

Final project Report

(subject : 2110366 Embedded Systems lab I)

Team นายแลปจอมป่วน กับ เด็กชายจอมแก่น หงษาม

Chatree kurupakorn 6330105721

Natanan Phumphuang 6330161821

Charnkij Suksuwanveeree 6330104021

Section 2

Table of Contents

Introduction.....	3	Role and
Responsibility.....	4	
Implementation.....		
Circuitry and Code.....	5-6	
Testing.....	6	
Results.....	7	
Appendices.....		
Bill of Materials.....	8-9	Source
Code.....	10-22	

Introduction

ระบบฝังตัวคือระบบคอมพิวเตอร์ขนาดจิ๋วที่ฝังไว้ในอุปกรณ์เครื่องใช้ไฟฟ้าและเครื่องเล่น อิเล็กทรอนิกส์ต่างๆ เป้าหมายของโปรเจกต์นี้คือ การสร้างระบบฝังตัวที่อยู่ใน แนวคิดของสังคม ไร้สัมผัส โดยจะใช้ STM32 microcontroller เพื่ออ่าน/ส่ง ข้อมูลจากเซนเซอร์ พร้อมกับ ESP-8266 (NodeMCU) เพื่ออำนวยความสะดวกในการสื่อสารกับ WiFi และ นำข้อมูล เหล่านั้นเข้าสู่ internet ซึ่งตรงกับหัวข้อ IoT หรือ The Internet of Thing

โดยที่มุ่งของพวกเราต้องการเปิดปิดไฟผ่านการตรวจสอบการเคลื่อน และส่งข้อมูลที่ได้จาก การตรวจจับอุณหภูมิขึ้น web server ซึ่งเมื่อมีการเคลื่อนที่ผ่าน ประตู จะทำการเปิดไฟพร้อม ด้วยกับการวัดอุณหภูมิเมื่อทำการเข้าใช้

ขอยกตัวอย่างสถานการณ์ดังนี้เมื่อเดินทางจากบ้านมาเห็นอย่างมาก มีอาจมีเชื้อโรคเพื่อเลี้ยงการสัมผัสกับอุปกรณ์เครื่องมือเครื่องใช้เลย ทำให้เราคิดอุปกรณ์ที่เมื่อเดินเข้าบ้านแล้วไม่อยากสัมผัสก็เปิดไฟได้และถ้าอยากรวัดอุณหภูมิก็สามารถวัดและเมื่อล้างมือเสร็จก็สามารถกดแสดงผล เพื่อคุณหภูมิที่เราได้วัดเมื่อตอนเข้าบ้าน

Role and Responsibility

หน้าที่ได้แบ่งออกเป็น 3 ส่วน 'ได้แก่'

- 1) Embedded System Development - Develop code on the MCU/ESP32
- 2) UI/UX Designer and Development - Design and develop user interface
- 3) Creative and management - Offer new ideas and integrate overall idea to the project

Chatree kurupakorn	UI/UX Designer and Development
Natanan Phumphuang	Embedded System Development
Charnkij Suksuwanveeree	Creative and Management

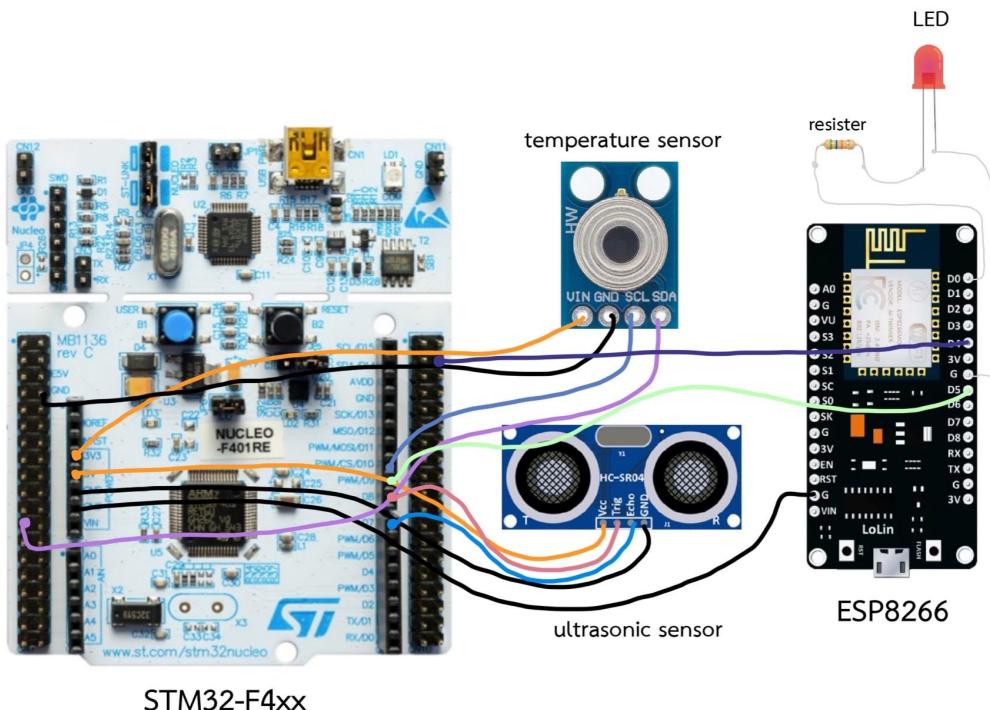
*หมายเหตุ แต่ละหน้าที่ถือเป็นหน้าที่หลักที่แต่ละคนได้รับมอบหมายแต่ ทุกคนยังต้องมี ส่วนในการ implement ตัวระบบฝั่งตัวนี้ ขึ้นมา

Topic	Date(May 2022)									
	22	23	24	25	26	27	28	29	30	
Brainstorming										
Preparing equipments										
Development process										
UX/UI design										
Github setup										
Summary + Making report										

ตารางที่ 1 : project plan

Implementation

Circuitry and Coding



รูปที่ 1: แผนผังวงจร

รูปที่ 1 แสดงแผนผังวงจร โดย STM32 microcontroller ถูกใช้ในการติดต่อสื่อสารกับ Ultrasonic sensor (HC-SR04) และ Temperature sensor (MLX-90614) ซึ่ง Ultrasonic sensor โดยใช้การรับส่งสัญญาณ TRIG และ ECHO ที่จะติดต่อผ่านทาง uart1 โดยต่อ TRIG ผ่าน GPIOA PIN 9 และ ECHO ผ่าน GPIOA PIN 8 สำหรับการติดต่อผ่าน uart1 และ Temperature sensor จะติดต่อผ่านทาง i2c โดยใช้ การรับส่งสัญญาณ SCL และ SDA ที่จะติดต่อผ่านทาง i2c1 โดยต่อ SCL ผ่าน GPIOB PIN 6 และ SDA ผ่าน GPIOB PIN 7 STM32 microcontroller จะเป็นตัวจัดสรรศักย์ไฟฟ้า 5 V และ 3.3 V ให้กับ Ultrasonic sensor และ Temperature sensor ตามลำดับ STM32 จะถูก เชื่อมต่อกับคอมพิวเตอร์ผ่านทาง mini-USB cable ที่จะเป็นตัว เข้าถึงและอัปโหลดโปรแกรมเข้าสู่ STM32 memory และเรียกข้อมูลที่เป็นเอกสารพุ ตคีนกลับมา จากตัวเซนเซอร์ทั้งสอง เตรียมไว้เพื่อนำไปใช้สู่กระบวนการต่อๆไป ซึ่งกระบวนการที่ว่าในที่นี้ STM32 จะทำการเชื่อมต่อกับ serial uart6 ผ่าน Rx และ Tx port บน GPIOC PIN 7 Rx และ GPIOC PIN 6 Tx ที่จะทำการติดต่อกับ ESP8266 ที่ถูก ตั้งค่า Serial port ผ่าน software serial บน ArduinoIDE ที่ port D4 คือ Rx port และ D5 คือ Tx port เป็นที่เรียบร้อยแล้ว

ชิ้นโค้ดที่ทีมของพวกเราออกแบบแบบถูกแบ่งออกเป็น 2 ส่วน ในส่วนแรกจะเป็น โปรแกรม ภาษา C ที่จะเป็น main program ของ STM32 ที่จะทำการรับข้อมูลผ่าน uart1 และ i2c1 และส่งออกข้อมูลผ่านทาง uart6 เพื่อติดต่อกับ ESP8266 โดยโค้ดส่วนนี้จะถูกเขียนใน stm32cubeIDE ภายในโปรแกรมหลัก (main.c) นั้นประกอบไปด้วยสามส่วนย่อย ได้แก่ โปรแกรมส่วนของ Ultrasonic sensor (HC-SR04), Temperature sensor (MLX-90614) และส่วนของการส่งข้อมูลไปให้ ESP8266 โดยใช้ UART (universal asynchronous receiver-transmitter)

ในส่วนที่สองจะเป็น Arduino code ที่จะเขียนด้วยภาษา C++ มาพร้อมกับ special methods and functions ที่จะเป็น main program ของ ESP8266 โดยจะทำการรับ signals และนำข้อมูลที่ได้มาแบ่งออกเป็น อุณหภูมิ ระยะห่าง สถานะ ที่จะนำข้อมูลเหล่านี้ขึ้น webServer และติดต่อผ่าน WiFi เพื่อประสบการณ์ Internet of Thing ที่ user สามารถได้รับโดยการ ติดต่อกับ ระบบเหล่านี้ผ่าน WiFi ซึ่งสามารถติดต่อได้ด้วยการส่ง ssid, password และ มีการ ส่ง IP address กลับมาแล้วทำการเริ่มต้น web server และเช็ค client ว่าพร้อมที่จะเริ่มต้น หรือยัง หลังจากนั้นจะเป็นการ implement webpage ขึ้นมาผ่าน client.println(..) method ที่จะทำการเพิ่ม html, css, javascript code ลงไว้ใน เพื่อ implement webpage ให้เสร็จ สมบูรณ์

Testing

เป้าหมายของ project นี้คือการสร้างอุปกรณ์รีสัมผัสหรือ contactless device เพื่อช่วยในการเปิดหรือปิดไฟและวัดอุณหภูมิ ร่างกายโดยใช้เซนเซอร์สองชนิด ได้แก่ Ultrasonic sensor (HC-SR04) ที่ใช้ในการตรวจสอบว่ามีคนเดินผ่านเซนเซอร์หรือไม่จากระยะห่างที่ เซนเซอร์ตัวนี้วัดได้ และอีกชนิดคือ Temperature sensor (MLX-90614) ที่ใช้ในการวัดอุณหภูมิ ของฝ่ามือ การติดต่อสื่อสารกันระหว่าง STM32 board (Nucleo-F411RE) และ ESP8266 STM32 board (Nucleo-F411RE) จะรับ input จากเซนเซอร์ทั้งสองชนิด และส่งข้อมูลเหล่านั้นไปยัง ESP8266 โดยใช้ UART (universal asynchronous receiver-transmitter) ในการส่ง ข้อมูล เพื่อที่จะนำข้อมูลเหล่านั้นไปดำเนินการต่อ ๆ ไปแก่ นำค่าอุณหภูมิไปแสดงใน web server เพื่อให้ผู้ใช้งานทราบค่า และสามารถรับมือในกรณีที่อุณหภูมิร่างกายนั้นสูงกว่าปกติ และนำค่าระยะห่างที่ Ultrasonic sensor (HC-SR04) วัดได้ไปพิจารณาสถานะการทำงาน ของหลอดไฟต่อไป

Results

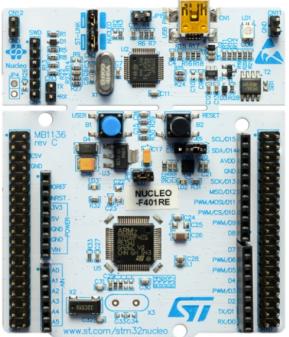
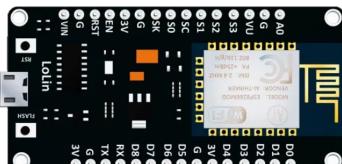


รูปที่ 2: Web server of ESP8266

ในส่วนของผลลัพธ์ เมื่อเราส่งข้อมูลที่ได้จากบอร์ด STM32 ซึ่งคืออุณหภูมิ และระยะห่าง ที่เซนเซอร์ที่จับได้ไปให้ ESP8266 จากนั้นเราก็เชื่อมไฟและเริ่มปล่อย web server จากบอร์ด ESP8266 ซึ่งจะได้ IP Address ออกมา (ตัวอย่าง 172.20.10.2) ซึ่งภายในเว็บจะประกอบด้วย อุณหภูมิที่วัดได้ สถานะของอุปกรณ์ และปุ่มเปิด-ปิด เมื่อกดปุ่มปิดสถานะจะเปลี่ยนเป็น off ทำให้ไฟ LED ปิด แต่ถ้าหากเม้าต์กลิ้นจะกดอุณหภูมิต่อๆ กันกว่า 40 เซนติเมตร สถานะจะถูกเปลี่ยนเป็น on และเปิด LED ซึ่งผู้ใช้สามารถกดปุ่มเปิดเองได้ อีกทั้งผู้ใช้งานสามารถอ่านอุณหภูมิได้โดยการนำเมา่มือหรือหน้าผากมาเข้าใกล้เซนเซอร์ ยังไม่หมดเพียงเท่านี้อุปกรณ์ของเรายังสามารถทำให้ผู้ใช้สามารถรับรู้อุณหภูมิภายในบ้าน ได้อีกด้วยเช่นกัน

APPENDICES

Appendix A: Bill of Materials

Device	Price	Quantity
STM32 board (Nucleo-F411RE)	590 บาท	1
		
Ultrasonic sensor (HC-SR04)	11 บาท	1
		
Temperature sensor (GY-906 MLX-90614)	385 บาท	1
		
ESP8266 (Node MCU)	120 บาท	1
		

Micro USB	40 บาท	1
Mini USB	130 บาท	1
Resistor $10\text{ k}\Omega \pm 5\%$	13 บาท	1
Red LED	10 บาท	1
Wire	40 บาท /100 สาย	16

Appendix B: embed Source Code

- main.c from stm32cubeIDE

```
/* USER CODE BEGIN Header */

/**
 * @file      : main.c
 * @brief     : Main program body
 *
 * @attention
 *
 * Copyright (c) 2022 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 <string.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 -----*/
I2C_HandleTypeDef hi2c1;

TIM_HandleTypeDef htim1;

UART_HandleTypeDef huart2;
UART_HandleTypeDef huart6;

/* USER CODE BEGIN PV */
char data[32] = " ";
char data1[32] = " ";
/* 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_TIM1_Init(void);
static void MX_USART6_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

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

void delay(uint16_t time) {
    __HAL_TIM_SET_COUNTER(&htim1, 0);
    while (__HAL_TIM_GET_COUNTER (&htim1) < time)
        ;
}

```

```

#define TRIG_PIN GPIO_PIN_9
#define TRIG_PORT GPIOA
#define ECHO_PIN GPIO_PIN_8
#define ECHO_PORT GPIOA
uint32_t pMillis;
uint32_t Value1 = 0;
uint32_t Value2 = 0;
uint16_t Distance = 0; // cm
uint8_t senddata[] = "Hello World";
char senddata_2[100];
char rec;
char b[100];
int i = 0;

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    if (rec != 13)
        b[i++] = rec;
    else if (rec == 13) {
        i = 0;
        HAL_UART_Transmit(&huart6, &b, sizeof(b), 100);
        for (int cnt = 0; cnt < sizeof(b); cnt++)
            b[cnt] = NULL;
    }
    HAL_UART_Receive_IT(&huart6, (uint8_t *) &rec, 1);
}

// Let's write the callback function

/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void) {
    /* USER CODE BEGIN 1 */
    HAL_StatusTypeDef ret;
    uint8_t buf[12];
    int16_t val;
}

```

```

float temp_c;
/* 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_TIM1_Init();
MX_USART6_UART_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start(&htim1);
HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_RESET); // pull the TRIG pin low
HAL_UART_Receive_IT(&huart6, (uint8_t*) &rec, 1);
uint8_t data_read[4];
uint8_t data_read1[4];
int16_t aux;
int16_t aux1;
int16_t ambient_temperature;
int16_t ambient_temperature1;
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */

```

```

HAL_StatusTypeDef status;
status = HAL_I2C_IsDeviceReady(&hi2c1, (0x5A << 1), 4, 100);
if (status == HAL_OK) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
}
while (1) {

    HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_SET); // pull the TRIG pin HIGH
    __HAL_TIM_SET_COUNTER(&htim1, 0);
    while (__HAL_TIM_GET_COUNTER (&htim1) < 10)
        ; // wait for 10 us
    HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_RESET); // pull the TRIG pin low

    pMillis = HAL_GetTick(); // used this to avoid infinite while loop (for timeout)
    // wait for the echo pin to go high
    while (!(HAL_GPIO_ReadPin(ECHO_PORT, ECHO_PIN))
        && pMillis + 10 > HAL_GetTick())
        ;
    Value1 = __HAL_TIM_GET_COUNTER(&htim1);

    pMillis = HAL_GetTick(); // used this to avoid infinite while loop (for timeout)
    // wait for the echo pin to go low
    while ((HAL_GPIO_ReadPin(ECHO_PORT, ECHO_PIN))
        && pMillis + 50 > HAL_GetTick())
        ;
    Value2 = __HAL_TIM_GET_COUNTER(&htim1);

    Distance = (Value2 - Value1) * 0.034 / 2;
    sprintf(data, "%d \r\n", Distance);
    HAL_UART_Transmit(&huart2, (uint8_t*) data, strlen(data), 1000);
    HAL_Delay(50);

    HAL_I2C_Mem_Read(&hi2c1, (0x5A << 1), 0x06, 1, (uint8_t*) data_read, 2,
                    100);

    aux = (int16_t) ((data_read[1] << 8) | data_read[0]);
    ambient_temperature = aux * 0.02 - 273.15;
}

```

```

    HAL_Delay(100);

    HAL_I2C_Mem_Read(&hi2c1, (0x5A << 1), 0x07, 1, (uint8_t*) data_read1, 2,
                      100);

    aux1 = (int16_t) ((data_read1[1] << 8) | data_read1[0]);
    ambient_temperature1 = aux1 * 0.02 - 273.15;

    HAL_Delay(100);

    sprintf(data, "f Ambient = %d \r\n", ambient_temperature);

    sprintf(data1, "%d %d \r\n", ambient_temperature1, Distance);

    HAL_UART_Transmit(&huart2, (uint8_t*) data, strlen(data), 1000);
    HAL_UART_Transmit(&huart2, (uint8_t*) data1, strlen(data1), 1000);
    HAL_UART_Transmit(&huart6, (uint8_t*) data1, strlen(data1), 1000);

    //HAL_UART_Transmit(&huart6, senddata, strlen(senddata), 100);
    //HAL_UART_Transmit(&huart2, buffer, strlen(buffer), 100);
    HAL_Delay(200);
    //sprintf(senddata_2, "%d", state);
    //HAL_UART_Transmit(&huart6, senddata_2, strlen(senddata_2), 100);
    HAL_Delay(1000);

//    HAL_UART_Transmit(&huart6, senddata, strlen(senddata), 100);
//    HAL_Delay(1000);

/* USER CODE END WHILE */

/*
 * USER CODE BEGIN 3 */
}

/* USER CODE END 3 */

}

/***
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void) {

```

```

RCC_OsclInitTypeDef RCC_OsclInitStruct = { 0 };

RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };

/** Configure the main internal regulator output voltage
 */
__HAL_RCC_PWR_CLK_ENABLE();
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

/** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OsclInitTypeDef structure.
 */
RCC_OsclInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OsclInitStruct.HSEState = RCC_HSE_ON;
RCC_OsclInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OsclInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OsclInitStruct.PLL.PLLM = 4;
RCC_OsclInitStruct.PLL.PLLN = 72;
RCC_OsclInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OsclInitStruct.PLL.PLLQ = 4;
if (HAL_RCC_OscConfig(&RCC_OsclInitStruct) != 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 I2C1 Initialization Function
 * @param None
 * @retval None
 */

```

```

/*
static void MX_I2C1_Init(void) {

    /* USER CODE BEGIN I2C1_Init 0 */

    /* USER CODE END I2C1_Init 0 */

    /* USER CODE BEGIN I2C1_Init 1 */

    /* USER CODE END I2C1_Init 1 */

    hi2c1.Instance = I2C1;
    hi2c1.Init.ClockSpeed = 100000;
    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 TIM1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM1_Init(void) {

    /* USER CODE BEGIN TIM1_Init 0 */

    /* USER CODE END TIM1_Init 0 */

```

```

TIM_MasterConfigTypeDef sMasterConfig = { 0 };

TIM_IC_InitTypeDef sConfigIC = { 0 };

/* USER CODE BEGIN TIM1_Init 1 */

/* USER CODE END TIM1_Init 1 */

htim1.Instance = TIM1;
htim1.Init.Prescaler = 72 - 1;
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
htim1.Init.Period = 0xffff - 1;
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim1.Init.RepetitionCounter = 0;
htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_IC_Init(&htim1) != HAL_OK) {
    Error_Handler();
}

sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig)
    != HAL_OK) {
    Error_Handler();
}

sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 0;
if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_1) != HAL_OK) {
    Error_Handler();
}

/* USER CODE BEGIN TIM1_Init 2 */

/* USER CODE END TIM1_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 USART6 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART6_UART_Init(void) {

    /* USER CODE BEGIN USART6_Init 0 */

    /* USER CODE END USART6_Init 0 */

    /* USER CODE BEGIN USART6_Init 1 */

```

```

/* USER CODE END USART6_Init 1 */

huart6.Instance = USART6;
huart6.Init.BaudRate = 115200;
huart6.Init.WordLength = UART_WORDLENGTH_8B;
huart6.Init.StopBits = UART_STOPBITS_1;
huart6.Init.Parity = UART_PARITY_NONE;
huart6.Init.Mode = UART_MODE_TX_RX;
huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart6.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart6) != HAL_OK) {
    Error_Handler();
}

/* USER CODE BEGIN USART6_Init 2 */

/* USER CODE END USART6_Init 2 */

}

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

    /* 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(GPIOA, LD2_Pin | GPIO_PIN_9, 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 pins : LD2_Pin PA9 */
    GPIO_InitStruct.Pin = LD2_Pin | GPIO_PIN_9;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
 * @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 */
}

#endif 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 */
```