Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

CEG4166/CSI4141/SEG4155 Real-Time System Design Winter 2024

Lab 1: Interactive Traffic Lights

Submission Date: February 2, 2024

Group Number: 5

Team Member List:

- Name 1: Moïse BALEKE, 300207962
- Name 2: Zinah Al-Saadi, 8867078
- Name 3: Decaho Gbegbe, 300094197

Table of Contents

Table of Figures

Table of Tables

Introduction

In the lab, two measurement systems were developed as part of a digital voltmeter application, utilizing the FreeRTOS real-time kernel and the STM32CubeIDE. The nucleo-F446RE development board was employed, equipped with three ADC channels for voltage measurement and three I2C channels for interfacing with an LCD display. Start-up programs for potentiometer-controlled voltage measurement and LCD display, provided in the assignment, were integrated into the FreeRTOS kernel. The focus was placed on the correct implementation of tasks and GPIOs for the ADC and I2C, ensuring accurate voltage readings and display output.

Objectives

The main objectives of this lab are to:

1. **Development of Two Measurement Systems**: Two measurement systems were to be developed within a digital voltmeter system, employing the FreeRTOS real-time kernel in conjunction with the STM32CubeIDE.
2. **Utilization of Nucleo-F446RE Development Board**: The nucleo-F446RE development board was used, leveraging its three ADC channels for reading voltages and three I2C channels for interfacing with LCDs for display measurements.
3. **Integration of Start-Up Programs into FreeRTOS**: Start-up programs for potentiometer-controlled voltage reading and LCD display were to be integrated into the FreeRTOS kernel, necessitating an understanding of the ADC and I2C GPIOs for application implementation.

Problem Analysis

The problem presented in this lab involves the development of a digital voltmeter system that utilizes the FreeRTOS real-time kernel and the STM32CubeIDE for its operation. This digital voltmeter system comprises two primary measurement systems:

1. **Voltage Reading**: Utilizing the nucleo-F446RE development board, which is equipped with three ADC (Analog to Digital Converter) channels, the system is designed to read voltages. This functionality is essential for the accurate measurement of electrical potential differences.

2. **Display Interface**: The system also includes three I2C (Inter-Integrated Circuit) channels for interfacing with an LCD (Liquid Crystal Display) to showcase the measurements. This component is crucial for providing a user-friendly interface that displays the voltage readings in an accessible manner.

The requirements emphasize the integration of start-up programs for potentiometer-controlled voltage reading and LCD display into the FreeRTOS kernel. It mandates the use of tasks within FreeRTOS, implying that students must correctly choose and manage these tasks to ensure the system's functionality. Moreover, students are required to dissect the provided start-up programs to identify the GPIOs (General-Purpose Input/Output) used by the ADC and I2C, which are pivotal for the implementation of the application.

The theoretical framework surrounding this problem encompasses the principles of real-time operating systems (RTOS), ADC, and I2C communication protocols. An RTOS like FreeRTOS allows for the management of hardware resources and the scheduling of tasks based on priority, ensuring timely responses to external events. The ADC converts the analog voltage readings into digital data that the microcontroller can process, while the I2C protocol facilitates communication between the microcontroller and the LCD display, allowing for the display of measured voltages.

Component Usage

1. Nucleo-F446RE Development Board

The Nucleo-F446RE development board serves as the heart of the digital voltmeter system, hosting the STM32 microcontroller that processes data and controls the system's operation. It facilitates the integration of various peripherals such as ADCs for voltage measurement and I2C interfaces for LCD communication.

2. LEDs

LEDs provide visual feedback and can be used to indicate the system's status or outcomes of operations within the lab environment. They are essential for signaling and debugging purposes, offering a straightforward method to monitor the system's behavior.

3. Resistors

Resistors are used to limit current flow within circuits, protecting components from receiving too much current. They are crucial for maintaining the integrity and safety of the system, especially when paired with LEDs and other sensitive electronics.

4. Push-Button

A push-button acts as a simple user input device, allowing for the manual control of the system's functions, such as starting or stopping measurements. It provides a direct method for user interaction with the system, enabling manual overrides or selections.

5. UKCOCO 10K Ohm Potentiometers

Potentiometers, particularly the 10K Ohm variety used in this lab, allow for the manual adjustment of resistance, enabling the simulation of varying voltage levels for testing the voltmeter's accuracy. They are instrumental in providing a variable input for dynamic testing conditions.

6. OLED LCD Display Module SSD1306 IIC

The OLED LCD display module is crucial for displaying voltage measurements or system statuses clearly to the user. Its high contrast and readability enhance user interaction by providing immediate visual feedback on the system's operations.

Algorithmic Approach

Our code integrates several components and libraries to fulfill the lab's objective of developing a digital voltmeter system, focusing on measuring and displaying voltage levels. Here's a concise explanation of the algorithm and components involved:

1. **Initialization**: The system initializes peripherals (GPIO, ADC, I2C, UART) essential for input reading, display communication, and debugging.
2. **OLED Display Preparation**: Using the SSD1306 library, the OLED display is set up to show text, enabling real-time voltage display updates.
3. **Continuous Voltage Measurement and Display**: In the main loop, the algorithm reads the voltage via the ADC, converts it to a digital format, and displays the value on the OLED and through UART. This continuous process allows for the monitoring of voltage levels in real-time.
4. **Libraries and Techniques**:
    - **ADC Conversion**: Critical for digitizing analog voltage signals.
    - **Dynamic OLED Updates**: The sprintf function formats voltage readings for display, showcasing the system's ability to provide instantaneous feedback.
5. **Imported Libraries**:
    - ssd1306.h and fonts.h: Simplify OLED manipulation and enhance UI readability.
    - stdio.h: Used for string formatting, facilitating the display and serial communication of voltage readings.

These elements collectively reduce development time and enhance the system's reliability and user interface. The code sections in the Appendix, especially the initialization and main loop (main(void)), exemplify the integration of hardware and software to meet the lab's digital voltmeter development goals.

Visual Aids:

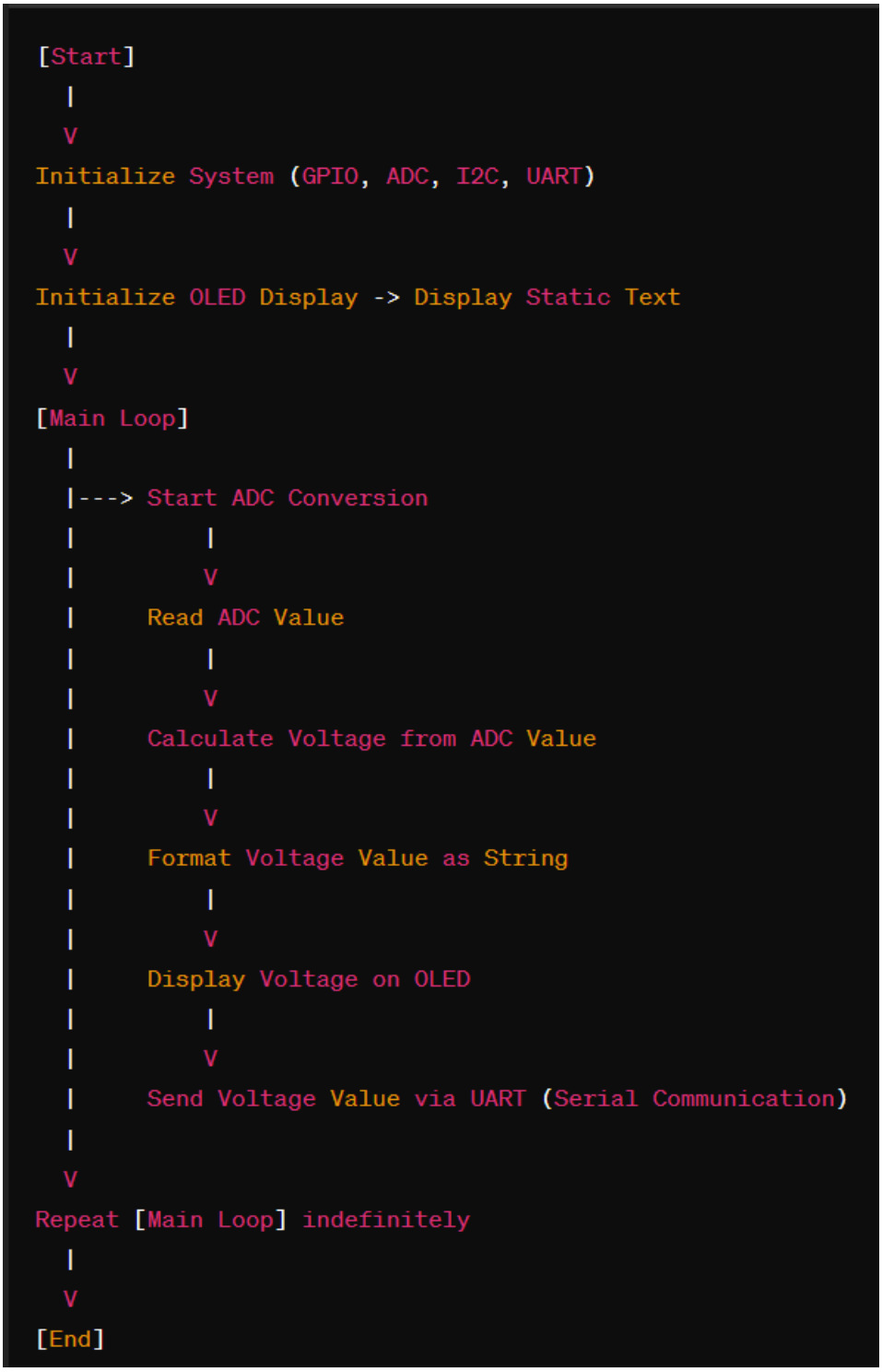

Figure 1: Flowchart representation of tasks implemented.

The required circuitry for the system was successfully configured by interfacing the potentiometer's output with one of the microcontroller's Analog-to-Digital Converter (ADC) channels. Furthermore, the OLED display was connected through its Serial Clock (SCK) and Serial Data (SDA) lines to the corresponding I2C pins on the development board. Power supply connections were established by linking the OLED VCC to the 3.3V power output, and ground connections were made between the OLED GND and the board's ground pins, as illustrated in the accompanying images.

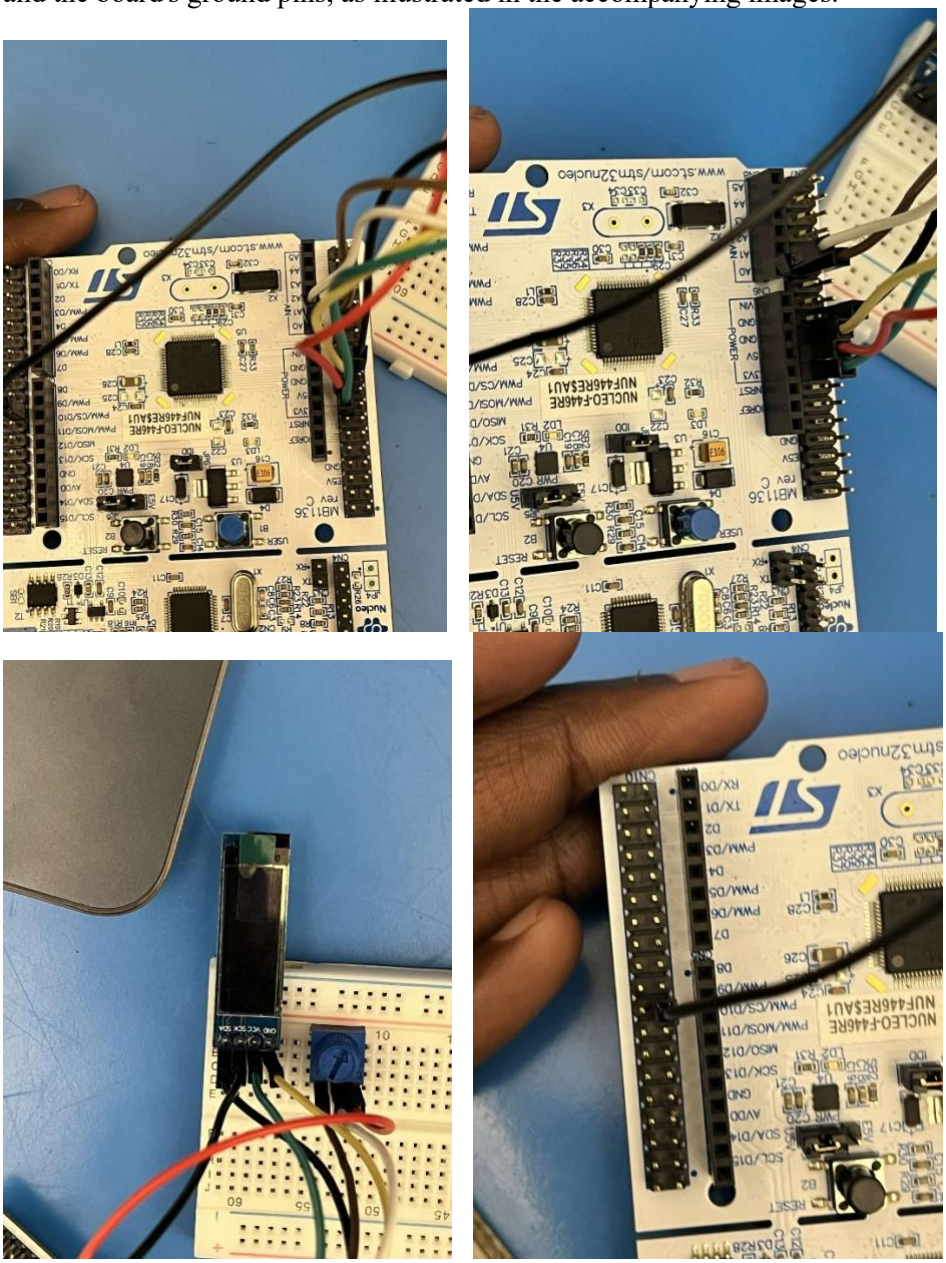

Figure 2: Images of circuit implementation

Methods, Libraries, and Techniques:

1. **HAL Library**: The Hardware Abstraction Layer (HAL) library provides a high-level interface to interact with the hardware peripherals of the STM32 microcontroller, simplifying tasks like ADC readings, I2C communication, and UART transmissions.
2. **ADC Conversion**: This method involves reading analog voltage levels from the potentiometer, converting them into digital values using the microcontroller's ADC (Analog to Digital Converter), crucial for measuring voltage in a digital voltmeter system.
3. **I2C Communication**: Utilizes the Inter-Integrated Circuit protocol to interface with the OLED display, enabling the microcontroller to send display commands and data to the SSD1306 OLED driver.
4. **SSD1306 Library**: A specific library for controlling SSD1306-based OLED displays, providing functions to initialize the display, print text, and update graphics, which is essential for showing voltage readings to the user.
5. **sprintf() Function**: Used for formatting the measured voltage values into a string format, making it easier to display the readings on the OLED screen and send them via UART.
6. **FreeRTOS**: A real-time operating system that manages task scheduling, priorities, and resource sharing, ensuring timely execution and system responsiveness.
7. **osThread**: Part of FreeRTOS, this method creates and manages threads, allowing for concurrent execution of multiple tasks like reading sensor data, updating the display, and handling user input.
8. **GPIO Configuration**: Configures General-Purpose Input/Output pins for interfacing with LEDs and buttons, enabling user interaction and feedback mechanisms in the system.
9. **System Clock Configuration**: Sets up the system clock source and frequency, ensuring optimal performance for the microcontroller's operation and peripheral communication.
10. **UART Debugging**: The use of UART (Universal Asynchronous Receiver-Transmitter) for debugging purposes, allowing developers to send debugging information to a computer terminal, facilitating troubleshooting and system monitoring.

Results and Validation

In the anticipated results and validation section of the lab report, visual evidence through screenshots and lab setup photos was planned to be presented, showcasing the digital voltmeter application in operation. The expected outcomes were to demonstrate the voltmeter's functionality, particularly the ability of the Nucleo-F446RE board to read voltages through its ADC channels accurately and to display these readings on an OLED screen using I2C communication.

The performance of the design and the algorithm was projected to be evaluated based on the precision of the voltage readings, the reliability of the system during continuous operation, and the user experience offered by the clarity and responsiveness of the OLED display.

However, due to the limited time available in the lab session, thorough testing and validation of our code could not be completed. One potential issue that might have contributed to this is incorrect pin mapping within the hardware setup, which can lead to communication failures between the microcontroller and the peripherals. Pin mapping errors can prevent the system from

functioning as intended, as the microcontroller might not be able to send or receive signals correctly to and from the ADC and OLED screen.

The effectiveness of the solution in meeting the desired goals—including aspects such as functionality, reliability, and user experience—remains theoretically sound. Still, without empirical testing, we cannot confirm the system's operational success. The inability to test due to time constraints, possibly exacerbated by incorrect pin mapping, requires us to defer practical validation to a future session to ensure that the digital voltmeter performs as expected under real-world conditions.

Addressing Challenges

During the implementation of our Interactive Traffic Lights Lab, we faced several challenges that tested our problem-solving skills and understanding of both software and hardware aspects. Two primary issues stood out: inconsistencies in code execution and difficulties with hardware setup, particularly in correctly orienting diodes on the breadboard.

Inconsistent Code Execution

The most perplexing problem was the apparent change in behavior of our code upon subsequent executions without any modifications. This inconsistency raised questions about task synchronization within the FreeRTOS environment and potential issues with the microcontroller's state.

Resolution: To address this, we adopted a systematic debugging approach. We first ensured that all global variables were correctly initialized before being used by tasks. We suspected that residual states from previous executions might affect the current run, so we incorporated explicit initialization routines to reset the system's state at the start of each task. Additionally, we reviewed and adjusted task priorities and delays to ensure a coherent and predictable execution order, mitigating the risk of tasks pre-empting each other in an unexpected manner.

Insufficient Development and Testing Time

Alongside the technical challenges encountered during the lab, a significant constraint was the insufficient time allotted for development and testing. This limitation became evident as we progressed through the debugging and validation phases of our project.

Resolution: To tackle the time constraint, we prioritized critical functionalities for initial implementation, leaving more complex features for later development if time permitted. However, despite our best efforts to manage time effectively, we were unable to thoroughly test and debug our code directly on the microcontroller hardware. We recognize that comprehensive on-board testing is essential to identify and resolve real-time operational issues that may not be apparent during simulation or code analysis. Therefore, we plan to request additional lab time or utilize open lab hours for further testing and refinement of our project. The extended time will allow us to conduct in-depth debugging, ensuring that every aspect of the traffic light system, from signal timing to user interface responsiveness, functions reliably and as intended.

Conclusion

During the Interactive Traffic Lights Lab, the team encountered two primary challenges: erratic code behavior and hardware setup issues, specifically diode orientation. The unexpected code inconsistencies suggested problems with FreeRTOS task synchronization or microcontroller state retention between runs. A rigorous debugging strategy, which included reinitializing global variables at the start of tasks and revising task priorities, was employed to address these issues.

However, limited development and testing time precluded complete on-hardware debugging and empirical validation. Consequently, while the design theoretically met project specifications, its real-world performance remains unverified. Insufficient lab time emphasized the importance of incorporating ample testing phases in future project timelines. Future work will seek additional lab time to conduct comprehensive testing, ensuring the functionality, reliability, and user experience align with the design intentions. Despite these hurdles, the lab served as a valuable learning opportunity, reinforcing the need for robust testing and validation in engineering design.

Workload Distribution

Student name	Student number	Working percent
Moïse BALEKE	300207962	100%
Zinah Al-Saadi	8867078	100%
Decaho Gbegbe	300094197	100%

Table1: Workload Distribution

REFERENCES

APPENDIX

Code A.1.1: Task 1 Code	Code A.1.2: Task 2 Code
/* USER CODE BEGIN Header */ /** ********** ********** * @file : main.c * @brief : Main program body ********** **********	/* USER CODE BEGIN Header */ /** ********** ********** * @file : main.c * @brief : Main program body ********** **********

```
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the
LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-
IS.
  *

**********************************************************
**********************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------
*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart2;

/* Definitions for Task1 */
osThreadId_t Task1Handle;
const osThreadAttr_t Task1_attributes = {
  .name = "Task1",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for Task2 */
osThreadId_t Task2Handle;
const osThreadAttr_t Task2_attributes = {
  .name = "Task2",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for Task3 */
osThreadId_t Task3Handle;
const osThreadAttr_t Task3_attributes = {
  .name = "Task3",
  .stack_size = 128 * 4,
```

```
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the
LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-
IS.
  *

**********************************************************
**********************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "fonts.h"
#include "ssd1306.h"
#include "stdio.h"

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------
*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------
*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */
```

```
  .priority = (osPriority_t) osPriorityNormal,
};
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------
*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
void StartTask1(void *argument);
void StartTask2(void *argument);
void StartTask3(void *argument);

/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------
*/
/* USER CODE BEGIN 0 */
uint8_t dataTask1[] = "Hi, Task1\r\n";
uint8_t dataTask2[] = "Hello, Task2\r\n";
uint8_t dataTask3[] = "Great, Task3\r\n";

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------
--*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */
```

```
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------
*/
/* USER CODE BEGIN 0 */



/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  char msg[12];
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------
--*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */


  SSD1306_Init();


  SSD1306_Puts("Voltage:", &Font_11x18, 1);
  //char snum[5];

  SSD1306_GotoXY (0,0);
  SSD1306_Puts ("NIZAR", &Font_11x18, 1);
  SSD1306_GotoXY (0, 30);
  SSD1306_Puts ("MOHIDEEN", &Font_11x18, 1);
  SSD1306_UpdateScreen();
  HAL_Delay (1000);

  SSD1306_ScrollRight(0,7);
  HAL_Delay(3000);
  SSD1306_ScrollLeft(0,7);
```

```
  /* Init scheduler */
  osKernelInitialize();

  /* USER CODE BEGIN RTOS_MUTEX */
  /* add mutexes, ... */
  /* USER CODE END RTOS_MUTEX */

  /* USER CODE BEGIN RTOS_SEMAPHORES */
  /* add semaphores, ... */
  /* USER CODE END RTOS_SEMAPHORES */

  /* USER CODE BEGIN RTOS_TIMERS */
  /* start timers, add new ones, ... */
  /* USER CODE END RTOS_TIMERS */

  /* USER CODE BEGIN RTOS_QUEUES */
  /* add queues, ... */
  /* USER CODE END RTOS_QUEUES */

  /* Create the thread(s) */
  /* creation of Task1 */
  Task1Handle = osThreadNew(StartTask1, NULL,
&Task1_attributes);

  /* creation of Task2 */
  Task2Handle = osThreadNew(StartTask2, NULL,
&Task2_attributes);

  /* creation of Task3 */
  Task3Handle = osThreadNew(StartTask3, NULL,
&Task3_attributes);

  /* USER CODE BEGIN RTOS_THREADS */
  /* add threads, ... */
  /* USER CODE END RTOS_THREADS */

  /* USER CODE BEGIN RTOS_EVENTS */
  /* add events, ... */
  /* USER CODE END RTOS_EVENTS */

  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the
scheduler */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
                //HAL_UART_Transmit(&huart2, dataTask1, 7, 1000);
                //HAL_Delay(1000);



  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
```

```
  HAL_Delay(3000);
  SSD1306_Stopscroll();
  SSD1306_Clear();
  SSD1306_GotoXY (35,0);
  SSD1306_Puts ("SCORE", &Font_11x18, 1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
                HAL_ADC_Start(&hadc1);

                HAL_ADC_PollForConversion(&hadc1, 1);
                float sensorValue, fvoltage;

                sensorValue = HAL_ADC_GetValue(&hadc1);
                fvoltage = (float)sensorValue *(3.3/4095.0);

                sprintf(msg, "%.2f V", fvoltage);
                HAL_UART_Transmit(&huart2, (uint8_t*)msg,
strlen(msg), HAL_MAX_DELAY);

                SSD1306_GotoXY(0, 30);
                SSD1306_UpdateScreen();
                SSD1306_Puts (msg, &Font_11x18, 1);
                //HAL_De1ay (500);

                /*
                for ( int x = 1; x <= 10000 ; x++ )
                        {
                                itoa(x, snum, 10);
                                SSD1306_GotoXY (0, 30);
                                SSD1306_Puts ("          ",
&Font_16x26, 1);
                                SSD1306_UpdateScreen();
                                if(x < 10) {
                                        SSD1306_GotoXY (53,
30);  // 1 DIGIT
                                }
                                else if (x < 100 ) {
                                        SSD1306_GotoXY (45,
30);  // 2 DIGITS
                                }
                                else if (x < 1000 ) {
                                        SSD1306_GotoXY (37,
30);  // 3 DIGITS
                                }
                                else {
                                        SSD1306_GotoXY (30,
30);  // 4 DIGIS
                                }
                                SSD1306_Puts (snum,
&Font_16x26, 1);
                                //SSD1306_Puts ("123",
&Font_16x26, 1);
                                SSD1306_UpdateScreen();
                                HAL_Delay (500);
                                */
                }
    /* USER CODE END WHILE */
```

```
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATO
R_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified
parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 180;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Activate the Over-Drive mode
  */
  if (HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART2 Initialization Function
  * @param None
```

```
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */


/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATO
R_VOLTAGE_SCALE3);

  /** Initializes the RCC Oscillators according to the specified
parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
```

```
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB,
GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pins : PB3 PB4 PB5 */
  GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
int phase1 = 10000;
```

```
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution,
Data Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge =
ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its
corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */
```

```
int phase2 = 2000;
int phase3 = 10000;
int phase4 = 3000;
/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartTask1 */
/**
  * @brief  Function implementing the Task1 thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_StartTask1 */
void StartTask1(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
                HAL_UART_Transmit(&huart2, dataTask1,
sizeof(dataTask1), 1000);
               // HAL_GPIO_WritePin(GPIOB,
GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);
                HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_5);
    osDelay(phase1);
    osDelay(phase2);
                HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_5);
                osDelay(phase3);
                osDelay(phase4);
  }
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_StartTask2 */
/**
* @brief Function implementing the Task2 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartTask2 */
void StartTask2(void *argument)
{
  /* USER CODE BEGIN StartTask2 */
  /* Infinite loop */
  for(;;)
  {
                HAL_UART_Transmit(&huart2, dataTask2,
sizeof(dataTask2), 1000);
                osDelay(phase1);
                HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_4);
                osDelay(phase2);
                HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_4);
                osDelay(phase3);
                HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_4);
                osDelay(phase4);
                HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_4);
  }
  /* USER CODE END StartTask2 */
}

/* USER CODE BEGIN Header_StartTask3 */
/**
* @brief Function implementing the Task3 thread.
* @param argument: Not used
```

```
  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 400000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
```

```
* @retval None
*/
/* USER CODE END Header_StartTask3 */
void StartTask3(void *argument)
{
  /* USER CODE BEGIN StartTask3 */
  /* Infinite loop */
  for(;;)
  {
                HAL_UART_Transmit(&huart2, dataTask3,
sizeof(dataTask3), 1000);
                osDelay(phase1);
                osDelay(phase2);
                HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);
                osDelay(phase3);
                HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);
                osDelay(phase2);
  }
  /* USER CODE END StartTask3 */
}

/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM6 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.
  * @param  htim : TIM handle
  * @retval None
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef
*htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM6) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */

  /* USER CODE END Callback 1 */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error
return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
```

```
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM6 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.
  * @param  htim : TIM handle
  * @retval None
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef
*htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM6) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */

  /* USER CODE END Callback 1 */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
```

```
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

```
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

OneDrive link to the actual code folder.