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

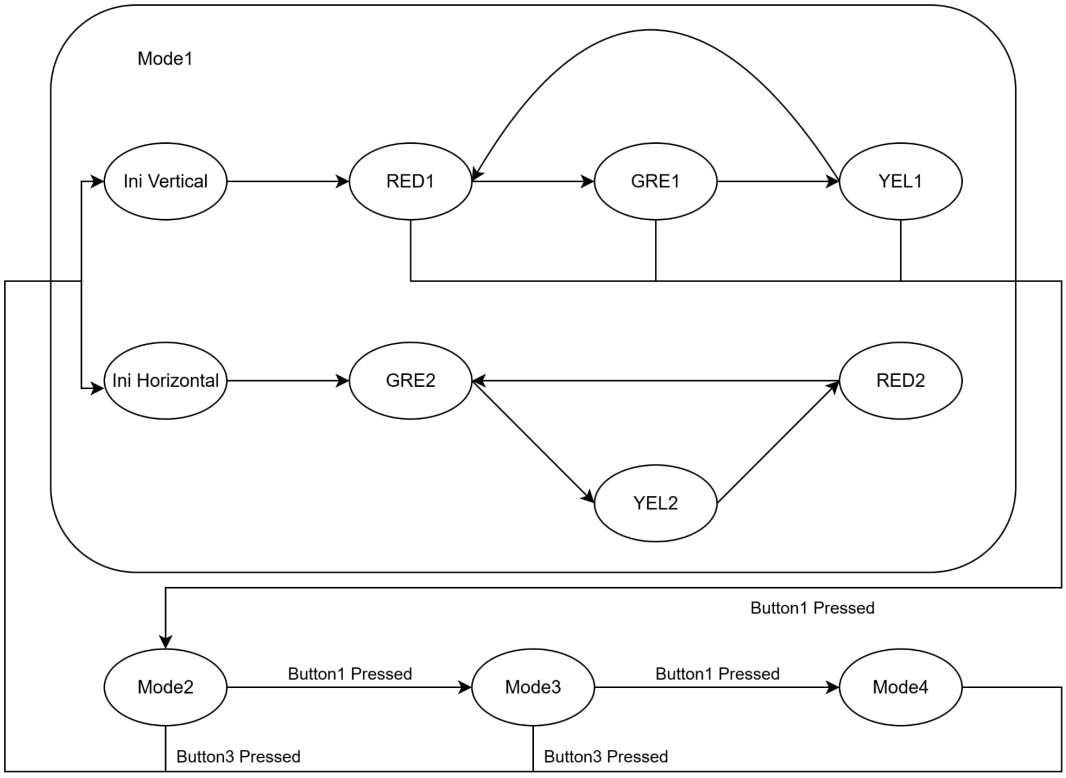
Lecturer: Phan Văn Sỹ
Students: Nguyễn Thế Khang - 2352490

HỒ CHÍ MINH CITY, October2025

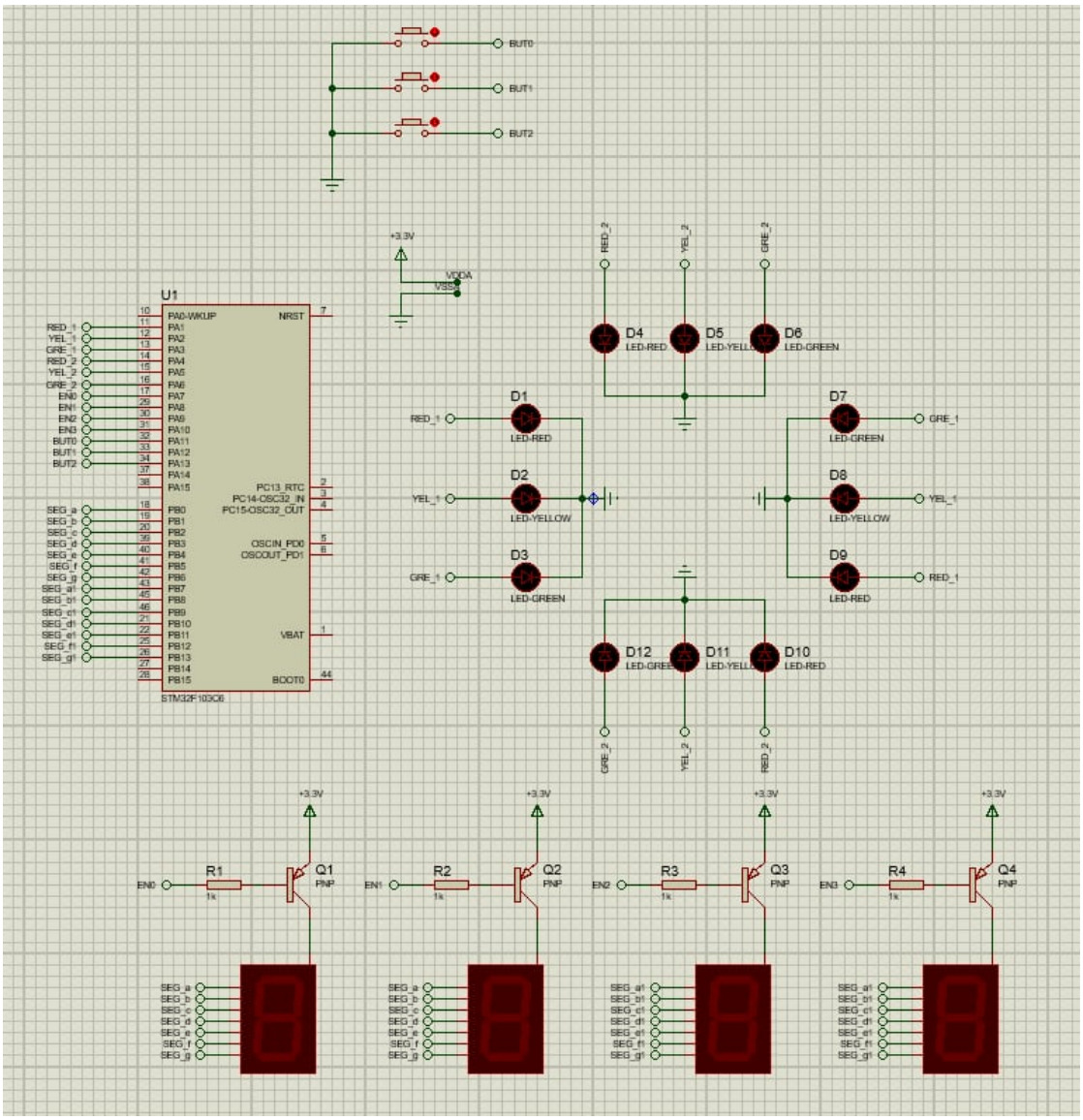
Contents

1 Exercise and Report

1.1 Exercise 1



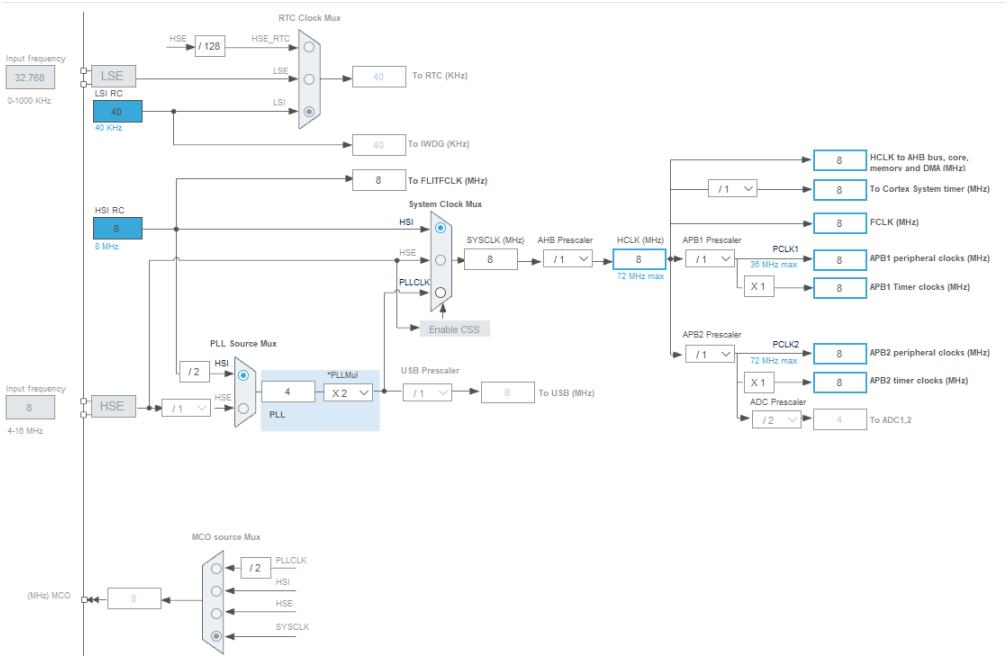
1.2 Exercise 2

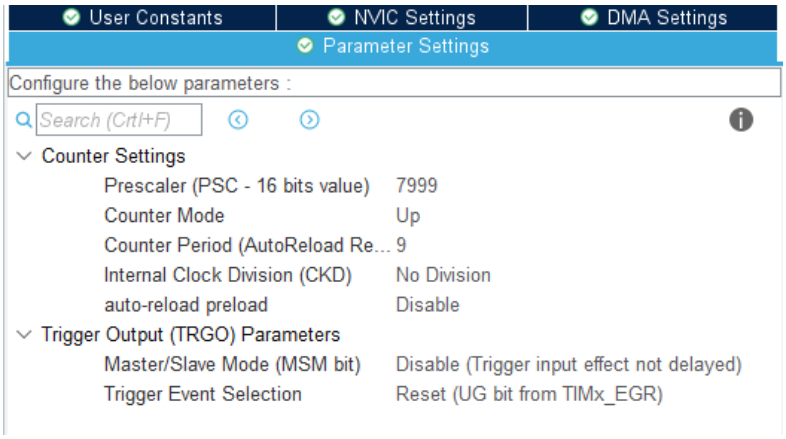


1.2.1 GitHub Repository

- **Schematic:** Lab3.pdsprj

1.3 Exercise 3





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

```
#ifndef INC_TIMER_H_
#define INC_TIMER_H_

#include "global.h"

#define CYCLE 10

extern int timer1, timer2, timer3;
extern int flag1, flag2, flag3;

void set1(int timer);
void set2(int timer);
void set3(int timer);

int isTimer1Expired(void);
int isTimer3Expired(void);

void resetTimer(int timer);
void timerRun();

#endif /* INC_TIMER_H_ */
```

- timer.c

```
#include "timer.h"

static int scan_state = 0;

int timer1 = 0, timer2 = 0, timer3 = 0;
int flag1 = 0, flag2 = 0, flag3 = 0;

void set1(int timer){
    timer1 = timer / CYCLE;
    flag1 = 0;
}

void set2(int timer){
    timer2 = timer / CYCLE;
    flag2 = 0;
}
```

```
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

```
#ifndef INC_INPUT_READING_H_
#define INC_INPUT_READING_H_

#include "global.h"

void button_reading(void);

unsigned char is_button_pressed(unsigned char index);
unsigned char is_button_pressed_1s(unsigned char index);

#endif /* INC_INPUT_READING_H_ */
```

- input_reading.c

```
#include "input_reading.h"

#define NO_OF_BUTTONS 3
#define DURATION_FOR_AUTO_INCREASING 100
#define BUTTON_IS_PRESSED GPIO_PIN_RESET
#define BUTTON_IS_RELEASED GPIO_PIN_SET

static GPIO_PinState buttonBuffer[NO_OF_BUTTONS];
static GPIO_PinState debounceButtonBuffer1[NO_OF_BUTTONS];
static GPIO_PinState debounceButtonBuffer2[NO_OF_BUTTONS];
static uint8_t flagForButtonPress1s[NO_OF_BUTTONS];
static uint16_t counterForButtonPress1s[NO_OF_BUTTONS];


void button_reading(void) {
    for (uint8_t i = 0; i < NO_OF_BUTTONS; i++) {
        debounceButtonBuffer2[i] = debounceButtonBuffer1[i];
        switch (i) {
            case 0:
                debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_11);
```

```
            break;
        case 1:
            debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_12);
            break;
        case 2:
            debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_13);
            break;
    }
    if (debounceButtonBuffer1[i] == debounceButtonBuffer2[i]) {
        buttonBuffer[i] = debounceButtonBuffer1[i];
    }
    if (buttonBuffer[i] == BUTTON_IS_PRESSED) {
        if (counterForButtonPress1s[i] < DURATION_FOR_AUTO_INCREASING) {
            counterForButtonPress1s[i]++;
        } else {
            flagForButtonPress1s[i] = 1;
        }
    } else {
        counterForButtonPress1s[i] = 0;
        flagForButtonPress1s[i] = 0;
    }
  }
}

unsigned char is_button_pressed(uint8_t index) {
  if (index >= NO_OF_BUTTONS) return 0;
  return (buttonBuffer[index] == BUTTON_IS_PRESSED);
}

unsigned char is_button_pressed_1s(uint8_t index) {
  if (index >= NO_OF_BUTTONS) return 0xff;
  return (flagForButtonPress1s[index] == 1);
}
```

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

```
#ifndef SRC_LED_DISPLAY_H_
#define SRC_LED_DISPLAY_H_

#include "global.h"

extern int segment_buffer[4];

void set7SegH(int);
void set7SegV(int);
void scan7Seg(int);

void updateSegment(int, int, int, int);
void updateSegment2Digits(int, int);

#endif /* SRC_LED_DISPLAY_H_ */
```

- led_display.c

```
#include "led_display.h"

int segment_buffer[4] = {0, 0, 0, 0};

GPIO_PinState pinArr[11][7] = {
    {0, 0, 0, 0, 0, 0, 1}, // 0
    {1, 0, 0, 1, 1, 1, 1}, // 1
    {0, 0, 1, 0, 0, 1, 0}, // 2
    {0, 0, 0, 0, 1, 1, 0}, // 3
    {1, 0, 0, 1, 1, 0, 0}, // 4
```

```
    {0, 1, 0, 0, 1, 0, 0}, // 5
    {0, 1, 0, 0, 0, 0, 0}, // 6
    {0, 0, 0, 1, 1, 1, 1}, // 7
    {0, 0, 0, 0, 0, 0, 0}, // 8
    {0, 0, 0, 0, 1, 0, 0}, // 9
    {1, 1, 1, 1, 1, 1, 1} // OFF
};

void setHorizontal(int num) {
    int index = (num >= 0 && num <= 9) ? num : 10;

    HAL_GPIO_WritePin(SEG_a_GPIO_Port, SEG_a_Pin, pinArr[index][0]);
    HAL_GPIO_WritePin(SEG_b_GPIO_Port, SEG_b_Pin, pinArr[index][1]);
    HAL_GPIO_WritePin(SEG_c_GPIO_Port, SEG_c_Pin, pinArr[index][2]);
    HAL_GPIO_WritePin(SEG_d_GPIO_Port, SEG_d_Pin, pinArr[index][3]);
    HAL_GPIO_WritePin(SEG_e_GPIO_Port, SEG_e_Pin, pinArr[index][4]);
    HAL_GPIO_WritePin(SEG_f_GPIO_Port, SEG_f_Pin, pinArr[index][5]);
    HAL_GPIO_WritePin(SEG_g_GPIO_Port, SEG_g_Pin, pinArr[index][6]);
}

void setVertical(int num) {
    int index = (num >= 0 && num <= 9) ? num : 10;

    HAL_GPIO_WritePin(SEG_a1_GPIO_Port, SEG_a1_Pin, pinArr[index][0]);
    HAL_GPIO_WritePin(SEG_b1_GPIO_Port, SEG_b1_Pin, pinArr[index][1]);
    HAL_GPIO_WritePin(SEG_c1_GPIO_Port, SEG_c1_Pin, pinArr[index][2]);
    HAL_GPIO_WritePin(SEG_d1_GPIO_Port, SEG_d1_Pin, pinArr[index][3]);
    HAL_GPIO_WritePin(SEG_e1_GPIO_Port, SEG_e1_Pin, pinArr[index][4]);
    HAL_GPIO_WritePin(SEG_f1_GPIO_Port, SEG_f1_Pin, pinArr[index][5]);
    HAL_GPIO_WritePin(SEG_g1_GPIO_Port, SEG_g1_Pin, pinArr[index][6]);
}

void scanSegment(int state) {
    switch (state % 2) {
    case 0:
        HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, RESET);
        HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, SET);
        HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin, RESET);
        HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin, SET);

        setHorizontal(segment_buffer[0]);
        setVertical(segment_buffer[2]);
        break;

    case 1:
        HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, SET);
        HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, RESET);
        HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin, SET);
        HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin, RESET);

        setHorizontal(segment_buffer[1]);
        setVertical(segment_buffer[3]);
        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

```h
#ifndef INC_INPUT_PROCESSING_H_
#define INC_INPUT_PROCESSING_H_

#include "global.h"

void fsm_for_input_processing(void);

#endif /* INC_INPUT_PROCESSING_H_ */
```

- input_processing.c

```c
#include "input_processing.h"

uint8_t systemMode = 0;
int temp_duration = 0;
static int auto_increment_timer = 0;

enum ButtonState { BUTTON_RELEASED, BUTTON_PRESSED, BUTTON_PRESSED_MORE_THAN_1_SECOND };
enum ButtonState button1State = BUTTON_RELEASED;

uint8_t getSystemMode(void) {
    return systemMode;
}

int is_button_just_pressed(int button_index) {
    static GPIO_PinState prev_state[3] = {SET, SET, SET};

    int is_pressed_now = is_button_pressed(button_index);

    if (is_pressed_now && (prev_state[button_index] == GPIO_PIN_SET)) {
        prev_state[button_index] = RESET;
        return 1;
    }

```

```
    if (!is_pressed_now) {
        prev_state[button_index] = SET;
    }

    return 0;
}

void fsm_for_input_processing(void) {
  // Button 1: changing mode
    if (is_button_just_pressed(0)) {
        systemMode = (systemMode + 1) % 4;

        switch(systemMode) {
            case 1:
                temp_duration = DURATION_RED_DEFAULT;
                break;
            case 2:
                temp_duration = DURATION_YELLOW_DEFAULT;
                break;
            case 3:
                temp_duration = DURATION_GREEN_DEFAULT;
                break;
            case 0:
                resetLEDsAndDisplay();
                break;
        }
    }
    // Button 2: changing duration
    if (systemMode > 0) {
        switch(button1State) {
            case BUTTON_RELEASED:
                if (is_button_just_pressed(1)) {
                    button1State = BUTTON_PRESSED;
                    temp_duration++;
                    if (temp_duration > 99) {
                        temp_duration = 1;
                    }
                }
                break;
            case BUTTON_PRESSED:
                if (!is_button_pressed(1)) {
                    button1State = BUTTON_RELEASED;
                } else if (is_button_pressed_1s(1)) {
                    button1State = BUTTON_PRESSED_MORE_THAN_1_SECOND;
                }
                break;
            // Increase value every 500ms (0.5s) after holding button > 1s
            case BUTTON_PRESSED_MORE_THAN_1_SECOND:
               if (!is_button_pressed(1)) {
                  button1State = BUTTON_RELEASED;
               } else {
                  if (auto_increment_timer > 0) {
                     auto_increment_timer--;
```

```
                }

                if (auto_increment_timer == 0) {
                   temp_duration++;
                if (temp_duration > 99) {
                   temp_duration = 1;
                }
                auto_increment_timer = 50;
                }
             }
              break;
        }
        //Button 3: confirm
        if (is_button_just_pressed(2)) {
            switch(systemMode) {
                case 1:
                    DURATION_RED_DEFAULT = temp_duration;
                    break;
                case 2:
                    DURATION_YELLOW_DEFAULT = temp_duration;
                    break;
                case 3:
                    DURATION_GREEN_DEFAULT = temp_duration;
                    break;
            }
            systemMode = 0;
            resetLEDsAndDisplay();
        }
    }
}
```

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

```
#ifndef INC_TRAFFIC_H_
#define INC_TRAFFIC_H_

#include "global.h"

void resetLEDsAndDisplay();
void completeTraffic_light();

#endif /* INC_TRAFFIC_H_ */
```

- traffic.c

```
#include "global.h"

static uint8_t ledState = 0;
static uint8_t trafficState = 0;
static uint8_t is_initialized = 0;
static uint8_t is_blink_timer_set = 0;

static int countdown1 = 0;
static int countdown2 = 0;

void resetLEDsAndDisplay(void) {
    HAL_GPIO_WritePin(GPIOA,
        RED_1_Pin|YEL_1_Pin|GRE_1_Pin|RED_2_Pin|YEL_2_Pin|GRE_2_Pin, RESET);
    updateSegment(10, 10, 10, 10);
    ledState = 0;
    is_initialized = 0;
    is_blink_timer_set = 0;
    set1(1000);
}
```

```
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,
                           RESET);
44                     if (countdown1 < 1) {
45                         trafficState = 1;
46                         countdown1 = DURATION_YELLOW_DEFAULT;
47                     }
48                     break;
49                 case 1: // State 1: Yellow horizontal, Red vertical
50                     HAL_GPIO_WritePin(GPIOA, YEL_1_Pin, SET);
51                     HAL_GPIO_WritePin(GPIOA, RED_2_Pin, SET);
52                     HAL_GPIO_WritePin(GPIOA, RED_1_Pin|GRE_1_Pin|YEL_2_Pin|GRE_2_Pin,
                           RESET);
53                     if (countdown1 < 1) {
54                         trafficState = 2;
55                         countdown1 = DURATION_RED_DEFAULT;
56                         countdown2 = DURATION_GREEN_DEFAULT;
57                     }
58                     break;
59                 case 2: // State 2: Red horizontal, Green vertical
60                     HAL_GPIO_WritePin(GPIOA, RED_1_Pin, SET);
61                     HAL_GPIO_WritePin(GPIOA, GRE_2_Pin, SET);
62                     HAL_GPIO_WritePin(GPIOA, YEL_1_Pin|GRE_1_Pin|RED_2_Pin|YEL_2_Pin,
                           RESET);
63                     if (countdown2 < 1) {
64                         trafficState = 3;
65                         countdown2 = DURATION_YELLOW_DEFAULT;
66                     }
67                     break;
68                 case 3: // State 3: Red horizontal, Yellow vertical
```

```
                HAL_GPIO_WritePin(GPIOA, RED_1_Pin, SET);
                HAL_GPIO_WritePin(GPIOA, YEL_2_Pin, SET);
                HAL_GPIO_WritePin(GPIOA, YEL_1_Pin|GRE_1_Pin|RED_2_Pin|GRE_2_Pin,
                    RESET);
                if (countdown2 < 1) {
                    trafficState = 0;
                    countdown1 = DURATION_GREEN_DEFAULT;
                    countdown2 = DURATION_RED_DEFAULT;
                }
                break;
            }
        }
    } else {
        is_initialized = 0;

        if (!is_blink_timer_set) {
            set3(250);
            is_blink_timer_set = 1;
        }
        if (isTimer3Expired()) {
            set3(250);
            ledState = !ledState;
        }
        // Select Mode 2, 3, 4
        if (mode == 1) {
            updateSegment(0, 2, temp_duration / 10, temp_duration % 10);
            HAL_GPIO_WritePin(GPIOA, RED_1_Pin | RED_2_Pin, ledState);
            HAL_GPIO_WritePin(GPIOA, YEL_1_Pin | YEL_2_Pin | GRE_1_Pin | GRE_2_Pin,
                RESET);
        } else if (mode == 2) {
            updateSegment(0, 3, temp_duration / 10, temp_duration % 10);
            HAL_GPIO_WritePin(GPIOA, YEL_1_Pin | YEL_2_Pin, ledState);
            HAL_GPIO_WritePin(GPIOA, RED_1_Pin | RED_2_Pin | GRE_1_Pin | GRE_2_Pin,
                RESET);
        } else if (mode == 3) {
            updateSegment(0, 4, temp_duration / 10, temp_duration % 10);
            HAL_GPIO_WritePin(GPIOA, GRE_1_Pin | GRE_2_Pin, ledState);
            HAL_GPIO_WritePin(GPIOA, RED_1_Pin | RED_2_Pin | YEL_1_Pin | YEL_2_Pin,
                RESET);
        }
    }
}
```

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

```
#ifndef INC_GLOBAL_H_
#define INC_GLOBAL_H_

#include "main.h"
#include "input_reading.h"
#include "input_processing.h"
#include "led_display.h"
#include "timer.h"
#include "traffic.h"

extern int DURATION_RED_DEFAULT;
extern int DURATION_YELLOW_DEFAULT;
extern int DURATION_GREEN_DEFAULT;

extern int temp_duration;

uint8_t getSystemMode(void);
void completeTraffic_light(void);

#endif /* INC_GLOBAL_H_ */
```

- global.c

```
#include "global.h"

// INPUT

int DURATION_RED_DEFAULT = 5 - 1;
int DURATION_YELLOW_DEFAULT = 2;
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