

IS1300 Project

Generated by Doxygen 1.9.1

1 File Index	1
1.1 File List	1
2 File Documentation	3
2.1 Core/Src/clock.c File Reference	3
2.1.1 Detailed Description	3
2.1.2 Function Documentation	3
2.1.2.1 get_time()	3
2.2 Core/Src/display.c File Reference	4
2.2.1 Detailed Description	5
2.2.2 Function Documentation	5
2.2.2.1 display_send_instruction()	5
2.2.2.2 display_transmit()	5
2.2.2.3 display_write()	5
2.2.2.4 display_write_row()	6
2.2.2.5 set_row()	6
2.2.2.6 split_byte()	6
2.3 Core/Src/error.c File Reference	7
2.3.1 Detailed Description	7
2.3.2 Function Documentation	7
2.3.2.1 handle_error()	7
2.4 Core/Src/freertos.c File Reference	8
2.4.1 Detailed Description	9
2.4.2 Function Documentation	9
2.4.2.1 MX_FREERTOS_Init()	9
2.4.2.2 startBacklightTask()	9
2.4.2.3 startClockTask()	10
2.4.2.4 StartDefaultTask()	10
2.4.3 Variable Documentation	10
2.4.3.1 backlightTask_attributes	10
2.4.3.2 clockTask_attributes	11
2.4.3.3 defaultTask_attributes	11
2.5 Core/Src/main.c File Reference	11
2.5.1 Detailed Description	12
2.5.2 Function Documentation	12
2.5.2.1 Error_Handler()	12
2.5.2.2 HAL_TIM_PeriodElapsedCallback()	12
2.5.2.3 HAL_UART_RxCpltCallback()	13
2.5.2.4 HAL_UART_TxCpltCallback()	13
2.5.2.5 main()	14
2.5.2.6 MX_FREERTOS_Init()	14
2.5.2.7 SystemClock_Config()	14

2.6 Core/Src/potentiometer.c File Reference	15
2.6.1 Detailed Description	15
2.6.2 Function Documentation	15
2.6.2.1 get_potentiometer_value()	15
2.7 Core/Src/red.c File Reference	16
2.7.1 Detailed Description	16
2.7.2 Function Documentation	16
2.7.2.1 set_brightness()	16
2.8 Core/Src/uart.c File Reference	17
2.8.1 Detailed Description	17
2.8.2 Function Documentation	17
2.8.2.1 uart_get_clock_input()	17
2.8.2.2 uart_println()	18
2.8.2.3 uart_printnum()	18
2.8.2.4 uart_receive()	18
2.8.2.5 uart_send()	19
Index	21

Chapter 1

File Index

1.1 File List

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

Core/Src/ clock.c	
This file provides code for starting and getting the rtc time	3
Core/Src/ display.c	
This file provides code for initialising and communicating with the display module	4
Core/Src/ error.c	
This file provides code for handling errors when they appear	7
Core/Src/ freertos.c	
This file contains the code that the program consists of, divided into tasks that will be run by the FreeRTOS scheduler	8
Core/Src/ main.c	
: Main program body. This is where the program is initialised	11
Core/Src/ potentiometer.c	
This file provides code for reading the potentiometer value	15
Core/Src/ red.c	
This file provides code for controlling the brightness of the red backlight	16
Core/Src/ uart.c	
This file contains functions for communicating via UART	17

Chapter 2

File Documentation

2.1 Core/Src/clock.c File Reference

This file provides code for starting and getting the rtc time.

```
#include "main.h"
#include "rtc.h"
```

Functions

- void [start_clock](#) (uint8_t hours, uint8_t minutes, uint8_t seconds)
start the RTC clock
- void [get_time](#) (RTC_TimeTypeDef *time)
Get the current RTC time.

2.1.1 Detailed Description

This file provides code for starting and getting the rtc time.

Author

William Asp

2.1.2 Function Documentation

2.1.2.1 [get_time\(\)](#)

```
void get_time (
    RTC_TimeTypeDef * time )
```

Get the current RTC time.

Parameters

out	time	The current time
-----	------	------------------

Returns

Pointer to the time struct

2.2 Core/Src/display.c File Reference

This file provides code for initialising and communicating with the display module.

```
#include "main.h"
#include "spi.h"
#include "error.h"
#include "red.h"
```

Functions

- void [hardware_reset](#) ()
Perform a hardware reset on the display Resets the display by writing to the displays hardware reset pin.
- void [test_backlight](#) ()
Test all backlight colors Run through each color of the display to see that they are lighting up.
- void [set_backlight](#) (uint8_t color, GPIO_PinState state)
Set a backlight color.
- void [split_byte](#) (uint8_t byte, uint8_t *buffer)
Split a byte to send to the display.
- int [display_transmit](#) (uint8_t startbyte, uint8_t *bytes, uint16_t length)
Send the display data or instructions.
- int [display_send_instruction](#) (uint8_t *instructions, uint16_t length)
Send instruction bytes via spi to the display.
- int [display_write](#) (char *characters, uint16_t length)
Write characters to the display where the cursor currently are.
- int [set_row](#) (uint8_t row)
Set the cursor on the display.
- int [display_write_row](#) (char *characters, uint16_t length, uint8_t row)
Write text to a specific row on the display.
- int [clear_display](#) ()
Clears the display.
- void [init_display](#) ()
Initialise the display.

Variables

- GPIO_TypeDef * [ports](#) [] = {Disp_White_GPIO_Port, Disp_Green_GPIO_Port}
The display backlight ports (without red)
- uint16_t [pins](#) [] = {Disp_White_Pin, Disp_Green_Pin}
The display backlight pins (without red)
- uint8_t [rows](#) [] = {0b10000000, 0b10100000, 0b11000000, 0b11100000}
The displays internal row addresses.

2.2.1 Detailed Description

This file provides code for initialising and communicating with the display module.

Author

William Asp

2.2.2 Function Documentation

2.2.2.1 display_send_instruction()

```
int display_send_instruction (
    uint8_t * instructions,
    uint16_t length )
```

Send instruction bytes via spi to the display.

Parameters

in	<i>instructions</i>	A pointer to the instructions to send to the display
in	<i>length</i>	The number of instructions

2.2.2.2 display_transmit()

```
int display_transmit (
    uint8_t startbyte,
    uint8_t * bytes,
    uint16_t length )
```

Send the display data or instructions.

Parameters

in	<i>startbyte</i>	The byte setting that initiates the transmit
in	<i>bytes</i>	The bytes that will be sent to the display
in	<i>length</i>	The number of bytes to send

2.2.2.3 display_write()

```
int display_write (
```

```
char * characters,  
uint16_t length )
```

Write characters to the display where the cursor currently are.

Parameters

<i>characters</i>	The characters to write
<i>length</i>	The number of characters

2.2.2.4 display_write_row()

```
int display_write_row (  
    char * characters,  
    uint16_t length,  
    uint8_t row )
```

Write text to a specific row on the display.

Parameters

in	<i>characters</i>	The characters to write
in	<i>length</i>	The number of characters
in	<i>row</i>	The row to write to

2.2.2.5 set_row()

```
int set_row (  
    uint8_t row )
```

Set the cursor on the display.

Parameters

in	<i>row</i>	The row to write to
----	------------	---------------------

2.2.2.6 split_byte()

```
void split_byte (  
    uint8_t byte,  
    uint8_t * buffer )
```

Split a byte to send to the display.

Parameters

in	<i>byte</i>	The byte to split into two
out	<i>buffer</i>	Where to place the two new bytes

2.3 Core/Src/error.c File Reference

This file provides code for handling errors when they appear.

```
#include "main.h"
```

Functions

- void [handle_error](#) ()
Show that an error has occurred by turning on LD2.

2.3.1 Detailed Description

This file provides code for handling errors when they appear.

Author

William Asp

2.3.2 Function Documentation

2.3.2.1 [handle_error\(\)](#)

```
void handle_error ( )
```

Show that an error has occurred by turning on LD2.

This function only turns on LD2 then runs in a while loop.

2.4 Core/Src/freertos.c File Reference

This file contains the code that the program consists of, divided into tasks that will be run by the FreeRTOS scheduler.

```
#include "FreeRTOS.h"
#include "task.h"
#include "main.h"
#include "cmsis_os.h"
#include "adc.h"
#include "rtc.h"
#include "spi.h"
#include "tim.h"
#include "gpio.h"
#include "stdio.h"
#include "display.h"
#include "error.h"
#include "uart.h"
#include "clock.h"
#include "red.h"
#include "potentiometer.h"
```

Macros

- `#define POT_MAX 4066`

Functions

- void [StartDefaultTask](#) (void *argument)
Function implementing the defaultTask thread.
- void [startBacklightTask](#) (void *argument)
Function implementing the backlightTask thread.
- void [startClockTask](#) (void *argument)
Function implementing the clockTask thread.
- void [MX_FREERTOS_Init](#) (void)
FreeRTOS initialization.

Variables

- osThreadId_t **defaultTaskHandle**
- const osThreadAttr_t **defaultTask_attributes**
- osThreadId_t **backlightTaskHandle**
- const osThreadAttr_t **backlightTask_attributes**
- osThreadId_t **clockTaskHandle**
- const osThreadAttr_t **clockTask_attributes**

2.4.1 Detailed Description

This file contains the code that the program consists of, divided into tasks that will be run by the FreeRTOS scheduler.

Author

William Asp

2.4.2 Function Documentation

2.4.2.1 MX_FREERTOS_Init()

```
void MX_FREERTOS_Init (  
    void )
```

FreeRTOS initialization.

Parameters

None	
------	--

Return values

None	
------	--

2.4.2.2 startBacklightTask()

```
void startBacklightTask (  
    void * argument )
```

Function implementing the backlightTask thread.

Parameters

<i>argument</i>	Not used
-----------------	----------

Return values

None	
------	--

2.4.2.3 startClockTask()

```
void startClockTask (
    void * argument )
```

Function implementing the clockTask thread.

Parameters

<i>argument</i>	Not used
-----------------	----------

Return values

<i>None</i>	
-------------	--

2.4.2.4 StartDefaultTask()

```
void StartDefaultTask (
    void * argument )
```

Function implementing the defaultTask thread.

Parameters

<i>argument</i>	Not used
-----------------	----------

Return values

<i>None</i>	
-------------	--

2.4.3 Variable Documentation

2.4.3.1 backlightTask_attributes

```
const osThreadAttr_t backlightTask_attributes
```

Initial value:

```
= {
    .name = "backlightTask",
    .stack_size = 128 * 4,
    .priority = (osPriority_t) osPriorityNormal,
}
```

2.4.3.2 clockTask_attributes

```
const osThreadAttr_t clockTask_attributes
```

Initial value:

```
= {
    .name = "clockTask",
    .stack_size = 128 * 4,
    .priority = (osPriority_t) osPriorityAboveNormal,
}
```

2.4.3.3 defaultTask_attributes

```
const osThreadAttr_t defaultTask_attributes
```

Initial value:

```
= {
    .name = "defaultTask",
    .stack_size = 128 * 4,
    .priority = (osPriority_t) osPriorityLow,
}
```

2.5 Core/Src/main.c File Reference

: Main program body. This is where the program is initialised.

```
#include "main.h"
#include "cmsis_os.h"
#include "adc.h"
#include "rtc.h"
#include "spi.h"
#include "tim.h"
#include "gpio.h"
#include "string.h"
#include "usart.h"
#include "stdio.h"
#include "display.h"
#include "error.h"
#include "uart.h"
#include "clock.h"
#include "red.h"
```

Functions

- void [SystemClock_Config](#) (void)
System Clock Configuration.
- void [MX_FREERTOS_Init](#) (void)
FreeRTOS initialization.
- void [HAL_UART_RxCpltCallback](#) (UART_HandleTypeDef *UartHandle)
Rx Transfer completed callback.
- void [HAL_UART_TxCpltCallback](#) (UART_HandleTypeDef *UartHandle)
Tx Transfer completed callback.
- int [main](#) (void)
The application entry point. This is where everything is initialised.
- void [HAL_TIM_PeriodElapsedCallback](#) (TIM_HandleTypeDef *htim)
Period elapsed callback in non blocking mode.
- void [Error_Handler](#) (void)
This function is executed in case of error occurrence.

Variables

- ITStatus **uartReady** = RESET

2.5.1 Detailed Description

: Main program body. This is where the program is initialised.

Attention

© Copyright (c) 2021 STMicroelectronics. All rights reserved.

This software component is licensed by ST under BSD 3-Clause license, the "License"; You may not use this file except in compliance with the License. You may obtain a copy of the License at: opensource.org/licenses/BSD-3-Clause

2.5.2 Function Documentation

2.5.2.1 Error_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

Return values

None	
------	--

2.5.2.2 HAL_TIM_PeriodElapsedCallback()

```
void HAL_TIM_PeriodElapsedCallback (
    TIM_HandleTypeDef * htim )
```

Period elapsed callback in non blocking mode.

Note

This function is called when TIM1 interrupt took place, inside HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment a global variable "uwTick" used as application time base.

Parameters

<i>htim</i>	: TIM handle
-------------	--------------

Return values

<i>None</i>	
-------------	--

2.5.2.3 HAL_UART_RxCpltCallback()

```
void HAL_UART_RxCpltCallback (
    UART_HandleTypeDef * UartHandle )
```

Rx Transfer completed callback.

Parameters

<i>UartHandle</i>	UART handle
-------------------	-------------

Note

This example shows a simple way to report end of IT Rx transfer, and you can add your own implementation.

Return values

<i>None</i>	
-------------	--

2.5.2.4 HAL_UART_TxCpltCallback()

```
void HAL_UART_TxCpltCallback (
    UART_HandleTypeDef * UartHandle )
```

Tx Transfer completed callback.

Parameters

<i>UartHandle</i>	UART handle.
-------------------	--------------

Note

This example shows a simple way to report end of IT Tx transfer, and you can add your own implementation.

Return values

<i>None</i>	
-------------	--

2.5.2.5 main()

```
int main (  
        void )
```

The application entry point. This is where everything is initialised.

The main function initialises all peripherals and starts up the program by taking user input and initialising the hardware before handing over the control to the FreeRTOS scheduler.

Return values

<i>int</i>	
------------	--

2.5.2.6 MX_FREERTOS_Init()

```
void MX_FREERTOS_Init (  
        void )
```

FreeRTOS initialization.

Parameters

<i>None</i>	
-------------	--

Return values

<i>None</i>	
-------------	--

2.5.2.7 SystemClock_Config()

```
void SystemClock_Config (  
        void )
```

System Clock Configuration.

Return values

<i>None</i>	
-------------	--

Configure the main internal regulator output voltage

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.

Initializes the CPU, AHB and APB buses clocks

2.6 Core/Src/potentiometer.c File Reference

This file provides code for reading the potentiometer value.

```
#include "main.h"
#include "adc.h"
```

Functions

- uint32_t [get_potentiometer_value](#) ()
Read the potentiometer value.

2.6.1 Detailed Description

This file provides code for reading the potentiometer value.

Author

William Asp

2.6.2 Function Documentation

2.6.2.1 [get_potentiometer_value\(\)](#)

```
uint32_t get_potentiometer_value ( )
```

Read the potentiometer value.

Returns

The value of the potentiometer

2.7 Core/Src/red.c File Reference

This file provides code for controlling the brightness of the red backlight.

```
#include "main.h"
#include "tim.h"
#include "uart.h"
#include "string.h"
```

Macros

- #define [CHANNEL](#) TIM_CHANNEL_2
The timer channel that is the PWM control.

Functions

- void [set_brightness](#) (double brightness)
Change the PWM pulse width of the red background light.

2.7.1 Detailed Description

This file provides code for controlling the brightness of the red backlight.

Author

William Asp

2.7.2 Function Documentation

2.7.2.1 [set_brightness\(\)](#)

```
void set_brightness (
    double brightness )
```

Change the PWM pulse width of the red background light.

Parameters

in	<i>brightness</i>	The brightness of the backlight from 0 to 1
----	-------------------	---------------------------------------------

2.8 Core/Src/uart.c File Reference

This file contains functions for communicating via UART.

```
#include "main.h"
#include "usart.h"
#include "string.h"
#include "stdio.h"
```

Macros

- `#define TIMEOUT 0xFFFFFFFF`
The polling timeout.

Functions

- `int uart_send (char *buffer, uint16_t length)`
Send a string over uart.
- `int uart_receive (char *buffer, uint16_t length)`
Recieve a string over uart.
- `int uart_println (char *string)`
send a string line to uart
- `int uart_printnum (uint32_t num)`
Print a number over uart.
- `void uart_get_clock_input (char *buffer)`
Let user input the time.

2.8.1 Detailed Description

This file contains functions for communicating via UART.

Author

William Asp

2.8.2 Function Documentation

2.8.2.1 `uart_get_clock_input()`

```
void uart_get_clock_input (
    char * buffer )
```

Let user input the time.

Parameters

<code>out</code>	<code>buffer</code>	The buffer to write to
------------------	---------------------	------------------------

Return values

<code>buffer</code>	The user entered time string
---------------------	------------------------------

2.8.2.2 uart_println()

```
int uart_println (
    char * string )
```

send a string line to uart

Parameters

<code>in</code>	<code>string</code>	The string to send
-----------------	---------------------	--------------------

Returns

HAL status of uart transmission

2.8.2.3 uart_printnum()

```
int uart_printnum (
    uint32_t num )
```

Print a number over uart.

Parameters

<code>in</code>	<code>num</code>	The number to be printed over UART
-----------------	------------------	------------------------------------

Returns

HAL status of uart transmission

2.8.2.4 uart_receive()

```
int uart_receive (
    char * buffer,
    uint16_t length )
```

Recieve a string over uart.

Parameters

out	<i>buffer</i>	The place to write the recieved string
in	<i>length</i>	The amount of data to read

Returns

HAL status of uart transmission

2.8.2.5 uart_send()

```
int uart_send (
    char * buffer,
    uint16_t length )
```

Send a string over uart.

Parameters

in	<i>message</i>	The character array to send
----	----------------	-----------------------------

Returns

HAL status of uart transmission

Index

backlightTask_attributes
freertos.c, [10](#)

clock.c
get_time, [3](#)
clockTask_attributes
freertos.c, [10](#)
Core/Src/clock.c, [3](#)
Core/Src/display.c, [4](#)
Core/Src/error.c, [7](#)
Core/Src/freertos.c, [8](#)
Core/Src/main.c, [11](#)
Core/Src/potentiometer.c, [15](#)
Core/Src/red.c, [16](#)
Core/Src/uart.c, [17](#)

defaultTask_attributes
freertos.c, [11](#)

display.c
display_send_instruction, [5](#)
display_transmit, [5](#)
display_write, [5](#)
display_write_row, [6](#)
set_row, [6](#)
split_byte, [6](#)
display_send_instruction
display.c, [5](#)
display_transmit
display.c, [5](#)
display_write
display.c, [5](#)
display_write_row
display.c, [6](#)

error.c
handle_error, [7](#)

Error_Handler
main.c, [12](#)

freertos.c
backlightTask_attributes, [10](#)
clockTask_attributes, [10](#)
defaultTask_attributes, [11](#)
MX_FREERTOS_Init, [9](#)
startBacklightTask, [9](#)
startClockTask, [9](#)
StartDefaultTask, [10](#)

get_potentiometer_value
potentiometer.c, [15](#)

get_time

clock.c, [3](#)

HAL_TIM_PeriodElapsedCallback
main.c, [12](#)

HAL_UART_RxCpltCallback
main.c, [13](#)

HAL_UART_TxCpltCallback
main.c, [13](#)

handle_error
error.c, [7](#)

main
main.c, [14](#)

main.c
Error_Handler, [12](#)
HAL_TIM_PeriodElapsedCallback, [12](#)
HAL_UART_RxCpltCallback, [13](#)
HAL_UART_TxCpltCallback, [13](#)
main, [14](#)
MX_FREERTOS_Init, [14](#)
SystemClock_Config, [14](#)
MX_FREERTOS_Init
freertos.c, [9](#)
main.c, [14](#)

potentiometer.c
get_potentiometer_value, [15](#)

red.c
set_brightness, [16](#)

set_brightness
red.c, [16](#)

set_row
display.c, [6](#)

split_byte
display.c, [6](#)

startBacklightTask
freertos.c, [9](#)

startClockTask
freertos.c, [9](#)

StartDefaultTask
freertos.c, [10](#)

SystemClock_Config
main.c, [14](#)

uart.c
uart_get_clock_input, [17](#)
uart_println, [18](#)
uart_printhnum, [18](#)
uart_receive, [18](#)

- uart_send, [19](#)
- uart_get_clock_input
 - uart.c, [17](#)
- uart_println
 - uart.c, [18](#)
- uart_printnum
 - uart.c, [18](#)
- uart_receive
 - uart.c, [18](#)
- uart_send
 - uart.c, [19](#)