

Two Counters

Generated by Doxygen 1.8.17

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 clcd_t Struct Reference	5
3.2 dataBuffer_t Struct Reference	5
3.3 frame_t Union Reference	6
3.3.1 Detailed Description	6
3.4 gpio_t Struct Reference	6
3.5 hUartConfig_t Struct Reference	6
3.6 led_t Struct Reference	7
3.7 NVIC_regMap Struct Reference	7
3.8 RCC_regMap Struct Reference	7
3.9 switch_t Struct Reference	8
3.10 SysTask Struct Reference	8
3.11 SYSTICK_regMap Struct Reference	8
3.12 Task Struct Reference	8
3.13 uart_t Struct Reference	9
4 File Documentation	11
4.1 App.c File Reference	11
4.1.1 Detailed Description	12
4.1.2 Function Documentation	12
4.1.2.1 APP_init()	12
4.1.2.2 APP_receiveFcn()	12
4.1.2.3 APP_sendTask()	13
4.2 App.h File Reference	13
4.2.1 Detailed Description	13
4.2.2 Function Documentation	13
4.2.2.1 APP_init()	14
4.2.2.2 APP_receiveFcn()	14
4.2.2.3 APP_sendTask()	14
4.3 Clcd.c File Reference	14
4.3.1 Detailed Description	15
4.3.2 Function Documentation	16
4.3.2.1 CLcd_ClearDisplay()	16
4.3.2.2 CLcd_ConfigCursor()	16
4.3.2.3 CLcd_ConfigDisplay()	16
4.3.2.4 CLcd_GotoXY()	17
4.3.2.5 CLcd_Init()	17

4.3.2.6 CLcd_SetDoneNotification()	18
4.3.2.7 CLcd_Task()	18
4.3.2.8 CLcd_WriteString()	18
4.4 CLcd.h File Reference	18
4.4.1 Detailed Description	19
4.4.2 Function Documentation	20
4.4.2.1 CLcd_ClearDisplay()	20
4.4.2.2 CLcd_ConfigCursor()	20
4.4.2.3 CLcd_ConfigDisplay()	20
4.4.2.4 CLcd_GotoXY()	21
4.4.2.5 CLcd_Init()	21
4.4.2.6 CLcd_SetDoneNotification()	22
4.4.2.7 CLcd_Task()	22
4.4.2.8 CLcd_WriteString()	22
4.5 CLcd_Cfg.c File Reference	22
4.5.1 Detailed Description	23
4.5.2 Variable Documentation	23
4.5.2.1 CLcd_clcd	23
4.6 Gpio.c File Reference	24
4.6.1 Detailed Description	24
4.6.2 Function Documentation	24
4.6.2.1 Gpio_InitPins()	24
4.6.2.2 Function: Gpio_InitPins	24
4.6.2.3 Gpio_ReadPin()	25
4.6.2.4 Function: Gpio_ReadPin	25
4.6.2.5 Gpio_WritePin()	25
4.6.2.6 Function: Gpio_WritePin	25
4.7 Gpio.h File Reference	26
4.7.1 Detailed Description	27
4.7.2 Function Documentation	27
4.7.2.1 Gpio_InitPins()	27
4.7.2.2 Function: Gpio_InitPins	27
4.7.2.3 Function: Gpio_InitPins	28
4.7.2.4 Gpio_ReadPin()	28
4.7.2.5 Function: Gpio_ReadPin	28
4.7.2.6 Function: Gpio_ReadPin	28
4.7.2.7 Gpio_WritePin()	29
4.7.2.8 Function: Gpio_WritePin	29
4.7.2.9 Function: Gpio_WritePin	29
4.8 HRcc.h File Reference	30
4.8.1 Detailed Description	30
4.8.2 Function Documentation	30

4.8.2.1 HRcc_EnPortClock()	30
4.8.2.2 HRcc_SystemClockInit()	31
4.9 HUART.c File Reference	31
4.9.1 Detailed Description	32
4.9.2 Function Documentation	32
4.9.2.1 HUART_Config()	32
4.9.2.2 HUART_Init()	33
4.9.2.3 HUART_Receive()	33
4.9.2.4 HUART_Send()	34
4.9.2.5 HUART_SetModule()	34
4.9.2.6 HUART_SetRxCb()	34
4.9.2.7 HUART_SetTxCb()	35
4.10 HUART.h File Reference	35
4.10.1 Detailed Description	36
4.10.2 Function Documentation	36
4.10.2.1 HUART_Config()	36
4.10.2.2 HUART_Init()	37
4.10.2.3 HUART_Receive()	37
4.10.2.4 HUART_Send()	38
4.10.2.5 HUART_SetModule()	38
4.10.2.6 HUART_SetRxCb()	38
4.10.2.7 HUART_SetTxCb()	39
4.11 HUART_Cfg.h File Reference	39
4.11.1 Detailed Description	39
4.12 LED.c File Reference	40
4.12.1 Detailed Description	40
4.12.2 Function Documentation	40
4.12.2.1 LED_Init()	40
4.12.2.2 Function: LED_Init	41
4.12.2.3 LED_SetLedOff()	41
4.12.2.4 Function: LED_SetLedOff	41
4.12.2.5 LED_SetLedOn()	41
4.12.2.6 Function: LED_SetLedOn	41
4.12.2.7 LED_SetLedStatus()	42
4.12.2.8 Function: LED_SetLedStatus	42
4.13 LED.h File Reference	42
4.13.1 Detailed Description	43
4.13.2 Function Documentation	43
4.13.2.1 LED_Init()	43
4.13.2.2 Function: LED_Init	43
4.13.2.3 Function: LED_Init	43
4.13.2.4 LED_SetLedOff()	43

4.13.2.5 Function: Led_SetLedOff	43
4.13.2.6 Function: Led_SetLedOff	44
4.13.2.7 Led_SetLedOn()	44
4.13.2.8 Function: Led_SetLedOn	44
4.13.2.9 Function: Led_SetLedOn	44
4.13.2.10 Led_SetLedStatus()	45
4.13.2.11 Function: Led_SetLedStatus	45
4.13.2.12 Function: Led_SetLedStatus	45
4.14 Led_Cfg.c File Reference	45
4.14.1 Detailed Description	46
4.14.2 Variable Documentation	46
4.14.2.1 Led_leds	46
4.15 Led_Cfg.h File Reference	46
4.15.1 Detailed Description	47
4.16 main.c File Reference	47
4.16.1 Detailed Description	47
4.17 NVIC.c File Reference	48
4.17.1 Detailed Description	48
4.17.2 Function Documentation	49
4.17.2.1 NVIC_configurePriority()	49
4.17.2.2 NVIC_controlAllPeripheral()	49
4.17.2.3 NVIC_controlFault()	49
4.17.2.4 NVIC_controlInterrupt()	50
4.17.2.5 NVIC_controlPendingFlag()	50
4.17.2.6 NVIC_filterInterrupts()	50
4.17.2.7 NVIC_getActiveFlagStatus()	50
4.17.2.8 NVIC_getPriority()	52
4.18 NVIC.h File Reference	52
4.18.1 Detailed Description	54
4.18.2 Macro Definition Documentation	54
4.18.2.1 NVIC_DISABLE	54
4.18.2.2 NVIC_IRQNUM_WWDG	55
4.18.2.3 NVIC_RESET	55
4.18.3 Function Documentation	55
4.18.3.1 NVIC_configurePriority()	55
4.18.3.2 NVIC_controlAllPeripheral()	55
4.18.3.3 NVIC_controlFault()	55
4.18.3.4 NVIC_controlInterrupt()	56
4.18.3.5 NVIC_controlPendingFlag()	56
4.18.3.6 NVIC_filterInterrupts()	56
4.18.3.7 NVIC_getActiveFlagStatus()	57
4.18.3.8 NVIC_getPriority()	57

4.19 RCC.h File Reference	57
4.19.1 Detailed Description	60
4.19.2 Function Documentation	60
4.19.2.1 RCC_configureMCO()	60
4.19.2.2 RCC_configurePLL()	61
4.19.2.3 RCC_configurePrescalers()	61
4.19.2.4 RCC_controlAHBPeripheral()	62
4.19.2.5 RCC_controlAPB1Peripheral()	62
4.19.2.6 RCC_controlAPB2Peripheral()	63
4.19.2.7 RCC_selectSystemClock()	63
4.19.2.8 RCC_setClockState()	63
4.20 SCHED.c File Reference	63
4.20.1 Detailed Description	64
4.20.2 Function Documentation	64
4.20.2.1 SCHED_createTask()	64
4.20.2.2 SCHED_init()	65
4.20.2.3 SCHED_start()	65
4.21 SCHED1.h File Reference	65
4.21.1 Detailed Description	66
4.21.2 Function Documentation	66
4.21.2.1 SCHED_createTask()	66
4.21.2.2 SCHED_init()	66
4.21.2.3 SCHED_start()	67
4.22 SCHED_CONF.h File Reference	67
4.22.1 Detailed Description	67
4.23 Std_Types.h File Reference	67
4.23.1 Detailed Description	68
4.24 Switch.c File Reference	69
4.24.1 Detailed Description	69
4.24.2 Function Documentation	69
4.24.2.1 Switch_GetSwitchStatus()	69
4.24.2.2 Function: Switch_GetSwitchStatus	69
4.24.2.3 Switch_Init()	70
4.24.2.4 Function: Switch_Init	70
4.24.2.5 Switch_Task()	70
4.25 Switch.h File Reference	70
4.25.1 Detailed Description	71
4.25.2 Function Documentation	71
4.25.2.1 Switch_GetSwitchStatus()	71
4.25.2.2 Function: Switch_GetSwitchStatus	71
4.25.2.3 Function: Switch_GetSwitchStatus	71
4.25.2.4 Switch_Init()	72

4.25.2.5 Function: Switch_Init	72
4.25.2.6 Function: Switch_Init	72
4.25.2.7 Switch_Task()	72
4.26 Switch_Cfg.c File Reference	72
4.26.1 Detailed Description	73
4.26.2 Variable Documentation	73
4.26.2.1 Switch_switches	73
4.27 Switch_Cfg.h File Reference	73
4.27.1 Detailed Description	73
4.28 SYSTICK.c File Reference	74
4.28.1 Detailed Description	74
4.28.2 Function Documentation	75
4.28.2.1 SysTick_Handler()	75
4.28.2.2 SYSTICK_init()	75
4.28.2.3 SYSTICK_setCallbackFcn()	75
4.28.2.4 SYSTICK_setTime()	75
4.28.2.5 SYSTICK_start()	76
4.28.2.6 SYSTICK_stop()	76
4.29 SYSTICK.h File Reference	76
4.29.1 Detailed Description	77
4.29.2 Function Documentation	77
4.29.2.1 SYSTICK_init()	77
4.29.2.2 SYSTICK_setCallbackFcn()	77
4.29.2.3 SYSTICK_setTime()	78
4.29.2.4 SYSTICK_start()	78
4.29.2.5 SYSTICK_stop()	78
4.30 SYSTICK_CONF.h File Reference	78
4.30.1 Detailed Description	79
4.31 Uart.c File Reference	79
4.31.1 Detailed Description	81
4.31.2 Function Documentation	81
4.31.2.1 UART4_IRQHandler()	81
4.31.2.2 UART5_IRQHandler()	81
4.31.2.3 Uart_Init()	81
4.31.2.4 Uart_Receive()	82
4.31.2.5 Uart_Send()	82
4.31.2.6 Uart_SetRxCb()	83
4.31.2.7 Uart_SetTxCb()	83
4.31.2.8 USART1_IRQHandler()	84
4.31.2.9 USART2_IRQHandler()	84
4.31.2.10 USART3_IRQHandler()	84
4.31.3 Variable Documentation	84

4.31.3.1 Uart_Address	84
4.32 Uart.h File Reference	85
4.32.1 Detailed Description	85
4.32.2 Function Documentation	86
4.32.2.1 Uart_Init()	86
4.32.2.2 Uart_Receive()	86
4.32.2.3 Uart_Send()	87
4.32.2.4 Uart_SetRxCb()	87
4.32.2.5 Uart_SetTxCb()	88
Index	89

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

clcd_t	5
dataBuffer_t	5
frame_t	6
This is the frame type of size 4 byte	6
gpio_t	6
hUartConfig_t	6
led_t	7
NVIC_regMap	7
RCC_regMap	7
switch_t	8
SysTask	8
SYSTICK_regMap	8
Task	8
uart_t	9

Chapter 2

File Index

2.1 File List

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

App.c	This is an application for testing the UART and the LCD drivers	11
App.h	This is the user interface for the two counters application	13
Clcd.c	This file contains the implementation for the Character LCD Driver	14
CLcd.h	This file is the user interface for the Character LCD Driver	18
CLcd_Cfg.c	The user's configurations	22
Gpio.c	This file is to be used as an implementation of the GPIO driver	24
Gpio.h	This file is to be used as an interface for the user of GPIO driver	26
HRcc.h	This is the user interface for the RCC Handler	30
HUart.c	This is the implementation for the UART handler	31
HUart.h	This is the user interface for the uart handler	35
HUart_Cfg.h	These are the user's configurations for the HUART driver	39
Led.c	This file is to be used as an implementation for the Led Handler	40
Led.h	This file is to be used as an interface for the user of the Led Handler	42
Led_Cfg.c	Those are the User's configurations for the LED Driver	45
Led_Cfg.h	This file is to be given to the user to configure the Led Handler	46
main.c	Here is the implementation for the main function fo the application and also the tasks	47
NVIC.c	This is the implementation for the NVIC Driver	48
NVIC.h	This is the user interface for the NVIC driver	52

RCC.h	This is the user interface for the RCC Driver	57
SCHED.c	This is the implementation of the scheduler	63
SCHED1.h	This is the user interface for the scheduler	65
SCHED_CONF.h	Those are the configurations for the Scheduler Driver	67
Std_Types.h	Those are the standard types used in the drivers	67
Switch.c	This file is to be used as an implementation for the Switch Handler	69
Switch.h	This file is to be used as an interface for the user of the Switch Handler	70
Switch_Cfg.c	This file is to be used as an implementation of the configurations the user configured in the Switch_Cfg.h	72
Switch_Cfg.h	This file is to be given to the user to configure the Switch Handler	73
SYSTICK.c	This is the SysTick driver implementation	74
SYSTICK.h	This is the user interface for the SysTick Driver	76
SYSTICK_CONF.h	Those are the configurations for the SysTick Driver	78
Uart.c	This is the implementation for the UART driver	79
Uart.h	This is the user interface for the UART driver	85

Chapter 3

Data Structure Documentation

3.1 clcd_t Struct Reference

Data Fields

- uint32_t **enPin**
- uint32_t **enPort**
- uint32_t **rwPin**
- uint32_t **rwPort**
- uint32_t **rsPin**
- uint32_t **rsPort**
- uint32_t **dPin** [CLCD_NUMBER_OF_DATA_PINS]
- uint32_t **dPort** [CLCD_NUMBER_OF_DATA_PINS]

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

- [CLcd.h](#)

3.2 dataBuffer_t Struct Reference

Data Fields

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

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

- [Uart.c](#)

3.3 frame_t Union Reference

This is the frame type of size 4 byte.

Data Fields

- uint8_t **data** [4]
- uint32_t **fullFrame**

3.3.1 Detailed Description

This is the frame type of size 4 byte.

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

- [App.c](#)

3.4 gpio_t Struct Reference

Data Fields

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

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

- [Gpio.h](#)

3.5 hUartConfig_t Struct Reference

Data Fields

- uint32_t **baudRate**
- uint32_t **stopBits**
- uint32_t **parity**
- uint32_t **flowControl**

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

- [HUart.c](#)

3.6 led_t Struct Reference

Data Fields

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

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

- [Led.h](#)

3.7 NVIC_regMap Struct Reference

Data Fields

- u32 **ISER** [3]
- u32 **RESERVED0** [29]
- u32 **ICER** [3]
- u32 **RESERVED1** [29]
- u32 **ISPR** [3]
- u32 **RESERVED2** [29]
- u32 **ICPR** [3]
- u32 **RESERVED3** [29]
- u32 **IABR** [3]
- u32 **RESERVED4** [29]
- u32 **IPR** [21]

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

- [NVIC.c](#)

3.8 RCC_regMap Struct Reference

Data Fields

- u32 **RCC_CR**
- u32 **RCC_CFGR**
- u32 **RCC_CIR**
- u32 **RCC_APB2RSTR**
- u32 **RCC_APB1RSTR**
- u32 **RCC_AHBENR**
- u32 **RCC_APB2ENR**
- u32 **RCC_APB1ENR**
- u32 **RCC_BDCR**
- u32 **RCC_CSR**

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

- [RCC.c](#)

3.9 switch_t Struct Reference

Data Fields

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

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

- [Switch.h](#)

3.10 SysTask Struct Reference

Data Fields

- [Task](#) * **appTask**
- u32 **RemainToExec**
- u32 **periodicTimeTicks**

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

- [SCHED.c](#)

3.11 SYSTICK_regMap Struct Reference

Data Fields

- u32 **CTRL**
- u32 **LOAD**
- u32 **VAL**
- u32 **CALIB**

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

- [SYSTICK.c](#)

3.12 Task Struct Reference

Data Fields

- taskRunnable **runnable**
- u32 **periodicTime**
- u32 **priority**

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

- [SCHED1.h](#)

3.13 uart_t Struct Reference

Data Fields

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

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

- [Uart.c](#)

Chapter 4

File Documentation

4.1 App.c File Reference

This is an application for testing the UART and the LCD drivers.

```
#include "Std_Types.h"
#include <stdlib.h>
#include "stdio.h"
#include "HUART_Cfg.h"
#include "HUART.h"
#include "CLCD.h"
#include "Switch_Cfg.h"
#include "Switch.h"
#include "Led_Cfg.h"
#include "Led.h"
#include "App.h"
```

Data Structures

- union [frame_t](#)

This is the frame type of size 4 byte.

Functions

- Std_ReturnType [APP_init](#) (void)

This is the initialization for the two counter application.

- void [APP_sendTask](#) (void)

The free running task that comes every 1 milli second.

- void [APP_receiveFcn](#) (void)

The receive function that will be called after each received frame.

Variables

- [frame_t](#) **recFrame**
- [frame_t](#) **sendFrame**

4.1.1 Detailed Description

This is an application for testing the UART and the LCD drivers.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-28

Copyright

Copyright (c) 2020

4.1.2 Function Documentation

4.1.2.1 APP_init()

```
Std_ReturnType APP_init (  
    void )
```

This is the initialization for the two counter application.

Returns

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

4.1.2.2 APP_receiveFcn()

```
void APP_receiveFcn (  
    void )
```

The receive function that will be called after each received frame.

4.1.2.3 APP_sendTask()

```
void APP_sendTask (
    void )
```

The free running task that comes every 1 milli second.

4.2 App.h File Reference

This is the user interface for the two counters application.

Functions

- Std_ReturnType [APP_init](#) (void)
This is the initialization for the two counter application.
- void [APP_sendTask](#) (void)
The free running task that comes every 1 milli second.
- void [APP_receiveFcn](#) (void)
The receive function that will be called after each received frame.

4.2.1 Detailed Description

This is the user interface for the two counters application.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-29

Copyright

Copyright (c) 2020

4.2.2 Function Documentation

4.2.2.1 APP_init()

```
Std_ReturnType APP_init (  
    void )
```

This is the initialization for the two counter application.

Returns

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

4.2.2.2 APP_receiveFcn()

```
void APP_receiveFcn (  
    void )
```

The receive function that will be called after each received frame.

4.2.2.3 APP_sendTask()

```
void APP_sendTask (  
    void )
```

The free running task that comes every 1 milli second.

4.3 Clcd.c File Reference

This file contains the implementation for the Character LCD Driver.

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

Macros

- #define CLCD_INITIALIZED 0
- #define CLCD_NOT_INITIALIZED 1
- #define CLCD_EMPTY_CMD 0x0
- #define CLCD_INIT_CONST 0x3
- #define CLCD_FUNC_SET 0x2
- #define CLCD_CLEAR_DISP 0x1
- #define CLCD_INC 0x6
- #define CLCD_DDRAM 0x80
- #define CLCD_SECOND_LINE 0x40
- #define CLCD_DISP_SETTING 0x8
- #define CLCD_CONFIG_DISP_CLR 0xF7

Enumerations

- enum `initState_t` {
 `hardwareInit_s`, `specialCaseFunctionSet_s`, `functionSet_s`, `display_s`,
 `clear_s`, `entry_s`}
- enum `writeState_t` { `setAddress_s`, `writeData_s` }
- enum `process_t` {
 `init_p`, `write_p`, `clear_p`, `goto_p`,
 `setup_p`, `idle_p` }
- enum `enable_t` { `low_s`, `high_s` }

Functions

- Std_ReturnType `CLcd_Init` (uint8_t nLines, uint8_t cursor, uint8_t blink)
 The Character LCD initialization.
- Std_ReturnType `CLcd_WriteString` (uint8_t *str, uint8_t x, uint8_t y)
 Writes a string on a specific location on the lcd display.
- Std_ReturnType `CLcd_ClearDisplay` (void)
 Clears the display.
- Std_ReturnType `CLcd_GotoXY` (uint8_t x, uint8_t y)
 jumps to a specific location on the lcd display
- Std_ReturnType `CLcd_ConfigCursor` (uint8_t cursor, uint8_t blink)
 Configures the cursor options.
- Std_ReturnType `CLcd_ConfigDisplay` (uint8_t disp)
 Sets the display on and off.
- Std_ReturnType `CLcd_SetDoneNotification` (lcdCb_t cb)
 Sets the callback function executed when done.
- void `CLcd_Task` (void)
 The running task that have to come every 1 milli second.

Variables

- const `clcd_t` `CLcd_clcd`

4.3.1 Detailed Description

This file contains the implementation for the Character LCD Driver.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-03-26

Copyright

Copyright (c) 2020

4.3.2 Function Documentation

4.3.2.1 CLcd_ClearDisplay()

```
Std_ReturnType CLcd_ClearDisplay (
    void )
```

Clears the display.

Returns

Std_ReturnType E_OK : If the clear operation started successfully E_NOT_OK : If the clear operation is not able to start right now

4.3.2.2 CLcd_ConfigCursor()

```
Std_ReturnType CLcd_ConfigCursor (
    uint8_t cursor,
    uint8_t blink )
```

Configures the cursor options.

Parameters

<i>cursor</i>	The State of the cursor (Visible or not) CLCD_CURSOR_ON CLCD_CURSOR_OFF
<i>blink</i>	The blinking option (no/off) CLCD_BLINKING_ON CLCD_BLINKING_OFF

Returns

Std_ReturnType E_OK : If the configuration started successfully E_NOT_OK : If the configuration is not able to start right now

4.3.2.3 CLcd_ConfigDisplay()

```
Std_ReturnType CLcd_ConfigDisplay (
    uint8_t disp )
```

Sets the display on and off.

Parameters

<i>disp</i>	the display state CLCD_DISP_ON CLCD_DISP_OFF
-------------	--

Returns

Std_ReturnType E_OK : If the configuration started successfully E_NOT_OK : If the configuration is not able to start right now

4.3.2.4 CLcd_GotoXY()

```
Std_ReturnType CLcd_GotoXY (
    uint8_t x,
    uint8_t y )
```

jumps to a specific location on the lcd display

Parameters

<i>x</i>	the location on the x-axis
<i>y</i>	the location on the y-axis

Returns

Std_ReturnType E_OK : If the goto operation started successfully E_NOT_OK : If the goto operation is not able to start right now

4.3.2.5 CLcd_Init()

```
Std_ReturnType CLcd_Init (
    uint8_t nLines,
    uint8_t cursor,
    uint8_t blink )
```

The Character LCD initialization.

Parameters

<i>nLines</i>	The number of lines on display CLCD_TWO_LINES : Two lines display CLCD_ONE_LINE : One line display
<i>cursor</i>	The State of the cursor (Visible or not) CLCD_CURSOR_ON CLCD_CURSOR_OFF
<i>blink</i>	The blinking option (no/off) CLCD_BLINKING_ON CLCD_BLINKING_OFF

Returns

Std_ReturnType E_OK : If the initialization started successfully E_NOT_OK : If the initialization is not able to start right now

4.3.2.6 CLcd_SetDoneNotification()

```
Std_ReturnType CLcd_SetDoneNotification (
    lcdCb_t cb )
```

Sets the callback function executed when done.

Parameters

<i>cb</i>	the callback function
-----------	-----------------------

Returns

Std_ReturnType

4.3.2.7 CLcd_Task()

```
void CLcd_Task (
    void )
```

The running task that have to come every 1 milli second.

4.3.2.8 CLcd_WriteString()

```
Std_ReturnType CLcd_WriteString (
    uint8_t * str,
    uint8_t x,
    uint8_t y )
```

Writes a string on a specific location on the lcd display.

Parameters

<i>str</i>	the string to write
<i>x</i>	the location on the x-axis
<i>y</i>	the location on the y-axis

Returns

Std_ReturnType E_OK : If the writing started successfully E_NOT_OK : If the write operation is not able to start right now

4.4 CLcd.h File Reference

This file is the user interface for the Character LCD Driver.

Data Structures

- struct [clcd_t](#)

Macros

- #define **CLCD_NUMBER_OF_DATA_PINS** 4
- #define **CLCD_TWO_LINES** 0x8
- #define **CLCD_ONE_LINE** 0x0
- #define **CLCD_DISP_ON** 0x4
- #define **CLCD_DISP_OFF** 0x0
- #define **CLCD_CURSOR_ON** 0x2
- #define **CLCD_CURSOR_OFF** 0x0
- #define **CLCD_BLINKING_ON** 0x1
- #define **CLCD_BLINKING_OFF** 0x0

Typedefs

- typedef void(* **lcdCb_t**) (void)

Functions

- Std_ReturnType [CLcd_Init](#) (uint8_t nLines, uint8_t cursor, uint8_t blink)
The Character LCD initialization.
- Std_ReturnType [CLcd_WriteString](#) (uint8_t *str, uint8_t x, uint8_t y)
Writes a string on a specific location on the lcd display.
- Std_ReturnType [CLcd_ClearDisplay](#) (void)
Clears the display.
- Std_ReturnType [CLcd_GotoXY](#) (uint8_t x, uint8_t y)
jumps to a specific location on the lcd display
- Std_ReturnType [CLcd_ConfigCursor](#) (uint8_t cursor, uint8_t blink)
Configures the cursor options.
- Std_ReturnType [CLcd_ConfigDisplay](#) (uint8_t disp)
Sets the display on and off.
- Std_ReturnType [CLcd_SetDoneNotification](#) (lcdCb_t cb)
Sets the callback function executed when done.
- void [CLcd_Task](#) (void)
The running task that have to come every 1 milli second.

4.4.1 Detailed Description

This file is the user interface for the Character LCD Driver.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-03-26

Copyright

Copyright (c) 2020

4.4.2 Function Documentation

4.4.2.1 CLcd_ClearDisplay()

```
Std_ReturnType CLcd_ClearDisplay (
    void )
```

Clears the display.

Returns

Std_ReturnType E_OK : If the clear operation started successfully E_NOT_OK : If the clear operation is not able to start right now

4.4.2.2 CLcd_ConfigCursor()

```
Std_ReturnType CLcd_ConfigCursor (
    uint8_t cursor,
    uint8_t blink )
```

Configures the cursor options.

Parameters

<i>cursor</i>	The State of the cursor (Visible or not) CLCD_CURSOR_ON CLCD_CURSOR_OFF
<i>blink</i>	The blinking option (no/off) CLCD_BLINKING_ON CLCD_BLINKING_OFF

Returns

Std_ReturnType E_OK : If the configuration started successfully E_NOT_OK : If the configuration is not able to start right now

4.4.2.3 CLcd_ConfigDisplay()

```
Std_ReturnType CLcd_ConfigDisplay (
    uint8_t disp )
```

Sets the display on and off.

Parameters

<i>disp</i>	the display state CLCD_DISP_ON CLCD_DISP_OFF
-------------	--

Returns

Std_ReturnType E_OK : If the configuration started successfully E_NOT_OK : If the configuration is not able to start right now

4.4.2.4 CLcd_GotoXY()

```
Std_ReturnType CLcd_GotoXY (
    uint8_t x,
    uint8_t y )
```

jumps to a specific location on the lcd display

Parameters

<i>x</i>	the location on the x-axis
<i>y</i>	the location on the y-axis

Returns

Std_ReturnType E_OK : If the goto operation started successfully E_NOT_OK : If the goto operation is not able to start right now

4.4.2.5 CLcd_Init()

```
Std_ReturnType CLcd_Init (
    uint8_t nLines,
    uint8_t cursor,
    uint8_t blink )
```

The Character LCD initialization.

Parameters

<i>nLines</i>	The number of lines on display CLCD_TWO_LINES : Two lines display CLCD_ONE_LINE : One line display
<i>cursor</i>	The State of the cursor (Visible or not) CLCD_CURSOR_ON CLCD_CURSOR_OFF
<i>blink</i>	The blinking option (no/off) CLCD_BLINKING_ON CLCD_BLINKING_OFF

Returns

Std_ReturnType E_OK : If the initialization started successfully E_NOT_OK : If the initialization is not able to start right now

4.4.2.6 CLcd_SetDoneNotification()

```
Std_ReturnType CLcd_SetDoneNotification (
    lcdCb_t cb )
```

Sets the callback function executed when done.

Parameters

<i>cb</i>	the callback function
-----------	-----------------------

Returns

Std_ReturnType

4.4.2.7 CLcd_Task()

```
void CLcd_Task (
    void )
```

The running task that have to come every 1 milli second.

4.4.2.8 CLcd_WriteString()

```
Std_ReturnType CLcd_WriteString (
    uint8_t * str,
    uint8_t x,
    uint8_t y )
```

Writes a string on a specific location on the lcd display.

Parameters

<i>str</i>	the string to write
<i>x</i>	the location on the x-axis
<i>y</i>	the location on the y-axis

Returns

Std_ReturnType E_OK : If the writing started successfully E_NOT_OK : If the write operation is not able to start right now

4.5 CLcd_Cfg.c File Reference

The user's configurations.


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

Variables

- const `clcd_t` `CLcd_clcd`

4.5.1 Detailed Description

The user's configurations.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-03-26

Copyright

Copyright (c) 2020

4.5.2 Variable Documentation

4.5.2.1 CLcd_clcd

```
const clcd_t CLcd_clcd
```

Initial value:

```
= {
    .enPin = GPIO_PIN_2,
    .enPort = GPIO_PORTA,
    .rwPin = GPIO_PIN_1,
    .rwPort = GPIO_PORTA,
    .rsPin = GPIO_PIN_0,
    .rsPort = GPIO_PORTA,
    .dPin = {GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5, GPIO_PIN_6},
    .dPort = {GPIO_PORTA, GPIO_PORTA, GPIO_PORTA, GPIO_PORTA}
}
```

4.6 Gpio.c File Reference

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

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

Macros

- #define **GPIO_CR** 0x00
- #define **GPIO_IDR** 0x08
- #define **GPIO_ODR** 0x0C
- #define **GPIO_BSR** 0x10
- #define **GPIO_BRR** 0x14
- #define **GPIO_LCK** 0x18
- #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)

4.6.1 Detailed Description

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

Author

Mark Attia

Date

February 6, 2020

4.6.2 Function Documentation

4.6.2.1 Gpio_InitPins()

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

Initializes pins mode and speed for a specific port.

4.6.2.2 Function: Gpio_InitPins

Parameters

<i>gpio</i>	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

4.6.2.3 Gpio_ReadPin()

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

Reads a value to a pin(0/1)

4.6.2.4 Function: Gpio_ReadPin

Parameters

<i>port</i>	The port you want to read from GPIO_PORTX : The pin number you want to read from
<i>pin</i>	The pin you want to read GPIO_PIN_X : The pin number you want to read //You can OR more than one pin\
<i>state</i>	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

4.6.2.5 Gpio_WritePin()

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

Write a value to a pin(0/1)

4.6.2.6 Function: Gpio_WritePin

Parameters

<i>port</i>	The port you want to configure GPIO_PORTX : The pin number you want to configure
<i>pin</i>	The pin you want to configure GPIO_PIN_X : The pin number you want to configure //You can OR more than one pin\
<i>pinStatus</i>	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

4.7 Gpio.h File Reference

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

Data Structures

- struct [gpio_t](#)

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)

4.7.1 Detailed Description

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

Author

Mark Attia

Date

February 6, 2020

4.7.2 Function Documentation

4.7.2.1 Gpio_InitPins()

```
Std_ReturnType Gpio_InitPins (  
    gpio\_t * gpio )
```

Initializes pins mode and speed for a specific port.

4.7.2.2 Function: Gpio_InitPins

Parameters

<i>gpio</i>	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

4.7.2.3 Function: Gpio_InitPins**Parameters**

<i>gpio</i>	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

4.7.2.4 Gpio_ReadPin()

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

Reads a value to a pin(0/1)

4.7.2.5 Function: Gpio_ReadPin**Parameters**

<i>port</i>	The port you want to read from GPIO_PORTX : The pin number you want to read from
<i>pin</i>	The pin you want to read GPIO_PIN_X : The pin number you want to read //You can OR more than one pin\
<i>state</i>	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

4.7.2.6 Function: Gpio_ReadPin

Parameters

<i>port</i>	The port you want to read from GPIO_PORTX : The pin number you want to read from
<i>pin</i>	The pin you want to read GPIO_PIN_X : The pin number you want to read //You can OR more than one pin\
<i>state</i>	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

4.7.2.7 Gpio_WritePin()

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

Write a value to a pin(0/1)

4.7.2.8 Function: Gpio_WritePin

Parameters

<i>port</i>	The port you want to configure GPIO_PORTX : The pin number you want to configure
<i>pin</i>	The pin you want to configure GPIO_PIN_X : The pin number you want to configure //You can OR more than one pin\
<i>pinStatus</i>	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

4.7.2.9 Function: Gpio_WritePin

Parameters

<i>port</i>	The port you want to configure GPIO_PORTX : The pin number you want to configure
<i>pin</i>	The pin you want to configure GPIO_PIN_X : The pin number you want to configure //You can OR more than one pin\
<i>pinStatus</i>	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

4.8 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.

4.8.1 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

4.8.2 Function Documentation

4.8.2.1 HRcc_EnPortClock()

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

This function initializes the clock for a specific GPIO port.

Parameters

<i>port</i>	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

4.8.2.2 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

4.9 HUART.c File Reference

This is the implementation for the UART handler.

```
#include "Std_Types.h"
#include "Uart.h"
#include "HUART_Cfg.h"
#include "HUART.h"
#include "NVIC.h"
#include "RCC.h"
#include "Gpio.h"
```

Data Structures

- struct [hUartConfig_t](#)

Macros

- #define **HUART_DEFAULT_MODULE** HUART_MODULE_1
- #define **UART_NUMBER_OF_MODULES** 5
- #define **HUART_NOT_INITIALIZED** 1
- #define **HUART_INITIALIZED** 0
- #define **HUART_NOT_CONFIGURED** 0
- #define **HUART_CONFIGURED** 1

Functions

- Std_ReturnType [HUart_Init](#) (void)
Initializes the UART Module.
- Std_ReturnType [HUart_Config](#) (uint32_t baudRate, uint32_t stopBits, uint32_t parity, uint32_t flowControl)
*Sets configurations for the UART module *The UART must be initialized after setting configurations to apply the changes.*
- Std_ReturnType [HUart_SetModule](#) (uint8_t uartModule)
Sets the module that you will be using.
- Std_ReturnType [HUart_Send](#) (uint8_t *data, uint16_t length)
Sends data through the UART.
- Std_ReturnType [HUart_Receive](#) (uint8_t *data, uint16_t length)
Receives data through the UART.
- Std_ReturnType [HUart_SetRxCb](#) (hUartRxCb_t func)
Sets the callback function that will be called when receive is completed.
- Std_ReturnType [HUart_SetTxCb](#) (hUartTxCb_t func)
Sets the callback function that will be called when transmission is completed.

4.9.1 Detailed Description

This is the implementation for the UART handler.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-03-29

Copyright

Copyright (c) 2020

4.9.2 Function Documentation

4.9.2.1 HUart_Config()

```
Std_ReturnType HUart_Config (
    uint32_t baudRate,
    uint32_t stopBits,
    uint32_t parity,
    uint32_t flowControl )
```

Sets configurations for the UART module *The UART must be initialized after setting configurations to apply the changes.

Parameters

<i>baudRate</i>	the baud rate of the UART (uint32_t)
<i>stopBits</i>	The number of the stop bits HUART_ONE_STOP_BIT HUART_TWO_STOP_BITS
<i>parity</i>	The parity of the transmission HUART_ODD_PARITY HUART_EVEN_PARITY HUART_NO_PARITY
<i>flowControl</i>	the flow control HUART_FLOW_CONTROL_EN HUART_FLOW_CONTROL_DIS

Returns

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

4.9.2.2 HUart_Init()

```
Std_ReturnType HUart_Init (
    void )
```

Initializes the UART Module.

Returns

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

4.9.2.3 HUart_Receive()

```
Std_ReturnType HUart_Receive (
    uint8_t * data,
    uint16_t length )
```

Receives data through the UART.

Parameters

<i>data</i>	The buffer to receive data in
<i>length</i>	the length of the data in bytes

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

4.9.2.4 HUart_Send()

```
Std_ReturnType HUart_Send (
    uint8_t * data,
    uint16_t length )
```

Sends data through the UART.

Parameters

<i>data</i>	The data to send
<i>length</i>	the length of the data in bytes

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

4.9.2.5 HUart_SetModule()

```
Std_ReturnType HUart_SetModule (
    uint8_t uartModule )
```

Sets the module that you will be using.

Parameters

<i>uartModule</i>	The UART module HUART_MODULE_1 HUART_MODULE_2 HUART_MODULE_3 HUART_MODULE_4 HUART_MODULE_5
-------------------	--

Returns

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

4.9.2.6 HUart_SetRxCb()

```
Std_ReturnType HUart_SetRxCb (
    hUartRxCb_t func )
```

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

Parameters

<i>func</i>	the callback function
-------------	-----------------------

Returns

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

4.9.2.7 HUART_SetTxCb()

```
Std_ReturnType HUART_SetTxCb (
    hUART_TxCb_t func )
```

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

Parameters

<i>func</i>	the callback function
-------------	-----------------------

Returns

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

4.10 HUART.h File Reference

This is the user interface for the uart handler.

Macros

- #define **HUART_MODULE_1** 0
- #define **HUART_MODULE_2** 1
- #define **HUART_MODULE_3** 2
- #define **HUART_MODULE_4** 3
- #define **HUART_MODULE_5** 4
- #define **HUART_ODD_PARITY** 0x00000200
- #define **HUART_EVEN_PARITY** 0x00000000
- #define **HUART_NO_PARITY** 0xFFFFFBFF
- #define **HUART_STOP_ONE_BIT** 0x00000000
- #define **HUART_STOP_TWO_BITS** 0x00003000
- #define **HUART_FLOW_CONTROL_EN** 0x00000100
- #define **HUART_FLOW_CONTROL_DIS** 0x00000000

Typedefs

- typedef void(* **hUART_TxCb_t**) (void)
- typedef void(* **hUART_RxCb_t**) (void)

Functions

- Std_ReturnType [HUart_Init](#) (void)
Initializes the UART Module.
- Std_ReturnType [HUart_Config](#) (uint32_t baudRate, uint32_t stopBits, uint32_t parity, uint32_t flowControl)
*Sets configurations for the UART module *The UART must be initialized after setting configurations to apply the changes.*
- Std_ReturnType [HUart_SetModule](#) (uint8_t uartModule)
Sets the module that you will be using.
- Std_ReturnType [HUart_Send](#) (uint8_t *data, uint16_t length)
Sends data through the UART.
- Std_ReturnType [HUart_Receive](#) (uint8_t *data, uint16_t length)
Receives data through the UART.
- Std_ReturnType [HUart_SetRxCb](#) (hUartRxCb_t func)
Sets the callback function that will be called when receive is completed.
- Std_ReturnType [HUart_SetTxCb](#) (hUartTxCb_t func)
Sets the callback function that will be called when transmission is completed.

4.10.1 Detailed Description

This is the user interface for the uart handler.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-03-29

Copyright

Copyright (c) 2020

4.10.2 Function Documentation

4.10.2.1 HUart_Config()

```
Std_ReturnType HUart_Config (
    uint32_t baudRate,
    uint32_t stopBits,
    uint32_t parity,
    uint32_t flowControl )
```

Sets configurations for the UART module *The UART must be initialized after setting configurations to apply the changes.

Parameters

<i>baudRate</i>	the baud rate of the UART (uint32_t)
<i>stopBits</i>	The number of the stop bits HUART_ONE_STOP_BIT HUART_TWO_STOP_BITS
<i>parity</i>	The parity of the transmission HUART_ODD_PARITY HUART_EVEN_PARITY HUART_NO_PARITY
<i>flowControl</i>	the flow control HUART_FLOW_CONTROL_EN HUART_FLOW_CONTROL_DIS

Returns

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

4.10.2.2 HUART_Init()

```
Std_ReturnType HUART_Init (  
    void )
```

Initializes the UART Module.

Returns

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

4.10.2.3 HUART_Receive()

```
Std_ReturnType HUART_Receive (  
    uint8_t * data,  
    uint16_t length )
```

Receives data through the UART.

Parameters

<i>data</i>	The buffer to receive data in
<i>length</i>	the length of the data in bytes

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

4.10.2.4 HUART_Send()

```
Std_ReturnType HUART_Send (
    uint8_t * data,
    uint16_t length )
```

Sends data through the UART.

Parameters

<i>data</i>	The data to send
<i>length</i>	the length of the data in bytes

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

4.10.2.5 HUART_SetModule()

```
Std_ReturnType HUART_SetModule (
    uint8_t uartModule )
```

Sets the module that you will be using.

Parameters

<i>uartModule</i>	The UART module HUART_MODULE_1 HUART_MODULE_2 HUART_MODULE_3 HUART_MODULE_4 HUART_MODULE_5
-------------------	--

Returns

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

4.10.2.6 HUART_SetRxCb()

```
Std_ReturnType HUART_SetRxCb (
    hUartRxCb_t func )
```

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

Parameters

<i>func</i>	the callback function
-------------	-----------------------

Returns

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

4.10.2.7 HUART_SetTxCb()

```
Std_ReturnType HUART_SetTxCb (
    hUART_TxCb_t func )
```

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

Parameters

<i>func</i>	the callback function
-------------	-----------------------

Returns

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

4.11 HUART_Cfg.h File Reference

These are the user's configurations for the HUART driver.

Macros

- #define **HUART_SYSTEM_CLK** 8000000
- #define **HUART_DEFAULT_BAUDRATE** 9600
- #define **HUART_DEFAULT_STOP_BITS** HUART_STOP_ONE_BIT
- #define **HUART_DEFAULT_PARITY** HUART_NO_PARITY
- #define **HUART_DEFAULT_FLOW_CONTROL** HUART_FLOW_CONTROL_DIS
- #define **HUART_DEFAULT_MODULE** HUART_MODULE_1

4.11.1 Detailed Description

These are the user's configurations for the HUART driver.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-03-27

Copyright

Copyright (c) 2020

4.12 Led.c File Reference

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

```
#include "Std_Types.h"
#include "Gpio.h"
#include "Hrcc.h"
#include "Led_Cfg.h"
#include "Led.h"
```

Functions

- Std_ReturnType [Led_Init](#) (void)
Initializes GPIOs for the LEDs.
- Std_ReturnType [Led_SetLedOn](#) (uint8_t ledName)
Sets the Led on.
- Std_ReturnType [Led_SetLedOff](#) (uint8_t ledName)
Sets the Led off.
- Std_ReturnType [Led_SetLedStatus](#) (uint8_t ledName, uint8_t status)
Sets the Led off.

Variables

- const [led_t](#) **Led_Ids** [LED_NUMBER_OF_LEDS]

4.12.1 Detailed Description

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

Author

Mark Attia

Date

January 22, 2020

4.12.2 Function Documentation

4.12.2.1 Led_Init()

```
Std_ReturnType Led_Init (  
    void )
```

Initializes GPIOs for the LEDs.

4.12.2.2 Function: Led_Init

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 Led_SetLedOff()

```
Std_ReturnType Led_SetLedOff (
    uint8_t ledName )
```

Sets the Led off.

4.12.2.4 Function: Led_SetLedOff

Parameters

<i>ledName</i>	The name of the LED
----------------	---------------------

Returns

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

4.12.2.5 Led_SetLedOn()

```
Std_ReturnType Led_SetLedOn (
    uint8_t ledName )
```

Sets the Led on.

4.12.2.6 Function: Led_SetLedOn

Parameters

<i>ledName</i>	The name of the LED
----------------	---------------------

Returns

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

4.12.2.7 Led_SetLedStatus()

```
Std_ReturnType Led_SetLedStatus (
    uint8_t ledName,
    uint8_t status )
```

Sets the Led off.

4.12.2.8 Function: Led_SetLedStatus

Parameters

<i>ledName</i>	The name of the LED
<i>pinStatus</i>	The status of the pin (GPIO_PIN_SET/GPIO_PIN_RESET) LED_ON : Sets the pin value to 1 LED_OFF : 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

4.13 Led.h File Reference

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

Data Structures

- struct [led_t](#)

Macros

- #define **LED_ON** 0
- #define **LED_OFF** !LED_ON

Functions

- Std_ReturnType [Led_Init](#) (void)
Initializes GPIOs for the LEDs.
- Std_ReturnType [Led_SetLedOn](#) (uint8_t ledName)
Sets the Led on.
- Std_ReturnType [Led_SetLedOff](#) (uint8_t ledName)
Sets the Led off.
- Std_ReturnType [Led_SetLedStatus](#) (uint8_t ledName, uint8_t status)
Sets the Led off.

4.13.1 Detailed Description

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

Author

Mark Attia

Date

January 22, 2020

4.13.2 Function Documentation

4.13.2.1 Led_Init()

```
Std_ReturnType Led_Init (  
    void )
```

Initializes GPIOs for the LEDs.

4.13.2.2 Function: Led_Init

Returns

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

4.13.2.3 Function: Led_Init

Returns

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

4.13.2.4 Led_SetLedOff()

```
Std_ReturnType Led_SetLedOff (  
    uint8_t ledName )
```

Sets the Led off.

4.13.2.5 Function: Led_SetLedOff

Parameters

<i>ledName</i>	The name of the LED
----------------	---------------------

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: Led_SetLedOff**Parameters**

<i>ledName</i>	The name of the LED
----------------	---------------------

Returns

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

4.13.2.7 Led_SetLedOn()

```
Std_ReturnType Led_SetLedOn (  
    uint8_t ledName )
```

Sets the Led on.

4.13.2.8 Function: Led_SetLedOn**Parameters**

<i>ledName</i>	The name of the LED
----------------	---------------------

Returns

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

4.13.2.9 Function: Led_SetLedOn**Parameters**

<i>ledName</i>	The name of the LED
----------------	---------------------

Returns

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

4.13.2.10 Led_SetLedStatus()

```
Std_ReturnType Led_SetLedStatus (
    uint8_t ledName,
    uint8_t status )
```

Sets the Led off.

4.13.2.11 Function: Led_SetLedStatus**Parameters**

<i>ledName</i>	The name of the LED
<i>pinStatus</i>	The status of the pin (GPIO_PIN_SET/GPIO_PIN_RESET) LED_ON : Sets the pin value to 1 LED_OFF : 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

4.13.2.12 Function: Led_SetLedStatus**Parameters**

<i>ledName</i>	The name of the LED
<i>pinStatus</i>	The status of the pin (GPIO_PIN_SET/GPIO_PIN_RESET) LED_ON : Sets the pin value to 1 LED_OFF : 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

4.14 Led_Cfg.c File Reference

Those are the User's configurations for the LED Driver.

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

Variables

- const `led_t` **Led_leds** [LED_NUMBER_OF_LEDS]

4.14.1 Detailed Description

Those are the User's configurations for the LED Driver.

Author

Mark Attia (markjosephattia@gmail.com)

Version

0.1

Date

2020-03-28

Copyright

Copyright (c) 2020

4.14.2 Variable Documentation

4.14.2.1 Led_leds

```
const led_t Led_leds[LED_NUMBER_OF_LEDS]
```

Initial value:

```
= {  
    {GPIO_PIN_13, GPIO_PORTC, GPIO_PIN_RESET}  
}
```

4.15 Led_Cfg.h File Reference

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

Macros

- `#define LED_NUMBER_OF_LEDS 1`
- `#define LED_1 0`
- `#define LED_2 1`
- `#define LED_3 2`

4.15.1 Detailed Description

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

Author

Mark Attia

Date

January 22, 2020

4.16 main.c File Reference

Here is the implementation for the main function fo the application and also the tasks.

```
#include "Std_Types.h"
#include "SCHED1.h"
#include "HRcc.h"
#include "CLcd.h"
#include "App.h"
#include "Switch.h"
```

Functions

- void **main** (void)

Variables

- Task **t1** = {APP_sendTask, 1000, 2}
- Task **t2** = {CLcd_Task, 1000, 1}
- Task **t3** = {Switch_Task, 1000, 0}

4.16.1 Detailed Description

Here is the implementation for the main function fo the application and also the tasks.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-28

Copyright

Copyright (c) 2020

4.17 NVIC.c File Reference

This is the implementation for the NVIC Driver.

```
#include "Std_Types.h"
#include "NVIC.h"
```

Data Structures

- struct [NVIC_regMap](#)

Macros

- #define **NVIC_BASE_ADDRESS** 0xE000E100
- #define **NVIC_peripheral** ((volatile [NVIC_regMap](#) *) NVIC_BASE_ADDRESS)
- #define **NVIC_IPR_SETMASK** 0x000000ff

Functions

- void [NVIC_controlInterrupt](#) (u8 interruptNum, u8 status)
Sets and resets the interrupts.
- void [NVIC_controlPendingFlag](#) (u8 interruptNum, u8 val)
Sets and resets The pending flag.
- u8 [NVIC_getActiveFlagStatus](#) (u8 interruptNum)
Gets the active flag state.
- void [NVIC_configurePriority](#) (u8 interruptNum, u8 priority)
Configures the periority of the interrupt.
- u8 [NVIC_getPriority](#) (u8 interruptNum)
Gets the priority of the interrupt.
- void [NVIC_controlAllPeripheral](#) (u8 status)
Controls All of the prephirals.
- void [NVIC_controlFault](#) (u8 status)
Controls The Fault flag.
- void [NVIC_filterInterrupts](#) (u8 priority)
Filters the interrupt.

4.17.1 Detailed Description

This is the implementation for the NVIC Driver.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-28

Copyright

Copyright (c) 2020

4.17.2 Function Documentation

4.17.2.1 NVIC_configurePriority()

```
void NVIC_configurePriority (
    u8 interruptNum,
    u8 priority )
```

Configures the periority of the interrupt.

Parameters

<i>interruptNum</i>	the number of the interrupt
<i>priority</i>	The periority

4.17.2.2 NVIC_controlAllPeripheral()

```
void NVIC_controlAllPeripheral (
    u8 status )
```

Controls All of the prephirals.

Parameters

<i>status</i>	NVIC_ENABLE NVIC_DISABLE
---------------	--------------------------

4.17.2.3 NVIC_controlFault()

```
void NVIC_controlFault (
    u8 status )
```

Controls The Fault flag.

Parameters

<i>status</i>	NVIC_ENABLE NVIC_DISABLE
---------------	--------------------------

4.17.2.4 NVIC_controlInterrupt()

```
void NVIC_controlInterrupt (
    u8 interruptNum,
    u8 status )
```

Sets and resets the interrupts.

Parameters

<i>interruptNum</i>	The Interrupt number
<i>status</i>	The state NVIC_DISABLE NVIC_ENABLE

4.17.2.5 NVIC_controlPendingFlag()

```
void NVIC_controlPendingFlag (
    u8 interruptNum,
    u8 val )
```

Sets and resets The pending flag.

Parameters

<i>interruptNum</i>	The Interrupt number
<i>val</i>	the value to be set NVIC_RESET NVIC_SET

4.17.2.6 NVIC_filterInterrupts()

```
void NVIC_filterInterrupts (
    u8 priority )
```

Filters the interrupt.

Parameters

<i>priority</i>	the priority of the interrupt
-----------------	-------------------------------

4.17.2.7 NVIC_getActiveFlagStatus()

```
u8 NVIC_getActiveFlagStatus (
    u8 interruptNum )
```

Gets the active flag state.

Parameters

<i>interruptNum</i>	the number of the interrupt
---------------------	-----------------------------

Returns

u8

4.17.2.8 NVIC_getPriority()

```
u8 NVIC_getPriority (
    u8 interruptNum )
```

Gets the priority of the interrupt.

Parameters

<i>interruptNum</i>	the number of the interrupt
---------------------	-----------------------------

Returns

u8

4.18 NVIC.h File Reference

This is the user interface for the NVIC driver.

Macros

- #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
- `#define NVIC_DISABLE` 0
- `#define NVIC_ENABLE` 1
- `#define NVIC_RESET` 0
- `#define NVIC_SET` 1

Functions

- void [NVIC_controlInterrupt](#) (u8 interruptNum, u8 status)
Sets and resets the interrupts.
- void [NVIC_controlPendingFlag](#) (u8 interruptNum, u8 val)
Sets and resets The pending flag.
- u8 [NVIC_getActiveFlagStatus](#) (u8 interruptNum)
Gets the active flag state.
- void [NVIC_configurePriority](#) (u8 interruptNum, u8 priority)
Configures the periority of the interrupt.
- u8 [NVIC_getPriority](#) (u8 interruptNum)
Gets the priority of the interrupt.
- void [NVIC_controlAllPeripheral](#) (u8 status)
Controls All of the prephirals.
- void [NVIC_controlFault](#) (u8 status)
Controls The Fault flag.
- void [NVIC_filterInterrupts](#) (u8 priority)
Filters the interrupt.

4.18.1 Detailed Description

This is the user interface for the NVIC driver.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-29

Copyright

Copyright (c) 2020

4.18.2 Macro Definition Documentation

4.18.2.1 NVIC_DISABLE

```
#define NVIC_DISABLE 0
```

status

4.18.2.2 NVIC_IRQNUM_WWDG

```
#define NVIC_IRQNUM_WWDG 0
```

interruptNum

4.18.2.3 NVIC_RESET

```
#define NVIC_RESET 0
```

val

4.18.3 Function Documentation

4.18.3.1 NVIC_configurePriority()

```
void NVIC_configurePriority (
    u8 interruptNum,
    u8 priority )
```

Configures the periority of the interrupt.

Parameters

<i>interruptNum</i>	the number of the interrupt
<i>priority</i>	The periority

4.18.3.2 NVIC_controlAllPeripheral()

```
void NVIC_controlAllPeripheral (
    u8 status )
```

Controls All of the prephirals.

Parameters

<i>status</i>	NVIC_ENABLE NVIC_DISABLE
---------------	--------------------------

4.18.3.3 NVIC_controlFault()

```
void NVIC_controlFault (
```

```
u8 status )
```

Controls The Fault flag.

Parameters

<i>status</i>	NVIC_ENABLE NVIC_DISABLE
---------------	--------------------------

4.18.3.4 NVIC_controlInterrupt()

```
void NVIC_controlInterrupt (
    u8 interruptNum,
    u8 status )
```

Sets and resets the interrupts.

Parameters

<i>interruptNum</i>	The Interrupt number
<i>status</i>	The state NVIC_DISABLE NVIC_ENABLE

4.18.3.5 NVIC_controlPendingFlag()

```
void NVIC_controlPendingFlag (
    u8 interruptNum,
    u8 val )
```

Sets and resets The pending flag.

Parameters

<i>interruptNum</i>	The Interrupt number
<i>val</i>	the value to be set NVIC_RESET NVIC_SET

4.18.3.6 NVIC_filterInterrupts()

```
void NVIC_filterInterrupts (
    u8 priority )
```

Filters the interrupt.

Parameters

<i>priority</i>	the priority of the interrupt
-----------------	-------------------------------

4.18.3.7 NVIC_getActiveFlagStatus()

```
u8 NVIC_getActiveFlagStatus (
    u8 interruptNum )
```

Gets the active flag state.

Parameters

<i>interruptNum</i>	the number of the interrupt
---------------------	-----------------------------

Returns

u8

4.18.3.8 NVIC_getPriority()

```
u8 NVIC_getPriority (
    u8 interruptNum )
```

Gets the priority of the interrupt.

Parameters

<i>interruptNum</i>	the number of the interrupt
---------------------	-----------------------------

Returns

u8

4.19 RCC.h File Reference

This is the user interface for the RCC Driver.

Macros

- #define **ENABLE** 1

- `#define DISABLE 0`
- `#define RCC_DMA1 0x00000001`
- `#define RCC_DMA2 0x00000002`
- `#define RCC_SRAM 0x00000004`
- `#define RCC_FLITF 0x00000010`
- `#define RCC_CRC 0x00000040`
- `#define RCC_FSMC 0x00000100`
- `#define RCC_SDIO 0x00000400`
- `#define RCC_AFIO 0x00000001`
- `#define RCC_GPIOA 0x00000004`
- `#define RCC_GPIOB 0x00000008`
- `#define RCC_GPIOC 0x00000010`
- `#define RCC_GPIOD 0x00000020`
- `#define RCC_GPIOE 0x00000040`
- `#define RCC_GPIOF 0x00000080`
- `#define RCC_GPIOG 0x00000100`
- `#define RCC_ADC1 0x00000200`
- `#define RCC_ADC2 0x00000400`
- `#define RCC_TIM1 0x00000800`
- `#define RCC_SPI1 0x00001000`
- `#define RCC_TIM8 0x00002000`
- `#define RCC_USART1 0x00004000`
- `#define RCC_ADC3 0x00008000`
- `#define RCC_TIM9 0x00080000`
- `#define RCC_TIM10 0x00100000`
- `#define RCC_TIM11 0x00200000`
- `#define RCC_TIM2 0x00000001`
- `#define RCC_TIM3 0x00000002`
- `#define RCC_TIM4 0x00000004`
- `#define RCC_TIM5 0x00000008`
- `#define RCC_TIM6 0x00000010`
- `#define RCC_TIM7 0x00000020`
- `#define RCC_TIM12 0x00000040`
- `#define RCC_TIM13 0x00000080`
- `#define RCC_TIM14 0x00000100`
- `#define RCC_WWDG 0x00000800`
- `#define RCC_SPI2_I2S 0x00004000`
- `#define RCC_SPI3_I2S 0x00008000`
- `#define RCC_USART2 0x00020000`
- `#define RCC_USART3 0x00040000`
- `#define RCC_UART4 0x00080000`
- `#define RCC_UART5 0x00100000`
- `#define RCC_I2C1 0x00200000`
- `#define RCC_I2C2 0x00400000`
- `#define RCC_USB 0x00800000`
- `#define RCC_CAN 0x02000000`
- `#define RCC_BKP 0x08000000`
- `#define RCC_PWR 0x10000000`
- `#define RCC_DAC 0x20000000`
- `#define RCC_sysClk_HSI 0x00000000`
- `#define RCC_sysClk_HSE 0x00000001`
- `#define RCC_sysClk_PLL 0x00000002`
- `#define RCC_OFF 0`
- `#define RCC_ON 1`
- `#define RCC_HSI_ON 0x00000001`

- `#define RCC_HSE_ON 0x00010000`
- `#define RCC_PLL_ON 0x01000000`
- `#define RCC_PLLSRC_HSI 0x00000000`
- `#define RCC_PLLSRC_HSE 0x00010000`
- `#define RCC_PLLSRC_HSE_DIV_2 0x00030000`
- `#define RCC_PLLMUL_SPEED_2 0x00000000`
- `#define RCC_PLLMUL_SPEED_3 0x00040000`
- `#define RCC_PLLMUL_SPEED_4 0x00080000`
- `#define RCC_PLLMUL_SPEED_5 0x000C0000`
- `#define RCC_PLLMUL_SPEED_6 0x00100000`
- `#define RCC_PLLMUL_SPEED_7 0x00140000`
- `#define RCC_PLLMUL_SPEED_8 0x00180000`
- `#define RCC_PLLMUL_SPEED_9 0x001C0000`
- `#define RCC_PLLMUL_SPEED_10 0x00200000`
- `#define RCC_PLLMUL_SPEED_11 0x00240000`
- `#define RCC_PLLMUL_SPEED_12 0x00280000`
- `#define RCC_PLLMUL_SPEED_13 0x002C0000`
- `#define RCC_PLLMUL_SPEED_14 0x00300000`
- `#define RCC_PLLMUL_SPEED_15 0x00340000`
- `#define RCC_PLLMUL_SPEED_16 0x00380000`
- `#define RCC_USB_PRESCALER 0x00400000`
- `#define RCC_ADC_PRESCALER 0x0000C000`
- `#define RCC_APB2_PRESCALER 0x00003800`
- `#define RCC_APB1_PRESCALER 0x00000700`
- `#define RCC_AHB_PRESCALER 0x000000F0`
- `#define RCC_USB_DIVIDED 0x00000000`
- `#define RCC_USB_NDIVIDED 0x00400000`
- `#define RCC_ADC_DIV_2 0x00000000`
- `#define RCC_ADC_DIV_4 0x00004000`
- `#define RCC_ADC_DIV_6 0x00008000`
- `#define RCC_ADC_DIV_8 0x0000C000`
- `#define RCC_APB2_NDIVIDED 0x00000000`
- `#define RCC_APB2_DIV_2 0x00002000`
- `#define RCC_APB2_DIV_4 0x00002800`
- `#define RCC_APB2_DIV_8 0x00003000`
- `#define RCC_APB2_DIV_16 0x00003800`
- `#define RCC_APB1_NDIVIDED 0x00000000`
- `#define RCC_APB1_DIV_2 0x00000400`
- `#define RCC_APB1_DIV_4 0x00000500`
- `#define RCC_APB1_DIV_8 0x00000600`
- `#define RCC_APB1_DIV_16 0x00000700`
- `#define RCC_AHB_NDIVIDED 0x00000000`
- `#define RCC_AHB_DIV_2 0x00000080`
- `#define RCC_AHB_DIV_4 0x00000090`
- `#define RCC_AHB_DIV_8 0x000000A0`
- `#define RCC_AHB_DIV_16 0x000000B0`
- `#define RCC_AHB_DIV_64 0x000000C0`
- `#define RCC_AHB_DIV_128 0x000000D0`
- `#define RCC_AHB_DIV_256 0x000000E0`
- `#define RCC_AHB_DIV_512 0x000000F0`
- `#define RCC_MCO_NOSRC 0x00000000`
- `#define RCC_MCO_SYSCLK 0x04000000`
- `#define RCC_MCO_HSI 0x05000000`
- `#define RCC_MCO_HSE 0x06000000`
- `#define RCC_MCO_PLL 0x07000000`

Functions

- void [RCC_controlAHBPeripheral](#) (u32 peripheralNum, u32 status)
- void [RCC_controlAPB2Peripheral](#) (u32 peripheralNum, u32 status)
- void [RCC_controlAPB1Peripheral](#) (u32 peripheralNum, u32 status)
- void [RCC_selectSystemClock](#) (u32 sysClkNum)
- void [RCC_setClockState](#) (u32 clkNum, u32 status)
- void [RCC_configurePLL](#) (u32 plISrc, u32 speedMul)
- void [RCC_configurePrescalers](#) (u32 target, u32 preValue)
- void [RCC_configureMCO](#) (u32 clkNum)

4.19.1 Detailed Description

This is the user interface for the RCC Driver.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-28

Copyright

Copyright (c) 2020

4.19.2 Function Documentation

4.19.2.1 [RCC_configureMCO\(\)](#)

```
void RCC_configureMCO (  
    u32 clkNum )
```

Function Name: [RCC_configureMCO](#) Usage: configure MCO source Function Arguments: u32 clkNum - takes one of these values [RCC_MCO_NOSRC](#) [RCC_MCO_SYSCLK](#) [RCC_MCO_HSI](#) [RCC_MCO_HSE](#) [RCC_MCO_PLL](#)

4.19.2.2 RCC_configurePLL()

```
void RCC_configurePLL (
    u32 pllSrc,
    u32 speedMul )
```

Function Name: RCC_configurePLL Usage: configure PLL source & speed Function Arguments: u32 pllSrc - takes one of these values RCC_PLLSRC_HSI
RCC_PLLSRC_HSE
RCC_PLLSRC_HSE_DIV_2

u32 speedMul - takes one of these values RCC_PLLMUL_SPEED_2 RCC_PLLMUL_SPEED_3 RCC_PLLMUL_SPEED_4 RCC_PLLMUL_SPEED_5 RCC_PLLMUL_SPEED_6 RCC_PLLMUL_SPEED_7 RCC_PLLMUL_SPEED_8 RCC_PLLMUL_SPEED_9 RCC_PLLMUL_SPEED_10 RCC_PLLMUL_SPEED_11 RCC_PLLMUL_SPEED_12 RCC_PLLMUL_SPEED_13 RCC_PLLMUL_SPEED_14 RCC_PLLMUL_SPEED_15 RCC_PLLMUL_SPEED_16

4.19.2.3 RCC_configurePrescalers()

```
void RCC_configurePrescalers (
    u32 target,
    u32 preValue )
```

Function Name: RCC_configurePrescalers Usage: configure prescalers for a specific target Function Arguments: u32 target - takes one of these values RCC_USB_PRESCALER RCC_ADC_PRESCALER RCC_APB2_PRESCALER RCC_APB1_PRESCALER RCC_AHB_PRESCALER

u32 preValue - takes one of these values RCC_USB_DIVIDED
RCC_USB_NDIVIDED RCC_ADC_DIV_2
RCC_ADC_DIV_4
RCC_ADC_DIV_6
RCC_ADC_DIV_6
RCC_APB2_NDIVIDED RCC_APB2_DIV_2
RCC_APB2_DIV_4
RCC_APB2_DIV_8
RCC_APB2_DIV_16
RCC_APB1_NDIVIDED RCC_APB1_DIV_2
RCC_APB1_DIV_4
RCC_APB1_DIV_8
RCC_APB1_DIV_16
RCC_AHB_NDIVIDED RCC_AHB_DIV_2
RCC_AHB_DIV_4
RCC_AHB_DIV_8
RCC_AHB_DIV_16
RCC_AHB_DIV_64
RCC_AHB_DIV_128 RCC_AHB_DIV_256 RCC_AHB_DIV_512

Function Name: RCC_configurePrescalers Usage: configure prescalers for a specific target Function Arguments: u32 target - takes one of these values RCC_USB_PRESCALER RCC_ADC_PRESCALER RCC_APB2_PRESCALER RCC_APB1_PRESCALER RCC_AHB_PRESCALER

u32 preValue - takes one of these values RCC_USB_DIVIDED
RCC_USB_NDIVIDED RCC_ADC_DIV_2
RCC_ADC_DIV_4
RCC_ADC_DIV_6
RCC_ADC_DIV_6

RCC_APB2_NDIVIDED RCC_APB2_DIV_2
 RCC_APB2_DIV_4
 RCC_APB2_DIV_8
 RCC_APB2_DIV_16
 RCC_APB1_NDIVIDED RCC_APB1_DIV_2
 RCC_APB1_DIV_4
 RCC_APB1_DIV_8
 RCC_APB1_DIV_16

4.19.2.4 RCC_controlAHBPeripheral()

```
void RCC_controlAHBPeripheral (
    u32 peripheralNum,
    u32 status )
```

Function Name: RCC_controlAHBPeripheral Usage: Disable/Enable peripherals on AHB Function Arguments: u32 peripheralNum - takes one of these values RCC_DMA1 RCC_DMA2 RCC_SRAM RCC_FLITF RCC_CRC RCC_FSMC RCC_SDIO

u32 status - takes one of these values ENABLE DISABLE

4.19.2.5 RCC_controlAPB1Peripheral()

```
void RCC_controlAPB1Peripheral (
    u32 peripheralNum,
    u32 status )
```

Function Name: RCC_controlAPB1Peripheral Usage: Disable/Enable peripherals on APB1 Function Arguments: u32 peripheralNum - takes one of these values RCC_TIM2

RCC_TIM3
 RCC_TIM4
 RCC_TIM5
 RCC_TIM6
 RCC_TIM7
 RCC_TIM12
 RCC_TIM13
 RCC_TIM14
 RCC_WWDG
 RCC_SPI2_I2S RCC_SPI3_I2S RCC_USART2
 RCC_USART3
 RCC_UART4
 RCC_UART5
 RCC_I2C1
 RCC_I2C2
 RCC_USB
 RCC_CAN
 RCC_BKP
 RCC_PWR
 RCC_DAC

u32 status - takes one of these values ENABLE DISABLE

4.19.2.6 RCC_controlAPB2Peripheral()

```
void RCC_controlAPB2Peripheral (
    u32 peripheralNum,
    u32 status )
```

Function Name: RCC_controlAPB2Peripheral Usage: Disable/Enable peripherals on APB2 Function Arguments:

u32 peripheralNum - takes one of these values RCC_AFIO

RCC_GPIOA RCC_GPIOB RCC_GPIOC RCC_GPIOD RCC_GPIOE RCC_GPIOF RCC_GPIOG RCC_ADC1

RCC_ADC2

RCC_TIM1

RCC_SPI1

RCC_TIM8

RCC_USART1 RCC_ADC3

RCC_TIM9

RCC_TIM10 RCC_TIM11

u32 status - takes one of these values ENABLE DISABLE

4.19.2.7 RCC_selectSystemClock()

```
void RCC_selectSystemClock (
    u32 sysClkNum )
```

Function Name: RCC_selectSystemClock Usage: Select clock source for the system Function Arguments: u32

sysClkNum - takes one of these values RCC_sysClk_HSI RCC_sysClk_HSE RCC_sysClk_PLL

4.19.2.8 RCC_setClockState()

```
void RCC_setClockState (
    u32 clkNum,
    u32 status )
```

Function Name: RCC_setClockState Usage: RCC_ON/RCC_OFF a clock source Function Arguments: u32

clkNum - takes one of these values RCC_HSI_ON RCC_HSE_ON RCC_PLL_ON

u32 status - takes one of these values RCC_ON RCC_OFF

4.20 SCHED.c File Reference

This is the implementation of the scheduler.

```
#include "Std_Types.h"
#include "RCC.h"
#include "SYSTICK.h"
#include "SCHED1.h"
#include "SCHED_CONF.h"
```

Data Structures

- struct [SysTask](#)

Functions

- void [SCHED_init](#) (void)
The initialization function.
- void [SCHED_createTask](#) ([Task](#) *appTask)
This function creates a task dynamically in the run time.
- void [SCHED_start](#) (void)
Starts The running scheduel.

4.20.1 Detailed Description

This is the implementation of the scheduler.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-28

Copyright

Copyright (c) 2020

4.20.2 Function Documentation

4.20.2.1 SCHED_createTask()

```
void SCHED_createTask (  
    Task * appTask )
```

This function creates a task dynamically in the run time.

Parameters

<i>appTask</i>	This is the application task desired to create
----------------	--

4.20.2.2 SCHED_init()

```
void SCHED_init (
    void )
```

The initialization function.

4.20.2.3 SCHED_start()

```
void SCHED_start (
    void )
```

Starts The running scheduel.

4.21 SCHED1.h File Reference

This is the user interface for the scheduler.

Data Structures

- struct [Task](#)

Typedefs

- typedef void(* **taskRunnable**) (void)

Functions

- void [SCHED_init](#) (void)
The initialization function.
- void [SCHED_createTask](#) ([Task](#) *appTask)
This function creates a task dynamically in the run time.
- void [SCHED_start](#) (void)
Starts The running scheduel.

4.21.1 Detailed Description

This is the user interface for the scheduler.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-29

Copyright

Copyright (c) 2020

4.21.2 Function Documentation

4.21.2.1 SCHED_createTask()

```
void SCHED_createTask (
    Task * appTask )
```

This function creates a task dynamically in the run time.

Parameters

<i>appTask</i>	This is the application task desired to create
----------------	--

4.21.2.2 SCHED_init()

```
void SCHED_init (
    void )
```

The initialization function.

4.21.2.3 SCHED_start()

```
void SCHED_start (
    void )
```

Starts The running scheduel.

4.22 SCHED_CONF.h File Reference

Those are the configurations for the Scheduler Driver.

Macros

- #define **SCHED_MAX_TASK_NUM** 3
- #define **SCHED_AHB_PREVAL** RCC_AHB_NDIVIDED
- #define **SCHED_AHB_CLOCK** 8000000
- #define **SCHED_TICK_TIME_US** 1000

4.22.1 Detailed Description

Those are the configurations for the Scheduler Driver.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-29

Copyright

Copyright (c) 2020

4.23 Std_Types.h File Reference

Those are the standard types used in the drivers.

Macros

- `#define NULL ((void*)0)`
- `#define E_OK (0)`
- `#define E_NOT_OK (1)`
- `#define STD_LOW (0)`
- `#define STD_HIGH (1)`
- `#define STD_IDLE (0)`
- `#define STD_ACTIVE (1)`
- `#define STD_OFF (0)`
- `#define STD_ON (1)`

Typedefs

- `typedef unsigned char u8`
- `typedef unsigned char uint8_t`
- `typedef signed char s8`
- `typedef signed char sint8_t`
- `typedef unsigned short int u16`
- `typedef unsigned short int uint16_t`
- `typedef signed short int s16`
- `typedef signed short int sint16_t`
- `typedef unsigned long int u32`
- `typedef unsigned long int uint32_t`
- `typedef signed long int s32`
- `typedef signed long int sint32_t`
- `typedef unsigned long long int u64`
- `typedef unsigned long long int uint64_t`
- `typedef signed long long int s64`
- `typedef signed long long int sint64_t`
- `typedef float f32`
- `typedef double f64`
- `typedef void(* callback_t) (void)`
- `typedef uint8_t Std_ReturnType`

4.23.1 Detailed Description

Those are the standard types used in the drivers.

Author

Mark Attia

Version

0.1

Date

2020-03-29

Copyright

Copyright (c) 2020

4.24 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_Cfg.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.
- void [Switch_Task](#) (void)
The running task of the switch driver to get the state of all of the switches.

Variables

- const [switch_t](#) **Switch_switches** [SWITCH_NUMBER_OF_SWITCHES]

4.24.1 Detailed Description

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

Author

Mark Attia

Date

January 22, 2020

4.24.2 Function Documentation

4.24.2.1 Switch_GetSwitchStatus()

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

Gets the status of the switch.

4.24.2.2 Function: Switch_GetSwitchStatus

Parameters

<i>switchName</i>	The name of the Switch
<i>state</i>	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

4.24.2.3 Switch_Init()

```
Std_ReturnType Switch_Init (
    void )
```

Initializes GPIOs for the Switches.

4.24.2.4 Function: Switch_Init**Returns**

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

4.24.2.5 Switch_Task()

```
void Switch_Task (
    void )
```

The running task of the switch driver to get the state of all of the switches.

4.25 Switch.h File Reference

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

Data Structures

- struct [switch_t](#)

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.
- void [Switch_Task](#) (void)
The running task of the switch driver to get the state of all of the switches.

4.25.1 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

4.25.2 Function Documentation

4.25.2.1 Switch_GetSwitchStatus()

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

Gets the status of the switch.

4.25.2.2 Function: Switch_GetSwitchStatus

Parameters

<i>switchName</i>	The name of the Switch
<i>state</i>	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

4.25.2.3 Function: Switch_GetSwitchStatus

Parameters

<i>switchName</i>	The name of the Switch
<i>state</i>	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

4.25.2.4 Switch_Init()

```
Std_ReturnType Switch_Init (
    void )
```

Initializes GPIOs for the Switches.

4.25.2.5 Function: Switch_Init**Returns**

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

4.25.2.6 Function: Switch_Init**Returns**

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

4.25.2.7 Switch_Task()

```
void Switch_Task (
    void )
```

The running task of the switch driver to get the state of all of the switches.

4.26 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]

4.26.1 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

4.26.2 Variable Documentation

4.26.2.1 Switch_switches

```
const switch\_t Switch_switches[SWITCH_NUMBER_OF_SWITCHES]
```

Initial value:

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

4.27 Switch_Cfg.h File Reference

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

Macros

- #define **SWITCH_USE_RTOS**
- #define **SWITCH_NUMBER_OF_SWITCHES** 1
- #define **SWITCH_1** 0
- #define **SWITCH_2** 1
- #define **SWITCH_3** 2

4.27.1 Detailed Description

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

Author

Mark Attia

Date

January 22, 2020

4.28 SYSTICK.c File Reference

This is the SysTick driver implementation.

```
#include "Std_Types.h"
#include "SYSTICK_CONF.h"
#include "SYSTICK.h"
```

Data Structures

- struct [SYSTICK_regMap](#)

Macros

- #define **SYSTICK_BASE_ADDRESS** 0xE000E010
- #define **SYSTICK_peripheral** ((volatile [SYSTICK_regMap](#) *) SYSTICK_BASE_ADDRESS)
- #define **SYSTICK_ENABLE_SETMASK** 0x00000001
- #define **SYSTICK_TICKINT_SETMASK** 0x00000002
- #define **SYSTICK_CLKSRC_SETMASK** 0x00000004

Functions

- void [SYSTICK_init](#) (void)
The initialization of the SysTick.
- void [SYSTICK_start](#) (void)
Starts the SysTick.
- void [SYSTICK_stop](#) (void)
Stops the timer.
- void [SYSTICK_setTime](#) (u32 time, u32 AHB_clockHz)
Sets the timer for a specific time.
- void [SYSTICK_setCallbackFcn](#) (SYSTICK_cbF cbF)
Sets the callback function.
- void [SysTick_Handler](#) (void)
The SysTick Handler.

4.28.1 Detailed Description

This is the SysTick driver implementation.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-28

Copyright

Copyright (c) 2020

4.28.2 Function Documentation

4.28.2.1 SysTick_Handler()

```
void SysTick_Handler (
    void )
```

The SysTick Handler.

4.28.2.2 SYSTICK_init()

```
void SYSTICK_init (
    void )
```

The initialization of the SysTick.

4.28.2.3 SYSTICK_setCallbackFcn()

```
void SYSTICK_setCallbackFcn (
    SYSTICK_cbF cbF )
```

Sets the callback function.

Parameters

<i>cbF</i>	the function to set
------------	---------------------

4.28.2.4 SYSTICK_setTime()

```
void SYSTICK_setTime (
    u32 time,
    u32 AHB_clockHz )
```

Sets the timer for a specific time.

Parameters

<i>time</i>	the time in milli seconds
<i>AHB_clockHz</i>	the AHB clock in Kilo Hz

4.28.2.5 SYSTICK_start()

```
void SYSTICK_start (  
    void )
```

Starts the SysTick.

4.28.2.6 SYSTICK_stop()

```
void SYSTICK_stop (  
    void )
```

Stops the timer.

4.29 SYSTICK.h File Reference

This is the user interface for the SysTick Driver.

Macros

- `#define SYSTICK_CLKSRC_AHB_DIV_8 0x00000000`
- `#define SYSTICK_CLKSRC_AHB 0x00000004`

Typedefs

- `typedef void(* SYSTICK_cbF) (void)`

Functions

- void [SYSTICK_init](#) (void)
The initialization of the SysTick.
- void [SYSTICK_start](#) (void)
Starts the SysTick.
- void [SYSTICK_stop](#) (void)
Stops the timer.
- void [SYSTICK_setTime](#) (u32 time, u32 AHB_clockHz)
Sets the timer for a specific time.
- void [SYSTICK_setCallbackFcn](#) (SYSTICK_cbF cbF)
Sets the callback function.

4.29.1 Detailed Description

This is the user interface for the SysTick Driver.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-29

Copyright

Copyright (c) 2020

4.29.2 Function Documentation

4.29.2.1 SYSTICK_init()

```
void SYSTICK_init (  
    void )
```

The initialization of the SysTick.

4.29.2.2 SYSTICK_setCallbackFcn()

```
void SYSTICK_setCallbackFcn (  
    SYSTICK_cbF cbF )
```

Sets the callback function.

Parameters

<i>cbF</i>	the function to set
------------	---------------------

4.29.2.3 SYSTICK_setTime()

```
void SYSTICK_setTime (
    u32 time,
    u32 AHB_clockHz )
```

Sets the timer for a specific time.

Parameters

<i>time</i>	the time in milli seconds
<i>AHB_clockHz</i>	the AHB clock in Kilo Hz

4.29.2.4 SYSTICK_start()

```
void SYSTICK_start (
    void )
```

Starts the Systick.

4.29.2.5 SYSTICK_stop()

```
void SYSTICK_stop (
    void )
```

Stops the timer.

4.30 SYSTICK_CONF.h File Reference

Those are the configurations for the Systick Driver.

Macros

- `#define SYSTICK_CLKSRC_PRE SYSTICK_CLKSRC_AHB`

4.30.1 Detailed Description

Those are the configurations for the Systick Driver.

Author

Mariam Mohammed

Version

0.1

Date

2020-03-29

Copyright

Copyright (c) 2020

4.31 Uart.c File Reference

This is the implementation for the UART driver.

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

Data Structures

- struct [uart_t](#)
- struct [dataBuffer_t](#)

Macros

- #define **UART_NUMBER_OF_MODULES** 5
- #define **UART_INT_NUMBER** 37
- #define **UART_BUFFER_IDLE** 0
- #define **UART_BUFFER_BUSY** 1
- #define **UART_TXE_CLR** 0xFFFFF7F
- #define **UART_TC_CLR** 0xFFFFFBF
- #define **UART_RXNE_CLR** 0xFFFFFDF
- #define **UART_PE_CLR** 0xFFFFFEE
- #define **UART_DR_CLR** 0xFFFFE00
- #define **UART_STOP_CLR** 0xFFFFCFFF
- #define **UART_TXEIE_CLR** 0xFFFFF7F
- #define **UART_PS_CLR** 0xFFFFDFF
- #define **UART_M_CLR** 0xFFFFEFF
- #define **UART_TXE_GET** 0x0000080

- `#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_RTSE_CLR 0xFFFFFEFF`
- `#define UART_NO_PRESCALER 0x1`

Functions

- void [USART1_IRQHandler](#) (void)
The UART 1 Handler.
- void [USART2_IRQHandler](#) (void)
The UART 2 Handler.
- void [USART3_IRQHandler](#) (void)
The UART 3 Handler.
- void [UART4_IRQHandler](#) (void)
The UART 4 Handler.
- void [UART5_IRQHandler](#) (void)
The UART 5 Handler.
- Std_ReturnType [Uart_Init](#) (uint32_t baudRate, uint32_t stopBits, uint32_t parity, uint32_t flowControl, uint32_t sysClk, uint8_t uartModule)
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_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.

Variables

- const uint32_t **Uart_Address** [UART_NUMBER_OF_MODULES]

4.31.1 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

4.31.2 Function Documentation

4.31.2.1 UART4_IRQHandler()

```
void UART4_IRQHandler (
    void )
```

The UART 4 Handler.

4.31.2.2 UART5_IRQHandler()

```
void UART5_IRQHandler (
    void )
```

The UART 5 Handler.

4.31.2.3 Uart_Init()

```
Std_ReturnType Uart_Init (
    uint32_t baudRate,
    uint32_t stopBits,
    uint32_t parity,
    uint32_t flowControl,
    uint32_t sysClk,
    uint8_t uartModule )
```

Initializes the UART.

Parameters

<i>baudRate</i>	the baud rate of the UART (uint32_t)
<i>stopBits</i>	The number of the stop bits UART_ONE_STOP_BIT UART_TWO_STOP_BITS
<i>parity</i>	The parity of the transmission UART_ODD_PARITY UART_EVEN_PARITY UART_NO_PARITY
<i>flowControl</i>	the flow control UART_FLOW_CONTROL_EN UART_FLOW_CONTROL_DIS
<i>sysClk</i>	the clock of the system
<i>uartModule</i>	the module number of the UART UART1 UART2 UART3 UART4 UART5

Returns

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

4.31.2.4 Uart_Receive()

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

Receives data through the UART.

Parameters

<i>data</i>	The buffer to receive data in
<i>length</i>	the length of the data in bytes
<i>uartModule</i>	the module number of the UART UART1 UART2 UART3 UART4 UART5

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

4.31.2.5 Uart_Send()

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

Sends data through the UART.

Parameters

<i>data</i>	The data to send
<i>length</i>	the length of the data in bytes
<i>uartModule</i>	the module number of the UART UART1 UART2 UART3 UART4 UART5

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

4.31.2.6 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

<i>func</i>	the callback function
<i>uartModule</i>	the module number of the UART UART1 UART2 UART3 UART4 UART5

Returns

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

4.31.2.7 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

<i>func</i>	the callback function
<i>uartModule</i>	the module number of the UART UART1 UART2 UART3 UART4 UART5

Returns

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

4.31.2.8 USART1_IRQHandler()

```
void USART1_IRQHandler (
    void )
```

The UART 1 Handler.

4.31.2.9 USART2_IRQHandler()

```
void USART2_IRQHandler (
    void )
```

The UART 2 Handler.

4.31.2.10 USART3_IRQHandler()

```
void USART3_IRQHandler (
    void )
```

The UART 3 Handler.

4.31.3 Variable Documentation**4.31.3.1 Uart_Address**

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

Initial value:

```
= {
    0x40013800,
    0x40004400,
    0x40004800,
    0x40004C00,
    0x40005000
}
```

4.32 Uart.h File Reference

This is the user interface for the UART driver.

Macros

- `#define UART1 0`
- `#define UART2 1`
- `#define UART3 2`
- `#define UART4 3`
- `#define UART5 4`
- `#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`

Typedefs

- `typedef void(* txCb_t) (void)`
- `typedef void(* rxCb_t) (void)`

Functions

- Std_ReturnType [Uart_Init](#) (uint32_t baudRate, uint32_t stopBits, uint32_t parity, uint32_t flowControl, uint32_t sysClk, uint8_t uartModule)
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_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.

4.32.1 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

4.32.2 Function Documentation

4.32.2.1 Uart_Init()

```
Std_ReturnType Uart_Init (
    uint32_t baudRate,
    uint32_t stopBits,
    uint32_t parity,
    uint32_t flowControl,
    uint32_t sysClk,
    uint8_t uartModule )
```

Initializes the UART.

Parameters

<i>baudRate</i>	the baud rate of the UART (uint32_t)
<i>stopBits</i>	The number of the stop bits UART_ONE_STOP_BIT UART_TWO_STOP_BITS
<i>parity</i>	The parity of the transmission UART_ODD_PARITY UART_EVEN_PARITY UART_NO_PARITY
<i>flowControl</i>	the flow control UART_FLOW_CONTROL_EN UART_FLOW_CONTROL_DIS
<i>sysClk</i>	the clock of the system
<i>uartModule</i>	the module number of the UART UART1 UART2 UART3 UART4 UART5

Returns

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

4.32.2.2 Uart_Receive()

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

Receives data through the UART.

Parameters

<i>data</i>	The buffer to receive data in
<i>length</i>	the length of the data in bytes
<i>uartModule</i>	the module number of the UART UART1 UART2 UART3 UART4 UART5

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

4.32.2.3 Uart_Send()

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

Sends data through the UART.

Parameters

<i>data</i>	The data to send
<i>length</i>	the length of the data in bytes
<i>uartModule</i>	the module number of the UART UART1 UART2 UART3 UART4 UART5

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

4.32.2.4 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

<i>func</i>	the callback function
<i>uartModule</i>	the module number of the UART UART1 UART2 UART3 UART4 UART5

Returns

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

4.32.2.5 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

<i>func</i>	the callback function
<i>uartModule</i>	the module number of the UART UART1 UART2 UART3 UART4 UART5

Returns

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

Index

App.c, [11](#)
 APP_init, [12](#)
 APP_receiveFcn, [12](#)
 APP_sendTask, [12](#)
App.h, [13](#)
 APP_init, [13](#)
 APP_receiveFcn, [14](#)
 APP_sendTask, [14](#)
APP_init
 App.c, [12](#)
 App.h, [13](#)
APP_receiveFcn
 App.c, [12](#)
 App.h, [14](#)
APP_sendTask
 App.c, [12](#)
 App.h, [14](#)

Clcd.c, [14](#)
 CLcd_ClearDisplay, [16](#)
 CLcd_ConfigCursor, [16](#)
 CLcd_ConfigDisplay, [16](#)
 CLcd_GotoXY, [17](#)
 CLcd_Init, [17](#)
 CLcd_SetDoneNotification, [17](#)
 CLcd_Task, [18](#)
 CLcd_WriteString, [18](#)
CLcd.h, [18](#)
 CLcd_ClearDisplay, [20](#)
 CLcd_ConfigCursor, [20](#)
 CLcd_ConfigDisplay, [20](#)
 CLcd_GotoXY, [21](#)
 CLcd_Init, [21](#)
 CLcd_SetDoneNotification, [21](#)
 CLcd_Task, [22](#)
 CLcd_WriteString, [22](#)
CLcd_Cfg.c, [22](#)
 CLcd_clcd, [23](#)
CLcd_clcd
 CLcd_Cfg.c, [23](#)
CLcd_ClearDisplay
 Clcd.c, [16](#)
 CLcd.h, [20](#)
CLcd_ConfigCursor
 Clcd.c, [16](#)
 CLcd.h, [20](#)
CLcd_ConfigDisplay
 Clcd.c, [16](#)
 CLcd.h, [20](#)
CLcd_GotoXY
 Clcd.c, [17](#)
 CLcd.h, [21](#)
 CLcd.c, [17](#)
 CLcd.h, [21](#)
CLcd_Init
 Clcd.c, [17](#)
 CLcd.h, [21](#)
CLcd_SetDoneNotification
 Clcd.c, [17](#)
 CLcd.h, [21](#)
clcd_t, [5](#)
CLcd_Task
 Clcd.c, [18](#)
 CLcd.h, [22](#)
CLcd_WriteString
 Clcd.c, [18](#)
 CLcd.h, [22](#)

dataBuffer_t, [5](#)

frame_t, [6](#)

Gpio.c, [24](#)
 Gpio_InitPins, [24](#)
 Gpio_ReadPin, [25](#)
 Gpio_WritePin, [25](#)
Gpio.h, [26](#)
 Gpio_InitPins, [27](#)
 Gpio_ReadPin, [28](#)
 Gpio_WritePin, [29](#)
Gpio_InitPins
 Gpio.c, [24](#)
 Gpio.h, [27](#)
Gpio_ReadPin
 Gpio.c, [25](#)
 Gpio.h, [28](#)
gpio_t, [6](#)
Gpio_WritePin
 Gpio.c, [25](#)
 Gpio.h, [29](#)

HRcc.h, [30](#)
 HRcc_EnPortClock, [30](#)
 HRcc_SystemClockInit, [31](#)
HRcc_EnPortClock
 HRcc.h, [30](#)
HRcc_SystemClockInit
 HRcc.h, [31](#)
HUART.c, [31](#)
 HUART_Config, [32](#)
 HUART_Init, [33](#)
 HUART_Receive, [33](#)

- HUart_Send, 33
 - HUart_SetModule, 34
 - HUart_SetRxCb, 34
 - HUart_SetTxCb, 35
- HUart.h, 35
 - HUart_Config, 36
 - HUart_Init, 37
 - HUart_Receive, 37
 - HUart_Send, 37
 - HUart_SetModule, 38
 - HUart_SetRxCb, 38
 - HUart_SetTxCb, 39
- HUart_Cfg.h, 39
- HUart_Config
 - HUart.c, 32
 - HUart.h, 36
- HUart_Init
 - HUart.c, 33
 - HUart.h, 37
- HUart_Receive
 - HUart.c, 33
 - HUart.h, 37
- HUart_Send
 - HUart.c, 33
 - HUart.h, 37
- HUart_SetModule
 - HUart.c, 34
 - HUart.h, 38
- HUart_SetRxCb
 - HUart.c, 34
 - HUart.h, 38
- HUart_SetTxCb
 - HUart.c, 35
 - HUart.h, 39
- hUartConfig_t, 6
- Led.c, 40
 - Led_Init, 40
 - Led_SetLedOff, 41
 - Led_SetLedOn, 41
 - Led_SetLedStatus, 41
- Led.h, 42
 - Led_Init, 43
 - Led_SetLedOff, 43
 - Led_SetLedOn, 44
 - Led_SetLedStatus, 44
- Led_Cfg.c, 45
 - Led_leds, 46
- Led_Cfg.h, 46
- Led_Init
 - Led.c, 40
 - Led.h, 43
- Led_leds
 - Led_Cfg.c, 46
- Led_SetLedOff
 - Led.c, 41
 - Led.h, 43
- Led_SetLedOn
 - Led.c, 41
- Led.h, 44
 - Led.c, 41
 - Led.h, 44
- led_t, 7
- main.c, 47
- NVIC.c, 48
 - NVIC_configurePriority, 49
 - NVIC_controlAllPeripheral, 49
 - NVIC_controlFault, 49
 - NVIC_controlInterrupt, 49
 - NVIC_controlPendingFlag, 50
 - NVIC_filterInterrupts, 50
 - NVIC_getActiveFlagStatus, 50
 - NVIC_getPriority, 52
- NVIC.h, 52
 - NVIC_configurePriority, 55
 - NVIC_controlAllPeripheral, 55
 - NVIC_controlFault, 55
 - NVIC_controlInterrupt, 56
 - NVIC_controlPendingFlag, 56
 - NVIC_DISABLE, 54
 - NVIC_filterInterrupts, 56
 - NVIC_getActiveFlagStatus, 57
 - NVIC_getPriority, 57
 - NVIC_IRQNUM_WWDG, 54
 - NVIC_RESET, 55
- NVIC_configurePriority
 - NVIC.c, 49
 - NVIC.h, 55
- NVIC_controlAllPeripheral
 - NVIC.c, 49
 - NVIC.h, 55
- NVIC_controlFault
 - NVIC.c, 49
 - NVIC.h, 55
- NVIC_controlInterrupt
 - NVIC.c, 49
 - NVIC.h, 56
- NVIC_controlPendingFlag
 - NVIC.c, 50
 - NVIC.h, 56
- NVIC_DISABLE
 - NVIC.h, 54
- NVIC_filterInterrupts
 - NVIC.c, 50
 - NVIC.h, 56
- NVIC_getActiveFlagStatus
 - NVIC.c, 50
 - NVIC.h, 57
- NVIC_getPriority
 - NVIC.c, 52
 - NVIC.h, 57
- NVIC_IRQNUM_WWDG
 - NVIC.h, 54
- NVIC_regMap, 7
- NVIC_RESET

- NVIC.h, 55
- RCC.h, 57
 - RCC_configureMCO, 60
 - RCC_configurePLL, 60
 - RCC_configurePrescalers, 61
 - RCC_controlAHBPeripheral, 62
 - RCC_controlAPB1Peripheral, 62
 - RCC_controlAPB2Peripheral, 62
 - RCC_selectSystemClock, 63
 - RCC_setClockState, 63
- RCC_configureMCO
 - RCC.h, 60
- RCC_configurePLL
 - RCC.h, 60
- RCC_configurePrescalers
 - RCC.h, 61
- RCC_controlAHBPeripheral
 - RCC.h, 62
- RCC_controlAPB1Peripheral
 - RCC.h, 62
- RCC_controlAPB2Peripheral
 - RCC.h, 62
- RCC_regMap, 7
- RCC_selectSystemClock
 - RCC.h, 63
- RCC_setClockState
 - RCC.h, 63
- SCHED.c, 63
 - SCHED_createTask, 64
 - SCHED_init, 65
 - SCHED_start, 65
- SCHED1.h, 65
 - SCHED_createTask, 66
 - SCHED_init, 66
 - SCHED_start, 66
- SCHED_CONF.h, 67
- SCHED_createTask
 - SCHED.c, 64
 - SCHED1.h, 66
- SCHED_init
 - SCHED.c, 65
 - SCHED1.h, 66
- SCHED_start
 - SCHED.c, 65
 - SCHED1.h, 66
- Std_Types.h, 67
- Switch.c, 69
 - Switch_GetSwitchStatus, 69
 - Switch_Init, 70
 - Switch_Task, 70
- Switch.h, 70
 - Switch_GetSwitchStatus, 71
 - Switch_Init, 72
 - Switch_Task, 72
- Switch_Cfg.c, 72
 - Switch_switches, 73
- Switch_Cfg.h, 73
- Switch_GetSwitchStatus
 - Switch.c, 69
 - Switch.h, 71
- Switch_Init
 - Switch.c, 70
 - Switch.h, 72
- Switch_switches
 - Switch_Cfg.c, 73
- switch_t, 8
- Switch_Task
 - Switch.c, 70
 - Switch.h, 72
- SysTask, 8
- SYSTICK.c, 74
 - SysTick_Handler, 75
 - SYSTICK_init, 75
 - SYSTICK_setCallbackFcn, 75
 - SYSTICK_setTime, 75
 - SYSTICK_start, 76
 - SYSTICK_stop, 76
- SYSTICK.h, 76
 - SYSTICK_init, 77
 - SYSTICK_setCallbackFcn, 77
 - SYSTICK_setTime, 77
 - SYSTICK_start, 78
 - SYSTICK_stop, 78
- SYSTICK_CONF.h, 78
- SysTick_Handler
 - SYSTICK.c, 75
- SYSTICK_init
 - SYSTICK.c, 75
 - SYSTICK.h, 77
- SYSTICK_regMap, 8
- SYSTICK_setCallbackFcn
 - SYSTICK.c, 75
 - SYSTICK.h, 77
- SYSTICK_setTime
 - SYSTICK.c, 75
 - SYSTICK.h, 77
- SYSTICK_start
 - SYSTICK.c, 76
 - SYSTICK.h, 78
- SYSTICK_stop
 - SYSTICK.c, 76
 - SYSTICK.h, 78
- Task, 8
- Uart.c, 79
 - UART4_IRQHandler, 81
 - UART5_IRQHandler, 81
 - Uart_Address, 84
 - Uart_Init, 81
 - Uart_Receive, 82
 - Uart_Send, 82
 - Uart_SetRxCb, 83
 - Uart_SetTxCb, 83
 - USART1_IRQHandler, 84
 - USART2_IRQHandler, 84

- USART3_IRQHandler, [84](#)
- Uart.h, [85](#)
 - Uart_Init, [86](#)
 - Uart_Receive, [86](#)
 - Uart_Send, [87](#)
 - Uart_SetRxCb, [87](#)
 - Uart_SetTxCb, [87](#)
- UART4_IRQHandler
 - Uart.c, [81](#)
- UART5_IRQHandler
 - Uart.c, [81](#)
- Uart_Address
 - Uart.c, [84](#)
- Uart_Init
 - Uart.c, [81](#)
 - Uart.h, [86](#)
- Uart_Receive
 - Uart.c, [82](#)
 - Uart.h, [86](#)
- Uart_Send
 - Uart.c, [82](#)
 - Uart.h, [87](#)
- Uart_SetRxCb
 - Uart.c, [83](#)
 - Uart.h, [87](#)
- Uart_SetTxCb
 - Uart.c, [83](#)
 - Uart.h, [87](#)
- uart_t, [9](#)
- USART1_IRQHandler
 - Uart.c, [84](#)
- USART2_IRQHandler
 - Uart.c, [84](#)
- USART3_IRQHandler
 - Uart.c, [84](#)