Ho Chi Minh city University of Technology
Computer Science and Engineering Faculty

Project – LAB3

Microcontroller – Microprocessor

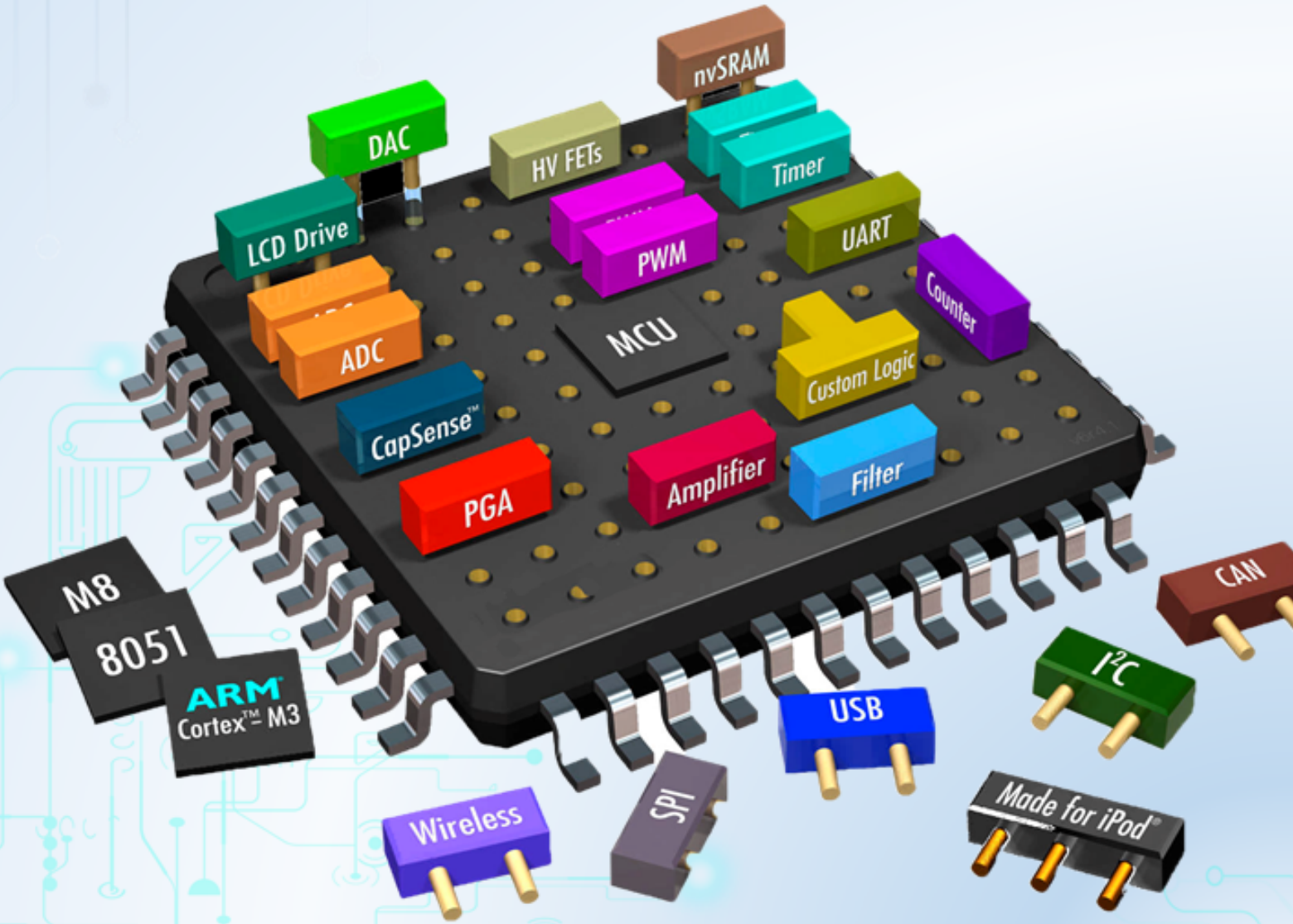
Instructor: Dr. Le Trong Nhan

Student: Nguyen Quoc Kiet - 1952802

Mục lục

CHƯƠNG 1

Buttons/Switches



GitHub Repository URL and Student Information:
https://github.com/codeKietne/LAB3

1 Objectives

In this lab, you will

- Learn how to add new C source files and C header files in an STM32 project,
- Learn how to read digital inputs and display values to LEDs using a timer interrupt of a microcontroller (MCU).
- Learn how to debounce when reading a button.
- Learn how to create an FSM and implement an FSM in an MCU.

2 Introduction

Embedded systems usually use buttons (or keys, or switches, or any form of mechanical contacts) as part of their user interface. This general rule applies from the most basic remote-control system for opening a garage door, right up to the most sophisticated aircraft autopilot system. Whatever the system you create, you need to be able to create a reliable button interface.

A button is generally hooked up to an MCU so as to generate a certain logic level when pushed or closed or “active" and the opposite logic level when unpushed or open or “inactive." The active logic level can be either ‘0’ or ‘1’, but for reasons both historical and electrical, an active level of ’0’ is more common.

We can use a button if we want to perform operations such as:

- Drive a motor while a switch is pressed.
- Switch on a light while a switch is pressed.
- Activate a pump while a switch is pressed.

These operations could be implemented using an electrical button without using an MCU; however, use of an MCU may well be appropriate if we require more complex behaviours. For example:

- Drive a motor while a switch is pressed.

  Condition: If the safety guard is not in place, don’t turn the motor. Instead sound a buzzer for 2 seconds.

- Switch on a light while a switch is pressed.

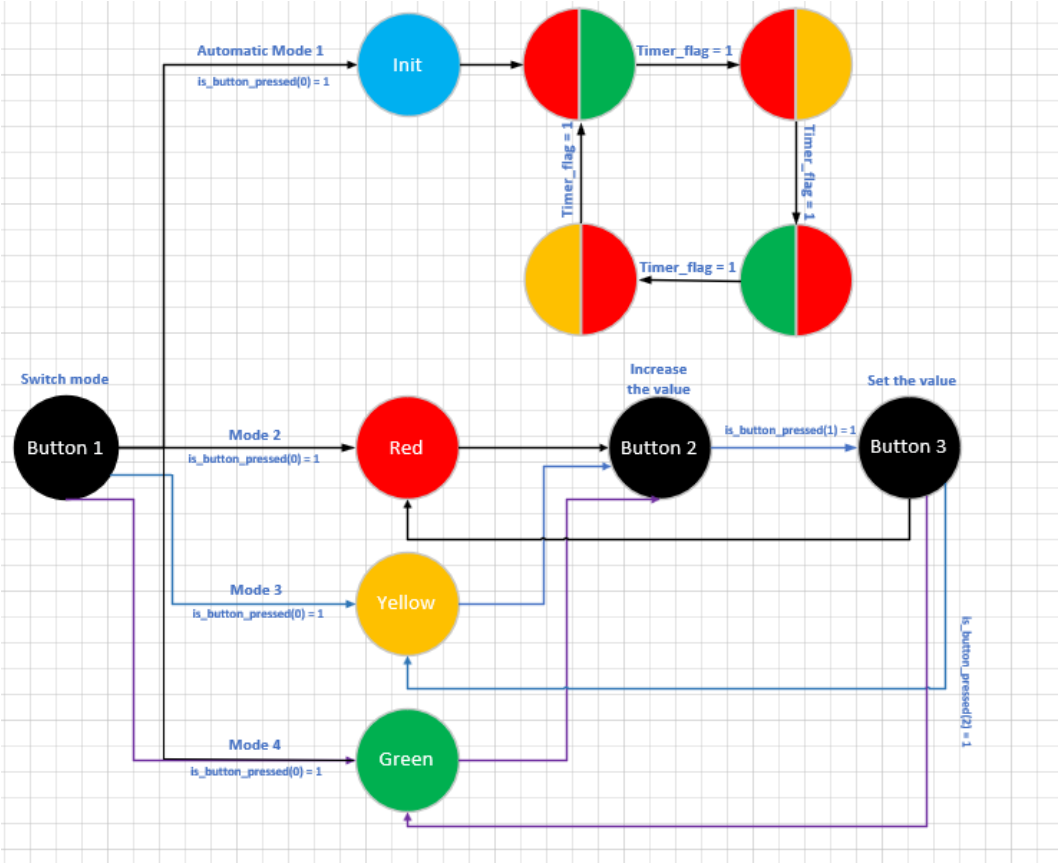
  Condition: To save power, ignore requests to turn on the light during daylight hours.

- Activate a pump while a switch is pressed

  Condition: If the main water reservoir is below 300 litres, do not start the main pump: instead, start the reserve pump and draw the water from the emergency tank.

2.1 Exercise1

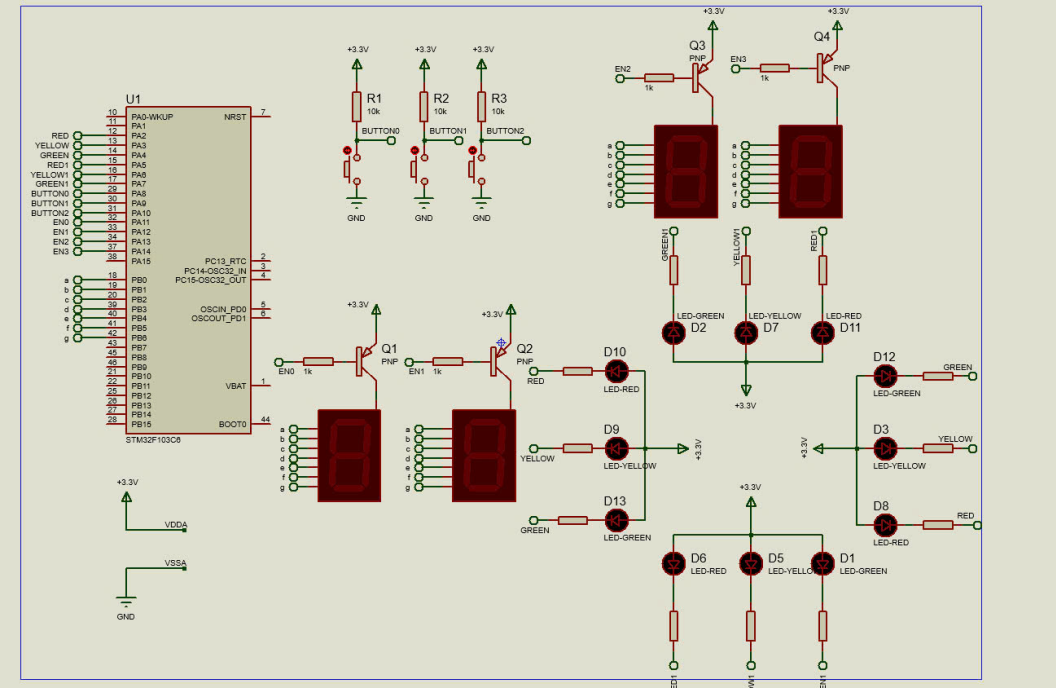
This exercise is to sketch an FSM that describes my idea of how to solve the problem.



Hình 1.1: Sketch an FSM

2.2 Exercise2

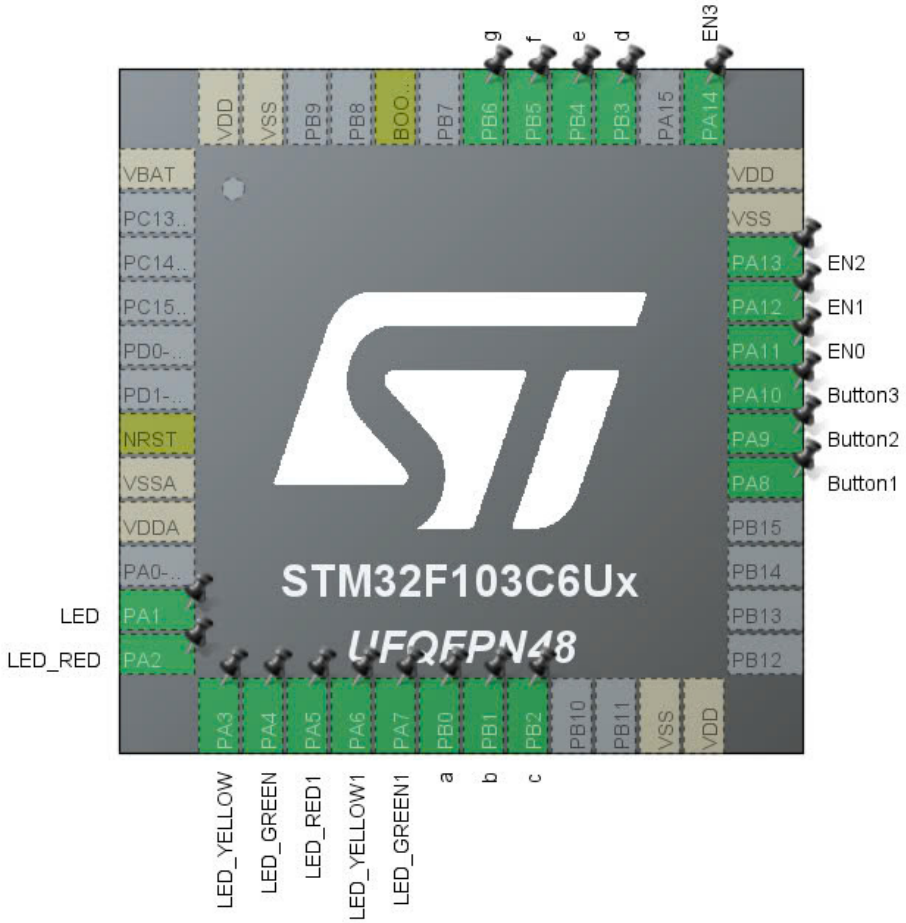
This task in this exercise is to draw a Proteus schematic for the problem above



Hình 1.2: Proteus Schematic

2.3 Exercise3

This task in this exercise is to draw a Proteus schematic for the problem above



Hình 1.3: Proteus Schematic

2.4 Exercise4

Modify Timer Parameters

```
#include "fsm_manual.h"

void fsm_manual_run()
{
  switch(status)
  {
  case MODE1:
    // return INIT status AUTO_RED_GREEN
    setTimer1(defaultTimeGreen*1000);
    setTimer2(500);
    timeRed = defaultTimeRed;
    timeGreen = defaultTimeGreen;
    tempYellow = defaultTimeYellow;
```

```
    updateClockBuffer(timeRed--, timeGreen--);
    status = AUTO_RED_GREEN;
    break;
  case MODE2:

    if(is_button_pressed(0) == 1)
    {
      status = MODE3;
      setTimer1(500);
      setTrafficYellowBlink();
    }
    //blinking red led
    if(timer1_flag == 1)
    {
      HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
      HAL_GPIO_TogglePin(LED_RED1_GPIO_Port, LED_RED1_Pin);
      setTimer1(500); //all LED 2Hz
    }

    if(is_button_pressed(1) == 1)
    {
      tempRed++;
      if(tempRed >99)
      tempRed = 1;
    }

    if(is_button_pressed(2) == 1)
    {
      defaultTimeRed = tempRed;
    }
    updateClockBuffer(2, defaultTimeRed);
    break;
  case MODE3:

    if(is_button_pressed(0) == 1)
    {
      status = MODE4;
      setTimer1(500);
      setTrafficGreenBlink();
    }
    //blinking yellow led
    if(timer1_flag == 1)
    {
      HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port,
 LED_YELLOW_Pin);
      HAL_GPIO_TogglePin(LED_YELLOW1_GPIO_Port,
 LED_YELLOW1_Pin);
      setTimer1(500);
    }
```

```
61
62      if(is_button_pressed(1) == 1)
63      {
64        tempYellow++;
65        if(tempYellow > 99) tempYellow = 1;
66      }
67      //if button3 is pressed, tempYellow is assign for
    dedefaultTimeYelow
68      if(is_button_pressed(2) == 1)
69      {
70        defaultTimeYellow = tempYellow ;
71      }
72      updateClockBuffer(3, defaultTimeYellow);
73
74      break;
75    case MODE4:
76      // if button0 is pressed, status will move to MAN_MODE1
77
78
79      if(is_button_pressed(0) == 1)
80      {
81        if (defaultTimeRed < (defaultTimeGreen +
    defaultTimeYellow))
82        {
83          defaultTimeRed = preTimeRed;
84          defaultTimeYellow = preTimeYellow;
85          defaultTimeGreen = preTimeGreen;
86        }
87        else
88        {
89          preTimeRed = defaultTimeRed;
90          preTimeYellow = defaultTimeYellow;
91          preTimeGreen = defaultTimeGreen;
92        }
93        status = MODE1;
94        setTimer1(500);
95      }
96
97      if(timer1_flag == 1)
98      {
99        HAL_GPIO_TogglePin(LED_GREEN1_GPIO_Port,
    LED_GREEN1_Pin);
100       HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin
    );
101       setTimer1(500);
102     }
103     //if button is pressed, tempGreen value 'll increase 1
    unit
104     //if value overcome 99, it'll return 1
```

```
105       if(is_button_pressed(1) == 1)
106       {
107         tempGreen++;
108         if(tempGreen > 99) tempGreen = 1;
109       }
110     //if button3 is pressed, tempGreen is assign for
    dedefaultTimeGreen
111       if(is_button_pressed(2) == 1)
112       {
113         defaultTimeGreen = tempGreen ;
114       }
115     updateClockBuffer(4, defaultTimeGreen);
116
117     break;
118   default:
119       break;
120   }
121 }
```

Program 1.1: Source code of Ex4

2.5 Exercise5

Adding code for button debouncing

```
1  #include <input_reading.h>
2
3  //we aim to work with more than one buttons
4  #define NO_OF_BUTTONS 3
5  //timer interrupt duration is 10ms, so to pass 1 second,
6  //we need to jump to the interrupt service routine 100 time
7  #define DURATION_FOR_AUTO_INCREASING 100
8  #define BUTTON_IS_PRESSED   GPIO_PIN_RESET
9  #define BUTTON_IS_RELEASED   GPIO_PIN_SET
10 #define BUTTON_PRESSED_MORE_THAN_1s   2
11
12 #define BUTTON_PRESSED GPIO_PIN_RESET // 0
13 #define BUTTON_RELEASED GPIO_PIN_SET  // 1
14 //the buffer that the final result is stored after
15 //debouncing
16 static GPIO_PinState buttonBuffer[NO_OF_BUTTONS];
17 //we define two buffers for debouncing
18 static GPIO_PinState debounceButtonBuffer1[NO_OF_BUTTONS];
19 static GPIO_PinState debounceButtonBuffer2[NO_OF_BUTTONS];
20 static GPIO_PinState debounceButtonBuffer3[NO_OF_BUTTONS];
21 //we define a flag for a button pressed more than 1 second.
22 static uint8_t flagForButtonPress1s[NO_OF_BUTTONS];
23 //we define counter for automatically increasing the value
24 //after the button is pressed more than 1 second.
```

```c
static uint16_t counterForButtonPress1s[NO_OF_BUTTONS];
static int buttonState[NO_OF_BUTTONS] = {BUTTON_IS_RELEASED
   , BUTTON_IS_RELEASED, BUTTON_IS_RELEASED};
static int button_flag[NO_OF_BUTTONS];

// this function turn on button_flag
void getKeyProcess(int index)
{
  if(index >= 0 && index < NO_OF_BUTTONS)
  {
    button_flag[index] = 1;
  }
}
// this function turn on flagForButtonPress1s
void get1sFlag(int index)
{
  if(index >= 0 && index < NO_OF_BUTTONS)
  {
    flagForButtonPress1s[index] = 1;
  }
}

void fsm_input_processing(GPIO_PinState buttonBuffer[], int
    index)
{
  switch(buttonState[index])
  {
  case BUTTON_IS_PRESSED:
    //if button is pressed more than 1s
    if(counterForButtonPress1s[index] <
   DURATION_FOR_AUTO_INCREASING)
    {
      counterForButtonPress1s[index]++;
      if(counterForButtonPress1s[index] ==
   DURATION_FOR_AUTO_INCREASING)
      {
        buttonState[index] = BUTTON_PRESSED_MORE_THAN_1s;
        counterForButtonPress1s[index] = 0;
        getKeyProcess(index);
        get1sFlag(index);
      }
    }
    //button is release
    if(buttonBuffer[index] == BUTTON_RELEASED){
      buttonState[index] = BUTTON_IS_RELEASED;
      counterForButtonPress1s[index] = 0;
    }
    break;
  case BUTTON_IS_RELEASED:
```

```
70      //button is pressed
71      if(buttonBuffer[index] == BUTTON_PRESSED)
72      {
73        buttonState[index] = BUTTON_IS_PRESSED;
74        getKeyProcess(index);
75      }
76      break;
77    case BUTTON_PRESSED_MORE_THAN_1s:
78      // if button is pressed more than 1s and button is
     continued pressing
79      // it'll execute previous status
80      if(counterForButtonPress1s[index] <
     DURATION_FOR_AUTO_INCREASING)
81      {
82            counterForButtonPress1s[index]++;
83            if(counterForButtonPress1s[index] ==
     DURATION_FOR_AUTO_INCREASING)
84            {
85              buttonState[index] =
     BUTTON_PRESSED_MORE_THAN_1s;
86              counterForButtonPress1s[index] = 0;
87              getKeyProcess(index);
88              get1sFlag(index);
89            }
90          }
91      //button is release
92      if(buttonBuffer[index] == BUTTON_RELEASED){
93        buttonState[index] = BUTTON_IS_RELEASED;
94        counterForButtonPress1s[index] = 0;
95      }
96      break;
97    default:
98      break;
99    }
100 }
101
102 void button_reading(void)
103 {
104   for(uint8_t i = 0; i < NO_OF_BUTTONS; i++)
105   {
106     debounceButtonBuffer3[i] = debounceButtonBuffer2[i];
107     debounceButtonBuffer2[i] = debounceButtonBuffer1[i];
108     // Choosing Which button is pressed.
109     switch(i)
110     {
111     case 0: // read signal from button0
112       debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(
     Button1_GPIO_Port, Button1_Pin);
113       break;
```

```
114     case 1://  read signal from button0
115       debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(
    Button2_GPIO_Port, Button2_Pin);
116       break;
117     case 2://  read signal from button0
118       debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(
    Button3_GPIO_Port, Button3_Pin);
119       break;
120     default:
121       break;
122     }
123     if((debounceButtonBuffer3[i] == debounceButtonBuffer2[i
    ]) && (debounceButtonBuffer2[i] == debounceButtonBuffer1
    [i]))
124     {
125       buttonBuffer[i] = debounceButtonBuffer3[i];
126       //call fsm_input_processing() function
127       fsm_input_processing(buttonBuffer,i);
128     }
129
130   }
131 }
132
133
134 //Determine whether a button is pressed or not
135 int is_button_pressed(uint8_t index)
136 {
137   if(index >= NO_OF_BUTTONS)
138   return 0;
139   if(button_flag[index] == 1)
140   {
141     //set button flag value = 0
142     button_flag[index] = 0;
143     return 1;
144   }
145   return 0;
146 }
147 //determine whether a button is pressed more than 1s or not
148 int  is_button_pressed_1s(uint8_t index)
149 {
150   if(index >= NO_OF_BUTTONS) return 0;
151   if(button_flag[index] == 1 && flagForButtonPress1s[index]
      == 1)
152   {
153     //set button value
154     button_flag[index] = 0;
155     flagForButtonPress1s[index] = 0;
156     return 1;
157   }
```

```
158   return 0;
159 }
```

Program 1.2: Source code of Ex5

2.6 Exercise6

Adding code for displaying modes

```
1  #include "led_7segment.h"
2
3  uint16_t led_matrix[MAX_MATRIX] = {0x3f, 0x06, 0x5b, 0x4f,
       0x66, 0x6d, 0x7D, 0x07, 0x7F, 0x6f};
4  int index_led = 0;
5  int led_buffer[MAX_BUFF] = {0,0,0,0};
6
7  //display LED 7 segment
8  void display7SEG(int number)
9  {
10   uint16_t bit_var = led_matrix[number];
11   HAL_GPIO_WritePin(GPIOB, bit_var, RESET);
12   HAL_GPIO_WritePin(GPIOB, ~bit_var, SET);
13 }
14 //if counter1 < 10, example value = 2, led7 1 display 0 and
       led 7 segment 2 display 2
15 //if counter1 > 10, example value = 12, led7 1 display 1
       and led 7 segment 2 display 2
16 void updateClockBuffer(int counter1, int counter2)
17 {
18       led_buffer[0] = counter1 / 10;
19     led_buffer[1] = counter1 % 10;
20     led_buffer[2] = counter2 / 10;
21     led_buffer[3] = counter2 % 10;
22  }
23 // show which led 7 segment is ON and the value it display
24 void update7SEG(int index)
25 {
26   switch(index)
27   {
28   case 0:
29     // Display the first 7 SEG with led_buffer [0]
30     HAL_GPIO_WritePin(GPIOA, EN0_Pin, RESET);
31     HAL_GPIO_WritePin(GPIOA, EN1_Pin | EN2_Pin | EN3_Pin,
       SET);
32     display7SEG(led_buffer[0]);
33     break;
34   case 1:
35     // Display the second 7 SEG with led_buffer [1]
36     HAL_GPIO_WritePin(GPIOA, EN1_Pin, RESET);
```

```
37      HAL_GPIO_WritePin(GPIOA, EN0_Pin | EN2_Pin | EN3_Pin,
    SET);
38      display7SEG(led_buffer[1]);
39      break;
40    case 2:
41      // Display the third 7 SEG with led_buffer [2]
42      HAL_GPIO_WritePin(GPIOA, EN2_Pin, RESET);
43      HAL_GPIO_WritePin(GPIOA, EN0_Pin | EN1_Pin | EN3_Pin,
    SET);
44      display7SEG(led_buffer[2]);
45      break;
46    case 3:
47      // Display the forth 7 SEG with led_buffer [3]
48      HAL_GPIO_WritePin(GPIOA, EN3_Pin, RESET);
49      HAL_GPIO_WritePin(GPIOA, EN0_Pin| EN1_Pin | EN2_Pin ,
    SET);
50      display7SEG(led_buffer[3]);
51      break;
52    default:
53      break;
54    }
55  }
56  // display time value in LED 7 SEGMENT
57  void scanLed()
58  {
59    if(timer3_flag == 1)
60    {
61      update7SEG(index_led++);
62      if(index_led > 3)
63      index_led = 0;
64      setTimer3(250);
65    }
66  }
```

Program 1.3: Source code of Ex6

2.7 Exercise7

Adding code for increasing time duration value for the red LEDs

```
1   case MODE2:
2
3     if(is_button_pressed(0) == 1)
4     {
5       status = MODE3;
6       setTimer1(500);
7       setTrafficYellowBlink();
8     }
9     //blinking red led
```

```
    if(timer1_flag == 1)
    {
      HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
      HAL_GPIO_TogglePin(LED_RED1_GPIO_Port, LED_RED1_Pin);
      setTimer1(500); //all LED 2Hz
    }

    if(is_button_pressed(1) == 1)
    {
      tempRed++;
      if(tempRed >99)
      tempRed = 1;
    }

    if(is_button_pressed(2) == 1)
    {
      defaultTimeRed = tempRed;
    }
    updateClockBuffer(2, defaultTimeRed);
    break;
```

Program 1.4: Source code of Ex7

2.8 Exercise8

Adding code for increasing time duration value for the red LEDs

```
  case MODE4:
    // if button0 is pressed, status will move to MAN_MODE1


    if(is_button_pressed(0) == 1)
    {
      if (defaultTimeRed < (defaultTimeGreen +
   defaultTimeYellow))
      {
        defaultTimeRed = preTimeRed;
        defaultTimeYellow = preTimeYellow;
        defaultTimeGreen = preTimeGreen;
      }
      else
      {
        preTimeRed = defaultTimeRed;
        preTimeYellow = defaultTimeYellow;
        preTimeGreen = defaultTimeGreen;
      }
      status = MODE1;
      setTimer1(500);
    }
```

```
22
23    if(timer1_flag == 1)
24    {
25      HAL_GPIO_TogglePin(LED_GREEN1_GPIO_Port,
    LED_GREEN1_Pin);
26      HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin
    );
27      setTimer1(500);
28    }
29    //if button is pressed, tempGreen value 'll increase 1
    unit
30    //if value overcome 99, it'll return 1
31      if(is_button_pressed(1) == 1)
32      {
33        tempGreen++;
34        if(tempGreen > 99) tempGreen = 1;
35      }
36    //if button3 is pressed, tempGreen is assign for
    dedefaultTimeGreen
37      if(is_button_pressed(2) == 1)
38      {
39        defaultTimeGreen = tempGreen ;
40      }
41    updateClockBuffer(4, defaultTimeGreen);
42
43    break;
```

Program 1.5: Source code of Ex8

2.9 Exercise9

Adding code for increasing time duration value for the red LEDs

```
1   case MODE3:
2
3     if(is_button_pressed(0) == 1)
4     {
5       status = MODE4;
6       setTimer1(500);
7       setTrafficGreenBlink();
8     }
9     //blinking yellow led
10    if(timer1_flag == 1)
11    {
12      HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port,
    LED_YELLOW_Pin);
13      HAL_GPIO_TogglePin(LED_YELLOW1_GPIO_Port,
    LED_YELLOW1_Pin);
14      setTimer1(500);
```

```
    }

    if(is_button_pressed(1) == 1)
    {
      tempYellow++;
      if(tempYellow > 99) tempYellow = 1;
    }
    //if button3 is pressed, tempYellow is assign for
   dedefaultTimeYelow
    if(is_button_pressed(2) == 1)
    {
      defaultTimeYellow = tempYellow ;
    }
    updateClockBuffer(3, defaultTimeYellow);

    break;
```

Program 1.6: Source code of Ex9

2.10 Exercise10

Adding code for increasing time duration value for the red LEDs

```
#include "main.h"
#include "global.h"
#include "input_reading.h"
#include "software_timer.h"
#include "traffic_light.h"
#include "led_7segment.h"
#include "fsm_automatic.h"
#include "fsm_manual.h"

/* Private includes
   ----------------------------------------------------------
   */
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef
   -----------------------------------------------------------
   */
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define
   ------------------------------------------------------------
   */
```

```
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro
   -------------------------------------------------------------
   */
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables
   ---------------------------------------------------------
   */
TIM_HandleTypeDef htim2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes
   -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code
   ---------------------------------------------------------
   */
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration
   ---------------------------------------------------------
   */

```

```
61   /* Reset of all peripherals, Initializes the Flash
      interface and the Systick. */
62   HAL_Init();
63
64   /* USER CODE BEGIN Init */
65
66   /* USER CODE END Init */
67
68   /* Configure the system clock */
69   SystemClock_Config();
70
71   /* USER CODE BEGIN SysInit */
72
73   /* USER CODE END SysInit */
74
75   /* Initialize all configured peripherals */
76   MX_GPIO_Init();
77   MX_TIM2_Init();
78   /* USER CODE BEGIN 2 */
79   HAL_TIM_Base_Start_IT(&htim2);
80   /* USER CODE END 2 */
81
82   /* Infinite loop */
83   /* USER CODE BEGIN WHILE */
84   status = INIT;
85   setTimer3(100);
86   while (1)
87   {
88     /* USER CODE END WHILE */
89     fsm_automatic_run();
90     fsm_manual_run();
91     scanLed();
92     /* USER CODE BEGIN 3 */
93   }
94   /* USER CODE END 3 */
95 }
96
97 /**
98   * @brief System Clock Configuration
99   * @retval None
100  */
101 void SystemClock_Config(void)
102 {
103   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
104   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
105
106   /** Initializes the RCC Oscillators according to the
      specified parameters
107   * in the RCC_OscInitTypeDef structure.
```

```
108   */
109   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI
      ;
110   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
111   RCC_OscInitStruct.HSICalibrationValue =
      RCC_HSICALIBRATION_DEFAULT;
112   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
113   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
114   {
115     Error_Handler();
116   }
117   /** Initializes the CPU, AHB and APB buses clocks
118   */
119   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|
      RCC_CLOCKTYPE_SYSCLK
120                                 |RCC_CLOCKTYPE_PCLK1|
      RCC_CLOCKTYPE_PCLK2;
121   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
122   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
123   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
124   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
125
126   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
      FLASH_LATENCY_0) != HAL_OK)
127   {
128     Error_Handler();
129   }
130 }
131
132 /**
133   * @brief TIM2 Initialization Function
134   * @param None
135   * @retval None
136   */
137 static void MX_TIM2_Init(void)
138 {
139
140   /* USER CODE BEGIN TIM2_Init 0 */
141
142   /* USER CODE END TIM2_Init 0 */
143
144   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
145   TIM_MasterConfigTypeDef sMasterConfig = {0};
146
147   /* USER CODE BEGIN TIM2_Init 1 */
148
149   /* USER CODE END TIM2_Init 1 */
150   htim2.Instance = TIM2;
151   htim2.Init.Prescaler = 7999;
```

```
152   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
153   htim2.Init.Period = 9;
154   htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
155   htim2.Init.AutoReloadPreload =
       TIM_AUTORELOAD_PRELOAD_DISABLE;
156   if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
157   {
158     Error_Handler();
159   }
160   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL
       ;
161   if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig
       ) != HAL_OK)
162   {
163     Error_Handler();
164   }
165   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
166   sMasterConfig.MasterSlaveMode =
       TIM_MASTERSLAVEMODE_DISABLE;
167   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &
       sMasterConfig) != HAL_OK)
168   {
169     Error_Handler();
170   }
171   /* USER CODE BEGIN TIM2_Init 2 */
172
173   /* USER CODE END TIM2_Init 2 */
174
175 }
176
177 /**
178   * @brief GPIO Initialization Function
179   * @param None
180   * @retval None
181   */
182 static void MX_GPIO_Init(void)
183 {
184   GPIO_InitTypeDef GPIO_InitStruct = {0};
185
186   /* GPIO Ports Clock Enable */
187   __HAL_RCC_GPIOA_CLK_ENABLE();
188   __HAL_RCC_GPIOB_CLK_ENABLE();
189
190   /*Configure GPIO pin Output Level */
191   HAL_GPIO_WritePin(GPIOA, LED_Pin|LED_RED_Pin|
       LED_YELLOW_Pin|LED_GREEN_Pin
192                             |LED_RED1_Pin|LED_YELLOW1_Pin|
       LED_GREEN1_Pin|EN0_Pin
```

```
193                           |EN1_Pin|EN2_Pin|EN3_Pin,
    GPIO_PIN_RESET);
194
195   /*Configure GPIO pin Output Level */
196   HAL_GPIO_WritePin(GPIOB, a_Pin|b_Pin|c_Pin|d_Pin
197                           |e_Pin|f_Pin|g_Pin,
    GPIO_PIN_RESET);
198
199   /*Configure GPIO pins : LED_Pin LED_RED_Pin
    LED_YELLOW_Pin LED_GREEN_Pin
200                            LED_RED1_Pin LED_YELLOW1_Pin
    LED_GREEN1_Pin EN0_Pin
201                            EN1_Pin EN2_Pin EN3_Pin */
202   GPIO_InitStruct.Pin = LED_Pin|LED_RED_Pin|LED_YELLOW_Pin|
    LED_GREEN_Pin
203                           |LED_RED1_Pin|LED_YELLOW1_Pin|
    LED_GREEN1_Pin|EN0_Pin
204                           |EN1_Pin|EN2_Pin|EN3_Pin;
205   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
206   GPIO_InitStruct.Pull = GPIO_NOPULL;
207   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
208   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
209
210   /*Configure GPIO pins : a_Pin b_Pin c_Pin d_Pin
211                            e_Pin f_Pin g_Pin */
212   GPIO_InitStruct.Pin = a_Pin|b_Pin|c_Pin|d_Pin
213                           |e_Pin|f_Pin|g_Pin;
214   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
215   GPIO_InitStruct.Pull = GPIO_NOPULL;
216   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
217   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
218
219   /*Configure GPIO pins : Button1_Pin Button2_Pin
    Button3_Pin */
220   GPIO_InitStruct.Pin = Button1_Pin|Button2_Pin|Button3_Pin
    ;
221   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
222   GPIO_InitStruct.Pull = GPIO_PULLUP;
223   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
224
225 }
226
227 /* USER CODE BEGIN 4 */
228 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
229 {
230   if(htim->Instance == TIM2){
231     button_reading();
232   }
233   timer_run();
```

```
234 }
235 /* USER CODE END 4 */
236
237 /**
238   * @brief  This function is executed in case of error
      occurrence.
239   * @retval None
240   */
241 void Error_Handler(void)
242 {
243   /* USER CODE BEGIN Error_Handler_Debug */
244   /* User can add his own implementation to report the HAL
      error return state */
245   __disable_irq();
246   while (1)
247   {
248   }
249   /* USER CODE END Error_Handler_Debug */
250 }
251
252 #ifdef  USE_FULL_ASSERT
253 /**
254   * @brief  Reports the name of the source file and the
      source line number
255   *         where the assert_param error has occurred.
256   * @param  file: pointer to the source file name
257   * @param  line: assert_param error line source number
258   * @retval None
259   */
260 void assert_failed(uint8_t *file, uint32_t line)
261 {
262   /* USER CODE BEGIN 6 */
263   /* User can add his own implementation to report the file
       name and line number,
264     ex: printf("Wrong parameters value: file %s on line %d
      \r\n", file, line) */
265   /* USER CODE END 6 */
266 }
267 #endif /* USE_FULL_ASSERT */
268
269 /************************ (C) COPYRIGHT STMicroelectronics
      *****END OF FILE****/
```

Program 1.7: Source code of Ex10