VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Microprocessors-Microcontrollers (CO3010)

Assignment

Traffic Light Circuit

Advisor: Nguyễn Thiên Ân (Tutor)

Lê Trọng Nhân (Head Teacher)

Group Members:
Lưu Bảo Long - 2252442
Hoàng Văn Phi - 2252608
Phạm Nguyễn Minh Hiếu - 2252215

Ho Chi Minh City, November 2024

Contents

1 Introduction

The project involves designing a traffic light control system using the STM32 microcontroller, an I2C-based 16x2 LCD, and LEDs to simulate the behavior of a real-world traffic signal. This system aims to demonstrate the basic operation of traffic lights, including adjustable timing for each signal phase, allowing dynamic control over the duration of red, amber, and green lights. The STM32 microcontroller serves as the core component, handling state changes and timing for the LEDs that represent different traffic light colors.

- **I2C LED**: This implementation uses a 16x2 LCD to provide a user-friendly interface, displaying relevant information such as the current mode, the countdown for each light, and configuration changes. The I2C communication protocol enables efficient data transfer between the LCD and the microcontroller, reducing the pin usage and simplifying the wiring.
- **Modes**: The system can operate in two primary modes: Normal Mode and Configuration Mode. In Normal Mode, the traffic lights cycle through the standard red, amber, and green phases. Configuration Mode allows the user to adjust the duration of each light phase, providing flexibility for various traffic conditions. Three buttons are used to control the system: one for toggling between modes, one for selecting the phase to adjust, and one for setting the desired time. This project provides a practical application of embedded systems concepts, integrating hardware and software components to create a simple yet effective traffic light simulator.

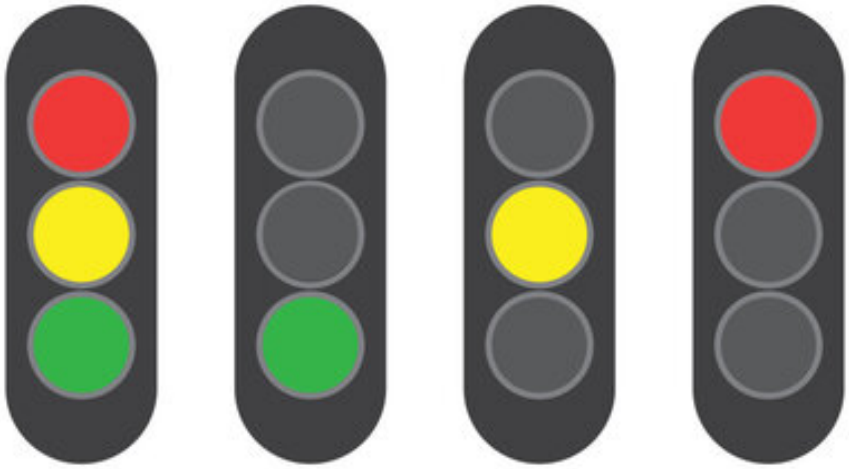

Figure 1

2 Report

2.1 FSM

Finite State Machine (FSM) Description

The following Finite State Machine (FSM) represents a traffic light control system with configurable timing for each LED color. Below is a detailed breakdown of the states and their transitions:

States

- **Normal Mode**:

  The default operational state where the traffic light cycles through the standard phases: red, amber, and green.

  - **Modify Red LED's Time**: This state allows the user to adjust the duration of the red LED. The system enters this state from Normal Mode upon pressing the "Button Mode."
  - **Modify Amber LED's Time**: In this state, the user can modify the duration of the amber LED. The system transitions here from the Modify Red LED's Time state upon pressing "Button Mode."
  - **Modify Green LED's Time**: This state is used to adjust the duration of the green LED. Transition to this state occurs from the Modify Amber LED's Time state upon pressing "Button Mode."
  - **Time Set (0 - 99)**: This state enables the user to set the time value for the selected LED. It is accessed from any of the Modify [Color] LED's Time states by pressing "Button 2 - Set Time."

Transitions

- **Button Mode**:
  - Pressing "Button Mode" in **Normal Mode** transitions the system to the **Modify Red LED's Time** state.
  - Pressing "Button Mode" in **Modify Red LED's Time** transitions the system to **Modify Amber LED's Time**.
  - Pressing "Button Mode" in **Modify Amber LED's Time** transitions the system to **Modify Green LED's Time**.
  - Pressing "Button Mode" in **Modify Green LED's Time** transitions the system back to **Normal Mode**.
- **Button 2 - Set Time**:
  - Pressing "Button 2" in any of the **Modify [Color] LED's Time** states transitions the system to the **Time Set (0 - 99)** state, where the user can configure the time duration.
- **Button 3 - Set**:
  - While in the **Time Set (0 - 99)** state, pressing "Button 3" saves the specified time value and transitions the system back to **Normal Mode**.

Summary

This FSM provides a user-friendly and configurable traffic light system. Users can navigate through different states using simple button presses, enabling the adjustment of durations for each LED color. The structured transitions ensure logical control and flexibility, allowing the system to adapt to various traffic management needs.

2.2 STM32 Setup

2.2.1 STM32CubeMx

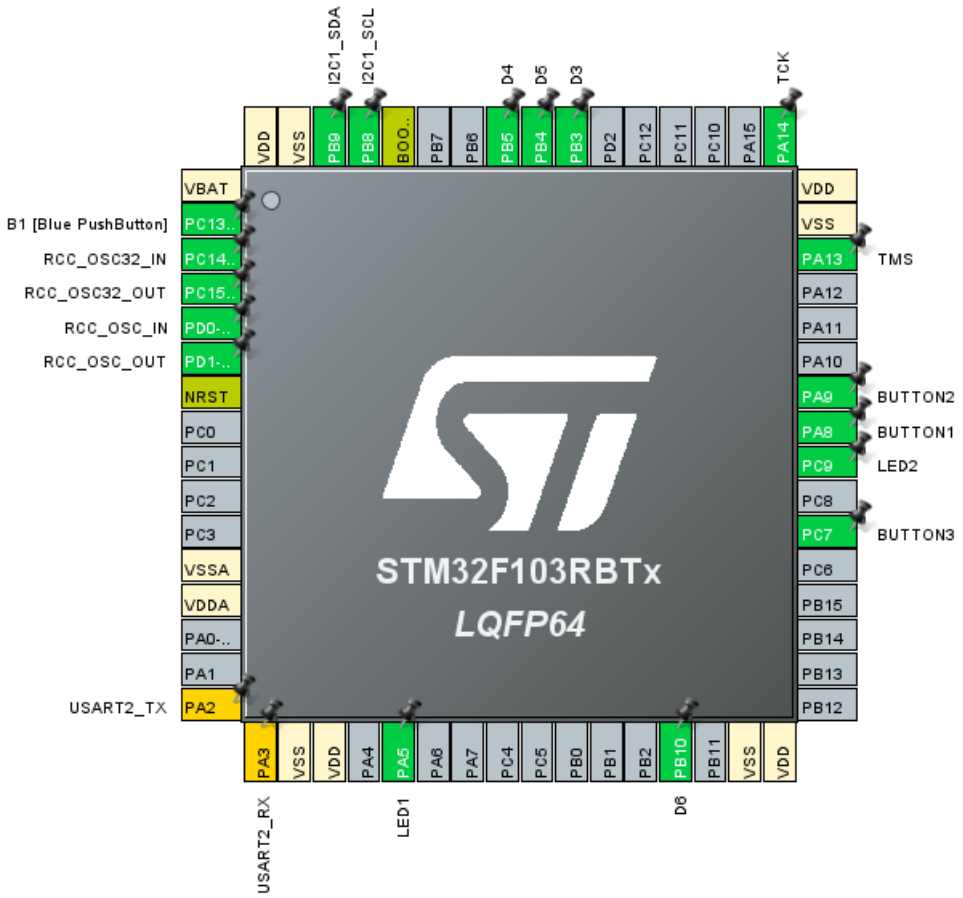

Figure 2

The image shows the pinout diagram of the STM32F103RBTx microcontroller in LQFP64 (Low-Profile Quad Flat Package with 64 pins). This pinout provides a visual representation of all the input/output pins available on the microcontroller, as well as their functions. Here are some notable aspects of the diagram:

- **Power Pins**:
  - VDD and VSS are power supply pins. VDD supplies voltage, and VSS is ground.

  - VBAT is for battery input to maintain backup registers and RTC (Real Time Clock) when VDD is off.

- **Reset Pin**:
  - **NRST** (Pin 7) is the reset pin used to reset the microcontroller.

- **General-Purpose Input/Output (GPIO) Pins**:
  - GPIO pins are labeled from **PA0** to **PA15**, **PB0** to **PB15**, **PC0** to **PC15**, and **PD0** to **PD1**. These pins can be used as digital I/O.
  - Specific GPIO pins are marked with labels like **LED1**, **BUTTON1**, **I2C1_SDA**, and **USART2_TX**, indicating their use for specific functions or peripherals.

- **Oscillator Pins**:
  - Pins labeled **RCC_OSC32_IN** and **RCC_OSC32_OUT** are for connecting an external 32 kHz crystal oscillator for the Real-Time Clock (RTC).
  - **RCC_OSC_IN** and **RCC_OSC_OUT** are for connecting the main crystal oscillator to drive the system clock.

- **Peripheral Pins**:
  - Several pins are labeled for specific peripheral functions:
    * **USART2_TX** and **USART2_RX** (PA2 and PA3) for serial communication.
    * **I2C1_SCL** and **I2C1_SDA** (PB6 and PB7) for I2C communication.
    * **TCK**, **TMS** (PA14, PA13) for debugging (JTAG/SWD interface).

- **Buttons and LEDs**:
  - The diagram shows multiple pins assigned specific functions, such as **BUTTON1**, **BUTTON2**, **BUTTON3**, and **LED1**, **LED2**.
  - These represent typical use cases where buttons and LEDs are directly connected to GPIO pins.

- **Debug Interface**:
  - Pins **PA13** (TMS) and **PA14** (TCK) are used for Serial Wire Debug (SWD), a debugging protocol supported by STM32 microcontrollers.

2.2.2 DEMO

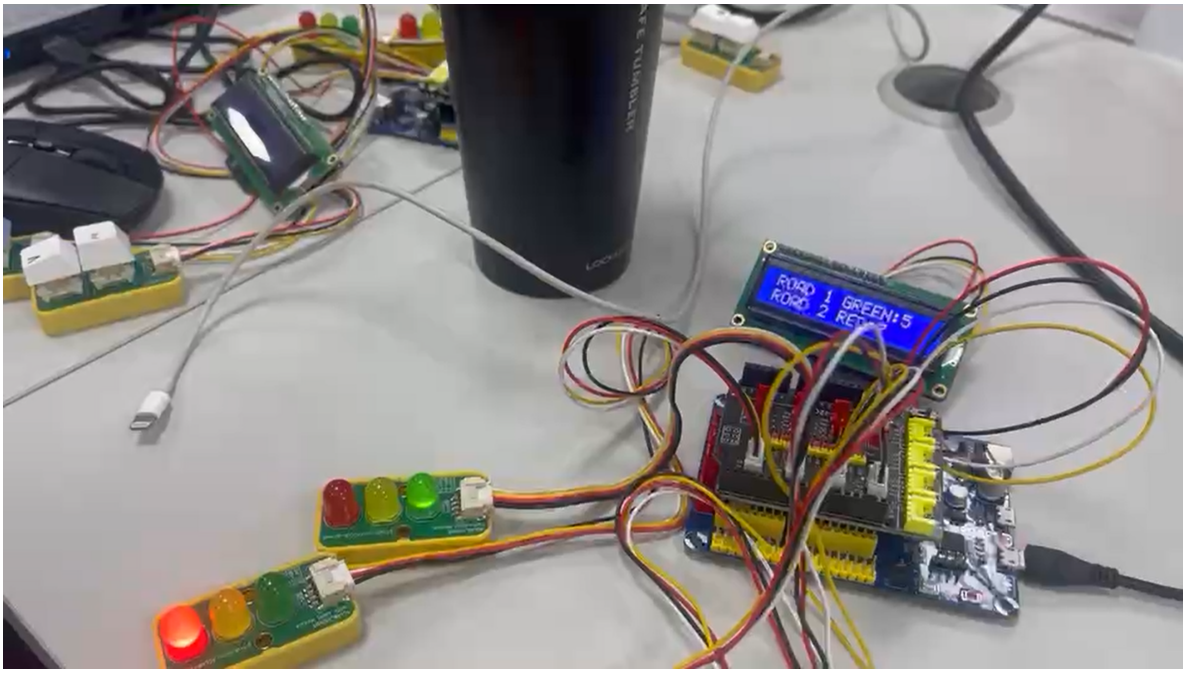

Figure 3

This image shows a traffic light simulation model with red, yellow, and green LEDs, along with an LCD screen displaying the status of each lane. On the LCD screen, it shows ”ROAD 1” with the status ”GREEN” and a countdown timer, while ”ROAD 2” is in the ”RED” state. The system is set up using microcontrollers and complex wiring to control the LEDs and display, simulating a part of a real road traffic system.

3 Code Showing

3.1 7SEG

3.1.1 7SEG.h

```c
/*
 * 7SEG.h
 *
 *  Created on: Aug 31, 2024
 *      Author: ADMIN
 */

#ifndef INC_7SEG_H_
#define INC_7SEG_H_

#include "main.h"

void display7SEG(int num);

typedef struct {
    GPIO_TypeDef *GPIO_Port;
    uint16_t GPIO_Pin;

} LED_CONTROL;
extern int led1;
extern int led2;
extern uint8_t datasend[100];
void mode1Counter2();
void update7SEG();
void mode1Counter();
void displayMode();
#endif /* INC_7SEG_H_ */
```

3.1.2 7SEG.c

```c
/*
 * 7SEG.c
 *
 *  Created on: Aug 31, 2024
 *      Author: ADMIN
 */

#include "7SEG.h"
#include "global.h"
#include "timer.h"
#include "FSM.h"
#include "i2c-lcd.h"
#include <stdio.h>
#include "string.h"

int led1 = RED_TIME;
int led2 = GREEN_TIME;
int state1 = AUTO_RED;
int state2 = AUTO_GREEN;
```

```
20
21  void resetCountValue() {
22      led1 = red_value;
23      led2 = green_value;
24      state1 = AUTO_RED;
25      state2 = AUTO_GREEN;
26  }
27
28  char str1[20];
29  char str2[20];
30
31  void mode1Counter() {
32      led1--;
33      led2--;
34
35      switch (state1) {
36          case AUTO_RED:
37              HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin, GPIO_PIN_SET);
38              HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin, GPIO_PIN_SET);
39              strcpy(str1, "RED:");
40              if (led1 <= 0) {
41                  led1 = green_value;
42                  state1 = AUTO_GREEN;
43                  strcpy(str1, "GREEN:");
44              }
45              break;
46
47          case AUTO_YELLOW:
48              HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin, GPIO_PIN_SET);
49              HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin, GPIO_PIN_RESET);
50              strcpy(str1, "YELLOW:");
51              if (led1 <= 0) {
52                  led1 = red_value;
53                  state1 = AUTO_RED;
54                  strcpy(str1, "RED:");
55              }
56              break;
57
58          case AUTO_GREEN:
59              HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin, GPIO_PIN_RESET);
60              HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin, GPIO_PIN_SET);
61              strcpy(str1, "GREEN:");
62              if (led1 <= 0) {
63                  led1 = yellow_value;
64                  state1 = AUTO_YELLOW;
65                  strcpy(str1, "YELLOW:");
66              }
67              break;
68
69          default:
70              break;
71      }
72
73      switch (state2) {
74          case AUTO_RED:
75              HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_SET);
```

```
            HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_SET);
            strcpy(str2, "RED:");
            if (led2 <= 0) {
                led2 = green_value;
                state2 = AUTO_GREEN;
                strcpy(str2, "GREEN:");
            }
            break;

        case AUTO_YELLOW:
            HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_SET);
            HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_RESET);
            strcpy(str2, "YELLOW:");
            if (led2 <= 0) {
                led2 = red_value;
                state2 = AUTO_RED;
                strcpy(str2, "RED:");
            }
            break;

        case AUTO_GREEN:
            HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_SET);
            strcpy(str2, "GREEN:");
            if (led2 <= 0) {
                led2 = yellow_value;
                state2 = AUTO_YELLOW;
                strcpy(str2, "YELLOW:");
            }
            break;

        default:
            break;
    }
}

static int get7SEG1() {
    switch (mode) {
        case 1:
            return led1;
        case 2:
            return red_temp;
        case 3:
            return green_temp;
        case 4:
            return yellow_temp;
        default:
            break;
    }
    return 0;
}

static int get7SEG2() {
    switch (mode) {
        case 1:
            return led2;
```

```
        case 2:
            return red_temp;
        case 3:
            return green_temp;
        case 4:
            return yellow_temp;
        default:
            break;
    }
    return 0;
}

char dataframe1[20];
char dataframe2[20];
char dataframe3[50];
char dataframe4[50];
char dataframe5[50];
char dataframe6[50];

void displayMode() {
    switch (mode) {
        case 1:
            if (get0flag()) {
                lcd_clear_display();
                sprintf(dataframe1, "ROAD 1 %s%d", str1, get7SEG1());
                lcd_goto_XY(1, 0);
                lcd_send_string(dataframe1);

                sprintf(dataframe2, "ROAD 2 %s%d", str2, get7SEG2());
                lcd_goto_XY(0, 0);
                lcd_send_string(dataframe2);

                setTimer0(100);
                mode1Counter();
            }
            break;

        case 2:
            if (get3flag()) {
                lcd_clear_display();
                sprintf(dataframe1, "Red Mode: %d", red_temp);
                lcd_goto_XY(1, 0);
                lcd_send_string(dataframe1);

                sprintf(dataframe2, "Mode: %d", mode);
                lcd_goto_XY(0, 0);
                lcd_send_string(dataframe2);

                setTimer3(50);
                HAL_GPIO_TogglePin(D4_GPIO_Port, D4_Pin);
                HAL_GPIO_TogglePin(D3_GPIO_Port, D3_Pin);
                HAL_GPIO_TogglePin(D5_GPIO_Port, D5_Pin);
                HAL_GPIO_TogglePin(D6_GPIO_Port, D6_Pin);
            }
            break;

```

```
        case 3:
            if (get3flag()) {
                lcd_clear_display();
                sprintf(dataframe1, "Green Mode: %d", green_temp);
                lcd_goto_XY(1, 0);
                lcd_send_string(dataframe1);

                sprintf(dataframe2, "Mode: %d", mode);
                lcd_goto_XY(0, 0);
                lcd_send_string(dataframe2);

                setTimer3(50);
                HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_RESET);
                HAL_GPIO_TogglePin(D3_GPIO_Port, D3_Pin);
                HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin, GPIO_PIN_RESET);
                HAL_GPIO_TogglePin(D5_GPIO_Port, D5_Pin);
            }
            break;

        case 4:
            if (get3flag()) {
                lcd_clear_display();
                sprintf(dataframe1, "Yellow Mode: %d", yellow_temp);
                lcd_goto_XY(1, 0);
                lcd_send_string(dataframe1);

                sprintf(dataframe2, "Mode: %d", mode);
                lcd_goto_XY(0, 0);
                lcd_send_string(dataframe2);

                setTimer3(50);
                HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_RESET);
                HAL_GPIO_TogglePin(D4_GPIO_Port, D4_Pin);
                HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin, GPIO_PIN_RESET);
                HAL_GPIO_TogglePin(D6_GPIO_Port, D6_Pin);
            }
            break;

        default:
            break;
    }
}
```

3.2 FSM

```c
/*
 * FSM.h
 *
 *  Created on: Sep 14, 2024
 *      Author: ADMIN
 */

#ifndef INC_FSM_H_
#define INC_FSM_H_

#include "global.h"

void fsm_run();
void fsm2_run();
void OtherMode();
#endif /* INC_FSM_H_ */
```

3.3 Global

3.3.1 Global.h

```c
/*
 * global.h
 *
 *  Created on: Sep 14, 2024
 *      Author: ADMIN
 */

#ifndef INC_GLOBAL_H_
#define INC_GLOBAL_H_
#include "timer.h"
#include "main.h"
#include "7SEG.h"
#include "input_reading.h"




#define INIT                1
#define AUTO_RED            2
#define AUTO_GREEN          3
#define AUTO_YELLOW         4

#define INIT1               1
#define AUTO_RED1           2
#define AUTO_GREEN1         3
#define AUTO_YELLOW1        4


#define RED_TIME        5
#define YELLOW_TIME     2
#define GREEN_TIME      3
extern int status;
```

```
extern int status1;
void LedInit();
extern int mode;
extern int red_value, yellow_value, green_value;
extern int red_temp, yellow_temp, green_temp;
#endif /* INC_GLOBAL_H_ */
```

3.3.2 Global.c

```
/*
 * global.c
 *
 *  Created on: Sep 14, 2024
 *      Author: ADMIN
 */

#include "global.h"

int mode;
int red_value, yellow_value, green_value;
int red_temp, yellow_temp, green_temp;
int status = 0;
int status1 = 0;
void LedInit() {
    red_value = RED_TIME;
    yellow_value = YELLOW_TIME;
    green_value = GREEN_TIME;
    red_temp = red_value;
    yellow_temp = yellow_value;
    green_temp = green_value;

}
```

3.4 I2C-LCD

3.4.1 I2C-LCD.h

```
/*
 * i2c-lcd.h
 *
 *  Created on: Oct 31, 2024
 *      Author: ADMIN
 */

#ifndef INC_I2C_LCD_H_
#define INC_I2C_LCD_H_

/**
Edit by modify: Ngoc Hang
**/

#include "stm32f1xx_hal.h"

void lcd_init (void);   // initialize lcd
```

```
18
19 void lcd_send_cmd (char cmd);  // send command to the lcd
20
21 void lcd_send_data (char data);  // send data to the lcd
22
23 void lcd_send_string (char *str);  // send string to the lcd
24
25 void lcd_clear_display (void);  //clear display lcd
26
27 void lcd_goto_XY (int row, int col); //set proper location on screen
28
29
30
31 #endif /* INC_I2C_LCD_H_ */
```

3.4.2 I2C-LCD.c

```
1  /**
2  Edit by modify: Ngoc Hang
3  **/
4
5  #include "i2c-lcd.h"
6  extern I2C_HandleTypeDef hi2c1;  // change your handler here accordingly
7
8  #define SLAVE_ADDRESS_LCD (0x21 << 1) // change this according to ur setup
9
10 void lcd_send_cmd (char cmd)
11 {
12   char data_u, data_l;
13     uint8_t data_t[4];
14     data_u = (cmd&0xf0);
15     data_l = ((cmd<<4)&0xf0);
16     data_t[0] = data_u|0x0C;  //en=1, rs=0
17     data_t[1] = data_u|0x08;  //en=0, rs=0
18     data_t[2] = data_l|0x0C;  //en=1, rs=0
19     data_t[3] = data_l|0x08;  //en=0, rs=0
20     HAL_I2C_Master_Transmit (&hi2c1, SLAVE_ADDRESS_LCD,(uint8_t *) data_t, 4, 100)
           ;
21 }
22
23 void lcd_send_data (char data)
24 {
25     char data_u, data_l;
26     uint8_t data_t[4];
27     data_u = (data&0xf0);
28     data_l = ((data<<4)&0xf0);
29     data_t[0] = data_u|0x0D;  //en=1, rs=0
30     data_t[1] = data_u|0x09;  //en=0, rs=0
31     data_t[2] = data_l|0x0D;  //en=1, rs=0
32     data_t[3] = data_l|0x09;  //en=0, rs=0
33     HAL_I2C_Master_Transmit (&hi2c1, SLAVE_ADDRESS_LCD,(uint8_t *) data_t, 4, 100)
           ;
34 }
35
36 void lcd_init (void) {
```

```
    lcd_send_cmd (0x33); /* set 4-bits interface */
    lcd_send_cmd (0x32);
    HAL_Delay(50);
    lcd_send_cmd (0x28); /* start to set LCD function */
    HAL_Delay(50);
    lcd_send_cmd (0x01); /* clear display */
    HAL_Delay(50);
    lcd_send_cmd (0x06); /* set entry mode */
    HAL_Delay(50);
    lcd_send_cmd (0x0c); /* set display to on */
    HAL_Delay(50);
    lcd_send_cmd (0x02); /* move cursor to home and set data address to 0 */
    HAL_Delay(50);
    lcd_send_cmd (0x80);
}

void lcd_send_string (char *str)
{
    while (*str) lcd_send_data (*str++);
}

void lcd_clear_display (void)
{

    lcd_goto_XY(0, 0);
    lcd_send_string("                ");


    lcd_goto_XY(1, 0);
    lcd_send_string("                ");


    lcd_goto_XY(0, 0);
}

void lcd_goto_XY (int row, int col)
{
    uint8_t pos_Addr;
    if(row == 1)
    {
        pos_Addr = 0x80 + row - 1 + col;
    }
    else
    {
        pos_Addr = 0x80 | (0x40 + col);
    }
    lcd_send_cmd(pos_Addr);
}
```

3.5 Input-processing

3.5.1 Input-processing.h

```
/*
 * input_process.h
```

```
 *
 *  Created on: Sep 18, 2024
 *      Author: ADMIN
 */

#ifndef INC_INPUT_PROCESS_H_
#define INC_INPUT_PROCESS_H_
#include "main.h"
#include "input_reading.h"
#include "7SEG.h"
#include "timer.h"
#include "global.h"

void ModeBuffer();
void DurationInit();
void Update();

#endif /* INC_INPUT_PROCESS_H_ */
```

3.5.2 Input-processing.c

```
/*
 * input_processing.c
 *
 *  Created on: Jul 25, 2023
 *      Author: BAO LONG
 */

#include "input_process.h"

enum ButtonState{BUTTON_RELEASED, BUTTON_PRESSED,
    BUTTON_PRESSED_MORE_THAN_1_SECOND};
enum ButtonState buttonState[3] = {BUTTON_RELEASED};

void ModeBuffer() {
    switch (buttonState[0]) {
        case BUTTON_RELEASED:
            if (is_button_pressed(0)) {
                buttonState[0] = BUTTON_PRESSED;
                mode++;
                if (mode > 4) {
                    mode = 1;
                    resetCountValue();
                }
            }
            break;
        case BUTTON_PRESSED:
            if (!is_button_pressed(0)) {
                buttonState[0] = BUTTON_RELEASED;
            }
            break;
        default:
            break;
    }
}
```

```
34
35  static void increaseDraftValue() {
36      switch (mode) {
37          case 2:
38              red_temp++;
39              if (red_temp > 99) red_temp = 0;
40              break;
41          case 3:
42
43              green_temp++;
44              if (green_temp > 99) green_temp = 0;
45              break;
46          case 4:
47              yellow_temp++;
48              if (yellow_temp > 99) yellow_temp = 0;
49              break;
50          default:
51              break;
52      }
53  }
54
55  static void setDurationValue() {
56      int diff = 0;
57      switch (mode) {
58          case 2:
59              diff = red_temp - red_value;
60              red_value = red_temp;
61              green_value += diff;
62              green_temp += diff;
63              break;
64          case 3:
65
66              diff = green_temp - green_value;
67              green_value = green_temp;
68              red_value += diff;
69              red_temp += diff;
70              break;
71          case 4:
72              diff = yellow_temp - yellow_value;
73              yellow_value = yellow_temp;
74              red_value += diff;
75              red_temp += diff;
76
77              break;
78          default:
79              break;
80      }
81  }
82
83  void Update() {
84      switch (buttonState[1]) {
85          case BUTTON_RELEASED:
86              if (is_button_pressed(1)) {
87                  buttonState[1] = BUTTON_PRESSED;
88                  increaseDraftValue();
89              }
```

```
            break;
        case BUTTON_PRESSED:
            if (!is_button_pressed(1)) {
                buttonState[1] = BUTTON_RELEASED;
            }
            if (is_button_pressed_1s(1)) {
                buttonState[1] = BUTTON_PRESSED_MORE_THAN_1_SECOND;
                increaseDraftValue();
            }
            break;
        case BUTTON_PRESSED_MORE_THAN_1_SECOND:
            if (!is_button_pressed(1)) {
                buttonState[1] = BUTTON_RELEASED;
            }
            if (is_button_held(1)) {
                reset_flagForButtonHold(1);
                increaseDraftValue();
            }
            break;
        default:
            break;
    }

    switch (buttonState[2]) {
        case BUTTON_RELEASED:
            if (is_button_pressed(2)) {
                buttonState[2] = BUTTON_PRESSED;
                setDurationValue();
                mode=1;
            }
            break;
        case BUTTON_PRESSED:
            if (!is_button_pressed(2)) {
                buttonState[2] = BUTTON_RELEASED;
            }
            break;
        default:
            break;
    }
}
```

3.6 Input-reading

3.6.1 Input-reading.h

```
/*
 * input_reading.h
 *
 *  Created on: Sep 13, 2024
 *      Author: ADMIN
 */

#ifndef INC_INPUT_READING_H_
#define INC_INPUT_READING_H_
#include "main.h"
```

```c
typedef struct {
    GPIO_TypeDef *pGPIOx;
    uint16_t pin;
} BUTTON_CONTROL;
extern BUTTON_CONTROL Button[];
void init_button();
void button_reading(void);
unsigned char is_button_pressed(unsigned char index);
unsigned char is_button_pressed_1s(unsigned char index);
unsigned char is_button_held(unsigned char index);
void reset_flagForButtonHold(unsigned char index);
#endif /* INC_INPUT_READING_H_ */
```

3.6.2 Input-reading.c

```c
/*
 * input_reading.c
 *
 *  Created on: Sep 13, 2024
 *      Author: ADMIN
 */

#include "input_reading.h"

#define NO_OF_BUTTONS                    3

BUTTON_CONTROL Button[] = {
    { BUTTON1_GPIO_Port, BUTTON1_Pin },
    { BUTTON2_GPIO_Port, BUTTON2_Pin },
    { BUTTON3_GPIO_Port, BUTTON3_Pin }
};

// Timer interrupt duration is 10ms, so to pass 1 second,
// we need to jump to the interrupt service routine 100 times
#define DURATION_FOR_AUTO_INCREASING     100
#define DURATION_FOR_HOLD                50
#define BUTTON_IS_PRESSED                GPIO_PIN_RESET
#define BUTTON_IS_RELEASED               GPIO_PIN_SET

// Buffer to store final result after debouncing
static GPIO_PinState buttonBuffer[NO_OF_BUTTONS];

// Buffers for debouncing
static GPIO_PinState debounceButtonBuffer1[NO_OF_BUTTONS];
static GPIO_PinState debounceButtonBuffer2[NO_OF_BUTTONS];
static GPIO_PinState debounceButtonBuffer3[NO_OF_BUTTONS];

// Flags and counters for button actions
static uint8_t flagForButtonPress1s[NO_OF_BUTTONS];
static uint8_t flagForButtonHold[NO_OF_BUTTONS];
static uint16_t counterForButtonPress1s[NO_OF_BUTTONS];
static uint16_t counterForButtonHold[NO_OF_BUTTONS];

// Initialize button states
void init_button() {
```

```
    for (int i = 0; i < NO_OF_BUTTONS; i++) {
        buttonBuffer[i] = BUTTON_IS_RELEASED;
        debounceButtonBuffer1[i] = BUTTON_IS_RELEASED;
        debounceButtonBuffer2[i] = BUTTON_IS_RELEASED;
        debounceButtonBuffer3[i] = BUTTON_IS_RELEASED;
        flagForButtonPress1s[i] = 0;
        flagForButtonHold[i] = 0;
        counterForButtonPress1s[i] = 0;
        counterForButtonHold[i] = 0;
    }
}

// Read button states with debouncing
void button_reading(void) {
    for (char i = 0; i < NO_OF_BUTTONS; i++) {
        debounceButtonBuffer3[i] = debounceButtonBuffer2[i];
        debounceButtonBuffer2[i] = debounceButtonBuffer1[i];
        debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(Button[i].pGPIOx, Button[i].
            pin);

        if (debounceButtonBuffer1[i] == debounceButtonBuffer2[i] &&
            debounceButtonBuffer3[i] == debounceButtonBuffer2[i]) {
            buttonBuffer[i] = debounceButtonBuffer1[i];
        }

        if (buttonBuffer[i] == BUTTON_IS_PRESSED) {
            // Reset all GPIO pins when button is pressed
            HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin, GPIO_PIN_RESET);

            // Increment counter for button press duration
            if (counterForButtonPress1s[i] < DURATION_FOR_AUTO_INCREASING) {
                counterForButtonPress1s[i]++;
            } else {
                // Set flag when button has been pressed for 1 second
                flagForButtonPress1s[i] = 1;

                // Handle button hold
                if (counterForButtonHold[i] < DURATION_FOR_HOLD) {
                    counterForButtonHold[i]++;
                    if (counterForButtonHold[i] >= DURATION_FOR_HOLD) {
                        counterForButtonHold[i] = 0;
                        flagForButtonHold[i] = 1;
                    }
                }
            }
        } else {
            // Reset counters and flags if button is released
            counterForButtonPress1s[i] = 0;
            flagForButtonPress1s[i] = 0;
            counterForButtonHold[i] = 0;
            flagForButtonHold[i] = 0;
        }
    }
```

```
}

// Check if button is pressed
unsigned char is_button_pressed(unsigned char index) {
    if (index >= NO_OF_BUTTONS) return 0;
    return (buttonBuffer[index] == BUTTON_IS_PRESSED);
}

// Check if button is pressed for 1 second
unsigned char is_button_pressed_1s(unsigned char index) {
    if (index >= NO_OF_BUTTONS) return 0;
    return (flagForButtonPress1s[index] == 1);
}

// Check if button is held
unsigned char is_button_held(unsigned char index) {
    if (index >= NO_OF_BUTTONS) return 0;
    return (flagForButtonHold[index] == 1);
}

// Reset flag for button hold
void reset_flagForButtonHold(unsigned char index) {
    if (index < NO_OF_BUTTONS) {
        flagForButtonHold[index] = 0;
    }
}
```

3.7 Timer

3.7.1 Timer.h

```
#ifndef INC_TIMER_H_
#define INC_TIMER_H_

extern int timer0_counter;
extern int timer0_flag;
extern int TIMER_CYCLE;
extern int timer1_counter;
extern int timer1_flag;
extern int timer2_counter;
extern int timer2_flag;
extern int timer3_counter;
extern int timer3_flag;

void setTimer0(int duration);
void setTimer1(int duration);
void setTimer2(int duration);
void setTimer3(int duration);
void timer_run();
int get0flag();
int get1flag();
int get2flag();
int get3flag();

#endif /* INC_TIMER_H_ */
```

3.7.2 Timer.c

```c
/*
 * timer.c
 *
 *  Created on: Sep 13, 2024
 *      Author: ADMIN
 */

/*
 * timer.c
 *
 *  Created on: Sep 7, 2024
 *      Author: ADMIN
 */

#include "timer.h"
#include "7SEG.h"
#define CASE0   0
#define CASE1   1
#define CASE2   2
#define CASE3   3
int timer0_counter = 0;
int timer0_flag = 0;
int TIMER_CYCLE = 1;
int timer1_counter = 0;
int timer1_flag = 0;
int timer2_counter = 0;
int timer2_flag = 0;
int timer3_counter = 0;
int timer3_flag = 0;
void setTimer0(int duration) {
    timer0_counter = duration / TIMER_CYCLE;
    timer0_flag = 0;

}

void setTimer1(int duration) {
    timer1_counter = duration / TIMER_CYCLE;
    timer1_flag = 0;

}
void setTimer2(int duration) {
    timer2_counter = duration / TIMER_CYCLE;
    timer2_flag = 0;

}
void setTimer3(int duration) {
    timer3_counter = duration / TIMER_CYCLE;
    timer3_flag = 0;

}
void timer_run() {
    if (timer0_counter > 0) {
        timer0_counter--;
        if (timer0_counter == 0) timer0_flag = 1;
```

```
55
56      }
57      if (timer1_counter > 0) {
58          timer1_counter--;
59          if (timer1_counter == 0) timer1_flag = 1;
60
61      }
62      if (timer2_counter > 0) {
63          timer2_counter--;
64          if (timer2_counter == 0) timer2_flag = 1;
65
66      }
67      if(timer3_counter>0){
68          timer3_counter--;
69          if(timer3_counter==0) timer3_flag =1;
70      }
71  }
72  int get0flag() {
73      return timer0_flag;
74  }
75  int get1flag() {
76      return timer1_flag;
77  }
78  int get2flag() {
79      return timer2_flag;
80  }
81  int get3flag() {
82      return timer3_flag;
83  }
```

3.8 MAIN

3.8.1 Main.h

```
1  /* USER CODE BEGIN Header */
2  /**
3    ******************************************************************************
4    * @file           : main.h
5    * @brief          : Header for main.c file.
6    *                   This file contains the common defines of the application.
7    ******************************************************************************
8    * @attention
9    *
10   * Copyright (c) 2024 STMicroelectronics.
11   * All rights reserved.
12   *
13   * This software is licensed under terms that can be found in the LICENSE file
14   * in the root directory of this software component.
15   * If no LICENSE file comes with this software, it is provided AS-IS.
16   *
17   ******************************************************************************
18   */
19 /* USER CODE END Header */
20
21 /* Define to prevent recursive inclusion -------------------------------------*/
```

```c
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f1xx_hal.h"

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

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define B1_EXTI_IRQn EXTI15_10_IRQn
#define LED1_Pin GPIO_PIN_5
#define LED1_GPIO_Port GPIOA
#define D6_Pin GPIO_PIN_10
#define D6_GPIO_Port GPIOB
#define BUTTON3_Pin GPIO_PIN_7
#define BUTTON3_GPIO_Port GPIOC
#define LED2_Pin GPIO_PIN_9
#define LED2_GPIO_Port GPIOC
#define BUTTON1_Pin GPIO_PIN_8
#define BUTTON1_GPIO_Port GPIOA
#define BUTTON2_Pin GPIO_PIN_9
#define BUTTON2_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
```

```
#define TCK_GPIO_Port GPIOA
#define D3_Pin GPIO_PIN_3
#define D3_GPIO_Port GPIOB
#define D5_Pin GPIO_PIN_4
#define D5_GPIO_Port GPIOB
#define D4_Pin GPIO_PIN_5
#define D4_GPIO_Port GPIOB

/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

3.8.2 Main.c

```
/* 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"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "timer.h"
#include "input_reading.h"
#include "global.h"
#include "7SEG.h"
#include "input_process.h"
#include "FSM.h"
#include "i2c-lcd.h"
#include "string.h"
/* USER CODE END Includes */

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

```
36
37 /* USER CODE END PTD */
38
39 /* Private define ------------------------------------------------------------*/
40 /* USER CODE BEGIN PD */
41 /* USER CODE END PD */
42
43 /* Private macro -------------------------------------------------------------*/
44 /* USER CODE BEGIN PM */
45
46 /* USER CODE END PM */
47
48 /* Private variables ---------------------------------------------------------*/
49 I2C_HandleTypeDef hi2c1;
50
51 TIM_HandleTypeDef htim2;
52
53 /* USER CODE BEGIN PV */
54
55 /* USER CODE END PV */
56
57 /* Private function prototypes -----------------------------------------------*/
58 void SystemClock_Config(void);
59 static void MX_GPIO_Init(void);
60 static void MX_TIM2_Init(void);
61 static void MX_I2C1_Init(void);
62 /* USER CODE BEGIN PFP */
63
64 /* USER CODE END PFP */
65
66 /* Private user code ---------------------------------------------------------*/
67 /* USER CODE BEGIN 0 */
68
69 /* USER CODE END 0 */
70
71 /**
72   * @brief  The application entry point.
73   * @retval int
74   */
75 int main(void)
76 {
77   /* USER CODE BEGIN 1 */
78
79   /* USER CODE END 1 */
80
81   /* MCU Configuration--------------------------------------------------------*/
82
83   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
84   HAL_Init();
85
86   /* USER CODE BEGIN Init */
87
88   /* USER CODE END Init */
89
90   /* Configure the system clock */
91   SystemClock_Config();
```

```
92
93    /* USER CODE BEGIN SysInit */
94
95    /* USER CODE END SysInit */
96
97    /* Initialize all configured peripherals */
98    MX_GPIO_Init();
99    MX_TIM2_Init();
100   MX_I2C1_Init();
101   /* USER CODE BEGIN 2 */
102   HAL_TIM_Base_Start_IT(&htim2);
103   /* USER CODE END 2 */
104
105   /* Infinite loop */
106   /* USER CODE BEGIN WHILE */
107   init_button();
108   LedInit();
109
110   setTimer0(100);
111   setTimer1(1);
112   setTimer2(90);
113
114   setTimer3(1);
115   lcd_init();
116
117     mode=1;
118 //  HAL_Delay(50);
119
120
121         while (1)
122         {
123             ModeBuffer();
124
125             displayMode();
126
127             Update();
128     /* USER CODE END WHILE */
129
130
131
132
133     /* USER CODE BEGIN 3 */
134         }
135   /* USER CODE END 3 */
136 }
137
138 /**
139   * @brief System Clock Configuration
140   * @retval None
141   */
142 void SystemClock_Config(void)
143 {
144   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
145   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
146
147   /** Initializes the RCC Oscillators according to the specified parameters
```

```
148   * in the RCC_OscInitTypeDef structure.
149   */
150   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
151   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
152   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
153   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
154   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
155   RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
156   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
157   {
158     Error_Handler();
159   }
160
161   /** Initializes the CPU, AHB and APB buses clocks
162   */
163   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
164                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
165   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
166   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV8;
167   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
168   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
169
170   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
171   {
172     Error_Handler();
173   }
174 }
175
176 /**
177   * @brief I2C1 Initialization Function
178   * @param None
179   * @retval None
180   */
181 static void MX_I2C1_Init(void)
182 {
183
184   /* USER CODE BEGIN I2C1_Init 0 */
185
186   /* USER CODE END I2C1_Init 0 */
187
188   /* USER CODE BEGIN I2C1_Init 1 */
189
190   /* USER CODE END I2C1_Init 1 */
191   hi2c1.Instance = I2C1;
192   hi2c1.Init.ClockSpeed = 100000;
193   hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
194   hi2c1.Init.OwnAddress1 = 0;
195   hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
196   hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
197   hi2c1.Init.OwnAddress2 = 0;
198   hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
199   hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
200   if (HAL_I2C_Init(&hi2c1) != HAL_OK)
201   {
202     Error_Handler();
203   }
```

```
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 7999;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 9;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_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_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, D6_Pin|D3_Pin|D5_Pin|D4_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);

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

  /*Configure GPIO pin : PA2 */
  GPIO_InitStruct.Pin = GPIO_PIN_2;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PA3 */
  GPIO_InitStruct.Pin = GPIO_PIN_3;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : LED1_Pin */
  GPIO_InitStruct.Pin = LED1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : D6_Pin D3_Pin D5_Pin D4_Pin */
  GPIO_InitStruct.Pin = D6_Pin|D3_Pin|D5_Pin|D4_Pin;
  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);

  /*Configure GPIO pin : BUTTON3_Pin */
  GPIO_InitStruct.Pin = BUTTON3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BUTTON3_GPIO_Port, &GPIO_InitStruct);
```

```
316
317   /*Configure GPIO pin : LED2_Pin */
318   GPIO_InitStruct.Pin = LED2_Pin;
319   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
320   GPIO_InitStruct.Pull = GPIO_NOPULL;
321   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
322   HAL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);
323
324   /*Configure GPIO pins : BUTTON1_Pin BUTTON2_Pin */
325   GPIO_InitStruct.Pin = BUTTON1_Pin|BUTTON2_Pin;
326   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
327   GPIO_InitStruct.Pull = GPIO_NOPULL;
328   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
329
330   /* EXTI interrupt init*/
331   HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
332   HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
333
334 }
335
336 /* USER CODE BEGIN 4 */
337
338 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
339
340         timer_run();
341         button_reading();
342
343
344
345  }
346 void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)
347 {
348
349 }
350 /* USER CODE END 4 */
351
352 /**
353   * @brief  This function is executed in case of error occurrence.
354   * @retval None
355   */
356 void Error_Handler(void)
357 {
358   /* USER CODE BEGIN Error_Handler_Debug */
359   /* User can add his own implementation to report the HAL error return state */
360   __disable_irq();
361   while (1)
362   {
363   }
364   /* USER CODE END Error_Handler_Debug */
365 }
366
367 #ifdef  USE_FULL_ASSERT
368 /**
369   * @brief  Reports the name of the source file and the source line number
370   *         where the assert_param error has occurred.
371   * @param  file: pointer to the source file name
```

```
372   * @param  line: assert_param error line source number
373   * @retval None
374   */
375 void assert_failed(uint8_t *file, uint32_t line)
376 {
377   /* USER CODE BEGIN 6 */
378   /* User can add his own implementation to report the file name and line number,
379      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
380   /* USER CODE END 6 */
381 }
382 #endif /* USE_FULL_ASSERT */
```

4 Project Summary

This project simulates and controls a simple traffic light system using STM32F103RBT6, with LEDs representing red, amber, and green lights. The system is designed to allow users to modify the timing for each LED and display the current status on an LCD.
Link GitHub (Name: SESSION1): https://github.com/lokx1/STM32Labs.git

4.0.1 Main Components

- **STM32F103RBT6**: The microcontroller used to control the states of the LEDs.
- **Red, Amber, Green LEDs**: Simulating the traffic lights for both lanes.
- **LCD 16x2**: Displays the current status of the traffic lights and countdown times.
- **Buttons**: Three buttons used to change modes and set the timing for the LEDs.

4.0.2 Main Functions

- **Normal Mode**: Traffic lights sequentially switch from red to amber to green according to the set cycle.
- **Modify Mode**: Allows users to adjust the timing of each LED.
  - Modify Red LED's Time: Adjust the duration of the red light.
  - Modify Amber LED's Time: Adjust the duration of the amber light.
  - Modify Green LED's Time: Adjust the duration of the green light.

4.0.3 Instructions for Use

1. **Normal Mode**:
   - When the system starts, the LEDs operate in Normal Mode.
   - The LCD displays the current status of each lane, for example: `ROAD 1 GREEN: 5`, indicating that lane 1 has a green light with a countdown of 5 seconds.
2. **Switch to Modify Mode**:
   - Press **Button 1** to switch to Modify Mode.
   - Each press cycles through options to modify the timing for the red, amber, and green lights.
3. **Adjust Timing**:

- Press **Button 2** to select the light to adjust.
- Press **Button 3** to increase the time (0-99 seconds). The time will be displayed on the LCD for user reference.
- Press **Button 2** to confirm the new value.

4. **Return to Normal Mode**:
    - After setting, press **Button 1** to return to Normal Mode. The system will now use the updated timing values.

4.0.4 Wiring Diagram

- **STM32F103RBT6**:
    - GPIO pins are used to control the red, amber, and green LEDs.
    - The LCD is connected to appropriate communication pins for status display.
    - Buttons are connected to GPIO pins to receive user control signals.

4.0.5 Notes

- Ensure correct and safe power supply to the STM32 and peripherals.
- Use the buttons carefully to avoid any errors during mode switching.

With this project, you can easily adjust the timing of the traffic lights to fit different traffic conditions, providing flexibility to the system.