VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
Faculty of Computer Science and Engineering

CC02 — Lab Report

Microprocessor - Microcontroller
Lab 3

Supervisors: Nguyen Thien An
Students: Nguyen Minh Dang 2252154

Ho Chi Minh City, November 8, 2024

Contents

1 Exercise

The schematic files and source code for each exercise lab are in this GitHub repository (included PNG, PDSPRJ and C) here or in this link: https://github.com/dangalpha78/Workspace-for-Microprocessor---Microcontroller/tree/main/Lab3. Also the link for simulation video is here simulation video.

The schematic for this lab:

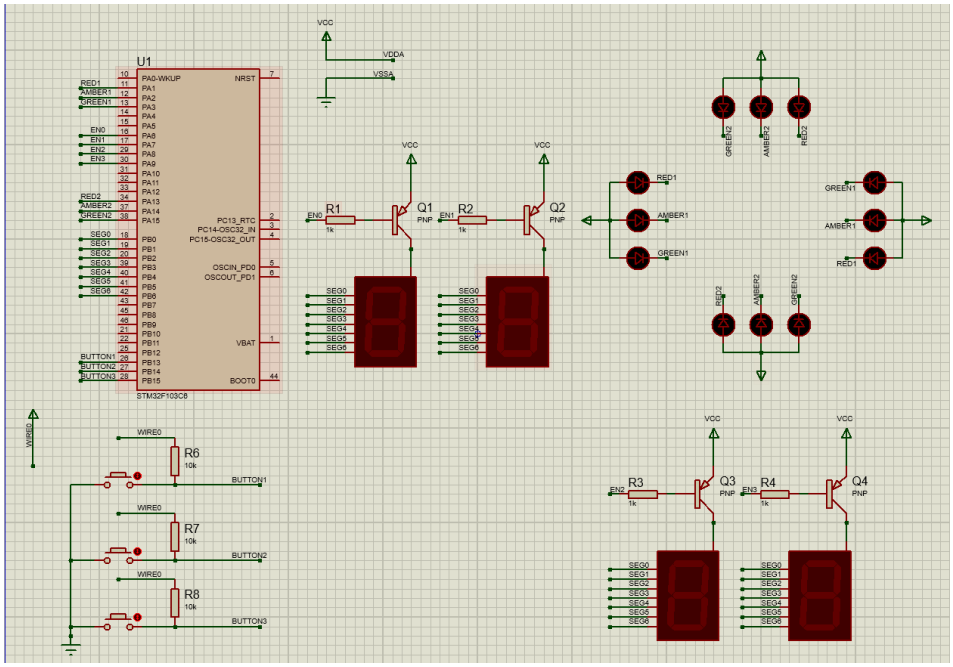

Figure 1: *The schematic for lab 3.*

Annotations:

- RED1, AMBER1, GREEN1 are LEDs belong to road 1; RED2, AMBER2, GREEN2 are LEDs belong to road 2.
- Mode 1 (Normal mode/Auto mode): Q1, Q2 are LEDs representing the time of road 1; Q3, Q4 are LEDs representing the time of road 1
- Mode 2, 3, 4 (Manual mode): Q1, Q2 are LEDs representing the timing of the red, amber, or green lights in sequence for both roads; Q3, Q4 are LEDs representing the current active mode.
- All buttons only has PRESS mode (**no function HOLD**).
- BUTTON1 is used to switch the current mode from 1 to 2 to 3 to 4, and then back to 1.
- BUTTON2 is used to increase the time value in modes 2, 3, 4 (no function in mode 1).
- BUTTON3 is used to set the time value in modes 2, 3, 4 (no function in mode 1). **If BUTTON3 is not pressed, the changed value will not be saved**.

1.1 Exercise 1: Sketch an FSM

The FSM schematic for this lab:

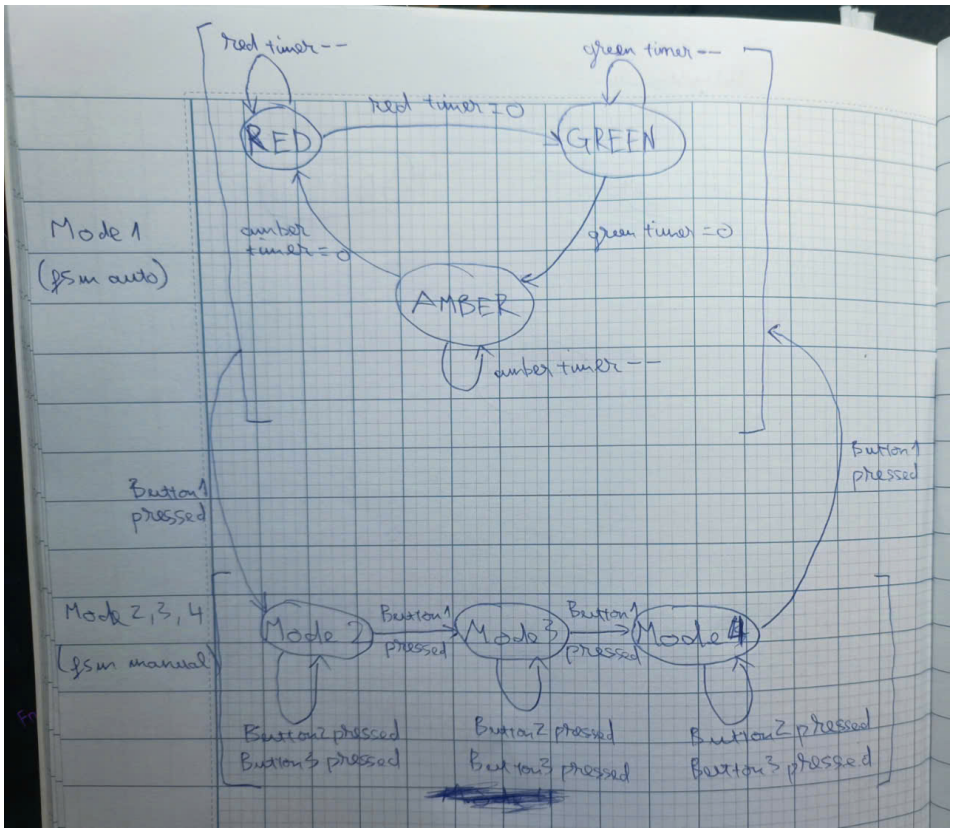

Figure 2: *The schematic for FSM of lab 3.*

Explanation:

• In this lab, I have divided the system into two FSMs: the auto FSM and the manual FSM. The auto FSM corresponds to mode 1, which is the normal mode, while the manual FSM corresponds to the modes for adjusting the timers of the lights (or more precisely, adjusting the counters). In this system, mode 2 is for adjusting the red light, mode 3 for the yellow light, and mode 4 for the green light.

• In mode 1, there are 3 states for road 1 and 3 states for road 2. These 3 states represent the light status (e.g., the red state means only the red light is on), and they automatically transition to another state when the individual timer for each light reaches zero. As shown in the diagram, the red light state will change to the green light state when the timer expires, then to amber, and then back to red. Since both roads have the same states (each with 3 states and independent timers for red, yellow, and green lights), I only illustrate the 3 states for the mode 1.

• However, in reality, the states should be labeled as RED1, AMBER1, GREEN1, RED2, AMBER2, GREEN2, and they run in parallel (the two FSMs run in parallel). For example, when

road 1 is in the red light state, road 2 is in the green light state, and so on. Both FSMs are completely independent of each other but run in parallel, differing only in the starting time of each FSM (road 1 starts with the red light, road 2 starts with the green light, but eventually, they both follow the red-amber-green cycle). Due to these similarities, I have only illustrated the 3 states for mode 1.

• In manual mode, which is the mode for adjusting the light counters, we can use button 1 to switch between modes (mode 1 $\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow$ 4, then back to mode 1). Buttons 2 and 3 have other functions, but they do not affect the states in the manual FSM, so pressing them will not cause a transition to another state.

1.2 Exercise 2: Proteus Schematic

The schematic is already shown above:

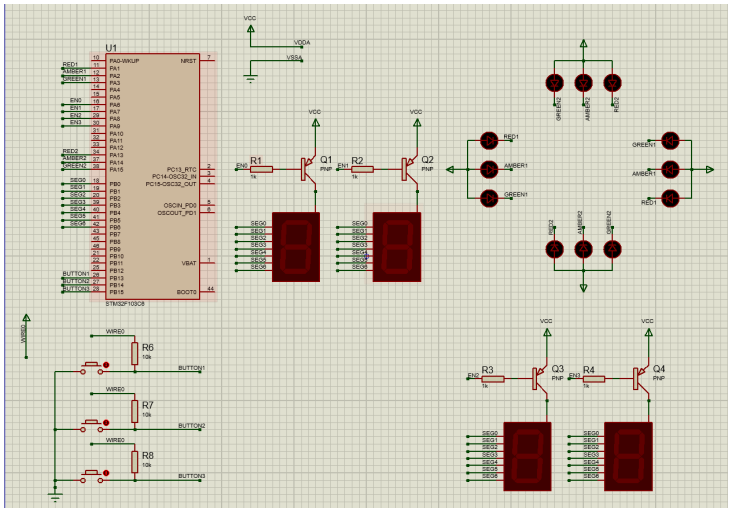

Figure 3: *The schematic for lab 3.*

Annotations:

- RED1, AMBER1, GREEN1 are LEDs belong to road 1; RED2, AMBER2, GREEN2 are LEDs belong to road 2.
- Mode 1 (Normal mode/Auto mode): Q1, Q2 are LEDs representing the time of road 1; Q3, Q4 are LEDs representing the time of road 1
- Mode 2, 3, 4 (Manual mode): Q1, Q2 are LEDs representing the timing of the red, amber, or green lights in sequence for both roads; Q3, Q4 are LEDs representing the current active mode.
- All buttons only has PRESS mode (**no function HOLD**).
- BUTTON1 is used to switch the current mode from 1 to 2 to 3 to 4, and then back to 1.
- BUTTON2 is used to increase the time value in modes 2, 3, 4 (no function in mode 1).
- BUTTON3 is used to set the time value in modes 2, 3, 4 (no function in mode 1). **If BUTTON3 is not pressed, the changed value will not be saved**.

1.3 Exercise 3: Create STM32 Project

The schematic of pins:

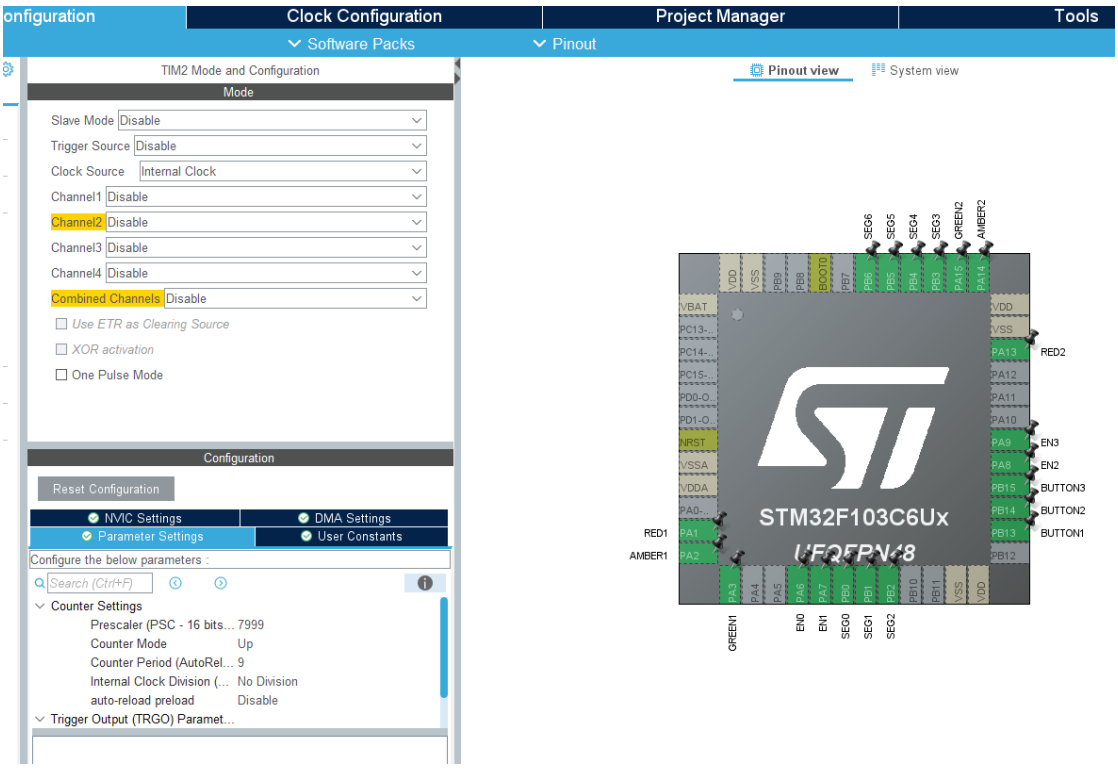

Figure 4: *The schematic of pins.*

File Overview:

- **button:** This file provides functions for debouncing and reading the states of multiple buttons, including detecting long presses exceeding one second.
- **fsm_auto:** This file implements the automatic finite state machine (FSM) for controlling LED states based on timers in mode 1.
- **fsm_manual:** This file implements the manual finite state machine (FSM) for handling button inputs, including state transitions and actions based on button presses.
- **global:** This file defines global variables, constants, and enumerations related to timers, LED states, and modes, and provides a function to set timer durations.
- **led_7segment:** This file provides functions to enable specific modes by controlling GPIO pins and display numbers on a 7-segment display based on the input number.
- **led_control:** This file manages timers and controls the LED lights for a traffic light system, including countdowns for red, amber, and green light states, as well as updating the values displayed on a 7-segment display.

- **led_traffic_light:** This file controls the LED traffic lights for two roads, turning the red, amber, and green LEDs on or off based on the traffic light state for each road.
- **mode_feature:** This file contains functions to manage and control different modes of a traffic light system, including updating LED states, handling button inputs, and controlling the 7-segment display, with the logic for mode transitions and timer-based actions.
- **mode_feature:** This file contains functions to manage and control different modes of a traffic light system, including updating LED states, handling button inputs, and controlling the 7-segment display, with the logic for mode transitions and timer-based actions.
- **software_timer:** This file contains the callback function for the timer interrupt, handling periodic timer events. It reads button states, manages traffic light timers for different modes (MODE1, MODE2, MODE3, MODE4), and updates the 7-segment display based on the current mode.

Add new files:

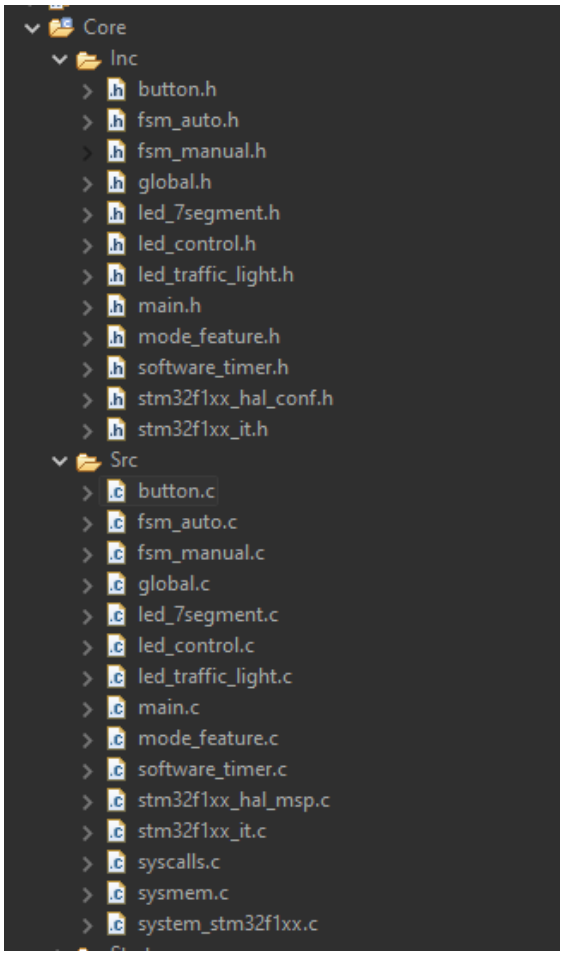

Figure 5: *Adding new files.*

1.4 Global and Initial declaration

1.4.1 global.h

This header file is used to declare definitions for the quantities of `TIMER_CYCLE`, `NO_OF_TIMERS`, and `NO_OF_LED_7SEGMENT`. It also contains the initial values for the traffic lights and the scan frequency for the 4-digit display. The file declares arrays and functions that will be used across different files. Additionally, it includes an enum for the states of the traffic light colors and mode statuses. Lastly, there is a variable `addValue`, which is used to store the incremented value when button 2 is pressed.

```
#ifndef INC_GLOBAL_H_
#define INC_GLOBAL_H_

#define TIMER_CYCLE 10
#define NO_OF_TIMERS 8
#define NO_LED_7SEGMENT 4
#define INITIAL_RED_LED_TIME 5
#define INITIAL_AMBER_LED_TIME 2
#define INITIAL_GREEN_LED_TIME 3
#define COUNTER_4DIGITS_LED 25 //2 hz

#define TIME_SCALE_COUNTER 100
#define TIME_SCALE_MS 1000

extern int timer_counter[NO_OF_TIMERS];
extern int timer_flag[NO_OF_TIMERS];
extern int led_buffer[NO_LED_7SEGMENT];

void setTimer(unsigned char index, int duration);

enum LedColorState{RED_LED, AMBER_LED, GREEN_LED};
extern enum LedColorState ledColorStateRoad1;
extern enum LedColorState ledColorStateRoad2;

extern int RED_LED_COUNTER;
extern int AMBER_LED_COUNTER;
extern int GREEN_LED_COUNTER;

extern int DIGITS_LED_COUNTER;

enum ModeState{MODE1, MODE2, MODE3, MODE4, MODE0};
extern enum ModeState currentMode;

extern int addValue;

#endif /* INC_GLOBAL_H_ */
```

1.4.2 global.c

In `global.c`, missing configurations from `global.h` are added, such as initializing counters for each traffic light, setting the scan frequency for the 4-digit display, and defining the initial states of the traffic lights, modes, and the `addValue` variable (set to 0).

```
#include "global.h"

int timer_counter[NO_OF_TIMERS] = {
  INITIAL_RED_LED_TIME * TIME_SCALE_COUNTER,
  INITIAL_AMBER_LED_TIME * TIME_SCALE_COUNTER,
  INITIAL_GREEN_LED_TIME  * TIME_SCALE_COUNTER,
  INITIAL_RED_LED_TIME * TIME_SCALE_COUNTER,
  INITIAL_AMBER_LED_TIME * TIME_SCALE_COUNTER,
  INITIAL_GREEN_LED_TIME * TIME_SCALE_COUNTER,
  COUNTER_4DIGITS_LED,
  COUNTER_4DIGITS_LED
};
int timer_flag[NO_OF_TIMERS] = {0};  // assuming all flags start at 0
int led_buffer[NO_LED_7SEGMENT] = {0, 0, 0, 0};

void setTimer(unsigned char index, int duration) {
  timer_counter[index] = duration / TIMER_CYCLE ;
  timer_flag[index] = 0;
}

enum LedColorState ledColorStateRoad1 = RED_LED; //led light state of road 1
enum LedColorState ledColorStateRoad2 = GREEN_LED; //led light state of road 2

int RED_LED_COUNTER = INITIAL_RED_LED_TIME * TIME_SCALE_MS;    //global variable
    red light counter
int AMBER_LED_COUNTER = INITIAL_AMBER_LED_TIME * TIME_SCALE_MS;  //global variable
     amber light counter
int GREEN_LED_COUNTER = INITIAL_GREEN_LED_TIME * TIME_SCALE_MS;  //global variable
     green light counter


int DIGITS_LED_COUNTER = 250;

enum ModeState currentMode = MODE0; //initial mode state (indefinite)

int addValue = 0; //handle the times click button 2
```

1.5 Button Debouncing

This part will handle the input from the buttons, processing noise by using debounce buffers, namely `debounceButtonBuffer1` and `debounceButtonBuffer2`. Additionally, a `counterForButtonPress1s` is used to manage the case where a button is held down.

```
#include "main.h"
// we aim to work with more than one buttons
#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

// the buffer that the final result is stored after debouncing
static GPIO_PinState buttonBuffer[NO_OF_BUTTONS];
// we define two buffers for debouncing
static GPIO_PinState debounceButtonBuffer1[NO_OF_BUTTONS];
static GPIO_PinState debounceButtonBuffer2[NO_OF_BUTTONS];
// we define a flag for a button pressed more than 1 second .
static uint8_t flagForButtonPress1s[NO_OF_BUTTONS];
// we define counter for automatically increasing the value
// after the button is pressed more than 1 second .
static uint16_t counterForButtonPress1s[NO_OF_BUTTONS];

void debounceButtonBuffer1_pin(void){
    debounceButtonBuffer1[0] = HAL_GPIO_ReadPin(BUTTON1_GPIO_Port, BUTTON1_Pin);
    debounceButtonBuffer1[1] = HAL_GPIO_ReadPin(BUTTON2_GPIO_Port, BUTTON2_Pin);
    debounceButtonBuffer1[2] = HAL_GPIO_ReadPin(BUTTON3_GPIO_Port, BUTTON3_Pin);
}

void button_reading(void) {
    for (uint8_t i = 0; i < NO_OF_BUTTONS ; i++) {
        debounceButtonBuffer2[i] = debounceButtonBuffer1[i];
        debounceButtonBuffer1_pin();
        if(debounceButtonBuffer1[i] == debounceButtonBuffer2[i])
            buttonBuffer[i] = debounceButtonBuffer1[i];

        if(buttonBuffer[i] == BUTTON_IS_PRESSED){
          // if a button is pressed , we start counting
          if(counterForButtonPress1s[i] < DURATION_FOR_AUTO_INCREASING){
              counterForButtonPress1s[i]++;
          }
          else {
            // the flag is turned on when 1 second has passed
            // since the button is pressed .
            flagForButtonPress1s[i] = 1;
            // todo
          }
        }

```

```
45          else {
46            counterForButtonPress1s[i] = 0;
47            flagForButtonPress1s[i] = 0;
48          }
49      }
50  }
51
52  unsigned char is_button_pressed(uint8_t index) {
53    if(index >= NO_OF_BUTTONS) return 0;
54    return (buttonBuffer[index] == BUTTON_IS_PRESSED);
55  }
56
57  unsigned char is_button_pressed_1s(unsigned char index) {
58    if(index >= NO_OF_BUTTONS ) return 0xff ;
59    return (flagForButtonPress1s[index] == 1) ;
60  }
```

1.6 FSM and Mode feature

1.6.1 FSM Auto

The FSM Auto is configured to automatically switch the state of the lights, where the light states of the two roads are independent but operate in parallel.

```
#include "main.h"
#include "global.h"
#include "led_control.h"

void fsm_auto(void) {
  if (currentMode == MODE1){
    switch(ledColorStateRoad1){
        case RED_LED:
            if(timer_flag_on(0)) {
              ledColorStateRoad1 = GREEN_LED;
            }
            ledOn(0);
            break;
        case AMBER_LED:
            if (timer_flag_on(1)) {
              ledColorStateRoad1 = RED_LED;
            }
            ledOn(1);
            break;
        case GREEN_LED:
          if (timer_flag_on(2)) {
            ledColorStateRoad1 = AMBER_LED;
          }
          ledOn(2);
            break;
    }

    switch(ledColorStateRoad2){
        case RED_LED:
            if(timer_flag_on(3)) {
              ledColorStateRoad2 = GREEN_LED;
            }
            ledOn(3);
            break;
        case AMBER_LED:
            if (timer_flag_on(4)) {
              ledColorStateRoad2 = RED_LED;
            }
            ledOn(4);
            break;
        case GREEN_LED:
          if (timer_flag_on(5)) {
            ledColorStateRoad2 = AMBER_LED;
```

```
            }
            ledOn(5);
              break;
    }
}
}
```

1.6.2 FSM Manual

FSM Manual is used when we want to adjust the `timer_counter` of the lights in modes 2, 3, and 4. The `currentModeRun()` function determines the current mode state; if it is mode 1, it switches to FSM Auto, while modes 2, 3, and 4 operate in FSM Manual. The button states function independently: button 1 has the `changModeState()` function to switch states when pressed, button 2 increments the value using `addValue`, and button 3 calls `setValue()` to update with a new value.

```
#include "main.h"
#include "button.h"
#include "mode_feature.h"
#include "global.h"

enum ButtonState{BUTTON_RELEASED, BUTTON_PRESSED,
    BUTTON_PRESSED_MORE_THAN_1_SECOND};
enum ButtonState buttonState1 = BUTTON_RELEASED;
enum ButtonState buttonState2 = BUTTON_RELEASED;
enum ButtonState buttonState3 = BUTTON_RELEASED;

void fsm_manual(void) {
  currentModeRun();
    switch(buttonState1){
        case BUTTON_RELEASED:
            if(is_button_pressed(0)) {
                buttonState1 = BUTTON_PRESSED;
                // INCREASE VALUE OF PORT A BY ONE UNIT
                changeModeState();
            }
            break;
        case BUTTON_PRESSED:
            if (!is_button_pressed(0)) {
                buttonState1 = BUTTON_RELEASED;
            } else {
                if(is_button_pressed_1s(0)) {
                    buttonState1 = BUTTON_PRESSED_MORE_THAN_1_SECOND;
                }
            }
            break;
        case BUTTON_PRESSED_MORE_THAN_1_SECOND:
            if (!is_button_pressed(0)) {
```

```
                buttonState1 = BUTTON_RELEASED;
            }
            // todo
            break;
    }

    switch(buttonState2){
        case BUTTON_RELEASED:
            if(is_button_pressed(1)) {
                buttonState2 = BUTTON_PRESSED;
                // INCREASE VALUE OF PORT A BY ONE UNIT
                addValue++;
            }
            break;
        case BUTTON_PRESSED:
            if (!is_button_pressed(1)) {
                buttonState2 = BUTTON_RELEASED;
            } else {
                if(is_button_pressed_1s(1)) {
                    buttonState2 = BUTTON_PRESSED_MORE_THAN_1_SECOND;
                }
            }
            break;
        case BUTTON_PRESSED_MORE_THAN_1_SECOND:
            if (!is_button_pressed(1)) {
                buttonState2 = BUTTON_RELEASED;
            }
            // todo
            break;
    }

    switch(buttonState3){
        case BUTTON_RELEASED:
            if(is_button_pressed(2)) {
                buttonState3 = BUTTON_PRESSED;
                // INCREASE VALUE OF PORT A BY ONE UNIT
                //increaseTime();
                setValue();
            }
            break;
        case BUTTON_PRESSED:
            if (!is_button_pressed(2)) {
                buttonState3 = BUTTON_RELEASED;
            } else {
                if(is_button_pressed_1s(2)) {
                    buttonState3 = BUTTON_PRESSED_MORE_THAN_1_SECOND;
                }
            }

```

```
            break;
        case BUTTON_PRESSED_MORE_THAN_1_SECOND:
            if (!is_button_pressed(2)) {
                buttonState3 = BUTTON_RELEASED;
            }
            // todo
            break;
  }
}
```

1.6.3 Mode Feature (Mode 2, 3, 4)

In Mode Feature, there are functions related to the actions triggered when buttons are pressed, as well as auxiliary functions and updates to the array displaying numbers on the 4-digit 7-segment display.

```
#include "main.h"
#include "global.h"
#include "fsm_auto.h"
#include "led_7segment.h"
#include "button.h"

#define MAX_VALUE 99

void ledOff(void){
  HAL_GPIO_WritePin(RED1_GPIO_Port, RED1_Pin, SET);
  HAL_GPIO_WritePin(AMBER1_GPIO_Port, AMBER1_Pin, SET);
  HAL_GPIO_WritePin(GREEN1_GPIO_Port, GREEN1_Pin, SET);
  HAL_GPIO_WritePin(RED2_GPIO_Port, RED2_Pin, SET);
  HAL_GPIO_WritePin(AMBER2_GPIO_Port, AMBER2_Pin, SET);
  HAL_GPIO_WritePin(GREEN2_GPIO_Port, GREEN2_Pin, SET);
}

void button_2_value(int newValue){
  led_buffer[0] = 0;

  switch(currentMode){
  case MODE2:
    led_buffer[1] = 2;
    break;
  case MODE3:
    led_buffer[1] = 3;
    break;
  case MODE4:
    led_buffer[1] = 4;
    break;
  default:
    break;
  }
```

```
34    newValue = newValue / TIME_SCALE_MS;
35    led_buffer[2] = newValue / 10;
36    led_buffer[3] = newValue % 10;
37  }
38
39  void changeModeState(void){
40    setTimer(7, DIGITS_LED_COUNTER);
41    addValue = 0;
42    switch(currentMode){
43    case MODE1:
44      ledOff();
45      currentMode = MODE2;
46      break;
47    case MODE2:
48      ledOff();
49      currentMode = MODE3;
50      break;
51    case MODE3:
52      ledOff();
53      currentMode = MODE4;
54      break;
55    case MODE4:
56      ledOff();
57      currentMode = MODE1;
58      break;
59    default:
60      ledOff();
61      currentMode = MODE1;
62      break;
63    }
64  }
65
66  void timer_run_mode(void){
67    if (timer_counter[7] > 0){
68      timer_counter[7]--;
69      if (timer_counter[7] <= 0) timer_flag[7] = 1;
70    }
71  }
72
73  unsigned char timer_mode_flag_on(uint8_t mode){
74    if (mode >= NO_OF_TIMERS) return 0;
75    if (timer_flag[mode] == 1){
76      setTimer(7, DIGITS_LED_COUNTER);
77      return 1;
78    }
79    return 0;
80  }
81
82
```

```
83  void mode2State(void){
84    if (timer_mode_flag_on(7) && currentMode == MODE2){
85      HAL_GPIO_TogglePin(RED1_GPIO_Port, RED1_Pin);
86      HAL_GPIO_TogglePin(RED2_GPIO_Port, RED2_Pin);
87    }
88  }
89
90  void mode3State(void){
91    if (timer_mode_flag_on(7) && currentMode == MODE3){
92      HAL_GPIO_TogglePin(AMBER1_GPIO_Port, AMBER1_Pin);
93      HAL_GPIO_TogglePin(AMBER2_GPIO_Port, AMBER2_Pin);
94    }
95  }
96
97  void mode4State(void){
98    if (timer_mode_flag_on(7) && currentMode == MODE4){
99      HAL_GPIO_TogglePin(GREEN1_GPIO_Port, GREEN1_Pin);
100     HAL_GPIO_TogglePin(GREEN2_GPIO_Port, GREEN2_Pin);
101   }
102 }
103
104 void currentModeRun(void){
105   switch(currentMode){
106   case MODE2:
107     mode2State();
108     break;
109   case MODE3:
110     mode3State();
111     break;
112   case MODE4:
113     mode4State();
114     break;
115   default:
116     break;
117   }
118 }
119
120 int led_enable_mode = 0;
121
122 void update7SEG_mode(uint8_t enable) {
123     enableMode(enable);
124     display7SEG(led_buffer[enable]);
125 }
126
127 void button_2_changed_value(void){
128   int newValue = 0;
129   int maxAdd = 0;
130   switch(currentMode){
131   case MODE2:
```

```
132     newValue = RED_LED_COUNTER + addValue * TIME_SCALE_MS;
133     maxAdd = MAX_VALUE - RED_LED_COUNTER / TIME_SCALE_MS;
134     break;
135   case MODE3:
136     newValue = AMBER_LED_COUNTER + addValue * TIME_SCALE_MS;
137     maxAdd = MAX_VALUE - AMBER_LED_COUNTER / TIME_SCALE_MS;
138     break;
139   case MODE4:
140     newValue = GREEN_LED_COUNTER + addValue * TIME_SCALE_MS;
141     maxAdd = MAX_VALUE - GREEN_LED_COUNTER / TIME_SCALE_MS;
142     break;
143   default:
144     break;
145   }
146 
147 
148   if (newValue > MAX_VALUE * TIME_SCALE_MS) {
149     addValue = maxAdd - MAX_VALUE + 1;
150     newValue = 1000;
151   }
152 
153   button_2_value(newValue);
154 }
155 
156 void setValue(void){
157   switch(currentMode){
158   case MODE2:
159     timer_counter[0] = timer_counter[0] + addValue * TIME_SCALE_COUNTER;
160     timer_counter[3] = timer_counter[3] + addValue * TIME_SCALE_COUNTER;
161     RED_LED_COUNTER = RED_LED_COUNTER + addValue * TIME_SCALE_MS;
162     break;
163   case MODE3:
164     timer_counter[1] = timer_counter[1] + addValue * TIME_SCALE_COUNTER;
165     timer_counter[4] = timer_counter[4] + addValue * TIME_SCALE_COUNTER;
166     AMBER_LED_COUNTER = AMBER_LED_COUNTER + addValue * TIME_SCALE_MS;
167     break;
168   case MODE4:
169     timer_counter[2] = timer_counter[2] + addValue * TIME_SCALE_COUNTER;
170     timer_counter[5] = timer_counter[5] + addValue * TIME_SCALE_COUNTER;
171     GREEN_LED_COUNTER = GREEN_LED_COUNTER + addValue * TIME_SCALE_MS;
172     break;
173   default:
174     break;
175   }
176   addValue = 0;
177 }
178 
179 void timer_run_4digits_mode(void){
180   if(timer_counter[6] > 0){
```

```
181     timer_counter[6]--;
182     if (timer_counter[6] <= 0) {
183       button_2_changed_value();
184       update7SEG_mode(led_enable_mode++);
185       if (led_enable_mode >= 4) led_enable_mode = 0;
186       setTimer(6, DIGITS_LED_COUNTER);
187     }
188   }
189 }
```

1.7 LEDs Control

1.7.1 7-segment 4 digits LEDs

The same as in previous labs 1 and 2:

```
void display7SEG(uint8_t num); //each led has cases 0 to 9
void enableMode(uint8_t en); //to turn on which led specify Ex: 0 - turn on led 1
```

1.7.2 Traffic lights LEDs

The same as in previous labs 1 and 2:

```
void ledOn(uint8_t led);
//turn on led
//Ex: 0 - led red road 1
//    1 - led amber road 1
//    2 - led green road 1
//    3 - led red road 2
//    4 - led amber road 2
//    5 - led green road 2
```

1.7.3 LEDs control in Mode 1

LEDs control is used to manage the counters of the traffic lights and update the time values for the 4-digit 7-segment display.

```
#include "main.h"
#include "led_traffic_light.h"
#include "software_timer.h"
#include "global.h"
#include "led_7segment.h"

void timer_run_road1(void){
  if(timer_counter[0] > 0 && ledColorStateRoad1 == RED_LED) {
    timer_counter[0]--;
    if(timer_counter[0] <= 0) timer_flag[0] = 1;
    updateLedBuffer(0, 1, timer_counter[0]);
  }
  if(timer_counter[1] > 0 && ledColorStateRoad1 == AMBER_LED) {
    timer_counter[1]--;
    if(timer_counter[1] <= 0) timer_flag[1] = 1;
    updateLedBuffer(0, 1, timer_counter[1]);
  }
  if(timer_counter[2] > 0 && ledColorStateRoad1 == GREEN_LED) {
    timer_counter[2]--;
    if(timer_counter[2] <= 0) timer_flag[2] = 1;
    updateLedBuffer(0, 1, timer_counter[2]);
  }
}
```

```
24
25 void timer_run_road2(void){
26   if(timer_counter[3] > 0 && ledColorStateRoad2 == RED_LED) {
27     timer_counter[3]--;
28     if(timer_counter[3] <= 0) timer_flag[3] = 1;
29     updateLedBuffer(2, 3, timer_counter[3]);
30   }
31   if(timer_counter[4] > 0 && ledColorStateRoad2 == AMBER_LED) {
32     timer_counter[4]--;
33     if(timer_counter[4] <= 0) timer_flag[4] = 1;
34     updateLedBuffer(2, 3, timer_counter[4]);
35   }
36   if(timer_counter[5] > 0 && ledColorStateRoad2 == GREEN_LED) {
37     timer_counter[5]--;
38     if(timer_counter[5] <= 0) timer_flag[5] = 1;
39     updateLedBuffer(2, 3, timer_counter[5]);
40   }
41 }
42
43 void resetTimer(uint8_t led){
44   switch(led){
45   case 0:
46     setTimer(led, RED_LED_COUNTER);
47     break;
48   case 1:
49     setTimer(led, AMBER_LED_COUNTER);
50     break;
51   case 2:
52     setTimer(led, GREEN_LED_COUNTER);
53     break;
54   case 3:
55     setTimer(led, RED_LED_COUNTER);
56     break;
57   case 4:
58     setTimer(led, AMBER_LED_COUNTER);
59     break;
60   case 5:
61     setTimer(led, GREEN_LED_COUNTER);
62     break;
63   }
64 }
65
66 unsigned char timer_flag_on(uint8_t led){
67   if (led >= NO_OF_TIMERS) return 0;
68   if (timer_flag[led] == 1){
69     resetTimer(led);
70     return 1;
71   }
72   return 0;
```

```
73 }
74
75 int led_enable = 0;
76
77 void updateLedBuffer(unsigned char enable0, unsigned char enable1, int value){
78   value = value / 100; //counter / 100
79   led_buffer[enable0] = value / 10;
80   led_buffer[enable1] = value % 10;
81 }
82
83 void update7SEG(uint8_t enable) {
84     enableMode(enable);
85     display7SEG(led_buffer[enable]);
86 }
87
88 void timer_run_4digits(void){
89   if(timer_counter[6] > 0){
90     timer_counter[6]--;
91     if (timer_counter[6] <= 0) {
92       update7SEG(led_enable++);
93       if (led_enable >= 4) led_enable = 0;
94       setTimer(6, DIGITS_LED_COUNTER);
95     }
96   }
97 }
```

1.8 Software Timer Interrupt and Main function

1.8.1 Timer Interrupt

The software timer includes the function `HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)`, which is called every 10ms. Within this function, `button_reading()` is executed every 10ms to read the button signals. Additionally, it handles the counters for the two road timers in mode 1 and the counter for the 4-digit 7-segment display in different modes. Since the counters for the two FSMs are different, two `if` statements are used to check the current mode and to call the appropriate `timer_run` functions as needed.

```
#include "main.h"
#include "button.h"
#include "led_control.h"
#include "global.h"
#include "mode_feature.h"

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef * htim)
{
    if(htim->Instance == TIM2){
        button_reading();

        if (currentMode == MODE1){
          timer_run_road1();

          timer_run_road2();

          timer_run_4digits();
        }

        if (currentMode == MODE2 || currentMode == MODE3 || currentMode == MODE4){
          timer_run_mode();
          //setTimer(6, DIGITS_LED_COUNTER);
          timer_run_4digits_mode();
        }

    }
}
```

1.8.2 Main

Finally, define two headers, `fsm_auto.h` and `fsm_manual.h`, in `main.c`. Call the function `HAL_TIM_Base_Start_IT` with the reference value `&htim2` in the main function. Combine this with running the two functions `fsm_auto()` and `fsm_manual()` in an infinite `while(1)` loop to complete the setup.

```
#include "main.h"
#include "fsm_auto.h"
```

```
#include "fsm_manual.h"

int main(void)
{
    HAL_TIM_Base_Start_IT(&htim2);
    while (1)
    {
    fsm_auto();
    fsm_manual();
    }
}
```

References