



MCU
REPORT

2023

(CO3009) Microprocessor – Microcontroller
Final Project

Submitted to: Mr. LÊ TRỌNG NHÂN

Tô Hoàng Phong - 2112012
Trần Tiến Đạt - 2113162
Trần Anh Tài - 2114700

Ho Chi Minh University of Technology (HCMUT)
Computer science and engineering faculty

Mục lục

1	Github repository URL and hardware implementation video	3
2	Finite state machine and behaviors of each state	3
2.1	System Finite State Machine	3
2.1.1	Finite State Machine	3
2.1.2	Behaviors	4
2.2	Button Finite State Machine	4
2.2.1	Finite State Machine	4
2.2.2	Behaviors	4
2.3	Buzzer Finite State Machine	5
2.3.1	Finite State Machine	6
2.3.2	Behaviors	6
3	Schematic and mapping table	6
4	File organization	7
5	Some important structs and functions	7
5.1	Pin Assignment	7
5.2	Scheduler	8
5.2.1	Update $O(1)$:	8
5.2.2	Add Task $O(n)$:	8
5.2.3	Remove Task:	9
5.2.4	Dispatch $O(1)$:	9
5.3	Initial Function	9
5.4	Running Scheduler Function	9
5.5	LED	9
5.5.1	LED Structure	9
5.5.2	LED Initialization	10
5.5.3	LED Control	10
5.6	Button	10
5.6.1	Button define	10
5.7	Button Structure	10
5.7.1	Button Initialization	11
5.7.2	Button Control	11
5.8	Finite state machine	12
5.8.1	All FSM Encode Number	12
5.8.2	FSM's variables Initialiaztion	12
5.8.3	System FSM	12
5.8.4	TrafficLight FSM	15
5.8.5	ManuallyTrafficLight FSM	16
5.8.6	LED FSM	16
5.8.7	Button FSM	17
5.8.8	Buzzer FSM	19
5.9	Main Loop and Set Up before Main Loop	19

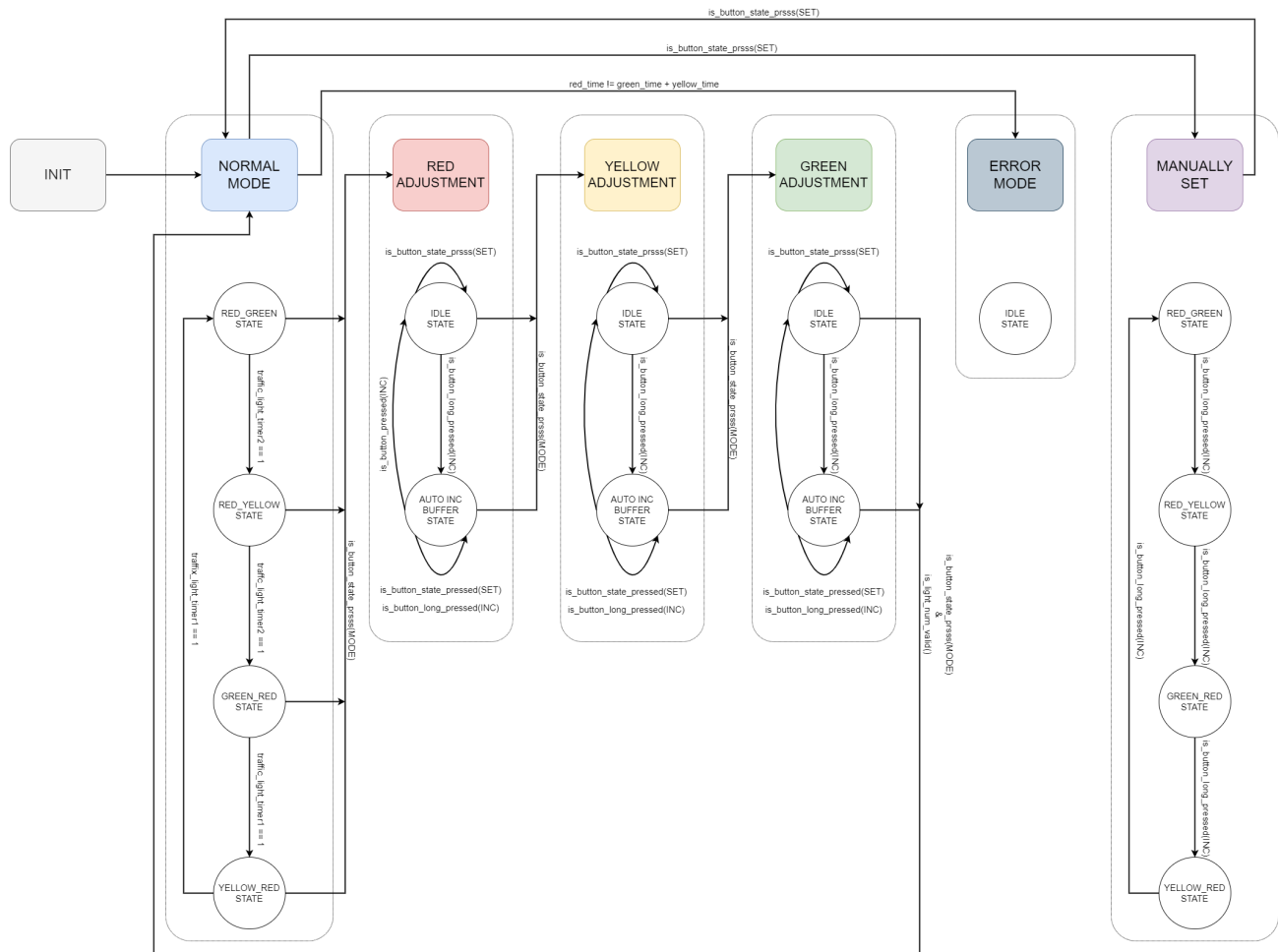
1 Github repository URL and hardware implementation video

- Github repository URL of the lab 3: https://github.com/kido2k3/TRAFFIC_LIGHT_PROJECT
- Hardware implementation video of the lab 3:

2 Finite state machine and behaviors of each state

2.1 System Finite State Machine

2.1.1 Finite State Machine



Hình 1: System Finite State Machine

2.1.2 Behaviors

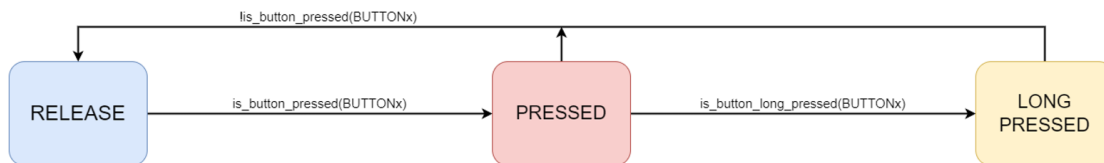
Short brief: System Finite State Machine has 5 states (TRAFFIC_LIGHT, RED_ADJUSTMENT, YELLOW_ADJUSTMENT, GREEN_ADJUSTMENT, ERROR_). Each state has some behaviors (see figure 2).

Name	Main functions
TRAFFIC_LIGHT	Display via UART Decrease traffic light timer every 1s light_pre_st = TRAFFIC_LIGHT Run Pedestrian finite state machine
RED_ADJUSTMENT	Blink red led in 2Hz Display red buffer via UART light_pre_st = RED_ADJUSTMENT
YELLOW_ADJUSTMENT	Blink yellow led in 2Hz Display yellow buffer via UART light_pre_st = YELLOW_ADJUSTMENT
GREEN_ADJUSTMENT	Blink green led in 2Hz Display green buffer via UART light_pre_st = GREEN_ADJUSTMENT
MANUALLY SET	Display current light
ERROR	Turn off all LEDs

Hình 2: Behavior of each state

2.2 Button Finite State Machine

2.2.1 Finite State Machine



Hình 3: State machine of each button

2.2.2 Behaviors

This project has 4 buttons: MODE button; SET Value; INC button; PEDES button. MODE, SET, PEDES buttons do not have LONG_PRESSED state in their fsm.

a. Button 0 (MODE)

The MODE button is used to transmit the current mode of traffic light

Name	Main functions
RELEASE	Do nothing
PRESSED	Change light state (light_st) Set up environment of next state

Hình 4: Behavior of each state

b. Button 1 (SET)

The SET button is used to set value to LED's buffer

Name	Main functions
RELEASE	Do nothing
PRESSED	Change light state (in TRAFFIC_LIGHT or MANUAL_SET state) Set the temporary buffer to current LED's buffer (in ADJUSTMENT state)

Hình 5: Behavior of each state

c. Button 2 (INC)

The INC button is used to increase the value of the LED's buffer.

Name	Main functions
RELEASE	Do nothing
PRESSED	Increase the current buffer (1 unit) (in ADJUSTMENT state)
LONG PRESSED	Increase the current buffer (every 0.25s) (in ADJUSTMENT state)

Hình 6: Behavior of each state

d. Button 3 (PEDES)

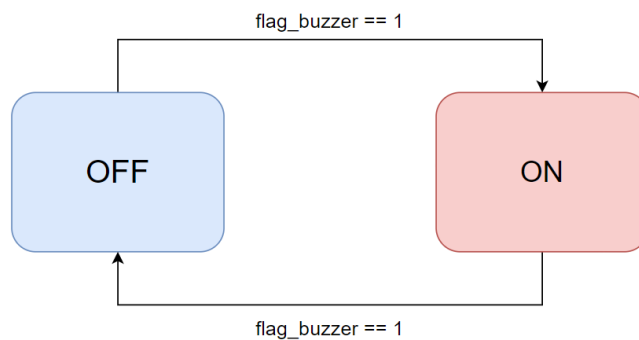
The PEDES button is used for walker (pedestrian light)

Name	Main functions
RELEASE	Do nothing
PRESSED	Run pedestrian countdown (in TRAFFIC_LIGHT or MANUALLY_SET state)

Hình 7: Behavior of each states

2.3 Buzzer Finite State Machine

2.3.1 Finite State Machine



Hình 8: Behavior of each state

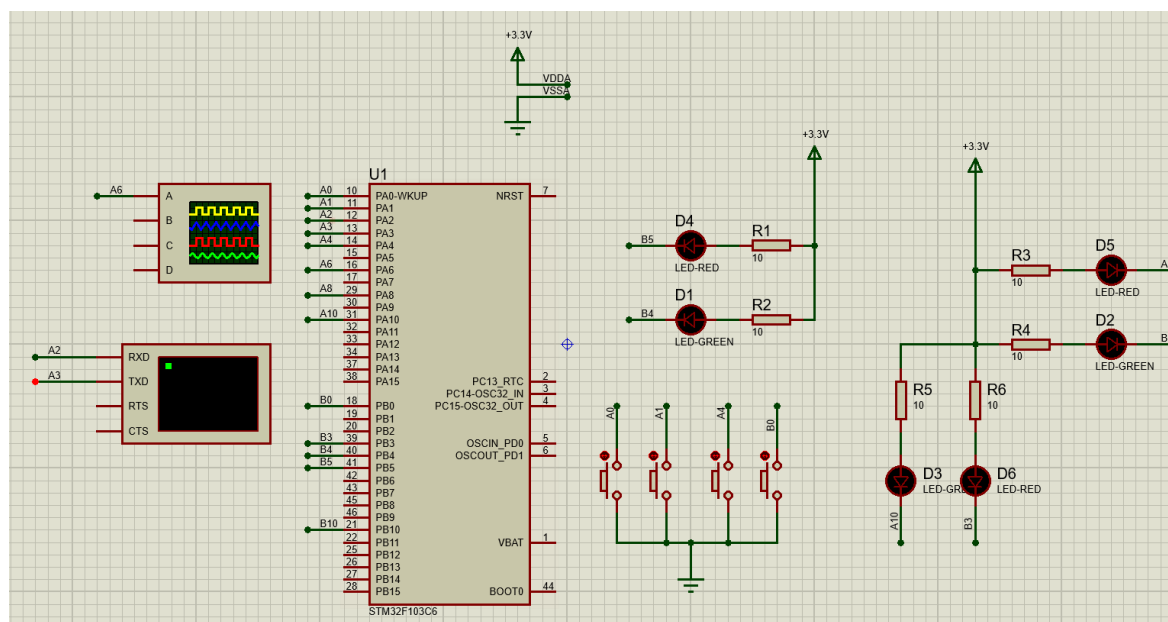
2.3.2 Behaviors

The buzzer's intensity is an attribute of the buzzer structure. Therefore, FSM will not execute it.

Name	Main functions
OFF	Set up environment of ON state
ON	Generate PWM depending on light countdown

Hình 9: Behavior of each state

3 Schematic and mapping table



Hình 10: Schematic

Number	Components	Pin
LED		
1	LED RED 1	PB4
2	LED GREEN 1	PB10
3	LED RED 2	PB3
4	LED GREEN 2	PB5
5	LED PEDES 1	PC7
6	LED PEDES 2	PA8
BUTTON		
7	MODE BUTTON	PA1
8	SET BUTTON	PA4
9	INC BUTTON	PB0
10	PEDES BUTTON	PA0
VOICE		
11	BUZZER	PA6
DISPLAY INFORMATION		
12	TERMINAL_RX	PA2
13	TERMINAL_TX	PA3

Hình 11: Pin Mapping Table

4 File organization

- my_definet.h: Define all ports used in the system
- my_button.h - my_button.c: Define the number of buttons, the structure of buttons, some button functions, hardware button-reading functions
- my_buzzer.h - my_buzzer.c: Define the number of buzzers, the structure of buzzers, some buzzer functions, API to generate PWN of buzzers.
- my_led.h - my_led.c: Control red, green and yellow light in traffic light
- my_system.h - my_system.c: Define the Interrupt Service Routine of timer, start timer function, and initialize hardware components.
- my_fsm.h - my_fsm.c: Define all fsm functions, their state variables.
- my_scheduler.h - my_scheduler.c: Define all scheduler APIs.

5 Some important structs and functions

5.1 Pin Assignment

```

1 // All buttons are pull-up
2 #define BUTTON1_PORT GPIOA
3 #define BUTTON1 GPIO_PIN_1
4 #define BUTTON2_PORT GPIOA
5 #define BUTTON2 GPIO_PIN_4
6 #define BUTTON3_PORT GPIOB
7 #define BUTTON3 GPIO_PIN_0
8 // define pedestrian button - pull up
9 #define BUTTON4_PORT GPIOA
10 #define BUTTON4 GPIO_PIN_0
11 // define output
12
13 #define TL_GREEN_PORT1 GPIOB // Traffic light 1st
14 #define TL_GREEN1 GPIO_PIN_10
15 #define TL_RED_PORT1 GPIOB // Traffic light 1st
16 #define TL_RED1 GPIO_PIN_4
17
18 #define TL_PORT2 GPIOB // Traffic light 2nd
19 #define TL_GREEN2 GPIO_PIN_5
20 #define TL_RED2 GPIO_PIN_3
21

```

```

22 #define TL_PED_GREEN_PORT GPIOC // Pedestrian light
23 #define TL_PED_GREEN GPIO_PIN_7
24 #define TL_PED_RED_PORT GPIOA // Pedestrian light
25 #define TL_PED_RED GPIO_PIN_8
26
27 typedef uint8_t bool;

```

5.2 Scheduler

5.2.1 Update O(1):

```

1 void sch_update(void)
2 {
3     if (stack_task.top == 0)
4         return;
5     if (stack_task.top->counter > 0)
6     {
7         stack_task.top->counter--;
8     }
9 }

```

5.2.2 Add Task O(n):

```

1 bool sch_add_task(void (*pTask)(), uint16_t delay, uint16_t period)
2 {
3     struct task *my_task = (struct task *)malloc(sizeof(struct task));
4     my_task->pTask = pTask;
5     my_task->counter = delay * FREQUENCY_OF_TIM/1000;
6     my_task->period = period;
7     my_task->next_task = 0;
8     if (stack_task.top == 0)
9     {
10         stack_task.top = my_task;
11         // stack_task.bottom = stack_task.top;
12         // stack_task.time_length = stack_task.top->counter;
13         return 1;
14     }
15     // if (delay >= stack_task.time_length)
16     // {
17     //     my_task->counter = delay - stack_task.time_length;
18     //     stack_task.bottom->next_task = my_task;
19     //     stack_task.bottom = stack_task.bottom->next_task;
20     //     stack_task.time_length += my_task->counter;
21     //     return 1;
22     // }
23     struct task *pre = stack_task.top;
24     struct task *cur = stack_task.top;
25     while (cur && my_task->counter >= cur->counter)
26     {
27         my_task->counter = my_task->counter - cur->counter;
28         pre = cur;
29         cur = cur->next_task;
30     }
31     if (pre != cur)
32     {
33         pre->next_task = my_task;
34         my_task->next_task = cur;
35     }
36     else
37     {
38         my_task->next_task = cur;
39         stack_task.top = my_task;
40     }
41     if (cur)
42         cur->counter -= my_task->counter;
43     return 1;
44 }

```


5.2.3 Remove Task:

```

1 void sch_delete_task(struct task *del_task)
2 {
3     if (del_task == 0)
4     {
5         return;
6     }
7     free(del_task);
8 }

```

5.2.4 Dispatch O(1):

```

1 bool sch_dispatch(void)
2 {
3     if (stack_task.top == 0)
4         return 0;
5     if (stack_task.top->counter == 0)
6     {
7         (*stack_task.top->pTask)();
8         struct task *temp = stack_task.top;
9         stack_task.top = stack_task.top->next_task;
10        temp->next_task = 0;
11        if (temp->period != 0)
12        {
13            sch_add_task(temp->pTask, temp->period, temp->period);
14        }
15        sch_delete_task(temp);
16        return 1;
17    }
18    return 0;
19 }

```

5.3 Initial Function

```

1 void init(void)
2 {
3     HAL_TIM_Base_Start_IT(&htim2);
4     button_init();
5     init_led();
6     uart_Init();
7     buzzer_init();
8     sch_add_task(button_read, 0, READ_BUTTON_TIME);
9 }

```

5.4 Running Scheduler Function

```

1 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
2 {
3     if (htim->Instance == htim2.Instance)
4     {
5         sch_update();
6     }
7 }

```

```

1 void loop(void)
2 {
3     sch_dispatch();
4     fsm();
5 }

```

5.5 LED

5.5.1 LED Structure

```

1 struct {
2     GPIO_TypeDef *green_port;
3     GPIO_TypeDef *red_port;
4     uint16_t green;
5     uint16_t red;
6 } traffic_light[2], pedestrian_light;

```

5.5.2 LED Initialization

LED Assignment

```

1 void init_led(void) {
2     init_traffic_light();
3 }
4 void init_traffic_light(void) {
5     traffic_light[0].green_port = TL_GREEN_PORT1;
6     traffic_light[0].green = TL_GREEN1;
7     traffic_light[0].red_port = TL_RED_PORT1;
8     traffic_light[0].red = TL_RED1;
9     traffic_light[1].green_port = TL_PORT2;
10    traffic_light[1].red_port = TL_PORT2;
11    traffic_light[1].green = TL_GREEN2;
12    traffic_light[1].red = TL_RED2;
13
14    pedestrian_light.green_port = TL_PED_GREEN_PORT;
15    pedestrian_light.red_port = TL_PED_RED_PORT;
16    pedestrian_light.red = TL_PED_RED;
17    pedestrian_light.green = TL_PED_GREEN;
18 }

```

5.5.3 LED Control

```

1 /*
2  * @brief: display traffic light function
3  * @para: i - id of traffic light
4  *         red, green - state of red and green led (1: on, 0: off)
5  * @retval: none*/
6 void control_traffic_light(unsigned i, GPIO_PinState red, GPIO_PinState green) {
7     HAL_GPIO_WritePin(traffic_light[i].red_port, traffic_light[i].red, red);
8     HAL_GPIO_WritePin(traffic_light[i].green_port, traffic_light[i].green, green);
9 }
10 /*
11  * @brief: display pedestrian light function
12  * @para: red, green - state of red and green led (1: on, 0: off)
13  * @retval: none*/
14 void control_pedestrian_light(GPIO_PinState red, GPIO_PinState green) {
15     HAL_GPIO_WritePin(pedestrian_light.red_port, pedestrian_light.red, red);
16     HAL_GPIO_WritePin(pedestrian_light.green_port, pedestrian_light.green, green);
17 }

```

5.6 Button

5.6.1 Button define

```

1 #define NUMBER_OF_BUTTON 4
2
3 #define RELEASE 1
4 #define PRESSED 0
5 #define LONG_PRESSED_TIME 150 // 1.5s

```

5.7 Button Structure

```

1 struct {
2     bool reg[3];
3     bool is_pressed;
4     bool is_long_pressed;
5     unsigned int timer;
6 }

```

```

7  GPIO_TypeDef *port;
8  uint16_t pin;
9  } button[NUMBER_OF_BUTTON];

```

5.7.1 Button Initialization

Button Assignment

```

1  /*
2  * button[0]: transitioning-mode button
3  * button[1]: increasing-value button
4  * button[2]: setting-value button
5  * button[3]: pedestrian button*/
6  void button_init(void) {
7      for (int i = 0; i < NUMBER_OF_BUTTON; i++) {
8          button[i].reg[0] = button[i].reg[1] = button[i].reg[2] = RELEASE;
9          button[i].is_long_pressed = 0;
10         button[i].is_pressed = 0;
11         button[i].timer = LONG_PRESSED_TIME;
12     }
13     // port and pin were matched by hand
14     button[0].port = BUTTON1_PORT;
15     button[0].pin = BUTTON1;
16     button[1].port = BUTTON2_PORT;
17     button[1].pin = BUTTON2;
18     button[2].port = BUTTON3_PORT;
19     button[2].pin = BUTTON3;
20     button[3].port = BUTTON4_PORT;
21     button[3].pin = BUTTON4;
22 }

```

5.7.2 Button Control

```

1  /*
2  * @brief: read the value of all buttons
3  * @para: none
4  * @retval: none
5  * */
6  void button_read(void) {
7      for (unsigned i = 0; i < NUMBER_OF_BUTTON; i++) {
8          button[i].reg[0] = button[i].reg[1];
9          button[i].reg[1] = button[i].reg[2];
10         button[i].reg[2] = HAL_GPIO_ReadPin(button[i].port, button[i].pin);
11         if (button[i].reg[0] == button[i].reg[1]
12             && button[i].reg[1] == button[i].reg[2]) {
13             //stable state, not bouncing
14             if (button[i].reg[2] == PRESSED) {
15                 button[i].is_pressed = 1;
16                 //decrease counter to toggle is_long_pressed flag
17                 if (button[i].timer > 0) {
18                     button[i].timer--;
19                 } else {
20                     button[i].is_long_pressed = 1;
21                 }
22             } else {
23                 button[i].is_long_pressed = button[i].is_pressed = 0;
24                 button[i].timer = LONG_PRESSED_TIME;
25             }
26         }
27     }
28 }
29 /*
30 * @brief: return the is_pressed flag
31 * @para: i - id of button
32 * @retval: is_pressed (0: released, 1: pressed)
33 * */
34 bool is_button_pressed(unsigned i) {
35     if (i >= NUMBER_OF_BUTTON)
36         return ERROR;
37     return button[i].is_pressed;
38 }
39 /*

```

```

40 * @brief: return the is_long_pressed flag
41 * @para: i - id of button
42 * @retval: is_pressed (1: long-pressed, 0: not)
43 */
44 bool is_button_long_pressed(unsigned i) {
45     if (i >= NUMBER_OF_BUTTON)
46         return ERROR;
47     return button[i].is_long_pressed;
48 }

```

5.8 Finite state machine

5.8.1 All FSM Encode Number

```

1 enum {
2     release, pressed, long_pressed
3 } /*state variable of button*/button_st[4];
4
5 enum {
6     TRAFFIC_LIGHT,
7     RED_ADJUSTMENT,
8     YELLOW_ADJUSTMENT,
9     GREEN_ADJUSTMENT,
10    SET_VALUE,
11    INCREASE_BY_1,
12    INCREASE_BY_1_OVER_TIME,
13    MANUALLY_SET
14 } /*state variable of system*/light_st = TRAFFIC_LIGHT,
15 /* previous state variable of system*/light_pre_st = TRAFFIC_LIGHT;
16 enum {
17     RED_GREEN, RED_YELLOW, GREEN_RED, YELLOW_RED, UNEQUAL
18 } /* state variable of traffic light*/tl_st = RED_GREEN, man_tl_st = RED_GREEN;
19 enum {
20     ON, OFF
21 } /* state variable of single led*/led_st, pedestrian_st = OFF;
22 enum {
23     BUZZER_ON, BUZZER_OFF
24 } /* state variable of single led*/bz_st = BUZZER_OFF;

```

5.8.2 FSM's variables Initialization

```

1 unsigned red_time = RED_TIME_INIT;
2 unsigned green_time = GREEN_TIME_INIT;
3 unsigned yellow_time = YELLOW_TIME_INIT;
4
5 unsigned red_time_buffer = RED_TIME_INIT;
6 unsigned green_time_buffer = GREEN_TIME_INIT;
7 unsigned yellow_time_buffer = YELLOW_TIME_INIT;
8
9 unsigned traffic_light_timer1 = RED_TIME_INIT;
10 unsigned traffic_light_timer2 = GREEN_TIME_INIT;
11
12 unsigned pedestrian_timer = 0; // timer to auto turn off pedestrian light when no one
    press for a while
13
14 bool flag_toggle_led = 1;
15 bool flag_countdown = 1;
16 bool flag_increase_over_time = 1;
17 bool flag_pedestrian_on = 1;
18 bool flag_buzzer = 1;

```

5.8.3 System FSM

```

1 /**
2  * @brief Top-layer finite state machine
3  * @param None
4  * @retval None
5  */
6 void fsm(void) {

```

```

7  switch (light_st) {
8  case TRAFFIC_LIGHT:
9      if (red_time != green_time + yellow_time) {
10         // off all leds
11         control_traffic_light(0, 0, 0);
12         control_traffic_light(1, 0, 0);
13     } else {
14         // decrease timer every 1s
15         if (flag_countdown == 1) {
16             flag_countdown = 0;
17             uart_SendTimeTraffic(0, traffic_light_timer1);
18             uart_SendTimeTraffic(1, traffic_light_timer2);
19             sch_add_task(task_countdown_1sec, ONE_SECOND, 0);
20         }
21         traffic_light_fsm();
22     }
23     if(!pedestrian_timer){
24         buzzer_calculation(6);
25         buzzer_off();
26
27         control_pedestrian_light(0, 0);
28     } else if(pedestrian_timer){
29         if (flag_pedestrian_on) {
30             flag_pedestrian_on = 0;
31             sch_add_task(task_countdown_pedestrian_timer, ONE_SECOND, 0);
32         }
33         switch(tl_st){
34         case RED_GREEN:
35             control_pedestrian_light(0, 1);
36             buzzer_fsm();
37             break;
38         case RED_YELLOW:
39             control_pedestrian_light(0, 1);
40             buzzer_fsm();
41             break;
42         case GREEN_RED:
43             control_pedestrian_light(1, 0);
44             buzzer_calculation(6);
45             buzzer_off();
46             break;
47         case YELLOW_RED:
48             control_pedestrian_light(1, 0);
49             buzzer_calculation(6);
50             buzzer_off();
51             break;
52         default:
53             break;
54         }
55     }
56     // transition to adjustment mode
57     button0_fsm();
58     // transistion to manual setting mode
59     button2_fsm();
60     // button to control pedestrian led
61     button3_fsm();
62     break;
63 case RED_ADJUSTMENT:
64     // update buffer of red with the condition that previous state has to be different
65     // from changing-value states
66     if (light_pre_st != INCREASE_BY_1 && light_pre_st != SET_VALUE
67         && light_pre_st != INCREASE_BY_1_OVER_TIME) {
68         red_time_buffer = red_time;
69     }
70     // update buffer of four 7-seg leds: value of red buffer and the mode (2)
71
72     fsm_led();
73     // transition mode function
74     button0_fsm();
75     button1_fsm();
76     button2_fsm();
77
78     break;
79 case YELLOW_ADJUSTMENT:
80     // update buffer of yellow with the condition that previous state has to be
81     // different from changing-value states

```

```

80     if (light_pre_st != INCREASE_BY_1 && light_pre_st != SET_VALUE
81         && light_pre_st != INCREASE_BY_1_OVER_TIME)
82         yellow_time_buffer = yellow_time;
83     // update buffer of four 7-seg leds: value of yellow buffer and the mode (3)
84
85     fsm_led();
86     // transition mode function
87     button0_fsm();
88     button1_fsm();
89     button2_fsm();
90
91     break;
92 case GREEN_ADJUSTMENT:
93     // update buffer of green with the condition that previous state has to be
94     // different from changing-value states
95     if (light_pre_st != INCREASE_BY_1 && light_pre_st != SET_VALUE
96         && light_pre_st != INCREASE_BY_1_OVER_TIME)
97         green_time_buffer = green_time;
98     // update buffer of four 7-seg leds: value of yellow buffer and the mode (4)
99
100    fsm_led();
101    // transition mode function
102    button0_fsm();
103    button1_fsm();
104    button2_fsm();
105
106    break;
107 case SET_VALUE:
108     // update the time value based-on previous state
109     if (light_pre_st == RED_ADJUSTMENT) {
110         red_time = red_time_buffer;
111     } else if (light_pre_st == YELLOW_ADJUSTMENT) {
112         yellow_time = yellow_time_buffer;
113     } else if (light_pre_st == GREEN_ADJUSTMENT) {
114         green_time = green_time_buffer;
115     }
116
117     light_st = light_pre_st;
118     light_pre_st = SET_VALUE;
119     break;
120 case INCREASE_BY_1:
121     // increase the time value based-on previous state (short-pressed)
122     increase_value();
123     switch (light_pre_st) {
124     case RED_ADJUSTMENT:
125         uart_SendMode(2);
126         uart_SendBufferRed(red_time_buffer);
127         break;
128     case YELLOW_ADJUSTMENT:
129         uart_SendMode(3);
130         uart_SendBufferYellow(yellow_time_buffer);
131         break;
132     case GREEN_ADJUSTMENT:
133         uart_SendMode(4);
134         uart_SendBufferGreen(green_time_buffer);
135         break;
136     default:
137         break;
138     }
139     light_st = light_pre_st;
140     light_pre_st = INCREASE_BY_1;
141     break;
142 case INCREASE_BY_1_OVER_TIME:
143     // increase the time value every 0.25s based-on previous state (short-pressed)
144     if (light_pre_st == RED_ADJUSTMENT) {
145
146     } else if (light_pre_st == YELLOW_ADJUSTMENT) {
147
148     } else if (light_pre_st == GREEN_ADJUSTMENT) {
149
150     }
151     if (flag_increase_over_time == 1) {
152         flag_increase_over_time = 0;
153         sch_add_task(task_increase_over_time, INCREASE_TIME, 0);
154     }

```

```

154     button1_fsm();
155     break;
156 case MANUALLY_SET:
157     manually_traffic_state();
158     if(!pedestrian_timer){
159         control_pedestrian_light(0, 0);
160     } else if(pedestrian_timer){
161         if (flag_pedestrian_on) {
162             flag_pedestrian_on = 0;
163             sch_add_task(task_countdown_pedestrian_timer, ONE_SECOND, 0);
164         }
165         switch(man_tl_st){
166             case RED_GREEN:
167                 control_pedestrian_light(0, 1);
168                 break;
169             case RED_YELLOW:
170                 control_pedestrian_light(0, 1);
171                 break;
172             case GREEN_RED:
173                 control_pedestrian_light(1, 0);
174                 break;
175             case YELLOW_RED:
176                 control_pedestrian_light(1, 0);
177                 break;
178             default:
179                 break;
180         }
181     }
182     //control traffic light
183     button1_fsm();
184     //return to traffic light mode
185     button2_fsm();
186     // button to control pedestrian led
187     button3_fsm();
188     break;
189 }
190 }
191 }

```

5.8.4 TrafficLight FSM

```

1  /*
2  * @brief:  finite state machine to control behavior of traffic light
3  * @para:  none
4  * @retval: none*/
5  void traffic_light_fsm(void) {
6      switch (tl_st) {
7          case RED_GREEN:
8              control_traffic_light(0, 1, 0);
9              control_traffic_light(1, 0, 1);
10             if (traffic_light_timer2 <= 0) {
11                 traffic_light_timer2 = yellow_time;
12                 tl_st = RED_YELLOW;
13             }
14             break;
15          case RED_YELLOW:
16              control_traffic_light(0, 1, 0);
17              control_traffic_light(1, 1, 1);
18              if (traffic_light_timer2 <= 0) {
19                 traffic_light_timer1 = green_time;
20                 traffic_light_timer2 = red_time;
21                 tl_st = GREEN_RED;
22             }
23             break;
24          case GREEN_RED:
25              control_traffic_light(0, 0, 1);
26              control_traffic_light(1, 1, 0);
27              if (traffic_light_timer1 <= 0) {
28                 traffic_light_timer1 = yellow_time;
29                 tl_st = YELLOW_RED;
30             }
31             break;
32          case YELLOW_RED:

```

```

33     control_traffic_light(0, 1, 1);
34     control_traffic_light(1, 1, 0);
35     if (traffic_light_timer1 <= 0) {
36         traffic_light_timer1 = red_time;
37         traffic_light_timer2 = green_time;
38         tl_st = RED_GREEN;
39     }
40     break;
41 default:
42     break;
43 }
44 }

```

5.8.5 ManuallyTrafficLight FSM

```

1  /*
2  * @brief:  finite state machine to display traffic light manually
3  * @para:  none
4  * @retval: none*/
5  void manually_traffic_state(void) {
6      switch (man_tl_st) {
7          case RED_GREEN:
8              control_traffic_light(0, 1, 0);
9              control_traffic_light(1, 0, 1);
10             break;
11
12          case GREEN_RED:
13              control_traffic_light(0, 0, 1);
14              control_traffic_light(1, 1, 0);
15             break;
16          default:
17              control_traffic_light(0, 1, 1);
18              control_traffic_light(1, 1, 1);
19             break;
20         }
21     }

```

5.8.6 LED FSM

```

1  /*@brief: state machine to blink led in 2Hz
2  * @para:  none
3  * @retval: none*/
4  void fsm_led(void) {
5      // transition state in 0.25s
6      if (flag_toggle_led) {
7          flag_toggle_led = 0;
8          sch_add_task(task_toggle_led, TOGGLE_TIME, 0);
9      }
10     switch (led_st) {
11         case ON:
12             switch (light_st) {
13                 case RED_ADJUSTMENT:
14                     // turn red led on
15                     control_traffic_light(0, 1, 0);
16                     control_traffic_light(1, 1, 0);
17                     break;
18                 case YELLOW_ADJUSTMENT:
19                     // turn yellow led on
20                     control_traffic_light(0, 1, 1);
21                     control_traffic_light(1, 1, 1);
22                     break;
23                 case GREEN_ADJUSTMENT:
24                     // turn green led on
25                     control_traffic_light(0, 0, 1);
26                     control_traffic_light(1, 0, 1);
27                     break;
28                 default:
29                     break;
30             }
31             break;
32         case OFF:
33             control_traffic_light(0, 0, 0);

```



```

34     control_traffic_light(1, 0, 0);
35     break;
36 }
37 }

```

5.8.7 Button FSM

```

1  /*
2  * @brief: mode button fsm - 2 states
3  * @para: none
4  * @retval: 1 - successful
5  *         0 - fail
6  * */
7  bool button0_fsm(void) {
8      switch (button_st[0]) {
9          case release:
10             if (is_button_pressed(0) == 1) {
11                 // to do
12                 light_pre_st = light_st;
13                 control_pedestrian_light(0, 0);
14                 pedestrian_timer = 0;
15                 switch (light_st) {
16                     case TRAFFIC_LIGHT:
17                         red_time_buffer = red_time;
18                         yellow_time_buffer = yellow_time;
19                         green_time_buffer = green_time;
20                         light_st = RED_ADJUSTMENT;
21                         uart_SendMode(2);
22                         uart_SendBufferRed(red_time_buffer);
23                         break;
24                     case RED_ADJUSTMENT:
25                         light_st = YELLOW_ADJUSTMENT;
26                         uart_SendMode(3);
27                         uart_SendBufferYellow(yellow_time_buffer);
28                         break;
29                     case YELLOW_ADJUSTMENT:
30                         light_st = GREEN_ADJUSTMENT;
31                         uart_SendMode(4);
32                         uart_SendBufferGreen(green_time_buffer);
33                         break;
34                     case GREEN_ADJUSTMENT:
35                         light_st = TRAFFIC_LIGHT;
36                         tl_st = RED_GREEN;
37                         traffic_light_timer1 = red_time;
38                         traffic_light_timer2 = green_time;
39                         break;
40                     default:
41                         break;
42                 }
43                 button_st[0] = pressed;
44             } else if (is_button_pressed(0) == ERROR)
45                 return 0;
46             break;
47          case pressed:
48             if (!is_button_pressed(0)) {
49                 button_st[0] = release;
50             } else {
51                 return 0;
52             }
53             break;
54          default:
55             return 0;
56      }
57      return 1;
58  }
59  /*
60  * @brief: setting-value button fsm - 2 states
61  * @para: none
62  * @retval: 1 - successful
63  *         0 - fail
64  * */
65  bool button2_fsm(void) {
66      switch (button_st[2]) {

```

```

67  case release:
68      if (is_button_pressed(2) == 1) {
69          // to do
70          light_pre_st = light_st;
71          switch (light_st) {
72              case TRAFFIC_LIGHT:
73                  light_st = MANUALLY_SET;
74                  break;
75              case MANUALLY_SET:
76                  light_st = TRAFFIC_LIGHT;
77                  break;
78              default:
79                  light_st = SET_VALUE;
80          }
81          button_st[2] = pressed;
82      } else if (is_button_pressed(2) == ERROR)
83          return 0;
84      break;
85  case pressed:
86      if (!is_button_pressed(2)) {
87          button_st[2] = release;
88      } else {
89          return 0;
90      }
91      break;
92  default:
93      return 0;
94      break;
95  }
96  return 1;
97 }
98
99 /*
100  * @brief:  increasing-value button fsm - 3 states
101  * @para:  none
102  * @retval: 1 - successful
103  *          0 - fail
104  */
105 bool button1_fsm(void) {
106     switch (button_st[1]) {
107         case release:
108             if (is_button_pressed(1) == 1) {
109                 // to do
110                 light_pre_st = light_st;
111                 switch (light_st) {
112                     case MANUALLY_SET:
113                         switch (man_tl_st) {
114                             case RED_GREEN:
115                                 man_tl_st = GREEN_RED;
116                                 break;
117                             case GREEN_RED:
118                                 man_tl_st = RED_GREEN;
119                                 break;
120                             default:
121                                 man_tl_st = GREEN_RED;
122                         }
123                         break;
124                     default:
125                         light_st = INCREASE_BY_1;
126                         break;
127                 }
128                 button_st[1] = pressed;
129             } else if (is_button_pressed(1) == ERROR)
130                 return 0;
131             break;
132         case pressed:
133             if (!is_button_pressed(1)) {
134                 button_st[1] = release;
135             } else if (is_button_long_pressed(1) == 1) {
136                 button_st[1] = long_pressed;
137             } else {
138                 return 0;
139             }
140             break;
141         case long_pressed:

```

```

142 // to do
143 if (light_st != INCREASE_BY_1_OVER_TIME) {
144     light_pre_st = light_st;
145     light_st = INCREASE_BY_1_OVER_TIME;
146 }
147 if (!is_button_pressed(1)) {
148     light_st = light_pre_st;
149     light_pre_st = INCREASE_BY_1_OVER_TIME;
150     button_st[1] = release;
151 }
152 break;
153 default:
154     return 0;
155 break;
156 }
157 return 1;
158 }
159 /*
160 * @brief: pedestrian button
161 * @para: none
162 * @retval: 1 - successful
163 *         0 - fail
164 * */
165 bool button3_fsm(void) {
166     switch (button_st[3]) {
167     case release:
168         if (is_button_pressed(3) == 1) {
169             // to do
170             pedestrian_timer = PEDESTRIAN_TIMER;
171             // flag_pedestrian_on = 1;
172             button_st[3] = pressed;
173         } else if (is_button_pressed(3) == ERROR)
174             return 0;
175         break;
176     case pressed:
177         if (!is_button_pressed(2)) {
178             button_st[3] = release;
179         } else {
180             return 0;
181         }
182         break;
183     default:
184         return 0;
185         break;
186     }
187     return 1;
188 }

```

5.8.8 Buzzer FSM

```

1 void buzzer_fsm(void){
2     switch(bz_st){
3     case BUZZER_ON:
4         if(flag_buzzer){
5             flag_buzzer = 0;
6             sch_add_task(task_buzzer_on, buzzer_getToggle_time(), 0);
7             bz_st = BUZZER_OFF;
8         }
9         break;
10    case BUZZER_OFF:
11        if(flag_buzzer){
12            flag_buzzer = 0;
13            sch_add_task(task_buzzer_off, buzzer_getToggle_time(), 0);
14            bz_st = BUZZER_ON;
15        }
16        break;
17    }
18 }

```

5.9 Main Loop and Set Up before Main Loop

```
1 int main(void)
2 {
3     /* USER CODE BEGIN 1 */
4
5     /* USER CODE END 1 */
6
7     /* MCU Configuration-----*/
8
9     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
10    HAL_Init();
11
12    /* USER CODE BEGIN Init */
13
14    /* USER CODE END Init */
15
16    /* Configure the system clock */
17    SystemClock_Config();
18
19    /* USER CODE BEGIN SysInit */
20
21    /* USER CODE END SysInit */
22
23    /* Initialize all configured peripherals */
24    MX_GPIO_Init();
25    MX_TIM2_Init();
26    MX_TIM3_Init();
27    MX_USART2_UART_Init();
28    /* USER CODE BEGIN 2 */
29    init();
30    /* USER CODE END 2 */
31
32    /* Infinite loop */
33    /* USER CODE BEGIN WHILE */
34    while (1) {
35        loop();
36        /* USER CODE END WHILE */
37
38        /* USER CODE BEGIN 3 */
39    }
40    /* USER CODE END 3 */
41 }
```