1_CLK_EN: USART 1 clock enable • RCC_ADC3_CLK_EN: ADC 3 clock enable • RCC_TIM9_CLK_EN: Timer 9 clock enable • RCC_TIM10_CLK_EN: Timer 10 clock enable • RCC_TIM11_CLK_EN: Timer 11 clock enable
state	The state of the clock (On / Off) • RCC_PERIPH_CLK_ON : To set the clock on • RCC_PERIPH_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.27 Function: Rcc_SetApb2PeriphClockState

Parameters

periph	The peripheral clock you want to configure • RCC_AFIO_CLK_EN: Alternate function input output clock enable • RCC_IOPA_CLK_EN: Port A input output clock enable • RCC_IOPB_CLK_EN: Port B input output clock enable • RCC_IOPC_CLK_EN: Port C input output clock enable • RCC_IOPD_CLK_EN: Port D input output clock enable • RCC_IOPE_CLK_EN: Port E input output clock enable • RCC_IOPF_CLK_EN: Port F input output clock enable • RCC_IOPG_CLK_EN: Port G input output clock enable • RCC_ADC1_CLK_EN: ADC 1 clock enable • RCC_ADC2_CLK_EN: ADC 2 clock enable • RCC_TIM1_CLK_EN: Timer 1 clock enable • RCC_SPI1_CLK_EN: SPI 1 clock enable • RCC_TIM8_CLK_EN: Timer 8 clock enable • RCC_USART1_CLK_EN: USART 1 clock enable • RCC_ADC3_CLK_EN: ADC 3 clock enable • RCC_TIM9_CLK_EN: Timer 9 clock enable • RCC_TIM10_CLK_EN: Timer 10 clock enable • RCC_TIM11_CLK_EN: Timer 11 clock enable
state	The state of the clock (On / Off) • RCC_PERIPH_CLK_ON : To set the clock on • RCC_PERIPH_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.28 Rcc_SetClockState()

```
Std_ReturnType Rcc_SetClockState (
            uint32_t clock,
            uint8_t state )
```

Choose a specific clock and changes its state (On / Off)

4.11.2.29 Function: Rcc_SetClockState

Parameters

clock	The clock you want to configure • RCC_HSI_SET: for the high speed internal clock • RCC_HSE_SET: for the high speed external clock • RCC_PLL_SET: for the PLL clock
state	: The state of the clock (On / Off) • RCC_CLK_ON : To set the clock on • RCC_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.30 Function: Rcc_SetClockState

Parameters

clock	The clock you want to configure • RCC_HSI_SET: for the high speed internal clock • RCC_HSE_SET: for the high speed external clock • RCC_PLL_SET: for the PLL clock
state	: The state of the clock (On / Off) • RCC_CLK_ON : To set the clock on • RCC_CLK_OFF : To set the clock off

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.31 Rcc_SetPllMultiplier()

```
Std_ReturnType Rcc_SetPllMultiplier (
            uint32_t pll )
```

Sets the PLL Multiplication factor.

4.11.2.32 Function: Rcc_SetPllMultiplier

Parameters

pll	: The PLL multiplication factor • RCC_PLL_MUL_XX : Set PLL multiplication factor

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.33 Function: Rcc_SetPllMultiplier

Parameters

pll	: The PLL multiplication factor • RCC_PLL_MUL_XX : Set PLL multiplication factor

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.34 Rcc_SetPllSource()

```
Std_ReturnType Rcc_SetPllSource (
            uint32_t source )
```

Chooses the PLL clock source.

4.11.2.35 Function: Rcc_SetPllSource

Parameters

source	: The PLL clock source • RCC_PLL_SRC_HSI : Choose high speed internal clock / 2 as a PLL clock source • RCC_PLL_SRC_HSE : Choose high speed external clock as a PLL clock source

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.36 Function: Rcc_SetPllSource

Parameters

source	: The PLL clock source • RCC_PLL_SRC_HSI : Choose high speed internal clock / 2 as a PLL clock source • RCC_PLL_SRC_HSE : Choose high speed external clock as a PLL clock source

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.37 Rcc_SetPrescaler()

```
Std_ReturnType Rcc_SetPrescaler (
            uint32_t clock,
            uint32_t value )
```

Sets the prescaler value for a specific clock.

4.11.2.38 Function: Rcc_SetPrescaler

Parameters

clock	The clock you want to configure • RCC_USB_PRE : For the USB prescaler • RCC_PLL_HSE_PRE : For the Pll prescaler • RCC_ADC_PRE : For the ADC prescaler • RCC_APB2_PRE : For the APB2 prescaler • RCC_APB1_PRE : For the APB1 prescaler • RCC_AHB_PRE : For the AHB prescaler
value	: The state of the clock (On / Off) • RCC_USB_PRE_1_5 : No USB prescaler value • RCC_USB_PRE_1_5 : USB prescaler 1.5 • RCC_PLL_HSE_PRE_X : PLL Prescaler value using high speed external clock • RCC_ADC_PRE_X : ADC Prescaler value • RCC_APB2_PRE_XX : APB2 prescater value • RCC_APB1_PRE_XX : APB1 prescater value • RCC_AHB_PRE_XXX : AHB prescater value

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.39 Function: Rcc_SetPrescaler

Parameters

clock	The clock you want to configure • RCC_USB_PRE : For the USB prescaler • RCC_PLL_HSE_PRE : For the Pll prescaler • RCC_ADC_PRE : For the ADC prescaler • RCC_APB2_PRE : For the APB2 prescaler • RCC_APB1_PRE : For the APB1 prescaler • RCC_AHB_PRE : For the AHB prescaler
value	: The state of the clock (On / Off) • RCC_USB_PRE_1_5 : No USB prescaler value • RCC_USB_PRE_1_5 : USB prescaler 1.5 • RCC_PLL_HSE_PRE_X : PLL Prescaler value using high speed external clock • RCC_ADC_PRE_X : ADC Prescaler value • RCC_APB2_PRE_XX : APB2 prescater value • RCC_APB1_PRE_XX : APB1 prescater value • RCC_AHB_PRE_XXX : AHB prescater value

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.40 Rcc_SwitchSystemClock()

```
Std_ReturnType Rcc_SwitchSystemClock (
             uint32_t clock )
```

Switches the system clock (HSI / HSE / PLL)

4.11.2.41 Function: Rcc_SwitchSystemClock

Parameters

clock	: The PLL clock source • RCC_SYS_CLK_SELECT_HSI : Select high speed internal clock as system clock • RCC_SYS_CLK_SELECT_HSE : Select high speed external clock as system clock • RCC_SYS_CLK_SELECT_PLL : Select PLL clock as system clock

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.11.2.42 Function: Rcc_SwitchSystemClock

Parameters

clock	: The PLL clock source • RCC_SYS_CLK_SELECT_HSI : Select high speed internal clock as system clock • RCC_SYS_CLK_SELECT_HSE : Select high speed external clock as system clock • RCC_SYS_CLK_SELECT_PLL : Select PLL clock as system clock

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.12 Switch.c File Reference

This file is to be used as an implementation for the Switch Handler.

```
#include "Std_Types.h"
#include "Gpio.h"
#include "HRcc.h"
#include "Switch.h"
```

Functions

- Std_ReturnType Switch_Init (void)

  Initializes GPIOs for the Switches.
- Std_ReturnType Switch_GetSwitchStatus (uint8_t switchName, uint8_t *state)

  Gets the status of the switch.

Variables

- const switch_t **Switch_switches** [SWITCH_NUMBER_OF_SWITCHES]

4.12.1 Detailed Description

This file is to be used as an implementation for the Switch Handler.

Author

Mark Attia

Date

January 22, 2020

4.12.2 Function Documentation

4.12.2.1 Switch_GetSwitchStatus()

```
Std_ReturnType Switch_GetSwitchStatus (
             uint8_t switchName,
             uint8_t * state )
```

Gets the status of the switch.

4.12.2.2 Function: Switch_GetSwitchStatus

Parameters

switchName	The name of the Switch
state	Save the status of the switch in • SWITCH_PRESSED : if the switch is pressed • SWITCH_NOT_PRESSED : if the switch is not pressed

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.12.2.3 Switch_Init()

```
Std_ReturnType Switch_Init (
             void  )
```

Initializes GPIOs for the Switches.

4.12.2.4 Function: Switch_Init

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.13 Switch.h File Reference

This file is to be used as an interface for the user of the Switch Handler.

```
#include "Switch_Cfg.h"
```

Data Structures

- struct switch_t

  The Switch pin layout.

Macros

- #define **SWITCH_PRESSED** 0
- #define **SWITCH_NOT_PRESSED** !SWITCH_PRESSED

Functions

- Std_ReturnType Switch_Init (void)

  Initializes GPIOs for the Switches.
- Std_ReturnType Switch_GetSwitchStatus (uint8_t switchName, uint8_t ∗state)

  Gets the status of the switch.

4.13.1 Detailed Description

This file is to be used as an interface for the user of the Switch Handler.

Author

Mark Attia

Date

January 22, 2020

4.13.2 Function Documentation

4.13.2.1 Switch_GetSwitchStatus()

```
Std_ReturnType Switch_GetSwitchStatus (
            uint8_t switchName,
            uint8_t ∗ state )
```

Gets the status of the switch.

4.13.2.2 Function: Switch_GetSwitchStatus

Parameters

switchName	The name of the Switch
state	Save the status of the switch in • SWITCH_PRESSED : if the switch is pressed • SWITCH_NOT_PRESSED : if the switch is not pressed

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.13.2.3 Function: Switch_GetSwitchStatus

Parameters

switchName	The name of the Switch
state	Save the status of the switch in • SWITCH_PRESSED : if the switch is pressed • SWITCH_NOT_PRESSED : if the switch is not pressed

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.13.2.4 Switch_Init()

```
Std_ReturnType Switch_Init (
            void  )
```

Initializes GPIOs for the Switches.

4.13.2.5 Function: Switch_Init

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.13.2.6 Function: Switch_Init

Returns

: A status E_OK : if the function is executed correctly E_NOT_OK : if the function is not executed correctly

4.14 Switch_Cfg.c File Reference

This file is to be used as an implementation of the configurations the user configured in the Switch_Cfg.h.

```
#include "Std_Types.h"
#include "Gpio.h"
#include "Switch_Cfg.h"
#include "Switch.h"
```

Variables

- const switch_t **Switch_switches** [SWITCH_NUMBER_OF_SWITCHES]

4.14.1 Detailed Description

This file is to be used as an implementation of the configurations the user configured in the Switch_Cfg.h.

Author

Mark Attia

Date

January 22, 2020

4.14.2 Variable Documentation

4.14.2.1 Switch_switches

```
const switch_t Switch_switches[SWITCH_NUMBER_OF_SWITCHES]
```

Initial value:

```
= {
    {GPIO_PIN_8, GPIO_PORTA, GPIO_PIN_RESET}
}
```

4.15 Switch_Cfg.h File Reference

This file is to be given to the user to configure the Switch Handler.

Macros

- #define **SWITCH_NUMBER_OF_SWITCHES** 1
- #define **BOOTLOADER_SWITCH** 0

4.15.1 Detailed Description

This file is to be given to the user to configure the Switch Handler.

Author

Mark Attia

Date

January 22, 2020

4.16 Uart.c File Reference

This is the implementation for the UART driver.

```
#include "Std_Types.h"
#include "Uart_Cfg.h"
#include "Uart.h"
#include "Dma.h"
```

Data Structures

- struct uart_t

  The UART Registers.
- struct dataBuffer_t

  The Uart Data Buffer.

Macros

- #define **UART_NUMBER_OF_MODULES** 3
- #define **UART_INT_NUMBER** 37
- #define **UART_BUFFER_IDLE** 0
- #define **UART_BUFFER_BUSY** 1
- #define **UART_TXE_CLR** 0xFFFFFF7F
- #define **UART_TC_CLR** 0xFFFFFFBF
- #define **UART_RXNE_CLR** 0xFFFFFFDF
- #define **UART_PE_CLR** 0xFFFFFFFE
- #define **UART_DR_CLR** 0xFFFFFE00
- #define **UART_STOP_CLR** 0xFFFFCFFF
- #define **UART_TXEIE_CLR** 0xFFFFFF7F
- #define **UART_PS_CLR** 0xFFFFFDFF
- #define **UART_M_CLR** 0xFFFFEFFF
- #define **UART_LBD_CLR** 0xFFFFFEFF
- #define **UART_LBDIE_CLR** 0xFFFFFFBF
- #define **UART_TXE_GET** 0x00000080
- #define **UART_TC_GET** 0x00000040
- #define **UART_RXNE_GET** 0x00000020
- #define **UART_PE_GET** 0x00000001

- #define **UART_UE_SET** 0x00002000
- #define **UART_PCE_SET** 0x00000400
- #define **UART_PEIE_SET** 0x00000100
- #define **UART_TXEIE_SET** 0x00000080
- #define **UART_TCIE_SET** 0x00000040
- #define **UART_RXNEIE_SET** 0x00000020
- #define **UART_IDLEIE_SET** 0x00000010
- #define **UART_TE_SET** 0x00000008
- #define **UART_RE_SET** 0x00000004
- #define **UART_M_SET** 0x00001000
- #define **UART_LBD_SET** 0x00000100
- #define **UART_LBDIE_SET** 0x00000040
- #define **UART_DMAT_SET** 0x00000080
- #define **UART_DMAR_SET** 0x00000040
- #define **UART_SBK_SET** 0x00000001
- #define **UART_LINEN_CLR** 0xFFFFBFFF
- #define **UART_RTSE_CLR** 0xFFFFFEFF
- #define **UART_NO_PRESCALER** 0x1
- #define **DMA_DID_NOT_RECEIVE** 0
- #define **DMA_RECEIVED** 1

Functions

- Std_ReturnType Uart_Init (Uart_cfg_t ∗cfgUart)

  Initializes the UART.
- Std_ReturnType Uart_Send (uint8_t ∗data, uint16_t length, uint8_t uartModule)

  Sends data through the UART.
- Std_ReturnType Uart_Receive (uint8_t ∗data, uint16_t length, uint8_t uartModule)

  Receives data through the UART.
- Std_ReturnType Uart_SendSync (uint8_t ∗data, uint16_t length, uint8_t uartModule)

  Sends data through the UART synchronously.
- Std_ReturnType Uart_ReceiveSync (uint8_t ∗data, uint16_t length, uint8_t uartModule)

  Receives data through the UART synchronously.
- Std_ReturnType Uart_SetTxCb (txCb_t func, uint8_t uartModule)

  Sets the callback function that will be called when transmission is completed.
- Std_ReturnType Uart_SetRxCb (rxCb_t func, uint8_t uartModule)

  Sets the callback function that will be called when receive is completed.
- Std_ReturnType Uart_SetBreakCb (brCb_t func, uint8_t uartModule)

  Sets the callback function that will be called when break happens.
- Std_ReturnType Uart_SendBreak (uint8_t uartModule)

  Sends a Lin break of 13 bit length.
- void USART1_IRQHandler (void)

  The UART 1 Handler.
- void USART2_IRQHandler (void)

  The UART 2 Handler.
- void USART3_IRQHandler (void)

  The UART 3 Handler.

Variables

- const uint32_t Uart_Address [UART_NUMBER_OF_MODULES]

  The Base Adresses of the UART module.
- const volatile uint8_t **Uart_DmaTxChannelNumber** [UART_NUMBER_OF_MODULES]
- const volatile uint8_t **Uart_DmaRxChannelNumber** [UART_NUMBER_OF_MODULES]

4.16.1 Detailed Description

This is the implementation for the UART driver.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-03-26

Copyright

Copyright (c) 2020

4.16.2 Function Documentation

4.16.2.1 Uart_Init()

```
Std_ReturnType Uart_Init (
            Uart_cfg_t * cfgUart )
```

Initializes the UART.

Parameters

cfg	The Uart Configurations

Returns

Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

4.16.2.2 Uart_Receive()

```
Std_ReturnType Uart_Receive (
             uint8_t * data,
             uint16_t length,
             uint8_t uartModule )
```

Receives data through the UART.

Parameters

data	The buffer to receive data in
length	the length of the data in bytes
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

Std_ReturnType A Status E_OK: If the driver is ready to receive E_NOT_OK: If the driver can't receive data right now

4.16.2.3 Uart_ReceiveSync()

```
Std_ReturnType Uart_ReceiveSync (
             uint8_t * data,
             uint16_t length,
             uint8_t uartModule )
```

Receives data through the UART synchronously.

Parameters

data	The buffer to receive data in
length	the length of the data in bytes
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

Std_ReturnType A Status E_OK: If the driver is ready to receive E_NOT_OK: If the driver can't receive data right now

4.16.2.4 Uart_Send()

```
Std_ReturnType Uart_Send (
            uint8_t * data,
            uint16_t length,
            uint8_t uartModule )
```

Sends data through the UART.

Parameters

data	The data to send
length	the length of the data in bytes
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

> Std_ReturnType A Status E_OK: If the driver is ready to send E_NOT_OK: If the driver can't send data right now

4.16.2.5 Uart_SendBreak()

```
Std_ReturnType Uart_SendBreak (
            uint8_t uartModule )
```

Sends a Lin break of 13 bit length.

Parameters

uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

> Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

4.16.2.6 Uart_SendSync()

```
Std_ReturnType Uart_SendSync (
            uint8_t * data,
            uint16_t length,
            uint8_t uartModule )
```

Sends data through the UART synchronously.

Parameters

data	The data to send
length	the length of the data in bytes
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

Std_ReturnType A Status E_OK: If the driver is ready to send E_NOT_OK: If the driver can't send data right now

4.16.2.7 Uart_SetBreakCb()

```
Std_ReturnType Uart_SetBreakCb (
            brCb_t func,
            uint8_t uartModule )
```

Sets the callback function that will be called when break happens.

Parameters

func	the callback function
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

4.16.2.8 Uart_SetRxCb()

```
Std_ReturnType Uart_SetRxCb (
             rxCb_t func,
             uint8_t uartModule )
```

Sets the callback function that will be called when receive is completed.

Parameters

func	the callback function
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

4.16.2.9 Uart_SetTxCb()

```
Std_ReturnType Uart_SetTxCb (
             txCb_t func,
             uint8_t uartModule )
```

Sets the callback function that will be called when transmission is completed.

Parameters

func	the callback function
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

4.16.2.10 USART1_IRQHandler()

```
void USART1_IRQHandler (
            void  )
```

The UART 1 Handler.

4.16.2.11 USART2_IRQHandler()

```
void USART2_IRQHandler (
            void  )
```

The UART 2 Handler.

4.16.2.12 USART3_IRQHandler()

```
void USART3_IRQHandler (
            void  )
```

The UART 3 Handler.

4.16.3 Variable Documentation

4.16.3.1 Uart_Address

```
const uint32_t Uart_Address[UART_NUMBER_OF_MODULES]
```

Initial value:

```
= {
  0x40013800,
  0x40004400,
  0x40004800
}
```

The Base Adresses of the UART module.

4.16.3.2 Uart_DmaRxChannelNumber

```
const volatile uint8_t Uart_DmaRxChannelNumber[UART_NUMBER_OF_MODULES]
```

Initial value:

```
=
{
  DMA_CH_5,
  DMA_CH_6,
  DMA_CH_3
}
```

4.16.3.3 Uart_DmaTxChannelNumber

```
const volatile uint8_t Uart_DmaTxChannelNumber[UART_NUMBER_OF_MODULES]
```

Initial value:

```
=
{
  DMA_CH_4,
  DMA_CH_7,
  DMA_CH_2
}
```

4.17 Uart.h File Reference

This is the user interface for the UART driver.

Data Structures

- struct Uart_cfg_t

Macros

- #define **UART1** 0
- #define **UART2** 1
- #define **UART3** 2
- #define **UART_ODD_PARITY** 0x00000200
- #define **UART_EVEN_PARITY** 0x00000000
- #define **UART_NO_PARITY** 0xFFFFFBFF
- #define **UART_STOP_ONE_BIT** 0x00000000
- #define **UART_STOP_TWO_BITS** 0x00003000
- #define **UART_FLOW_CONTROL_EN** 0x00000100
- #define **UART_FLOW_CONTROL_DIS** 0x00000000
- #define **UART_LIN_EN** 0x00004000
- #define **UART_LIN_DIS** 0x00000000
- #define **UART_INTERRUPT_DIS** 0
- #define **UART_INTERRUPT_TXE** 1
- #define **UART_INTERRUPT_TC** 2
- #define **UART_INTERRUPT_RXNE** 4
- #define **UART_INTERRUPT_LBD** 8
- #define **UART_MODE_ASYNC** 0
- #define **UART_MODE_DMA** 1

Typedefs

- typedef void(∗ **txCb_t**) (uint8_t)
- typedef void(∗ **rxCb_t**) (uint8_t)
- typedef void(∗ **brCb_t**) (uint8_t)

Functions

- Std_ReturnType Uart_Init (Uart_cfg_t ∗cfgUart)

  Initializes the UART.
- Std_ReturnType Uart_Send (uint8_t ∗data, uint16_t length, uint8_t uartModule)

  Sends data through the UART.
- Std_ReturnType Uart_SendBreak (uint8_t uartModule)

  Sends a Lin break of 13 bit length.
- Std_ReturnType Uart_Receive (uint8_t ∗data, uint16_t length, uint8_t uartModule)

  Receives data through the UART.
- Std_ReturnType Uart_SendSync (uint8_t ∗data, uint16_t length, uint8_t uartModule)

  Sends data through the UART synchronously.
- Std_ReturnType Uart_ReceiveSync (uint8_t ∗data, uint16_t length, uint8_t uartModule)

  Receives data through the UART synchronously.
- Std_ReturnType Uart_SetTxCb (txCb_t func, uint8_t uartModule)

  Sets the callback function that will be called when transmission is completed.
- Std_ReturnType Uart_SetRxCb (rxCb_t func, uint8_t uartModule)

  Sets the callback function that will be called when receive is completed.
- Std_ReturnType Uart_SetBreakCb (brCb_t func, uint8_t uartModule)

  Sets the callback function that will be called when break happens.

4.17.1 Detailed Description

This is the user interface for the UART driver.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-03-26

Copyright

Copyright (c) 2020

4.17.2 Function Documentation

4.17.2.1 Uart_Init()

```
Std_ReturnType Uart_Init (
              Uart_cfg_t ∗ cfgUart )
```

Initializes the UART.

Parameters

cfgUart	The Uart Configurations

Returns

Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

Parameters

cfg	The Uart Configurations

Returns

Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

4.17.2.2 Uart_Receive()

```
Std_ReturnType Uart_Receive (
            uint8_t * data,
            uint16_t length,
            uint8_t uartModule )
```

Receives data through the UART.

Parameters

data	The buffer to receive data in
length	the length of the data in bytes
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

Std_ReturnType A Status E_OK: If the driver is ready to receive E_NOT_OK: If the driver can't receive data right now

4.17.2.3 Uart_ReceiveSync()

```
Std_ReturnType Uart_ReceiveSync (
            uint8_t * data,
            uint16_t length,
            uint8_t uartModule )
```

Receives data through the UART synchronously.

Parameters

data	The buffer to receive data in
length	the length of the data in bytes
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

> Std_ReturnType A Status E_OK: If the driver is ready to receive E_NOT_OK: If the driver can't receive data right now

4.17.2.4 Uart_Send()

```
Std_ReturnType Uart_Send (
            uint8_t * data,
            uint16_t length,
            uint8_t uartModule )
```

Sends data through the UART.

Parameters

data	The data to send
length	the length of the data in bytes
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

> Std_ReturnType A Status E_OK: If the driver is ready to send E_NOT_OK: If the driver can't send data right now

4.17.2.5 Uart_SendBreak()

```
Std_ReturnType Uart_SendBreak (
             uint8_t uartModule )
```

Sends a Lin break of 13 bit length.

Parameters

uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

4.17.2.6 Uart_SendSync()

```
Std_ReturnType Uart_SendSync (
             uint8_t * data,
             uint16_t length,
             uint8_t uartModule )
```

Sends data through the UART synchronously.

Parameters

data	The data to send
length	the length of the data in bytes
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

Std_ReturnType A Status E_OK: If the driver is ready to send E_NOT_OK: If the driver can't send data right now

4.17.2.7 Uart_SetBreakCb()

```
Std_ReturnType Uart_SetBreakCb (
            brCb_t func,
            uint8_t uartModule )
```

Sets the callback function that will be called when break happens.

Parameters

func	the callback function
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

> Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

4.17.2.8 Uart_SetRxCb()

```
Std_ReturnType Uart_SetRxCb (
            rxCb_t func,
            uint8_t uartModule )
```

Sets the callback function that will be called when receive is completed.

Parameters

func	the callback function
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

> Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

4.17.2.9 Uart_SetTxCb()

```
Std_ReturnType Uart_SetTxCb (
            txCb_t func,
            uint8_t uartModule )
```

Sets the callback function that will be called when transmission is completed.

Parameters

func	the callback function
uartModule	the module number of the UART • UART1 • UART2 • UART3

Returns

Std_ReturnType A Status E_OK: If the function executed successfully E_NOT_OK: If the did not execute successfully

4.18 Uart_Cfg.h File Reference

Those are the user configurations for the Uart Driver.

Macros

- #define **UART_MODE** UART_MODE_ASYNC

4.18.1 Detailed Description

Those are the user configurations for the Uart Driver.

Author

Mark Attia (markjosephattia@gmailcom)

Version

0.1

Date

2020-04-04

Copyright

Copyright (c) 2020

Index