

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY  
UNIVERSITY OF TECHNOLOGY  
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



# Microprocessors-Microcontrollers (CO3010)

---

## Assignment Traffic Light Circuit

---

<b>Advisor:</b>	Nguyễn Thiên Ân (Tutor)
	Lê Trọng Nhân (Head Teacher)
<b>Group Members:</b>	Lưu Bảo Long - 2252442
	Hoàng Văn Phi - 2252608
	Phạm Nguyễn Minh Hiếu - 2252215

Ho Chi Minh City, November 2024

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Report</b>	<b>3</b>
2.1	FSM . . . . .	3
2.2	STM32 Setup . . . . .	4
2.2.1	STM32CubeMx . . . . .	4
2.2.2	DEMO . . . . .	6
<b>3</b>	<b>Code Showing</b>	<b>7</b>
3.1	7SEG . . . . .	7
3.1.1	7SEG.h . . . . .	7
3.1.2	7SEG.c . . . . .	7
3.2	FSM . . . . .	12
3.3	Global . . . . .	12
3.3.1	Global.h . . . . .	12
3.3.2	Global.c . . . . .	13
3.4	I2C-LCD . . . . .	13
3.4.1	I2C-LCD.h . . . . .	13
3.4.2	I2C-LCD.c . . . . .	14
3.5	Input-processing . . . . .	15
3.5.1	Input-processing.h . . . . .	15
3.5.2	Input-processing.c . . . . .	16
3.6	Input-reading . . . . .	18
3.6.1	Input-reading.h . . . . .	18
3.6.2	Input-reading.c . . . . .	19
3.7	Timer . . . . .	21
3.7.1	Timer.h . . . . .	21
3.7.2	Timer.c . . . . .	22
3.8	MAIN . . . . .	23
3.8.1	Main.h . . . . .	23
3.8.2	Main.c . . . . .	25
<b>4</b>	<b>Project Summary</b>	<b>32</b>
4.0.1	Main Components . . . . .	32
4.0.2	Main Functions . . . . .	32
4.0.3	Instructions for Use . . . . .	32
4.0.4	Wiring Diagram . . . . .	33
4.0.5	Notes . . . . .	33

# 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.

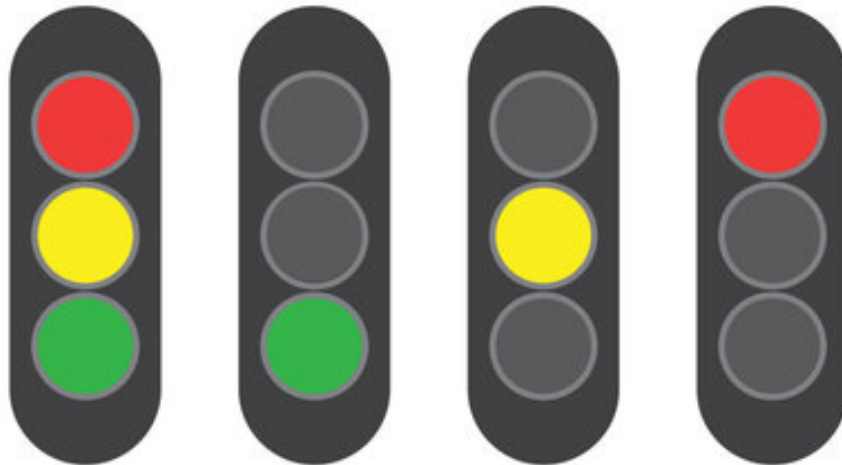


Figure 1

## 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

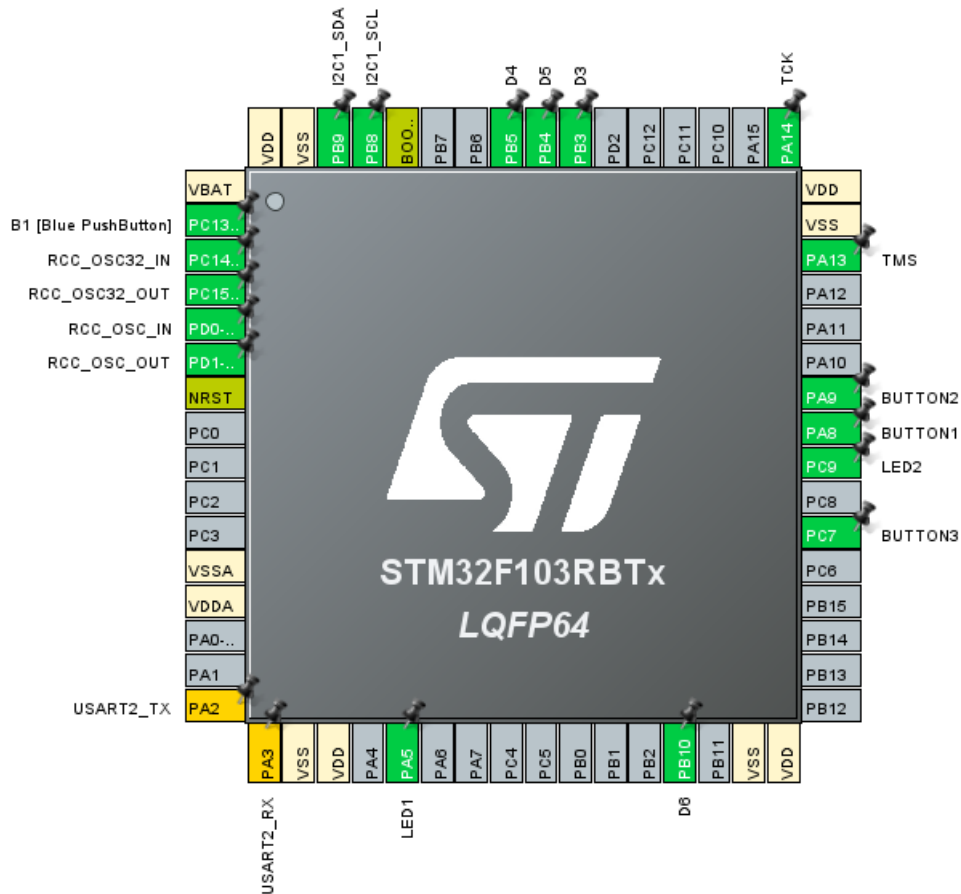


Figure 2

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

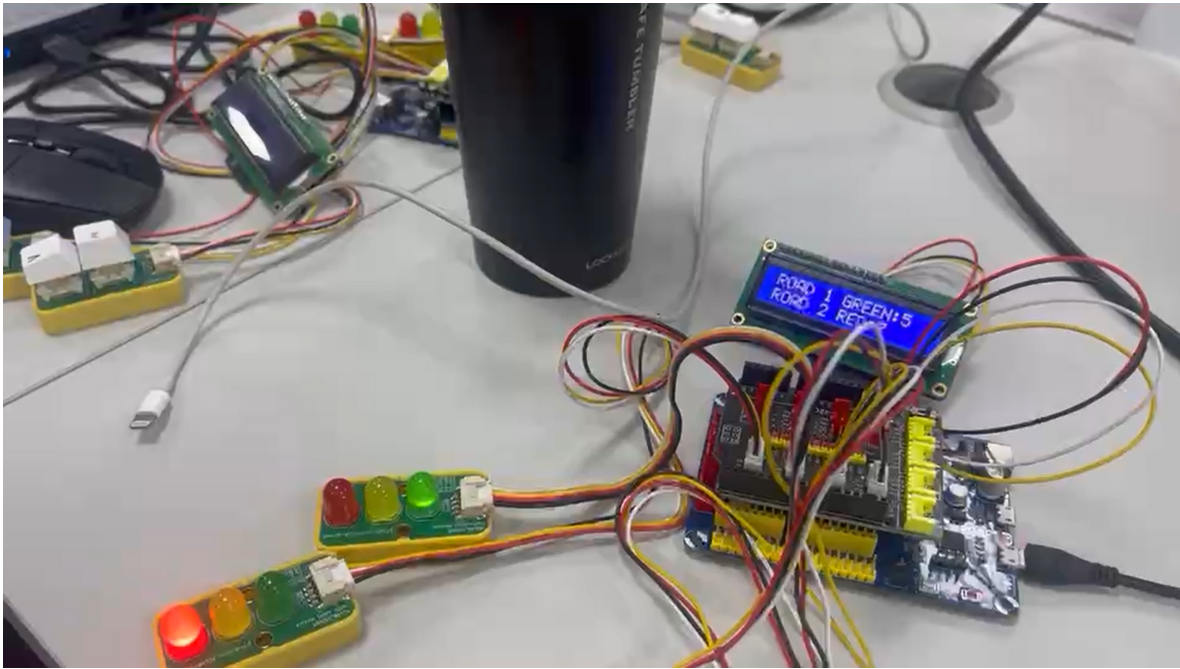


Figure 3

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

```
1  /*
2   * 7SEG.h
3   *
4   * Created on: Aug 31, 2024
5   * Author: ADMIN
6   */
7
8  #ifndef INC_7SEG_H_
9  #define INC_7SEG_H_
10
11  #include "main.h"
12
13  void display7SEG(int num);
14
15  typedef struct {
16      GPIO_TypeDef *GPIO_Port;
17      uint16_t GPIO_Pin;
18  } LED_CONTROL;
19
20  extern int led1;
21  extern int led2;
22  extern uint8_t datasend[100];
23  void modelCounter2();
24  void update7SEG();
25  void modelCounter();
26  void displayMode();
27  #endif /* INC_7SEG_H_ */
```

#### 3.1.2 7SEG.c

```
1  /*
2   * 7SEG.c
3   *
4   * Created on: Aug 31, 2024
5   * Author: ADMIN
6   */
7
8  #include "7SEG.h"
9  #include "global.h"
10 #include "timer.h"
11 #include "FSM.h"
12 #include "i2c-lcd.h"
13 #include <stdio.h>
14 #include "string.h"
15
16 int led1 = RED_TIME;
17 int led2 = GREEN_TIME;
18 int state1 = AUTO_RED;
19 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);

```

```

76     HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_SET);
77     strcpy(str2, "RED:");
78     if (led2 <= 0) {
79         led2 = green_value;
80         state2 = AUTO_GREEN;
81         strcpy(str2, "GREEN:");
82     }
83     break;
84
85     case AUTO_YELLOW:
86         HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_SET);
87         HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_RESET);
88         strcpy(str2, "YELLOW:");
89         if (led2 <= 0) {
90             led2 = red_value;
91             state2 = AUTO_RED;
92             strcpy(str2, "RED:");
93         }
94         break;
95
96     case AUTO_GREEN:
97         HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_RESET);
98         HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_SET);
99         strcpy(str2, "GREEN:");
100         if (led2 <= 0) {
101             led2 = yellow_value;
102             state2 = AUTO_YELLOW;
103             strcpy(str2, "YELLOW:");
104         }
105         break;
106
107     default:
108         break;
109 }
110 }
111
112 static int get7SEG1() {
113     switch (mode) {
114         case 1:
115             return led1;
116         case 2:
117             return red_temp;
118         case 3:
119             return green_temp;
120         case 4:
121             return yellow_temp;
122         default:
123             break;
124     }
125     return 0;
126 }
127
128 static int get7SEG2() {
129     switch (mode) {
130         case 1:
131             return led2;

```

```

132     case 2:
133         return red_temp;
134     case 3:
135         return green_temp;
136     case 4:
137         return yellow_temp;
138     default:
139         break;
140 }
141 return 0;
142 }
143
144 char dataframe1[20];
145 char dataframe2[20];
146 char dataframe3[50];
147 char dataframe4[50];
148 char dataframe5[50];
149 char dataframe6[50];
150
151 void displayMode() {
152     switch (mode) {
153         case 1:
154             if (get0flag()) {
155                 lcd_clear_display();
156                 sprintf(dataframe1, "ROAD 1 %s%d", str1, get7SEG1());
157                 lcd_goto_XY(1, 0);
158                 lcd_send_string(dataframe1);
159
160                 sprintf(dataframe2, "ROAD 2 %s%d", str2, get7SEG2());
161                 lcd_goto_XY(0, 0);
162                 lcd_send_string(dataframe2);
163
164                 setTimer0(100);
165                 mode1Counter();
166             }
167             break;
168
169         case 2:
170             if (get3flag()) {
171                 lcd_clear_display();
172                 sprintf(dataframe1, "Red Mode: %d", red_temp);
173                 lcd_goto_XY(1, 0);
174                 lcd_send_string(dataframe1);
175
176                 sprintf(dataframe2, "Mode: %d", mode);
177                 lcd_goto_XY(0, 0);
178                 lcd_send_string(dataframe2);
179
180                 setTimer3(50);
181                 HAL_GPIO_TogglePin(D4_GPIO_Port, D4_Pin);
182                 HAL_GPIO_TogglePin(D3_GPIO_Port, D3_Pin);
183                 HAL_GPIO_TogglePin(D5_GPIO_Port, D5_Pin);
184                 HAL_GPIO_TogglePin(D6_GPIO_Port, D6_Pin);
185             }
186             break;
187

```

```

188     case 3:
189         if (get3flag()) {
190             lcd_clear_display();
191             sprintf(dataframe1, "Green Mode: %d", green_temp);
192             lcd_goto_XY(1, 0);
193             lcd_send_string(dataframe1);
194
195             sprintf(dataframe2, "Mode: %d", mode);
196             lcd_goto_XY(0, 0);
197             lcd_send_string(dataframe2);
198
199             setTimer3(50);
200             HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_RESET);
201             HAL_GPIO_TogglePin(D3_GPIO_Port, D3_Pin);
202             HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin, GPIO_PIN_RESET);
203             HAL_GPIO_TogglePin(D5_GPIO_Port, D5_Pin);
204         }
205         break;
206
207     case 4:
208         if (get3flag()) {
209             lcd_clear_display();
210             sprintf(dataframe1, "Yellow Mode: %d", yellow_temp);
211             lcd_goto_XY(1, 0);
212             lcd_send_string(dataframe1);
213
214             sprintf(dataframe2, "Mode: %d", mode);
215             lcd_goto_XY(0, 0);
216             lcd_send_string(dataframe2);
217
218             setTimer3(50);
219             HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_RESET);
220             HAL_GPIO_TogglePin(D4_GPIO_Port, D4_Pin);
221             HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin, GPIO_PIN_RESET);
222             HAL_GPIO_TogglePin(D6_GPIO_Port, D6_Pin);
223         }
224         break;
225
226     default:
227         break;
228 }
229 }

```

## 3.2 FSM

```
1  /*
2  * FSM.h
3  *
4  * Created on: Sep 14, 2024
5  * Author: ADMIN
6  */
7
8  #ifndef INC_FSM_H_
9  #define INC_FSM_H_
10
11 #include "global.h"
12
13 void fsm_run();
14 void fsm2_run();
15 void OtherMode();
16 #endif /* INC_FSM_H_ */
```

## 3.3 Global

### 3.3.1 Global.h

```
1  /*
2  * global.h
3  *
4  * Created on: Sep 14, 2024
5  * Author: ADMIN
6  */
7
8  #ifndef INC_GLOBAL_H_
9  #define INC_GLOBAL_H_
10 #include "timer.h"
11 #include "main.h"
12 #include "7SEG.h"
13 #include "input_reading.h"
14
15
16
17
18 #define INIT 1
19 #define AUTO_RED 2
20 #define AUTO_GREEN 3
21 #define AUTO_YELLOW 4
22
23 #define INIT1 1
24 #define AUTO_RED1 2
25 #define AUTO_GREEN1 3
26 #define AUTO_YELLOW1 4
27
28
29 #define RED_TIME 5
30 #define YELLOW_TIME 2
31 #define GREEN_TIME 3
32 extern int status;
```

```

33 extern int status1;
34 void LedInit();
35 extern int mode;
36 extern int red_value, yellow_value, green_value;
37 extern int red_temp, yellow_temp, green_temp;
38 #endif /* INC_GLOBAL_H_ */

```

### 3.3.2 Global.c

```

1  /*
2   * global.c
3   *
4   * Created on: Sep 14, 2024
5   * Author: ADMIN
6   */
7
8  #include "global.h"
9
10 int mode;
11 int red_value, yellow_value, green_value;
12 int red_temp, yellow_temp, green_temp;
13 int status = 0;
14 int status1 = 0;
15 void LedInit() {
16     red_value = RED_TIME;
17     yellow_value = YELLOW_TIME;
18     green_value = GREEN_TIME;
19     red_temp = red_value;
20     yellow_temp = yellow_value;
21     green_temp = green_value;
22 }
23

```

## 3.4 I2C-LCD

### 3.4.1 I2C-LCD.h

```

1  /*
2   * i2c-lcd.h
3   *
4   * Created on: Oct 31, 2024
5   * Author: ADMIN
6   */
7
8  #ifndef INC_I2C_LCD_H_
9  #define INC_I2C_LCD_H_
10
11 /**
12  Edit by modify: Ngoc Hang
13  */
14
15 #include "stm32f1xx_hal.h"
16
17 void lcd_init (void);    // initialize lcd

```

```

18 void lcd_send_cmd (char cmd); // send command to the lcd
19
20 void lcd_send_data (char data); // send data to the lcd
21
22 void lcd_send_string (char *str); // send string to the lcd
23
24 void lcd_clear_display (void); //clear display lcd
25
26 void lcd_goto_XY (int row, int col); //set proper location on screen
27
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
24 void lcd_send_data (char data)
25 {
26     char data_u, data_l;
27     uint8_t data_t[4];
28     data_u = (data&0xf0);
29     data_l = ((data<<4)&0xf0);
30     data_t[0] = data_u|0x0D; //en=1, rs=0
31     data_t[1] = data_u|0x09; //en=0, rs=0
32     data_t[2] = data_l|0x0D; //en=1, rs=0
33     data_t[3] = data_l|0x09; //en=0, rs=0
34     HAL_I2C_Master_Transmit (&hi2c1, SLAVE_ADDRESS_LCD,(uint8_t *) data_t, 4, 100)
35     ;
36 }
37
38 void lcd_init (void) {

```

```

37     lcd_send_cmd (0x33); /* set 4-bits interface */
38     lcd_send_cmd (0x32);
39     HAL_Delay(50);
40     lcd_send_cmd (0x28); /* start to set LCD function */
41     HAL_Delay(50);
42     lcd_send_cmd (0x01); /* clear display */
43     HAL_Delay(50);
44     lcd_send_cmd (0x06); /* set entry mode */
45     HAL_Delay(50);
46     lcd_send_cmd (0x0c); /* set display to on */
47     HAL_Delay(50);
48     lcd_send_cmd (0x02); /* move cursor to home and set data address to 0 */
49     HAL_Delay(50);
50     lcd_send_cmd (0x80);
51 }
52
53 void lcd_send_string (char *str)
54 {
55     while (*str) lcd_send_data (*str++);
56 }
57
58 void lcd_clear_display (void)
59 {
60
61     lcd_goto_XY(0, 0);
62     lcd_send_string("                ");
63
64
65     lcd_goto_XY(1, 0);
66     lcd_send_string("                ");
67
68
69     lcd_goto_XY(0, 0);
70 }
71
72 void lcd_goto_XY (int row, int col)
73 {
74     uint8_t pos_Addr;
75     if(row == 1)
76     {
77         pos_Addr = 0x80 + row - 1 + col;
78     }
79     else
80     {
81         pos_Addr = 0x80 | (0x40 + col);
82     }
83     lcd_send_cmd(pos_Addr);
84 }

```

## 3.5 Input-processing

### 3.5.1 Input-processing.h

```

1  /*
2  * input_process.h

```



```

3  *
4  *   Created on: Sep 18, 2024
5  *       Author: ADMIN
6  */
7
8  #ifndef INC_INPUT_PROCESS_H_
9  #define INC_INPUT_PROCESS_H_
10 #include "main.h"
11 #include "input_reading.h"
12 #include "7SEG.h"
13 #include "timer.h"
14 #include "global.h"
15
16 void ModeBuffer();
17 void DurationInit();
18 void Update();
19
20 #endif /* INC_INPUT_PROCESS_H_ */

```

### 3.5.2 Input-processing.c

```

1  /*
2  *   input_processing.c
3  *
4  *   Created on: Jul 25, 2023
5  *       Author: BAO LONG
6  */
7
8  #include "input_process.h"
9
10 enum ButtonState{BUTTON_RELEASED, BUTTON_PRESSED,
11                 BUTTON_PRESSED_MORE_THAN_1_SECOND};
12
13 enum ButtonState buttonState[3] = {BUTTON_RELEASED};
14
15 void ModeBuffer() {
16     switch (buttonState[0]) {
17         case BUTTON_RELEASED:
18             if (is_button_pressed(0)) {
19                 buttonState[0] = BUTTON_PRESSED;
20                 mode++;
21                 if (mode > 4) {
22                     mode = 1;
23                     resetCountValue();
24                 }
25             }
26             break;
27         case BUTTON_PRESSED:
28             if (!is_button_pressed(0)) {
29                 buttonState[0] = BUTTON_RELEASED;
30             }
31             break;
32         default:
33             break;
34     }
35 }

```

```

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             }

```

```

90         break;
91     case BUTTON_PRESSED:
92         if (!is_button_pressed(1)) {
93             buttonState[1] = BUTTON_RELEASED;
94         }
95         if (is_button_pressed_1s(1)) {
96             buttonState[1] = BUTTON_PRESSED_MORE_THAN_1_SECOND;
97             increaseDraftValue();
98         }
99         break;
100     case BUTTON_PRESSED_MORE_THAN_1_SECOND:
101         if (!is_button_pressed(1)) {
102             buttonState[1] = BUTTON_RELEASED;
103         }
104         if (is_button_held(1)) {
105             reset_flagForButtonHold(1);
106             increaseDraftValue();
107         }
108         break;
109     default:
110         break;
111 }
112
113 switch (buttonState[2]) {
114     case BUTTON_RELEASED:
115         if (is_button_pressed(2)) {
116             buttonState[2] = BUTTON_PRESSED;
117             setDurationValue();
118             mode=1;
119         }
120         break;
121     case BUTTON_PRESSED:
122         if (!is_button_pressed(2)) {
123             buttonState[2] = BUTTON_RELEASED;
124         }
125         break;
126     default:
127         break;
128 }
129 }

```

## 3.6 Input-reading

### 3.6.1 Input-reading.h

```

1  /*
2  * input_reading.h
3  *
4  * Created on: Sep 13, 2024
5  * Author: ADMIN
6  */
7
8  #ifndef INC_INPUT_READING_H_
9  #define INC_INPUT_READING_H_
10 #include "main.h"

```

```

11 typedef struct {
12     GPIO_TypeDef *pGPIOx;
13     uint16_t pin;
14 } BUTTON_CONTROL;
15 extern BUTTON_CONTROL Button[];
16 void init_button();
17 void button_reading(void);
18 unsigned char is_button_pressed(unsigned char index);
19 unsigned char is_button_pressed_1s(unsigned char index);
20 unsigned char is_button_held(unsigned char index);
21 void reset_flagForButtonHold(unsigned char index);
22 #endif /* INC_INPUT_READING_H_ */

```

### 3.6.2 Input-reading.c

```

1  /*
2   * input_reading.c
3   *
4   * Created on: Sep 13, 2024
5   * Author: ADMIN
6   */
7
8  #include "input_reading.h"
9
10 #define NO_OF_BUTTONS 3
11
12 BUTTON_CONTROL Button[] = {
13     { BUTTON1_GPIO_Port, BUTTON1_Pin },
14     { BUTTON2_GPIO_Port, BUTTON2_Pin },
15     { BUTTON3_GPIO_Port, BUTTON3_Pin }
16 };
17
18 // Timer interrupt duration is 10ms, so to pass 1 second,
19 // we need to jump to the interrupt service routine 100 times
20 #define DURATION_FOR_AUTO_INCREASING 100
21 #define DURATION_FOR_HOLD 50
22 #define BUTTON_IS_PRESSED GPIO_PIN_RESET
23 #define BUTTON_IS_RELEASED GPIO_PIN_SET
24
25 // Buffer to store final result after debouncing
26 static GPIO_PinState buttonBuffer[NO_OF_BUTTONS];
27
28 // Buffers for debouncing
29 static GPIO_PinState debounceButtonBuffer1[NO_OF_BUTTONS];
30 static GPIO_PinState debounceButtonBuffer2[NO_OF_BUTTONS];
31 static GPIO_PinState debounceButtonBuffer3[NO_OF_BUTTONS];
32
33 // Flags and counters for button actions
34 static uint8_t flagForButtonPress1s[NO_OF_BUTTONS];
35 static uint8_t flagForButtonHold[NO_OF_BUTTONS];
36 static uint16_t counterForButtonPress1s[NO_OF_BUTTONS];
37 static uint16_t counterForButtonHold[NO_OF_BUTTONS];
38
39 // Initialize button states
40 void init_button() {

```

```

41     for (int i = 0; i < NO_OF_BUTTONS; i++) {
42         buttonBuffer[i] = BUTTON_IS_RELEASED;
43         debounceButtonBuffer1[i] = BUTTON_IS_RELEASED;
44         debounceButtonBuffer2[i] = BUTTON_IS_RELEASED;
45         debounceButtonBuffer3[i] = BUTTON_IS_RELEASED;
46         flagForButtonPress1s[i] = 0;
47         flagForButtonHold[i] = 0;
48         counterForButtonPress1s[i] = 0;
49         counterForButtonHold[i] = 0;
50     }
51 }
52
53 // Read button states with debouncing
54 void button_reading(void) {
55     for (char i = 0; i < NO_OF_BUTTONS; i++) {
56         debounceButtonBuffer3[i] = debounceButtonBuffer2[i];
57         debounceButtonBuffer2[i] = debounceButtonBuffer1[i];
58         debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(Button[i].pGPIOx, Button[i].
59             pin);
60
61         if (debounceButtonBuffer1[i] == debounceButtonBuffer2[i] &&
62             debounceButtonBuffer3[i] == debounceButtonBuffer2[i]) {
63             buttonBuffer[i] = debounceButtonBuffer1[i];
64         }
65
66         if (buttonBuffer[i] == BUTTON_IS_PRESSED) {
67             // Reset all GPIO pins when button is pressed
68             HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_RESET);
69             HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_RESET);
70             HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin, GPIO_PIN_RESET);
71             HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin, GPIO_PIN_RESET);
72
73             // Increment counter for button press duration
74             if (counterForButtonPress1s[i] < DURATION_FOR_AUTO_INCREASING) {
75                 counterForButtonPress1s[i]++;
76             } else {
77                 // Set flag when button has been pressed for 1 second
78                 flagForButtonPress1s[i] = 1;
79
80                 // Handle button hold
81                 if (counterForButtonHold[i] < DURATION_FOR_HOLD) {
82                     counterForButtonHold[i]++;
83                     if (counterForButtonHold[i] >= DURATION_FOR_HOLD) {
84                         counterForButtonHold[i] = 0;
85                         flagForButtonHold[i] = 1;
86                     }
87                 }
88             } else {
89                 // Reset counters and flags if button is released
90                 counterForButtonPress1s[i] = 0;
91                 flagForButtonPress1s[i] = 0;
92                 counterForButtonHold[i] = 0;
93                 flagForButtonHold[i] = 0;
94             }
95     }

```

```

96 }
97
98 // Check if button is pressed
99 unsigned char is_button_pressed(unsigned char index) {
100     if (index >= NO_OF_BUTTONS) return 0;
101     return (buttonBuffer[index] == BUTTON_IS_PRESSED);
102 }
103
104 // Check if button is pressed for 1 second
105 unsigned char is_button_pressed_1s(unsigned char index) {
106     if (index >= NO_OF_BUTTONS) return 0;
107     return (flagForButtonPress1s[index] == 1);
108 }
109
110 // Check if button is held
111 unsigned char is_button_held(unsigned char index) {
112     if (index >= NO_OF_BUTTONS) return 0;
113     return (flagForButtonHold[index] == 1);
114 }
115
116 // Reset flag for button hold
117 void reset_flagForButtonHold(unsigned char index) {
118     if (index < NO_OF_BUTTONS) {
119         flagForButtonHold[index] = 0;
120     }
121 }

```

## 3.7 Timer

### 3.7.1 Timer.h

```

1  #ifndef INC_TIMER_H_
2  #define INC_TIMER_H_
3
4  extern int timer0_counter;
5  extern int timer0_flag;
6  extern int TIMER_CYCLE;
7  extern int timer1_counter;
8  extern int timer1_flag;
9  extern int timer2_counter;
10 extern int timer2_flag;
11 extern int timer3_counter;
12 extern int timer3_flag;
13
14 void setTimer0(int duration);
15 void setTimer1(int duration);
16 void setTimer2(int duration);
17 void setTimer3(int duration);
18 void timer_run();
19 int get0flag();
20 int get1flag();
21 int get2flag();
22 int get3flag();
23
24 #endif /* INC_TIMER_H_ */

```

### 3.7.2 Timer.c

```
1  /*
2      * timer.c
3      *
4      * Created on: Sep 13, 2024
5      * Author: ADMIN
6      */
7
8  /*
9      * timer.c
10     *
11     * Created on: Sep 7, 2024
12     * Author: ADMIN
13     */
14
15 #include "timer.h"
16 #include "7SEG.h"
17 #define CASE0 0
18 #define CASE1 1
19 #define CASE2 2
20 #define CASE3 3
21 int timer0_counter = 0;
22 int timer0_flag = 0;
23 int TIMER_CYCLE = 1;
24 int timer1_counter = 0;
25 int timer1_flag = 0;
26 int timer2_counter = 0;
27 int timer2_flag = 0;
28 int timer3_counter = 0;
29 int timer3_flag = 0;
30 void setTimer0(int duration) {
31     timer0_counter = duration / TIMER_CYCLE;
32     timer0_flag = 0;
33 }
34
35 void setTimer1(int duration) {
36     timer1_counter = duration / TIMER_CYCLE;
37     timer1_flag = 0;
38 }
39
40 void setTimer2(int duration) {
41     timer2_counter = duration / TIMER_CYCLE;
42     timer2_flag = 0;
43 }
44
45 void setTimer3(int duration) {
46     timer3_counter = duration / TIMER_CYCLE;
47     timer3_flag = 0;
48 }
49
50 void timer_run() {
51     if (timer0_counter > 0) {
52         timer0_counter--;
53         if (timer0_counter == 0) timer0_flag = 1;
54     }
```

```

55     }
56     if (timer1_counter > 0) {
57         timer1_counter--;
58         if (timer1_counter == 0) timer1_flag = 1;
59     }
60
61     if (timer2_counter > 0) {
62         timer2_counter--;
63         if (timer2_counter == 0) timer2_flag = 1;
64     }
65
66     if(timer3_counter>0){
67         timer3_counter--;
68         if(timer3_counter==0) timer3_flag =1;
69     }
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 -----*/

```



```

22 #ifndef __MAIN_H
23 #define __MAIN_H
24
25 #ifdef __cplusplus
26 extern "C" {
27 #endif
28
29 /* Includes -----*/
30 #include "stm32f1xx_hal.h"
31
32 /* Private includes -----*/
33 /* USER CODE BEGIN Includes */
34
35 /* USER CODE END Includes */
36
37 /* Exported types -----*/
38 /* USER CODE BEGIN ET */
39
40 /* USER CODE END ET */
41
42 /* Exported constants -----*/
43 /* USER CODE BEGIN EC */
44
45 /* USER CODE END EC */
46
47 /* Exported macro -----*/
48 /* USER CODE BEGIN EM */
49
50 /* USER CODE END EM */
51
52 /* Exported functions prototypes -----*/
53 void Error_Handler(void);
54
55 /* USER CODE BEGIN EFP */
56
57 /* USER CODE END EFP */
58
59 /* Private defines -----*/
60 #define B1_Pin GPIO_PIN_13
61 #define B1_GPIO_Port GPIOC
62 #define B1_EXTI_IRQn EXTI15_10_IRQn
63 #define LED1_Pin GPIO_PIN_5
64 #define LED1_GPIO_Port GPIOA
65 #define D6_Pin GPIO_PIN_10
66 #define D6_GPIO_Port GPIOB
67 #define BUTTON3_Pin GPIO_PIN_7
68 #define BUTTON3_GPIO_Port GPIOC
69 #define LED2_Pin GPIO_PIN_9
70 #define LED2_GPIO_Port GPIOC
71 #define BUTTON1_Pin GPIO_PIN_8
72 #define BUTTON1_GPIO_Port GPIOA
73 #define BUTTON2_Pin GPIO_PIN_9
74 #define BUTTON2_GPIO_Port GPIOA
75 #define TMS_Pin GPIO_PIN_13
76 #define TMS_GPIO_Port GPIOA
77 #define TCK_Pin GPIO_PIN_14

```

```

78 #define TCK_GPIO_Port GPIOA
79 #define D3_Pin GPIO_PIN_3
80 #define D3_GPIO_Port GPIOB
81 #define D5_Pin GPIO_PIN_4
82 #define D5_GPIO_Port GPIOB
83 #define D4_Pin GPIO_PIN_5
84 #define D4_GPIO_Port GPIOB
85
86 /* USER CODE BEGIN Private defines */
87
88 /* USER CODE END Private defines */
89
90 #ifdef __cplusplus
91 }
92 #endif
93
94 #endif /* __MAIN_H */

```

### 3.8.2 Main.c

```

1  /* USER CODE BEGIN Header */
2  /**
3   *
4   * @file          : main.c
5   * @brief         : Main program body
6   *
7   *
8   *
9   * Copyright (c) 2024 STMicroelectronics.
10  * All rights reserved.
11  *
12  * This software is licensed under terms that can be found in the LICENSE file
13  * in the root directory of this software component.
14  * If no LICENSE file comes with this software, it is provided AS-IS.
15  *
16  *
17  */
18  /* USER CODE END Header */
19  /* Includes -----*/
20  #include "main.h"
21
22  /* Private includes -----*/
23  /* USER CODE BEGIN Includes */
24  #include "timer.h"
25  #include "input_reading.h"
26  #include "global.h"
27  #include "7SEG.h"
28  #include "input_process.h"
29  #include "FSM.h"
30  #include "i2c-lcd.h"
31  #include "string.h"
32  /* USER CODE END Includes */
33
34  /* Private typedef -----*/
35  /* 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.HSISState = 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_SYSCCLKSOURCE_PLLCLK;
166  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCCLK_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     }

```

```

204  /* USER CODE BEGIN I2C1_Init 2 */
205
206  /* USER CODE END I2C1_Init 2 */
207
208  }
209
210  /**
211   * @brief TIM2 Initialization Function
212   * @param None
213   * @retval None
214   */
215  static void MX_TIM2_Init(void)
216  {
217
218      /* USER CODE BEGIN TIM2_Init 0 */
219
220      /* USER CODE END TIM2_Init 0 */
221
222      TIM_ClockConfigTypeDef sClockSourceConfig = {0};
223      TIM_MasterConfigTypeDef sMasterConfig = {0};
224
225      /* USER CODE BEGIN TIM2_Init 1 */
226
227      /* USER CODE END TIM2_Init 1 */
228      htim2.Instance = TIM2;
229      htim2.Init.Prescaler = 7999;
230      htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
231      htim2.Init.Period = 9;
232      htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
233      htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
234      if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
235      {
236          Error_Handler();
237      }
238      sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
239      if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
240      {
241          Error_Handler();
242      }
243      sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
244      sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
245      if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
246      {
247          Error_Handler();
248      }
249      /* USER CODE BEGIN TIM2_Init 2 */
250
251      /* USER CODE END TIM2_Init 2 */
252
253  }
254
255  /**
256   * @brief GPIO Initialization Function
257   * @param None
258   * @retval None
259   */

```

```

260 static void MX_GPIO_Init(void)
261 {
262     GPIO_InitTypeDef GPIO_InitStruct = {0};
263
264     /* GPIO Ports Clock Enable */
265     __HAL_RCC_GPIOC_CLK_ENABLE();
266     __HAL_RCC_GPIOD_CLK_ENABLE();
267     __HAL_RCC_GPIOA_CLK_ENABLE();
268     __HAL_RCC_GPIOB_CLK_ENABLE();
269
270     /*Configure GPIO pin Output Level */
271     HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
272
273     /*Configure GPIO pin Output Level */
274     HAL_GPIO_WritePin(GPIOB, D6_Pin|D3_Pin|D5_Pin|D4_Pin, GPIO_PIN_RESET);
275
276     /*Configure GPIO pin Output Level */
277     HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
278
279     /*Configure GPIO pin : B1_Pin */
280     GPIO_InitStruct.Pin = B1_Pin;
281     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
282     GPIO_InitStruct.Pull = GPIO_NOPULL;
283     HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
284
285     /*Configure GPIO pin : PA2 */
286     GPIO_InitStruct.Pin = GPIO_PIN_2;
287     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
288     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
289     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
290
291     /*Configure GPIO pin : PA3 */
292     GPIO_InitStruct.Pin = GPIO_PIN_3;
293     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
294     GPIO_InitStruct.Pull = GPIO_NOPULL;
295     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
296
297     /*Configure GPIO pin : LED1_Pin */
298     GPIO_InitStruct.Pin = LED1_Pin;
299     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
300     GPIO_InitStruct.Pull = GPIO_NOPULL;
301     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
302     HAL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
303
304     /*Configure GPIO pins : D6_Pin D3_Pin D5_Pin D4_Pin */
305     GPIO_InitStruct.Pin = D6_Pin|D3_Pin|D5_Pin|D4_Pin;
306     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
307     GPIO_InitStruct.Pull = GPIO_NOPULL;
308     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
309     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
310
311     /*Configure GPIO pin : BUTTON3_Pin */
312     GPIO_InitStruct.Pin = BUTTON3_Pin;
313     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
314     GPIO_InitStruct.Pull = GPIO_NOPULL;
315     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  #ifndef 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 include Demo Clip (Name: SESSION1):** <https://github.com/loxx1/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.