

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



Microprocessors-Microcontrollers

COURSE ID: CO3010 - HK251 CLASS: CC02

Lab 3 Report

Lecturer: Phan Văn Sỹ
Students: Nguyễn Thế Khang - 2352490

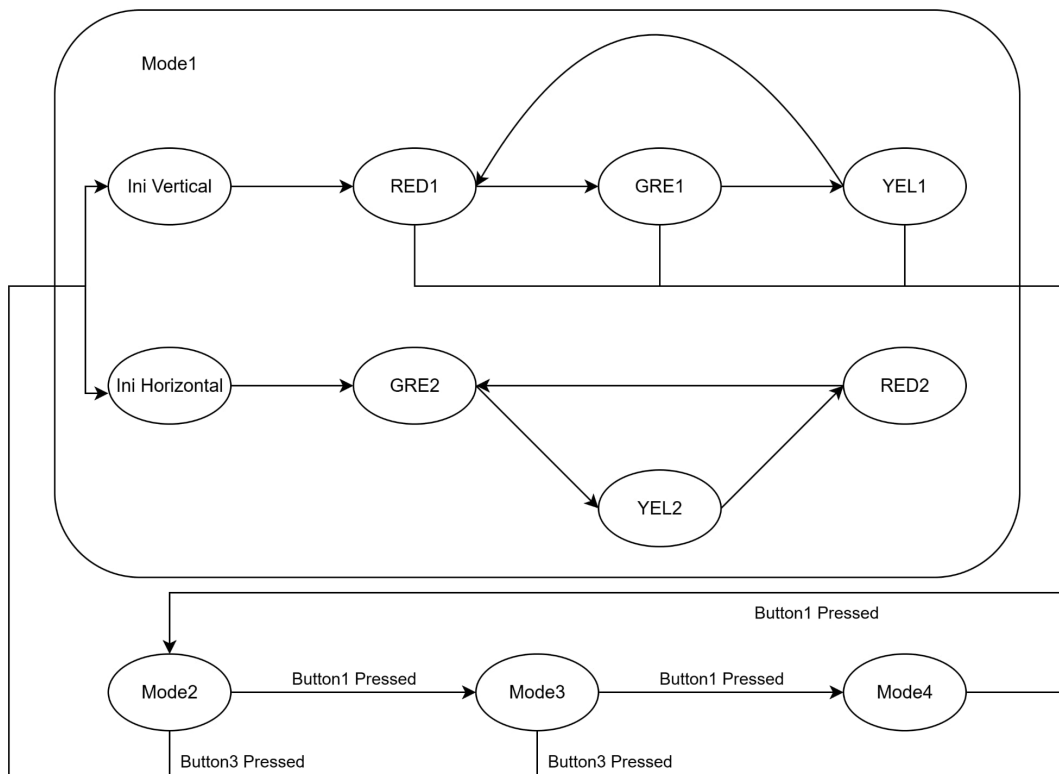
HỒ CHÍ MINH CITY, October 2025

Contents

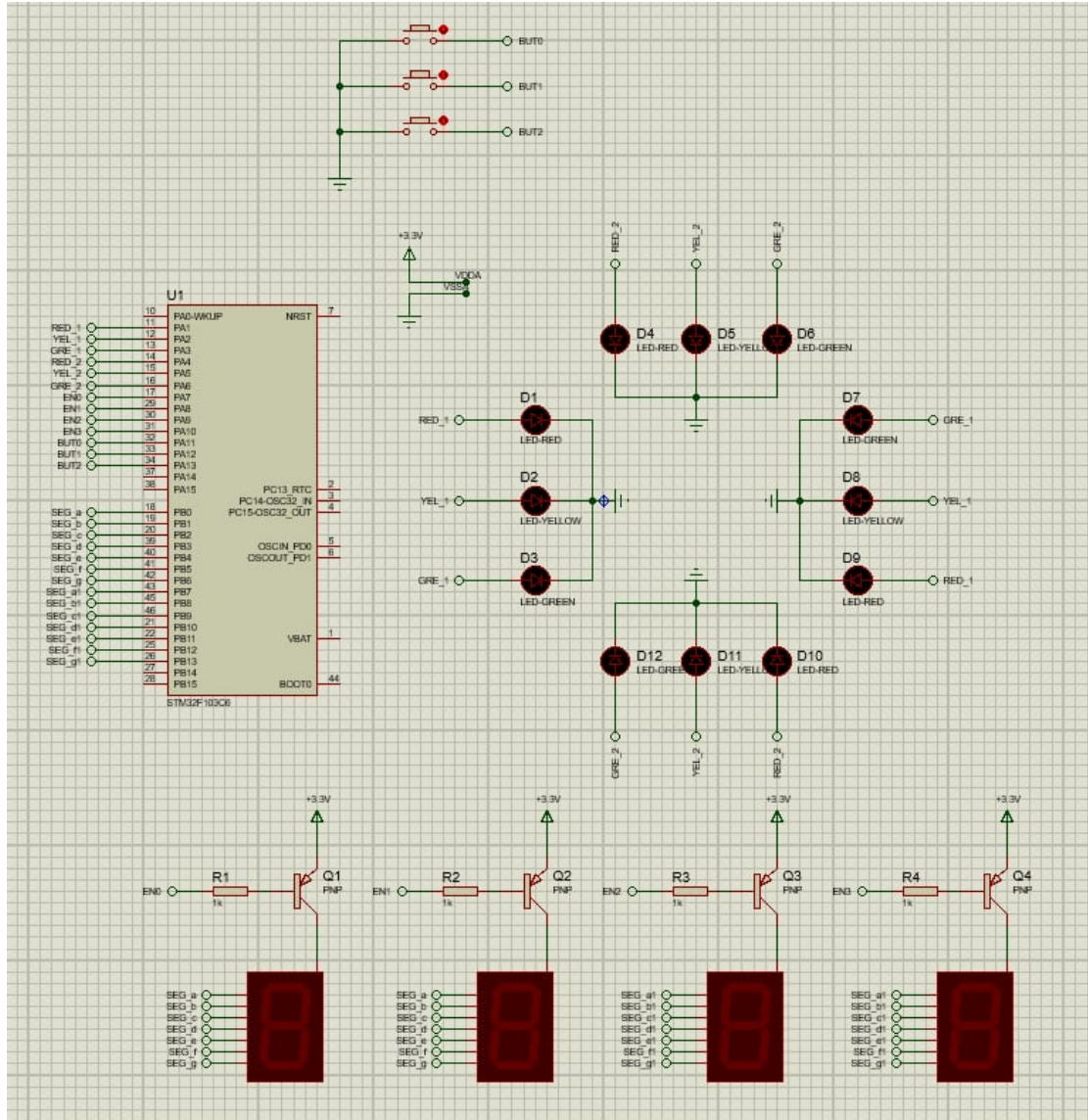
1	Exercise and Report	2
1.1	Exercise 1	2
1.2	Exercise 2	3
1.2.1	GitHub Repository	3
1.3	Exercise 3	4
1.4	Exercise 4	5
1.4.1	timer functions	5
1.4.2	GitHub Repository	7
1.5	Exercise 5	8
1.5.1	input_reading functions	8
1.5.2	GitHub Repository	9
1.6	Exercise 6	10
1.6.1	led_display functions	10
1.6.2	GitHub Repository	12
1.7	Exercise 7 & 8 & 9	13
1.7.1	GitHub Repository	15
1.7.2	traffic functions	16
1.7.3	GitHub Repository	18
1.7.4	global functions	19
1.7.5	GitHub Repository	19
1.8	Demonstration Link	20

1 Exercise and Report

1.1 Exercise 1



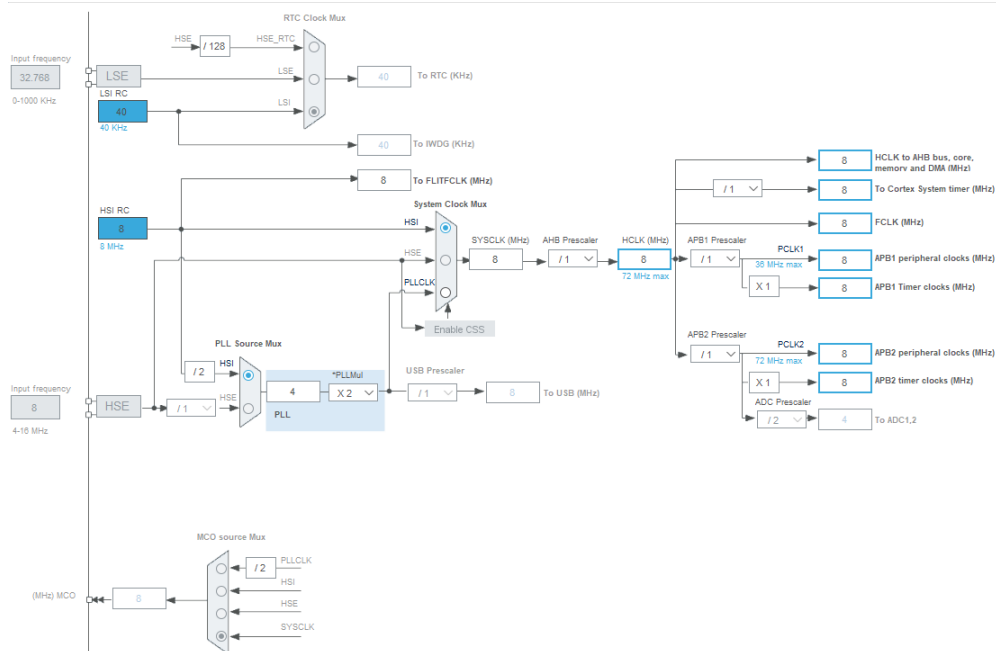
1.2 Exercise 2



1.2.1 GitHub Repository

- Schematic: [Lab3.pdsprj](#)

1.3 Exercise 3



User Constants	NVIC Settings	DMA Settings
Parameter Settings		
Configure the below parameters :		
<input type="text" value="Search (Ctrl+F)"/>		
<div> <div>Counter Settings</div> <div> Prescaler (PSC - 16 bits value) 7999 Counter Mode Up Counter Period (AutoReload Re... 9 Internal Clock Division (CKD) No Division auto-reload preload Disable </div> </div>		
<div> <div>Trigger Output (TRGO) Parameters</div> <div> Master/Slave Mode (MSM bit) Disable (Trigger input effect not delayed) Trigger Event Selection Reset (UG bit from TIMx_EGR) </div> </div>		



1.4 Exercise 4

1.4.1 timer functions

timer.h and timer.c implementing a constant *CYCLE* to be able to modify the time duration of the timer interrupt, without affecting the overall system.

- timer.h

```
1 #ifndef INC_TIMER_H_
2 #define INC_TIMER_H_
3
4 #include "global.h"
5
6 #define CYCLE 10
7
8 extern int timer1, timer2, timer3;
9 extern int flag1, flag2, flag3;
10
11 void set1(int timer);
12 void set2(int timer);
13 void set3(int timer);
14
15 int isTimer1Expired(void);
16 int isTimer3Expired(void);
17
18 void resetTimer(int timer);
19 void timerRun();
20
21 #endif /* INC_TIMER_H_ */
```

- timer.c

```
1 #include "timer.h"
2
3 static int scan_state = 0;
4
5 int timer1 = 0, timer2 = 0, timer3 = 0;
6 int flag1 = 0, flag2 = 0, flag3 = 0;
7
8 void set1(int timer){
9     timer1 = timer / CYCLE;
10    flag1 = 0;
11 }
12
13 void set2(int timer){
14     timer2 = timer / CYCLE;
15     flag2 = 0;
16 }
```



```
17
18 void set3(int timer){
19     timer3 = timer / CYCLE;
20     flag3 = 0;
21 }
22
23 void resetTimer(int timer){
24     switch(timer){
25     case 1:
26         timer1 = 0;
27         flag1 = 0;
28         break;
29     case 2:
30         timer2 = 0;
31         flag2 = 0;
32         break;
33     case 3:
34         timer3 = 0;
35         flag3 = 0;
36         break;
37     default:
38         timer1 = 0;
39         break;
40     }
41 }
42
43 void timerRun(){
44     timer1--;
45     timer2--;
46     timer3--;
47     if (timer1 == 0){
48         flag1 = 1;
49     }
50     if (timer2 == 0){
51         flag2 = 1;
52     }
53     if (timer3 == 0){
54         flag3 = 1;
55     }
56 }
57
58 int isTimer1Expired(void){
59     if(flag1 == 1){
60         flag1 = 0;
61         return 1;
62     }
63     return 0;
64 }
65
66 int isTimer3Expired(void){
67     if(flag3 == 1){
68         flag3 = 0;
69         return 1;

```



```
70 }
71 return 0;
72 }
73
74 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
75     if (htim->Instance == TIM2) {
76         button_reading();
77         fsm_for_input_processing();
78         timerRun();
79         if (flag2) {
80             scanSegment(scan_state);
81             scan_state = (scan_state + 1) % 2;
82             set2(250);
83         }
84     }
85 }
```

1.4.2 GitHub Repository

- Source: [timer.h](#)
- Source: [timer.c](#)



1.5 Exercise 5

1.5.1 input_reading functions

Initializes a maximum of three buttons.

- **button_reading**: Debounces and reads three buttons (GPIOA pins 11–13), updates buttonBuffer, and tracks 1-second presses.
- **is_button_pressed**: Returns 1 if a button is pressed, 0 otherwise.
- **is_button_pressed_1s**: Returns 1 for a 1-second button press, 0xff for an invalid index.
- input_reading.h

```
1 #ifndef INC_INPUT_READING_H_
2 #define INC_INPUT_READING_H_
3
4 #include "global.h"
5
6 void button_reading(void);
7
8 unsigned char is_button_pressed(unsigned char index);
9 unsigned char is_button_pressed_1s(unsigned char index);
10
11 #endif /* INC_INPUT_READING_H_ */
```

- input_reading.c

```
1 #include "input_reading.h"
2
3 #define NO_OF_BUTTONS 3
4 #define DURATION_FOR_AUTO_INCREASING 100
5 #define BUTTON_IS_PRESSED GPIO_PIN_RESET
6 #define BUTTON_IS_RELEASED GPIO_PIN_SET
7
8 static GPIO_PinState buttonBuffer[NO_OF_BUTTONS];
9 static GPIO_PinState debounceButtonBuffer1[NO_OF_BUTTONS];
10 static GPIO_PinState debounceButtonBuffer2[NO_OF_BUTTONS];
11 static uint8_t flagForButtonPress1s[NO_OF_BUTTONS];
12 static uint16_t counterForButtonPress1s[NO_OF_BUTTONS];
13
14
15 void button_reading(void) {
16     for (uint8_t i = 0; i < NO_OF_BUTTONS; i++) {
17         debounceButtonBuffer2[i] = debounceButtonBuffer1[i];
18         switch (i) {
19             case 0:
20                 debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_11);
```

```
21         break;
22     case 1:
23         debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_12);
24         break;
25     case 2:
26         debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_13);
27         break;
28     }
29     if (debounceButtonBuffer1[i] == debounceButtonBuffer2[i]) {
30         buttonBuffer[i] = debounceButtonBuffer1[i];
31     }
32     if (buttonBuffer[i] == BUTTON_IS_PRESSED) {
33         if (counterForButtonPress1s[i] < DURATION_FOR_AUTO_INCREASING) {
34             counterForButtonPress1s[i]++;
35         } else {
36             flagForButtonPress1s[i] = 1;
37         }
38     } else {
39         counterForButtonPress1s[i] = 0;
40         flagForButtonPress1s[i] = 0;
41     }
42 }
43 }
44
45 unsigned char is_button_pressed(uint8_t index) {
46     if (index >= NO_OF_BUTTONS) return 0;
47     return (buttonBuffer[index] == BUTTON_IS_PRESSED);
48 }
49
50 unsigned char is_button_pressed_1s(uint8_t index) {
51     if (index >= NO_OF_BUTTONS) return 0xff;
52     return (flagForButtonPress1s[index] == 1);
53 }
```

1.5.2 GitHub Repository

- Source: [input_reading.h](#)
- Source: [input_reading.c](#)



1.6 Exercise 6

1.6.1 led_display functions

led_display.h and led_display.c are used to implement the four the 7-Segment LEDs. The 7-Segmen LEDs are indexed as 0, 1, 2, 3.

- **setHorizontal**: Sets pins for a 7-segment display for horizontal path to show a digit (0–9).
- **setVertical**: Sets pins for a 7-segment display for vertical path to show a digit (0–9).
- **scanSegment**: Alternates between displaying two pairs of 7-segment digits (horizontal: segment_buffer[0,1], vertical: segment_buffer[2,3]).
- **updateSegment**: Updates segment_buffer with four digits for 7-segment display.
- **updateSegment2Digits**: Converts two numbers to four digits and updates segment_buffer for two 2-digit 7-segment displays.

- led_display.h

```
1 #ifndef SRC_LED_DISPLAY_H_
2 #define SRC_LED_DISPLAY_H_
3
4 #include "global.h"
5
6 extern int segment_buffer[4];
7
8 void setHorizontal(int);
9 void setVertical(int);
10 void scanSegment(int);
11
12 void updateSegment(int, int, int, int);
13 void updateSegment2Digits(int, int);
14
15 #endif /* SRC_LED_DISPLAY_H_ */
```

- led_display.c

```
1 #include "led_display.h"
2
3 int segment_buffer[4] = {0, 0, 0, 0};
4
5 GPIO_PinState pinArr[11][7] = {
6     {0, 0, 0, 0, 0, 0, 1}, // 0
7     {1, 0, 0, 1, 1, 1, 1}, // 1
8     {0, 0, 1, 0, 0, 1, 0}, // 2
9     {0, 0, 0, 0, 1, 1, 0}, // 3
10    {1, 0, 0, 1, 1, 0, 0}, // 4
```



```
11 {0, 1, 0, 0, 1, 0, 0}, // 5
12 {0, 1, 0, 0, 0, 0, 0}, // 6
13 {0, 0, 0, 1, 1, 1, 1}, // 7
14 {0, 0, 0, 0, 0, 0, 0}, // 8
15 {0, 0, 0, 0, 1, 0, 0}, // 9
16 {1, 1, 1, 1, 1, 1, 1} // OFF
17};
18
19void setHorizontal(int num) {
20    int index = (num >= 0 && num <= 9) ? num : 10;
21
22    HAL_GPIO_WritePin(SEG_a_GPIO_Port, SEG_a_Pin, pinArr[index][0]);
23    HAL_GPIO_WritePin(SEG_b_GPIO_Port, SEG_b_Pin, pinArr[index][1]);
24    HAL_GPIO_WritePin(SEG_c_GPIO_Port, SEG_c_Pin, pinArr[index][2]);
25    HAL_GPIO_WritePin(SEG_d_GPIO_Port, SEG_d_Pin, pinArr[index][3]);
26    HAL_GPIO_WritePin(SEG_e_GPIO_Port, SEG_e_Pin, pinArr[index][4]);
27    HAL_GPIO_WritePin(SEG_f_GPIO_Port, SEG_f_Pin, pinArr[index][5]);
28    HAL_GPIO_WritePin(SEG_g_GPIO_Port, SEG_g_Pin, pinArr[index][6]);
29}
30
31void setVertical(int num) {
32    int index = (num >= 0 && num <= 9) ? num : 10;
33
34    HAL_GPIO_WritePin(SEG_a1_GPIO_Port, SEG_a1_Pin, pinArr[index][0]);
35    HAL_GPIO_WritePin(SEG_b1_GPIO_Port, SEG_b1_Pin, pinArr[index][1]);
36    HAL_GPIO_WritePin(SEG_c1_GPIO_Port, SEG_c1_Pin, pinArr[index][2]);
37    HAL_GPIO_WritePin(SEG_d1_GPIO_Port, SEG_d1_Pin, pinArr[index][3]);
38    HAL_GPIO_WritePin(SEG_e1_GPIO_Port, SEG_e1_Pin, pinArr[index][4]);
39    HAL_GPIO_WritePin(SEG_f1_GPIO_Port, SEG_f1_Pin, pinArr[index][5]);
40    HAL_GPIO_WritePin(SEG_g1_GPIO_Port, SEG_g1_Pin, pinArr[index][6]);
41}
42
43void scanSegment(int state) {
44    switch (state % 2) {
45        case 0:
46            HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, RESET);
47            HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, SET);
48            HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin, RESET);
49            HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin, SET);
50
51            setHorizontal(segment_buffer[0]);
52            setVertical(segment_buffer[2]);
53            break;
54
55        case 1:
56            HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, SET);
57            HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, RESET);
58            HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin, SET);
59            HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin, RESET);
60
61            setHorizontal(segment_buffer[1]);
62            setVertical(segment_buffer[3]);
63            break;
```



```
64
65     default:
66         break;
67     }
68 }
69
70 void updateSegment(int a, int b, int c, int d) {
71     segment_buffer[0] = a;
72     segment_buffer[1] = b;
73     segment_buffer[2] = c;
74     segment_buffer[3] = d;
75 }
76
77 void updateSegment2Digits(int firstNum, int secNum) {
78     segment_buffer[0] = firstNum / 10;
79     segment_buffer[1] = firstNum % 10;
80     segment_buffer[2] = secNum / 10;
81     segment_buffer[3] = secNum % 10;
82 }
```

1.6.2 GitHub Repository

- Source: [led_display.h](#)
- Source: [led_display.c](#)

1.7 Exercise 7 & 8 & 9

- **getSystemMode**: Indexes the current system mode (0–3) for traffic light operation.
- **fsm_for_input_processing**: Finite state machine processing button inputs: Button 1 cycles through modes (0: normal, 1: red, 2: yellow, 3: green) and resets LEDs/display; Button 2 increments `temp_duration` (1–99) on press or every 0.5s after a 1s hold in modes 1–3; Button 3 confirms new duration to set `DURATION_RED/YELLOW/GREEN_DEFAULT` and resets to mode 1.
- **is_button_just_pressed**: Detects a button's transition from released to pressed, returning 1 for a new press, 0 otherwise, using stored previous states.
- `input_processing.h`

```
1 #ifndef INC_INPUT_PROCESSING_H_
2 #define INC_INPUT_PROCESSING_H_
3
4 #include "global.h"
5
6 void fsm_for_input_processing(void);
7
8 #endif /* INC_INPUT_PROCESSING_H_ */
```

- `input_processing.c`

```
1 #include "input_processing.h"
2
3 uint8_t systemMode = 0;
4 int temp_duration = 0;
5 static int auto_increment_timer = 0;
6
7 enum ButtonState { BUTTON_RELEASED, BUTTON_PRESSED, BUTTON_PRESSED_MORE_THAN_1_SECOND };
8 enum ButtonState button1State = BUTTON_RELEASED;
9
10 uint8_t getSystemMode(void) {
11     return systemMode;
12 }
13
14 int is_button_just_pressed(int button_index) {
15     static GPIO_PinState prev_state[3] = {SET, SET, SET};
16
17     int is_pressed_now = is_button_pressed(button_index);
18
19     if (is_pressed_now && (prev_state[button_index] == GPIO_PIN_SET)) {
20         prev_state[button_index] = RESET;
21         return 1;
22     }
23 }
```

```
24     if (!is_pressed_now) {
25         prev_state[button_index] = SET;
26     }
27
28     return 0;
29 }
30
31 void fsm_for_input_processing(void) {
32     // Button 1: changing mode
33     if (is_button_just_pressed(0)) {
34         systemMode = (systemMode + 1) % 4;
35
36         switch(systemMode) {
37             case 1:
38                 temp_duration = DURATION_RED_DEFAULT;
39                 break;
40             case 2:
41                 temp_duration = DURATION_YELLOW_DEFAULT;
42                 break;
43             case 3:
44                 temp_duration = DURATION_GREEN_DEFAULT;
45                 break;
46             case 0:
47                 resetLEDsAndDisplay();
48                 break;
49         }
50     }
51     // Button 2: changing duration
52     if (systemMode > 0) {
53         switch(button1State) {
54             case BUTTON_RELEASED:
55                 if (is_button_just_pressed(1)) {
56                     button1State = BUTTON_PRESSED;
57                     temp_duration++;
58                     if (temp_duration > 99) {
59                         temp_duration = 1;
60                     }
61                 }
62                 break;
63             case BUTTON_PRESSED:
64                 if (!is_button_pressed(1)) {
65                     button1State = BUTTON_RELEASED;
66                 } else if (is_button_pressed_1s(1)) {
67                     button1State = BUTTON_PRESSED_MORE_THAN_1_SECOND;
68                 }
69                 break;
70             // Increase value every 500ms (0.5s) after holding button > 1s
71             case BUTTON_PRESSED_MORE_THAN_1_SECOND:
72                 if (!is_button_pressed(1)) {
73                     button1State = BUTTON_RELEASED;
74                 } else {
75                     if (auto_increment_timer > 0) {
76                         auto_increment_timer--;
```

```
77         }
78
79         if (auto_increment_timer == 0) {
80             temp_duration++;
81             if (temp_duration > 99) {
82                 temp_duration = 1;
83             }
84             auto_increment_timer = 50;
85         }
86     }
87     break;
88 }
89 //Button 3: confirm
90 if (is_button_just_pressed(2)) {
91     switch(systemMode) {
92         case 1:
93             DURATION_RED_DEFAULT = temp_duration;
94             break;
95         case 2:
96             DURATION_YELLOW_DEFAULT = temp_duration;
97             break;
98         case 3:
99             DURATION_GREEN_DEFAULT = temp_duration;
100            break;
101     }
102     systemMode = 0;
103     resetLEDsAndDisplay();
104 }
105 }
106 }
```

1.7.1 GitHub Repository

- Source: [input_processing.h](#)
- Source: [input_processing.c](#)

1.7.2 traffic functions

Implement the main logic for traffic light and displaying user input values for LED_RED, LED_YEL and LED_GRE.

- **resetLEDsAndDisplay**: Resets all LEDs (red, yellow, green for horizontal and vertical) and 7-segment displays to off, clears states.
- **completeTraffic_light**: Manages traffic light operation based on system mode: In mode 1 (indexed as 0), cycles through four states performing normal traffic operations with countdowns (`countdown1`, `countdown2`) updated every 1s via `timer1`, controlling LEDs and 7-segment displays; in modes 2–4, blinks red, yellow, or green LEDs every 250ms using `timer3`, displays current mode and value of selected LED on 7-segment displays and resets initialization for mode transitions.

- traffic.h

```
1 #ifndef INC_TRAFFIC_H_
2 #define INC_TRAFFIC_H_
3
4 #include "global.h"
5
6 void resetLEDsAndDisplay();
7 void completeTraffic_light();
8
9 #endif /* INC_TRAFFIC_H_ */
```

- traffic.c

```
1 #include "global.h"
2
3 static uint8_t ledState = 0;
4 static uint8_t trafficState = 0;
5 static uint8_t is_initialized = 0;
6 static uint8_t is_blink_timer_set = 0;
7
8 static int countdown1 = 0;
9 static int countdown2 = 0;
10
11 void resetLEDsAndDisplay(void) {
12     HAL_GPIO_WritePin(GPIOA,
13         RED_1_Pin|YEL_1_Pin|GRE_1_Pin|RED_2_Pin|YEL_2_Pin|GRE_2_Pin, RESET);
14     updateSegment(10, 10, 10, 10);
15     ledState = 0;
16     is_initialized = 0;
17     is_blink_timer_set = 0;
18     set1(1000);
19 }
```



```
19
20 void completeTraffic_light(void) {
21     uint8_t mode = getSystemMode();
22
23     if (mode == 0) {
24         is_blink_timer_set = 0;
25
26         if (!is_initialized) {
27             trafficState = 0;
28             countdown1 = DURATION_GREEN_DEFAULT;
29             countdown2 = DURATION_RED_DEFAULT;
30             is_initialized = 1;
31         }
32
33         if (isTimer1Expired()) {
34             set1(1000);
35             updateSegment2Digits(countdown1, countdown2);
36             countdown1--;
37             countdown2--;
38
39             switch (trafficState) {
40                 case 0: // State 0: Green horizontal, Red vertical
41                     HAL_GPIO_WritePin(GPIOA, GRE_1_Pin, SET);
42                     HAL_GPIO_WritePin(GPIOA, RED_2_Pin, SET);
43                     HAL_GPIO_WritePin(GPIOA, RED_1_Pin|YEL_1_Pin|YEL_2_Pin|GRE_2_Pin,
44                                     RESET);
45                     if (countdown1 < 1) {
46                         trafficState = 1;
47                         countdown1 = DURATION_YELLOW_DEFAULT;
48                     }
49                     break;
50                 case 1: // State 1: Yellow horizontal, Red vertical
51                     HAL_GPIO_WritePin(GPIOA, YEL_1_Pin, SET);
52                     HAL_GPIO_WritePin(GPIOA, RED_2_Pin, SET);
53                     HAL_GPIO_WritePin(GPIOA, RED_1_Pin|GRE_1_Pin|YEL_2_Pin|GRE_2_Pin,
54                                     RESET);
55                     if (countdown1 < 1) {
56                         trafficState = 2;
57                         countdown1 = DURATION_RED_DEFAULT;
58                         countdown2 = DURATION_GREEN_DEFAULT;
59                     }
60                     break;
61                 case 2: // State 2: Red horizontal, Green vertical
62                     HAL_GPIO_WritePin(GPIOA, RED_1_Pin, SET);
63                     HAL_GPIO_WritePin(GPIOA, GRE_2_Pin, SET);
64                     HAL_GPIO_WritePin(GPIOA, YEL_1_Pin|GRE_1_Pin|RED_2_Pin|YEL_2_Pin,
65                                     RESET);
66                     if (countdown2 < 1) {
67                         trafficState = 3;
68                         countdown2 = DURATION_YELLOW_DEFAULT;
69                     }
70                     break;
71                 case 3: // State 3: Red horizontal, Yellow vertical
```

```
69         HAL_GPIO_WritePin(GPIOA, RED_1_Pin, SET);
70         HAL_GPIO_WritePin(GPIOA, YEL_2_Pin, SET);
71         HAL_GPIO_WritePin(GPIOA, YEL_1_Pin|GRE_1_Pin|RED_2_Pin|GRE_2_Pin,
72             RESET);
73         if (countdown2 < 1) {
74             trafficState = 0;
75             countdown1 = DURATION_GREEN_DEFAULT;
76             countdown2 = DURATION_RED_DEFAULT;
77         }
78         break;
79     }
80 } else {
81     is_initialized = 0;
82
83     if (!is_blink_timer_set) {
84         set3(250);
85         is_blink_timer_set = 1;
86     }
87     if (isTimer3Expired()) {
88         set3(250);
89         ledState = !ledState;
90     }
91     // Select Mode 2, 3, 4
92     if (mode == 1) {
93         updateSegment(0, 2, temp_duration / 10, temp_duration % 10);
94         HAL_GPIO_WritePin(GPIOA, RED_1_Pin | RED_2_Pin, ledState);
95         HAL_GPIO_WritePin(GPIOA, YEL_1_Pin | YEL_2_Pin | GRE_1_Pin | GRE_2_Pin,
96             RESET);
97     } else if (mode == 2) {
98         updateSegment(0, 3, temp_duration / 10, temp_duration % 10);
99         HAL_GPIO_WritePin(GPIOA, YEL_1_Pin | YEL_2_Pin, ledState);
100        HAL_GPIO_WritePin(GPIOA, RED_1_Pin | RED_2_Pin | GRE_1_Pin | GRE_2_Pin,
101            RESET);
102    } else if (mode == 3) {
103        updateSegment(0, 4, temp_duration / 10, temp_duration % 10);
104        HAL_GPIO_WritePin(GPIOA, GRE_1_Pin | GRE_2_Pin, ledState);
105        HAL_GPIO_WritePin(GPIOA, RED_1_Pin | RED_2_Pin | YEL_1_Pin | YEL_2_Pin,
106            RESET);
107    }
108 }
```

1.7.3 GitHub Repository

- Source: [traffic.h](#)
- Source: [traffic.c](#)



1.7.4 global functions

Initializes all functions and default inputs for traffic light:
DURATION_RED_DEFAULT, DURATION_YELLOW_DEFAULT, DURATION_GREEN_DEFAULT.

- global.h

```
1 #ifndef INC_GLOBAL_H_
2 #define INC_GLOBAL_H_
3
4 #include "main.h"
5 #include "input_reading.h"
6 #include "input_processing.h"
7 #include "led_display.h"
8 #include "timer.h"
9 #include "traffic.h"
10
11 extern int DURATION_RED_DEFAULT;
12 extern int DURATION_YELLOW_DEFAULT;
13 extern int DURATION_GREEN_DEFAULT;
14
15 extern int temp_duration;
16
17 uint8_t getSystemMode(void);
18 void completeTraffic_light(void);
19
20 #endif /* INC_GLOBAL_H_ */
```

- global.c

```
1 #include "global.h"
2
3 // INPUT
4
5 int DURATION_RED_DEFAULT = 5 - 1;
6 int DURATION_YELLOW_DEFAULT = 2;
7 int DURATION_GREEN_DEFAULT = 3;
```

1.7.5 GitHub Repository

- Source: [global.h](#)
- Source: [global.c](#)



1.8 Demonstration Link

YouTube link for the demonstration: https://youtu.be/f_2Q6d8wJJQ