ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

VI XỬ LÝ - VI ĐIỀU KHIỂN

Assignment Project

Traffic Light using STM32F103RB

GVHD: Lê Trọng Nhân.

Lớp: L02

Sinh viên: Nguyễn Tuyết Vy – 2012458.

Phạm Văn Nhật Vũ - 2110676.

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 12/2023

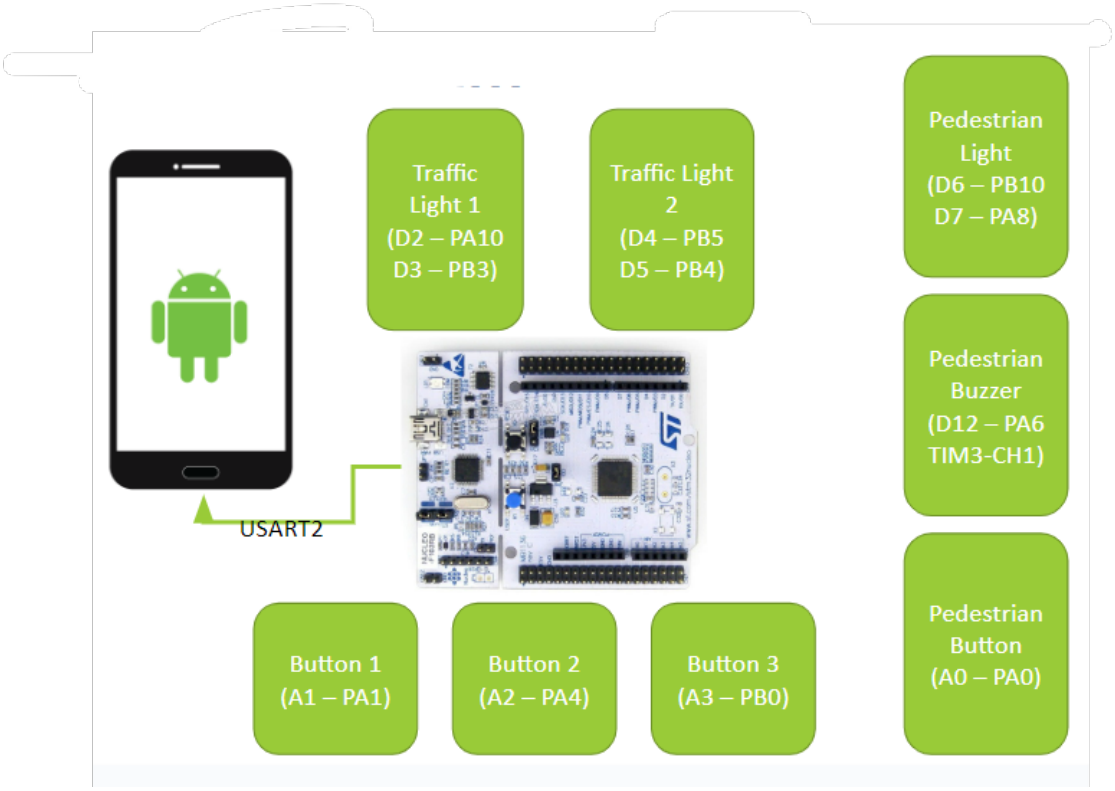
Nội dung

1 Đặc tả yêu cầu

In this project, the STM32F103RB is used to simulate the 2-way traffic light system, having some main features:

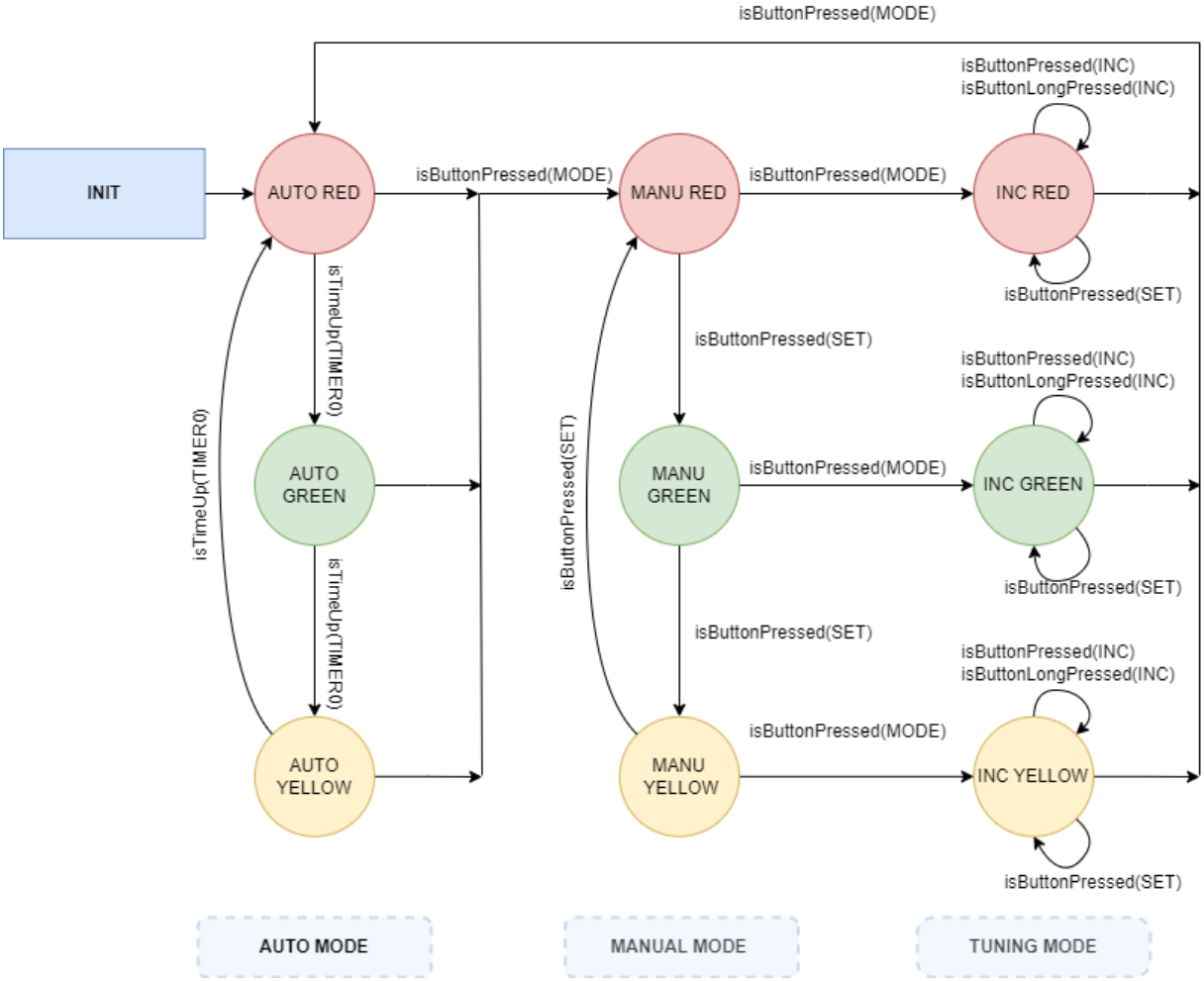
- **Automatic mode**: The system operates as normal. The light colors are red, yellow, and green.
- **Manual mode**: A button is used to switch the light colors in this mode
- **Tuning mode**: This mode is used to modify the light timing length
- **Pedestrian scramble**: when the button for pedestrian is pressed, its light is turned on and operates reversely to the light of vehicles

Block Diagram:

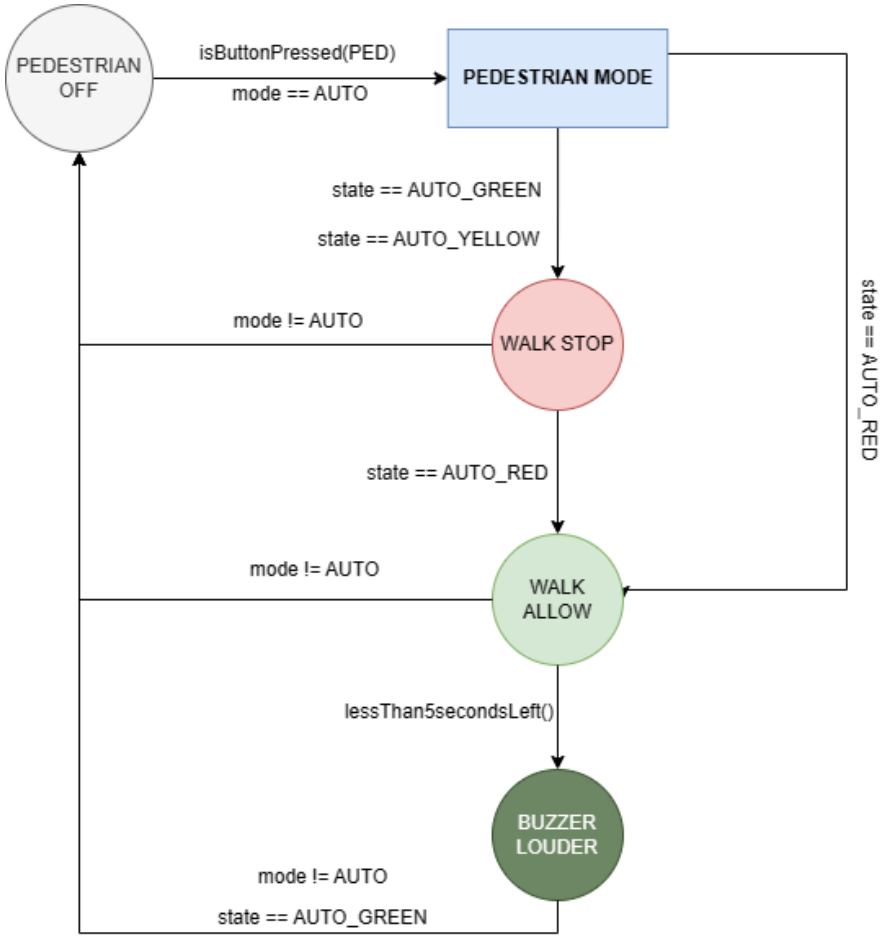


Hình 1: Block Diagram

2 Thiết kế FSM



Hình 2: FSM của 3 chế độ chính



Hình 3: FSM của chế độ cho người đi bộ

Ý tưởng:

- Mỗi hướng đường (ngang hoặc dọc) tại ngã tư sẽ được đặt:
  - 1 đèn đỏ (RED), 1 đèn vàng (YELLOW), 1 đèn xanh (GREEN) cho các phương tiện giao thông
  - 1 đèn đỏ, 1 đèn xanh và 1 loa cho người đi bộ.
- Có 5 nút nhấn:
  - **MODE**: dùng để lựa chọn các chế độ Auto mode, Manual mode, Tuning mode.
  - **SET**: dùng để chọn màu đèn được điều chỉnh thời gian sáng và lưu lại thời gian sáng của các đèn trong chế độ. **TUNING**.
  - **INC**: dùng để thay đổi thời gian sáng của các đèn, giá trị thay đổi này chỉ được lưu khi nút nhấn **SET** được nhấn.
  - **Hai nút nhấn PED tại mỗi hướng đường**: dùng cho người đi bộ sử dụng để xin đường (hệ thống đi vào chế độ **PEDESTRIAN MODE**).

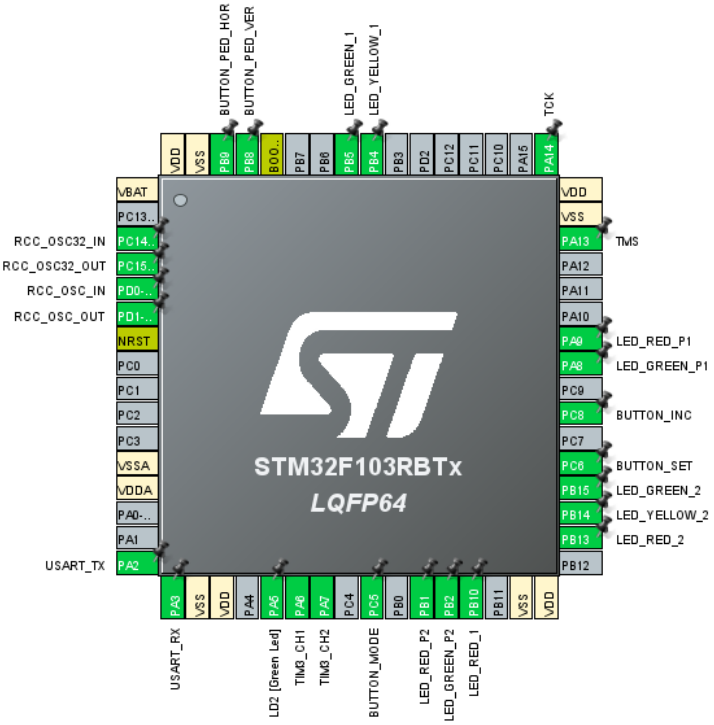
- Có 3 chế độ trong hệ thống đèn giao thông (cho các phương tiện giao thông):
  - **Chế độ 1: AUTO MODE**
    - Các đèn tại mỗi hướng đường hoạt động bình thường theo cơ chế đèn giao thông. Khi bắt đầu khởi động chế độ 1 thì đường theo **hướng ngang sáng đèn đỏ** và đường theo **hướng dọc sáng đèn xanh**.
    - Do thời gian sáng của mỗi đèn có thể được điều chỉnh trong chế độ **TUNING MODE**, cơ chế thời gian đèn giao thông có thể không tuân thủ theo công thức: PERIOD_RED = PERIOD_YELLOW + PERIOD_GREEN.
    - Tại đây, nếu nút **MODE** được nhấn, hệ thống sẽ thay đổi sang chế độ 2.
  - **Chế độ 2: MANUAL MODE**
    - Chọn màu đèn cần thay đổi thời gian sáng, khi bắt đầu chế độ 2 thì màu đèn mặc định là màu đỏ.
    - Khi nhấn nút **SET**, màu được chọn sẽ được thay đổi theo thứ tự: đỏ - xanh - vàng - đỏ - xanh - vàng - đỏ....
    - Khi nhấn nút **MODE**, hệ thống sẽ chuyển sang chế độ 3 để điều chỉnh thời gian sáng cho các đèn có màu đang được chọn.
  - **Chế độ 3: TUNING MODE**
    - Thay đổi thời gian sáng cho các đèn của một màu đã được chọn ở chế độ 2.
    - Mỗi khi nút **INC** được nhấn, thời gian sáng sẽ tăng lên 1 đơn vị. Nếu nút này được nhấn đè trong 1 giây, thì cứ mỗi 500ms, thời gian sáng cũng sẽ tăng lên 1 đơn vị, tới khi nút **INC** được nhả. Giá trị của thời gian sáng sẽ nằm trong đoạn từ 1 tới 99, nếu giá trị này vượt quá 99 thì sẽ được đặt về lại từ 1.
    - Thời gian sáng thay đổi bằng nút **INC** chỉ được lưu lại sau khi nút **SET** được nhấn. Nếu nút **SET** không được nhấn, thời gian của các đèn tại chế độ 1 sẽ được giữ nguyên như trước khi hệ thống đi vào chế độ 3. Nút **SET** có thể được nhấn nhiều lần để lưu lại các giá trị thời gian sáng khác nhau và giá trị ngay trước lần nhấn nút **SET** cuối cùng sẽ là thời gian sáng của các đèn khi hệ thống quay về chế độ **AUTO MDOE**.
    - Khi nhấn nút **MODE**, hệ thống sẽ trở về chế độ 1.
- Bên cạnh đó, người đi bộ tại mỗi hướng đường có thể nhấn nút **PED** tương ứng để xin đường và kích hoạt chế độ **PEDESTRIAN MODE** tại hướng đường tương ứng.
  - Chế độ **PEDESTRIAN MODE** chỉ được kích hoạt khi hệ thống đang ở chế độ **AUTO MODE**. Và khi hệ thống được chuyển sang các chế độ khác (**MANUAL MODE**, **TUNING MODE**), chế độ **PEDESTRIAN MODE** sẽ bị bất hoạt (trở về trạng thái **PEDESTRIAN_OFF**).
  - Nếu khi nhấn nút **PED**, hướng đường tương ứng đang sáng đèn xanh hoặc sáng đèn vàng thì đèn đỏ cho người đi bộ sẽ sáng, đồng thời loa sẽ kêu với lớn và nhanh để cảnh báo cho người đi bộ không được qua đường. Ở đây, loa được sử dụng để cung cấp khả năng tiếp cận (**accessibility**) cho người khiếm thị, đảm bảo những người đó cũng có khả năng nhận biết được trạng thái của hệ thống như người bình thường.
  - Nếu khi nhấn nút **PED**, hướng đường tương ứng đang sáng đỏ hoặc khi hướng đường tương ứng chuyển từ sáng đèn vàng sang sáng đèn đỏ (sau khi nút **PED** được nhấn) thì đèn xanh cho người đi bộ sẽ sáng, loa sẽ kêu với nhỏ và chậm hơn để thông báo cho người đi bộ có thể qua đường. Sau đó, khi thời gian đèn đỏ cho các phương tiện

giao thông còn 5 giây hoặc ít hơn, đèn xanh cho người đi bộ sẽ nhấp nháy, và loa sẽ kêu ngày càng lớn và ngày càng nhanh, cảnh báo cho người đi bộ sắp hết thời gian qua đường. Cuối cùng, khi thời gian đèn đỏ kết thúc, hướng đường tương ứng chuyển sang sáng đèn xanh thì chế độ cho người đi bộ sẽ trở về trạng thái **PEDESTRIAN OFF**, kết thúc một chu kỳ hoạt động.

- Ngoài ra, giá trị của bộ đếm xuống hiện tại trong chế độ **AUTO MODE** và thời gian sáng đang được điều chỉnh trong chế độ **TUNING MODE** sẽ được gửi qua kết nối UART để thuận tiện cho việc quan sát trạng thái hiện tại của hệ thống.

3 Tạo project STM32

Nhóm cấu hình trạng thái cho các chân output dùng để điều khiển các LED, các chân input dùng để đọc tín hiệu từ các nút nhấn, 2 chân UART Tx, Rx và 2 chân PWM để điều khiển cường độ của các loa:



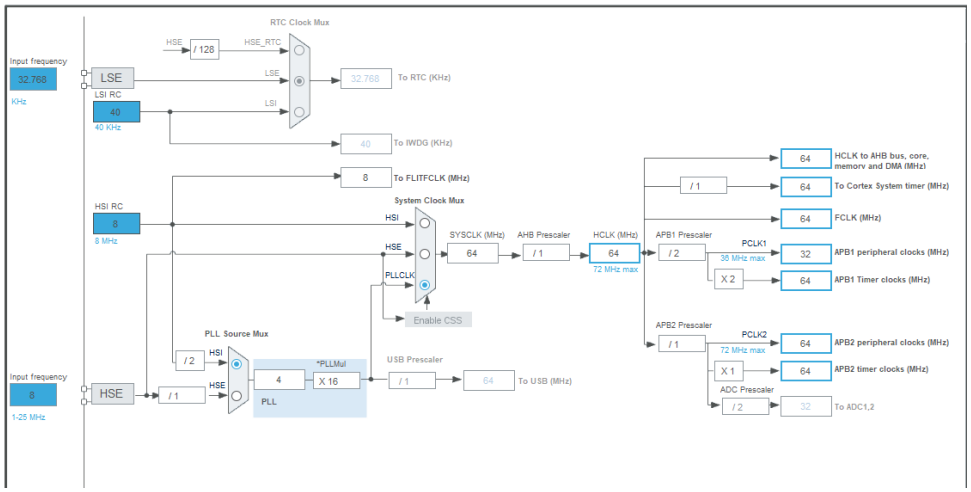
Hình 4: Khởi tạo project STM32

Trong đó, các chân input được cấu hình **pull-up** như sau:

PB8	n/a	n/a	Input m...	Pull-up	n/a	BUTTO...	☑
PB9	n/a	n/a	Input m...	Pull-up	n/a	BUTTO...	☑
PB10	n/a	Low	Output ...	No pull-...	Low	LED_R...	☑
PB13	n/a	Low	Output ...	No pull-...	Low	LED_R...	☑
PB14	n/a	Low	Output ...	No pull-...	Low	LED_Y...	☑
PB15	n/a	Low	Output ...	No pull-...	Low	LED_G...	☑
PC5	n/a	n/a	Input m...	Pull-up	n/a	BUTTO...	☑
PC6	n/a	n/a	Input m...	Pull-up	n/a	BUTTO...	☑
PC8	n/a	n/a	Input m...	Pull-up	n/a	BUTTO...	☑

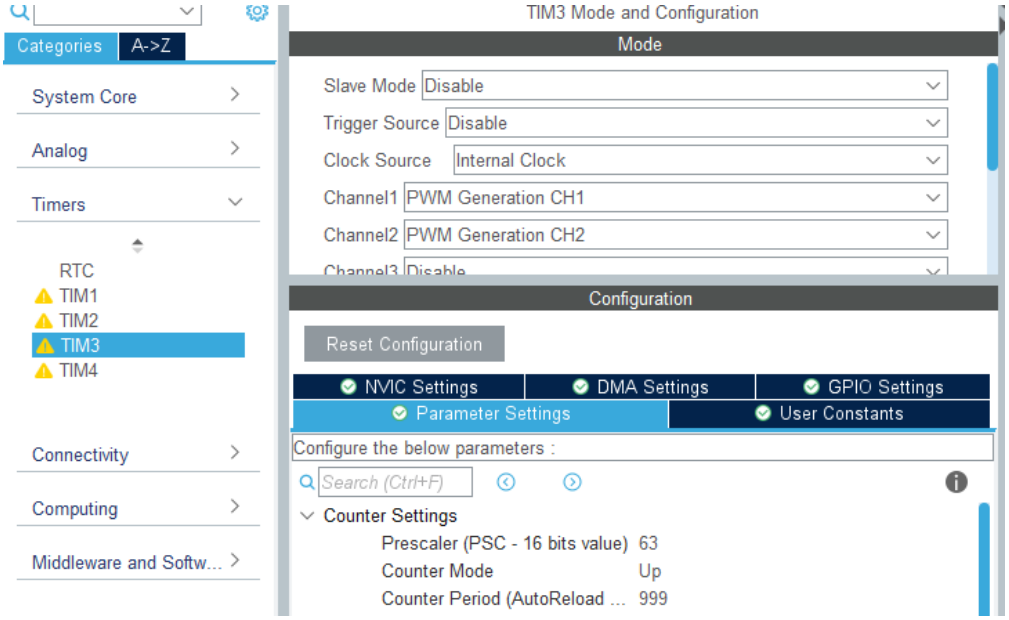
Hình 5: Cấu hình các chân input

Quan sát clock diagram của project, nhóm nhận thấy clock nội có tần số 64 MHz.



Hình 6: Clock diagram của project

Để cấu hình timer interrupt với tần số 1 kHz (được gọi mỗi 1 milli giây), nhóm cài đặt các tham số Presacler và Counter Period lần lượt là 63 và 999



Hình 7: Cấu hình timer interrupt

Thật vậy, theo cấu hình trên, timer 3 sẽ có tần số ngắt bằng

$$f = \frac{f_{internal}}{(63+1)(999+1)} = 1000(Hz) = 1Khz$$

4 Hiện thực

Khung chương trình chính được hiện thực trong hàm **main()** của file **main.c** như sau:

```
...
void testIO()
{
  if (isButtonPressed(BUTTON_PED_HOR))
  {
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
  }
  if (isButtonPressed(BUTTON_PED_VER))
  {
    HAL_GPIO_TogglePin(LED_GREEN_1_GPIO_Port, LED_GREEN_1_Pin);
  }
  if (isButtonPressed(BUTTON_MODE))
  {
    HAL_GPIO_TogglePin(LED_YELLOW_1_GPIO_Port, LED_YELLOW_1_Pin);
  }
  if (isButtonPressed(BUTTON_SET))
  {
    HAL_GPIO_TogglePin(LED_RED_1_GPIO_Port, LED_RED_1_Pin);
  }
  if (isButtonPressed(BUTTON_INC))
  {
    HAL_GPIO_TogglePin(LED_GREEN_P1_GPIO_Port, LED_GREEN_P1_Pin);
  }
}

void testBuzzer()
{
  for (int i = 500; i < 1000; i += 100)
  {
    __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 500);
    HAL_Delay(1000);
    __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 0);
    HAL_Delay(1000);
    __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 600);
    HAL_Delay(1000);
    __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 0);
    HAL_Delay(1000);
    __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 800);
    HAL_Delay(1000);
    __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 0);
    HAL_Delay(1000);
  }
}

int main(void)
```

```
{
    ...
    /* USER CODE BEGIN WHILE */
    HAL_TIM_Base_Start_IT(&htim3);
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);

    setTimer(7, BUTTON_TIME_STEP);
    while (1)
    {
    //    testBuzzer();
      fsmAutoModeRun();
      fsmManualModeRun();
      fsmTuningModeRun();
      fsmPedestrianModeRun();
      uartProcessing();
      buttonProcessing();
    //    testIO();
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}

/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    timerRun();
}
```

Program 1: Khung chương trình chính hiện thực trong hàm **main()**

Hàm **testIO()** được sử dụng để kiểm tra hoạt động của các đèn LED, nút nhấn kết nối với các chân input, output của STM32. Tương tự hàm **testBuzzer()** để kiểm tra hoạt động của các loa.

Hàm **HAL_TIM_PeriodElapsedCallback()** là hàm interrupt service routine cho timer 3 được cấu hình ở phần 3, do đó, hàm **timerRun()** sẽ được gọi mỗi 1 milli giây.

Vòng lặp super loop gọi các hàm chạy FSM của từng chế độ **AUTO MODE**, **MANU MODE**, **TUNING MODE** của hệ thống đèn giao thông chính, FSM của hệ thống đèn, loa cho người đi bộ cũng như hàm xử lý quá trình truyền UART và xử lý tín hiệu từ các nút nhấn.

Trong đó:

- Hàm **fsmAutoModeRun()** được hiện thực như sau:

```
...
#include "fsm_auto_mode.h"
void fsmAutoModeRun()
{
    switch (mode)
    {
        case INIT:
            mode = AUTO_MODE;
            resetLight();
            setTimer(6, UART_TIME_STEP);
            break;
        case AUTO_MODE:
            // Switch to MANUAL MODE when press MODE button
            if (isButtonPressed(BUTTON_MODE))
            {
                mode = MANU_MODE;
            }

            // Display single LEDs according to the traffic
    rules
            displayLeds();
            break;
        default:
            break;
    }
}
```

Program 2: Hàm **fsmAutoModeRun()** trong file **fsm_auto_mode.c**

Khi hệ thống đang ở chế độ **INIT**, đèn tại 2 hướng đường ngang và dọc được reset về trạng thái ban đầu, một software timer được cài đặt để truyền giá trị của bộ đếm xuống tại hướng đường ngang qua kết nối UART và hệ thống sẽ ngay lập tức chuyển sang chế độ **AUTO MODE**.

Khi hệ thống đang ở chế độ **AUTO MODE**, nếu nút nhấn **MODE** được nhấn thì hệ thống sẽ chuyển sang chế độ **MANU MODE**. Nếu không, các đèn tại mỗi hướng đường sẽ được hiển thị bình thường theo luật giao thông bằng hàm **displayLeds**, được hiện thực trong file **f_display_LED.c**:

```
#include "f_display_LED.h"

int horizontal_state = INIT;
int vertical_state = INIT;

void fsmHorLeds()
{
    switch (horizontal_state)
    {
```

```
        case INIT:
            horizontal_state = AUTO_RED;
            setTimer(0, RED_time);
            break;
        case AUTO_RED:
            if (isTimerUp(0))
            {
                horizontal_state = AUTO_GREEN;
                setTimer(0, GREEN_time);
            }
            break;
        case AUTO_GREEN:
            if(isTimerUp(0))
            {
                horizontal_state = AUTO_YELLOW;
                setTimer(0, YELLOW_time);
            }
            break;
        case AUTO_YELLOW:
            if(isTimerUp(0))
            {
                horizontal_state = AUTO_RED;
                setTimer(0, RED_time);
            }
            break;
        default:
            break;
    }
}

void fsmVerLeds()
{
    switch (vertical_state)
    {
        case INIT:
            vertical_state = AUTO_GREEN;
            setTimer(1, GREEN_time);
            break;
        case AUTO_RED:
            if (isTimerUp(1))
            {
                vertical_state = AUTO_GREEN;
                setTimer(1, GREEN_time);
            }
            break;
        case AUTO_GREEN:
            if(isTimerUp(1))
            {
                vertical_state = AUTO_YELLOW;
```

```
59                  setTimer(1, YELLOW_time);
60              }
61              break;
62          case AUTO_YELLOW:
63              if(isTimerUp(1))
64              {
65                  vertical_state = AUTO_RED;
66                  setTimer(1, RED_time);
67              }
68              break;
69          default:
70              break;
71      }
72  }
73
74  void displayLeds()
75  {
76      fsmHorLeds();
77      fsmVerLeds();
78
79      switch (horizontal_state)
80      {
81          case AUTO_RED:
82              HAL_GPIO_WritePin(LED_RED_1_GPIO_Port,
    LED_RED_1_Pin, LED_ON);
83              HAL_GPIO_WritePin(LED_GREEN_1_GPIO_Port,
    LED_GREEN_1_Pin, LED_OFF);
84              HAL_GPIO_WritePin(LED_YELLOW_1_GPIO_Port,
    LED_YELLOW_1_Pin, LED_OFF);
85              break;
86
87          case AUTO_GREEN:
88              HAL_GPIO_WritePin(LED_RED_1_GPIO_Port,
    LED_RED_1_Pin, LED_OFF);
89              HAL_GPIO_WritePin(LED_GREEN_1_GPIO_Port,
    LED_GREEN_1_Pin, LED_ON);
90              HAL_GPIO_WritePin(LED_YELLOW_1_GPIO_Port,
    LED_YELLOW_1_Pin, LED_OFF);
91              break;
92
93          case AUTO_YELLOW:
94              HAL_GPIO_WritePin(LED_RED_1_GPIO_Port,
    LED_RED_1_Pin, LED_OFF);
95              HAL_GPIO_WritePin(LED_GREEN_1_GPIO_Port,
    LED_GREEN_1_Pin, LED_OFF);
96              HAL_GPIO_WritePin(LED_YELLOW_1_GPIO_Port,
    LED_YELLOW_1_Pin, LED_ON);
97              break;
98
```

```
99          default:
100             break;
101     }
102
103     switch (vertical_state)
104     {
105         case AUTO_RED:
106             HAL_GPIO_WritePin(LED_RED_2_GPIO_Port,
    LED_RED_2_Pin, LED_ON);
107             HAL_GPIO_WritePin(LED_GREEN_2_GPIO_Port,
    LED_GREEN_2_Pin, LED_OFF);
108             HAL_GPIO_WritePin(LED_YELLOW_2_GPIO_Port,
    LED_YELLOW_2_Pin, LED_OFF);
109             break;
110
111         case AUTO_GREEN:
112             HAL_GPIO_WritePin(LED_RED_2_GPIO_Port,
    LED_RED_2_Pin, LED_OFF);
113             HAL_GPIO_WritePin(LED_GREEN_2_GPIO_Port,
    LED_GREEN_2_Pin, LED_ON);
114             HAL_GPIO_WritePin(LED_YELLOW_2_GPIO_Port,
    LED_YELLOW_2_Pin, LED_OFF);
115             break;
116
117         case AUTO_YELLOW:
118             HAL_GPIO_WritePin(LED_RED_2_GPIO_Port,
    LED_RED_2_Pin, LED_OFF);
119             HAL_GPIO_WritePin(LED_GREEN_2_GPIO_Port,
    LED_GREEN_2_Pin, LED_OFF);
120             HAL_GPIO_WritePin(LED_YELLOW_2_GPIO_Port,
    LED_YELLOW_2_Pin, LED_ON);
121             break;
122         default:
123             break;
124     }
125 }
```

Program 3: Module **f_display_LED**

Trong hàm **displayLEDs**, trước khi đèn màu phù hợp với trạng thái hiện tại của mỗi hướng đường được hiển thị, 2 hàm **fsmHorLeds()** và **fsmVerLeds()** được gọi để kiểm tra bộ đếm hiện tại của mỗi hướng đường đã về 0 hay chưa. Nếu bộ đếm đã về 0, trạng thái của hướng đường tương ứng sẽ và sẽ thay đổi để chuyển sang sáng đèn màu khác.

- Hàm **fsmManualModeRun()** được hiện thực như sau:

```
#include "fsm_manual_mode.h"

void fsmManualModeRun()
{
  switch (mode) {
    case MANU_MODE:
      mode = MANU_RED;
      break;

    case MANU_RED:
      // Turn on red lights
      HAL_GPIO_WritePin(LED_RED_1_GPIO_Port, LED_RED_1_Pin,
    LED_ON);
      HAL_GPIO_WritePin(LED_GREEN_1_GPIO_Port, LED_GREEN_1_Pin
    , LED_OFF);
      HAL_GPIO_WritePin(LED_YELLOW_1_GPIO_Port,
    LED_YELLOW_1_Pin, LED_OFF);

      HAL_GPIO_WritePin(LED_RED_2_GPIO_Port, LED_RED_2_Pin,
    LED_ON);
      HAL_GPIO_WritePin(LED_GREEN_2_GPIO_Port, LED_GREEN_2_Pin
    , LED_OFF);
      HAL_GPIO_WritePin(LED_YELLOW_2_GPIO_Port,
    LED_YELLOW_2_Pin, LED_OFF);

      // Switch to MAN_GREEN when press SET button
      if (isButtonPressed(BUTTON_SET))
      {
        mode = MANU_GREEN;
      }

      // Switch to TUNING MODE when press MODE button

      if(isButtonPressed(BUTTON_MODE))
      {
        fsmButtonReset(BUTTON_INC);
        mode = INC_RED;
        time_count = RED_time / 1000;
      }
      break;

    case MANU_GREEN:
      // Turn on green lights
      HAL_GPIO_WritePin(LED_RED_1_GPIO_Port, LED_RED_1_Pin,
    LED_OFF);
      HAL_GPIO_WritePin(LED_GREEN_1_GPIO_Port, LED_GREEN_1_Pin
    , LED_ON);
      HAL_GPIO_WritePin(LED_YELLOW_1_GPIO_Port,
```

```
    LED_YELLOW_1_Pin, LED_OFF);

      HAL_GPIO_WritePin(LED_RED_2_GPIO_Port, LED_RED_2_Pin,
    LED_OFF);
      HAL_GPIO_WritePin(LED_GREEN_2_GPIO_Port, LED_GREEN_2_Pin
    , LED_ON);
      HAL_GPIO_WritePin(LED_YELLOW_2_GPIO_Port,
    LED_YELLOW_2_Pin, LED_OFF);

      // Switch to MAN_YELLOW when press SET button
      if (isButtonPressed(BUTTON_SET))
      {
        mode = MANU_YELLOW;
      }

      // Switch to TUNING MODE when press MODE button

      if(isButtonPressed(BUTTON_MODE))
      {
        fsmButtonReset(BUTTON_INC);
        mode = INC_GREEN;
        time_count = GREEN_time / 1000;
      }
      break;

   case MANU_YELLOW:
      // Turn on yellow lights
      HAL_GPIO_WritePin(LED_RED_1_GPIO_Port, LED_RED_1_Pin,
    LED_OFF);
      HAL_GPIO_WritePin(LED_GREEN_1_GPIO_Port, LED_GREEN_1_Pin
    , LED_OFF);
      HAL_GPIO_WritePin(LED_YELLOW_1_GPIO_Port,
    LED_YELLOW_1_Pin, LED_ON);

      HAL_GPIO_WritePin(LED_RED_2_GPIO_Port, LED_RED_2_Pin,
    LED_OFF);
      HAL_GPIO_WritePin(LED_GREEN_2_GPIO_Port, LED_GREEN_2_Pin
    , LED_OFF);
      HAL_GPIO_WritePin(LED_YELLOW_2_GPIO_Port,
    LED_YELLOW_2_Pin, LED_ON);

      // Switch to MAN_RED when press SET button
      if (isButtonPressed(BUTTON_SET))
      {
        mode = MANU_RED;
      }

      // Switch to TUNING MODE when press MODE button

```

```
80        if(isButtonPressed(BUTTON_MODE))
81        {
82          fsmButtonReset(BUTTON_INC);
83          mode = INC_YELLOW;
84          time_count = YELLOW_time / 1000;
85        }
86        break;
87
88      default:
89        break;
90    }
91  }
92
```

Program 4: Hàm **fsmManulModeRun()** hiện thực trong file **fsm_manual_mode.c**

Nếu hệ thống đang ở chế độ **MANU MODE**, tức là ngay sau khi nút nhấn **MODE** được nhấn chế độ trước đó là **AUTO MODE**, hệ thống sẽ ngay lập tức chuyển sang chế độ **MANU RED**. Tại chế độ **MANU RED**, các đèn đỏ tại 2 hướng đường sáng, các đèn còn lại tắt, nếu người dùng nhấn nút **INC**, hệ thống sẽ chuyển sang chế độ **INC RED** để điều chỉnh thời gian sáng của các đèn đỏ, và trạng thái của nút nhấn **INC** được reset bằng hàm **fsmButtonReset()**, để hiện thực chức năng nhấn đè (sẽ được trình bày sau). Mặt khác, nếu tại chế độ **MANU RED**, người dùng nhấn nút **SET** thì hệ thống sẽ chuyển sang chế độ **MANU GREEN**. Cơ chế tương tự cũng được áp dụng cho chế độ **MANU GREEN** và **MANU YELLOW**.

- Hàm **fsmTuningModeRun** được hiện thực như sau:

```
1  void fsmTuningModeRun()
2  {
3    switch (mode)
4    {
5      case INC_RED:
6        // Switch to AUTO MODE when press MODE button
7        if (isButtonPressed(BUTTON_MODE))
8        {
9          mode = AUTO_MODE;
10         resetLight();
11       }
12       // Set the time duration of the red LEDs
13       //when press SET button
14       if(isButtonPressed(BUTTON_SET))
15       {
16         RED_time = time_count * 1000;
17       }
18
19   // Increase the time duration of the red LEDs
20   // when INC button is pressed or auto increased the red
21   //LEDs every 500ms if button is pressed longer than 1 second
22       fsmButtonRun(BUTTON_INC);
23       break;
```

```
24
25      case INC_GREEN:
26        // Switch to AUTO MODE when press MODE button
27        if (isButtonPressed(BUTTON_MODE))
28        {
29          mode = AUTO_MODE;
30          resetLight();
31        }
32
33        // Set the time duration of the green LEDs
34        //when press SET button
35        if(isButtonPressed(BUTTON_SET))
36        {
37          GREEN_time = time_count * 1000;
38        }
39
40      // Increase the time duration of the green LEDs
41      // when INC button is pressed or auto increased the green
42  //LEDs every 500ms if button is pressed longer than 1 second
43        fsmButtonRun(BUTTON_INC);
44        break;
45
46      case INC_YELLOW:
47        // Switch to AUTO MODE when press MODE button
48        if (isButtonPressed(BUTTON_MODE))
49        {
50          mode = AUTO_MODE;
51          resetLight();
52        }
53        // Set the time duration of the yellow LEDs
54        //when press SET button
55        if(isButtonPressed(BUTTON_SET))
56        {
57          YELLOW_time = time_count * 1000;
58        }
59
60        // Increase the time duration of the red LEDs
61        // when INC button is pressed or auto increased the red
62        // LEDs every 500ms if button is pressed longer than 1
     second
63        fsmButtonRun(BUTTON_INC);
64        break;
65
66      default:
67        break;
68    }
69  }
70
```

Program 5: Hàm **fsmTuningModeRun** được hiện thực trong file **fsm_tuning_mode.c**

Khi hệ thống đang ở chế độ **INC RED**, nếu người dùng nhấn nút **MODE** thì hệ thống sẽ trở về trạng thái **AUTO MODE**, các đèn tại hai hướng đường được reset về trạng thái ban đầu thông qua hàm **resetLight()**. Mặt khác, nếu người dùng nhấn nút **SET**, hệ thống sẽ cập nhật thời gian sáng của các đèn đỏ từ giá trị đang được điều chỉnh trong chế độ **TUNING MODE** này (biến **time_count**). Cuối cùng, nếu người dùng nhấn nút **INC**, giá trị của biến **time_count** sẽ được thay đổi theo ý tưởng đã nêu ở phần 2, hiện thực bởi hàm **fsmButtonRun()** trong file **f_button.c**, các cơ chế này cũng được áp dụng tương tự khi hệ thống ở chế độ **INC GREEN** và **INC YELLOW**.

```
#include "f_button.h"

int button_flag[NUM_OF_BUTTONS];
int button_long_flag[NUM_OF_BUTTONS];
GPIO_PinState key_reg[NUM_OF_BUTTONS][4];
int counter_for_key_pressed[NUM_OF_BUTTONS];
int button_status[NUM_OF_BUTTONS];
int button_released[NUM_OF_BUTTONS];

int isButtonReleased(int button)
{
  if (button_released[button])
  {
    button_released[button] = 0;
    return 1;
  }

  else return 0;
}

int isButtonPressed(int button)
{
  if (button_flag[button])
  {
    button_flag[button] = 0;
    return 1;
  }

  else return 0;
}

int isButtonLongPressed(int button)
{
  if (button_long_flag[button])
  {
    button_long_flag[button] = 0;
    return 1;
  }

  else return 0;
}
```

```
42
43 void subKeyProcess(int button)
44 {
45   button_flag[button] = 1;
46 }
47
48 void subKeyLongProcess(int button)
49 {
50   button_long_flag[button] = 1;
51 }
52
53 void resetKey(int button)
54 {
55   button_flag[button] = 0;
56   button_long_flag[button] = 0;
57 }
58
59 void setKeyTimer(int button)
60 {
61   counter_for_key_pressed[button] = timer_for_key_pressed /
      timer_cycle;
62 }
63
64 void getKeyInput()
65 {
66   for (int button = 0; button < NUM_OF_BUTTONS; button++)
67   {
68     key_reg[button][0] = key_reg[button][1];
69     key_reg[button][1] = key_reg[button][2];
70     switch (button)
71     {
72       case BUTTON_MODE:
73         key_reg[button][2] = HAL_GPIO_ReadPin(
     BUTTON_MODE_GPIO_Port, BUTTON_MODE_Pin);
74         break;
75       case BUTTON_SET:
76         key_reg[button][2] = HAL_GPIO_ReadPin(
     BUTTON_SET_GPIO_Port, BUTTON_SET_Pin);
77         break;
78       case BUTTON_INC:
79         key_reg[button][2] = HAL_GPIO_ReadPin(
     BUTTON_INC_GPIO_Port, BUTTON_INC_Pin);
80         break;
81       case BUTTON_PED_HOR:
82         key_reg[button][2] = HAL_GPIO_ReadPin(
     BUTTON_PED_HOR_GPIO_Port, BUTTON_PED_HOR_Pin);
83         break;
84       case BUTTON_PED_VER:
85         key_reg[button][2] = HAL_GPIO_ReadPin(
```

```
    BUTTON_PED_VER_GPIO_Port, BUTTON_PED_VER_Pin);
86        break;
87      default:
88        break;
89    }
90
91    if ((key_reg[button][0] == key_reg[button][1]) && (key_reg
    [button][1] == key_reg[button][2]))
92    {
93
94      // Press button, then release
95      if (key_reg[button][3] != key_reg[button][2])
96      {
97        key_reg[button][3] = key_reg[button][2];
98
99        if (key_reg[button][2] == PRESSED_STATE)
100       {
101         subKeyProcess(button);
102         setKeyTimer(button);
103         button_released[button] = 0;
104       } else {
105         resetKey(button);
106         button_released[button] = 1;
107       }
108     }
109     // Press and hold button
110     else
111     {
112       counter_for_key_pressed[button]--;
113       if (counter_for_key_pressed[button] <= 0)
114       {
115         if (key_reg[button][2] == PRESSED_STATE)
116         {
117           subKeyLongProcess(button);
118           button_released[button] = 0;
119         }
120         else resetKey(button);
121         setKeyTimer(button);
122       }
123     }
124   }
125 }
126 }
127
128 void buttonProcessing()
129 {
130   if (isTimerUp(7))
131   {
132     setTimer(7, BUTTON_TIME_STEP);
```

```
133     getKeyInput();
134   }
135 }
136 
137 void fsmButtonReset(int button)
138 {
139   button_status[button] = BUTTON_INIT;
140 }
141 
142 void fsmButtonRun(int button)
143 {
144   switch (button_status[button])
145   {
146     case BUTTON_INIT:
147       button_status[button] = BUTTON_RELEASED;
148     case BUTTON_RELEASED:
149       if (isButtonPressed(button))
150       {
151         button_status[button] = BUTTON_PRESSED;
152         ++time_count;
153         if (time_count > 99)
154         {
155           time_count = 1;
156         }
157       }
158       break;
159     case BUTTON_PRESSED:
160       if (isButtonReleased(button))
161       {
162         button_status[button] = BUTTON_RELEASED;
163       }
164       if (isButtonLongPressed(button))
165       {
166         setTimer(8, BUTTON_AUTO_INC_TIME);
167         button_status[button] = BUTTON_LONG_PRESSED;
168       }
169       break;
170     case BUTTON_LONG_PRESSED:
171       if (isTimerUp(8))
172       {
173         setTimer(8, BUTTON_AUTO_INC_TIME);
174         ++time_count;
175         if (time_count > 99)
176         {
177           time_count = 1;
178         }
179       }
180       if (isButtonReleased(button))
181       {
```

```
182         button_status[button] = BUTTON_RELEASED;
183       }
184       break;
185   }
186 }
187
```

Program 6: Module **f_button**

Trong module này, hàm **getKeyInput()** được sử dụng để đọc trạng thái của các nút nhấn thực hiện kỹ thuật chống rung nút nhấn (button deboucning) thông qua việc lưu trữ 4 trạng thái gần nhất đọc từ được mỗi nút nhấn vào mảng **keyReg** và so sánh các trạng thái đó. Các hàm **isButtonReleased()**, **isButtonPressed()**, **isButtonLongPressed()** lần lượt có nhiệm vụ kiểm tra nút nhấn có đang được thả, được nhấn và được nhấn đè (trong 1 giây hoặc lâu hơn). Hàm **buttonProcessing()** được gọi trong vòng lặp super loop để hiện thực việc đọc nút nhấn mỗi *10ms* sử dụng một software timer. Hàm **fsmButtonReset()** được sử dụng để reset lại trạng thái ban đầu cho FSM của mỗi nút nhấn trong khi **fsmButtonRun()** được sử dụng để hiện thực các FSM đó.

- Hàm **fsmPedestrianModeRun()** được hiện thực như sau:

```
// configure the running environment of states
void fsmPedestrianModeRun()
{
  fsmVerPedRun();
  fsmHorPedRun();

  switch (ver_ped)
  {
    case PED_OFF:
      HAL_GPIO_WritePin(LED_RED_P2_GPIO_Port, LED_RED_P2_Pin,
   LED_OFF);
      HAL_GPIO_WritePin(LED_GREEN_P2_GPIO_Port,
   LED_GREEN_P2_Pin, LED_OFF);
      //The vertical buzzer is off
      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 0);
      break;
    case PED_ON:
      HAL_GPIO_WritePin(LED_RED_P2_GPIO_Port, LED_RED_P2_Pin,
   LED_OFF);
      HAL_GPIO_WritePin(LED_GREEN_P2_GPIO_Port,
   LED_GREEN_P2_Pin, LED_OFF);
      //The vertical buzzer is off
      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 0);
      break;
    case WALK_STOP:
      HAL_GPIO_WritePin(LED_RED_P2_GPIO_Port, LED_RED_P2_Pin,
   LED_ON);
      HAL_GPIO_WritePin(LED_GREEN_P2_GPIO_Port,
   LED_GREEN_P2_Pin, LED_OFF);
      if (isTimerUp(3))
```

```
25      {
26        setTimer(3, BUZZER_time_stop);
27        switch (ver_buzzer_state)
28        {
29          case BUZZER_ON:
30            ver_buzzer_state = BUZZER_OFF;
31            __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 0);
32            break;
33          case BUZZER_OFF:
34            ver_buzzer_state = BUZZER_ON;
35            __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2,
    BUZZER_pul_stop);
36            break;
37          default:
38            break;
39        }
40      }
41      break;
42    case WALK_ALLOW:
43      HAL_GPIO_WritePin(LED_RED_P2_GPIO_Port, LED_RED_P2_Pin,
    LED_OFF);
44      HAL_GPIO_WritePin(LED_GREEN_P2_GPIO_Port,
    LED_GREEN_P2_Pin, LED_ON);
45      if (isTimerUp(3))
46      {
47        setTimer(3, BUZZER_time_allow);
48        switch (ver_buzzer_state)
49        {
50          case BUZZER_ON:
51            ver_buzzer_state = BUZZER_OFF;
52            __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 0);
53            break;
54          case BUZZER_OFF:
55            ver_buzzer_state = BUZZER_ON;
56            __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2,
    BUZZER_pul_allow);
57            break;
58          default:
59            break;
60        }
61      }
62      break;
63    case BUZZER_LOUDER:
64      HAL_GPIO_WritePin(LED_RED_P2_GPIO_Port, LED_RED_P2_Pin,
    LED_OFF);
65      if (isTimerUp(5))
66      {
67        setTimer(5, PED_LED_TOGGLE_DURATION);
68        HAL_GPIO_TogglePin(LED_GREEN_P2_GPIO_Port,
```

```
   LED_GREEN_P2_Pin);
69      }
70
71      if (isTimerUp(3))
72      {
73        switch (ver_buzzer_state)
74        {
75          case BUZZER_ON:
76            ver_buzzer_state = BUZZER_OFF;
77            __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, 0);
78            break;
79          case BUZZER_OFF:
80            // Buzzer is louder and faster after each cycle
81            ver_buzzer_state = BUZZER_ON;
82            if (ver_buzzer_time - 200 >= 200)
83            {
84              ver_buzzer_time -= 200;
85            }
86            if (ver_pul + 100 <= 950)
87            {
88              ver_pul += 100;
89            }
90            __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2,
   ver_pul);
91            break;
92          default:
93            break;
94        }
95        setTimer(3, ver_buzzer_time);
96      }
97      break;
98    default:
99      break;
100 }
101
102 switch (hor_ped)
103 {
104   case PED_OFF:
105     HAL_GPIO_WritePin(LED_RED_P1_GPIO_Port, LED_RED_P1_Pin,
   LED_OFF);
106     HAL_GPIO_WritePin(LED_GREEN_P1_GPIO_Port,
   LED_GREEN_P1_Pin, LED_OFF);
107     //The horizontal buzzer is off
108     __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, 0);
109     break;
110   case PED_ON:
111     HAL_GPIO_WritePin(LED_RED_P1_GPIO_Port, LED_RED_P1_Pin,
   LED_OFF);
112     HAL_GPIO_WritePin(LED_GREEN_P1_GPIO_Port,
```

```
    LED_GREEN_P1_Pin, LED_OFF);
113       //The horizontal buzzer is off
114       __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, 0);
115       break;
116     case WALK_STOP:
117       HAL_GPIO_WritePin(LED_RED_P1_GPIO_Port, LED_RED_P1_Pin,
    LED_ON);
118       HAL_GPIO_WritePin(LED_GREEN_P1_GPIO_Port,
    LED_GREEN_P1_Pin, LED_OFF);
119       if (isTimerUp(2))
120       {
121         setTimer(2, BUZZER_time_stop);
122         switch (hor_buzzer_state)
123         {
124           case BUZZER_ON:
125             hor_buzzer_state = BUZZER_OFF;
126             __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, 0);
127             break;
128           case BUZZER_OFF:
129             hor_buzzer_state = BUZZER_ON;
130             __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1,
    BUZZER_pul_stop);
131             break;
132           default:
133             break;
134         }
135       }
136       break;
137     case WALK_ALLOW:
138       HAL_GPIO_WritePin(LED_RED_P1_GPIO_Port, LED_RED_P1_Pin,
    LED_OFF);
139       HAL_GPIO_WritePin(LED_GREEN_P1_GPIO_Port,
    LED_GREEN_P1_Pin, LED_ON);
140       if (isTimerUp(2))
141       {
142         setTimer(2, BUZZER_time_allow);
143         switch (hor_buzzer_state)
144         {
145           case BUZZER_ON:
146             hor_buzzer_state = BUZZER_OFF;
147             __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, 0);
148             break;
149           case BUZZER_OFF:
150             hor_buzzer_state = BUZZER_ON;
151             __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1,
    BUZZER_pul_allow);
152             break;
153           default:
154             break;
```

```
155         }
156       }
157       break;
158     case BUZZER_LOUDER:
159       HAL_GPIO_WritePin(LED_RED_P1_GPIO_Port, LED_RED_P1_Pin,
    LED_OFF);
160       if (isTimerUp(4))
161       {
162         setTimer(4, PED_LED_TOGGLE_DURATION);
163         HAL_GPIO_TogglePin(LED_GREEN_P1_GPIO_Port,
    LED_GREEN_P1_Pin);
164       }
165 
166       if (isTimerUp(2))
167       {
168         switch (hor_buzzer_state)
169         {
170           case BUZZER_ON:
171             hor_buzzer_state = BUZZER_OFF;
172             __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, 0);
173             break;
174           case BUZZER_OFF:
175             // Buzzer is louder and faster after each cycle
176             hor_buzzer_state = BUZZER_ON;
177             if (hor_buzzer_time - 200 >= 200)
178             {
179               hor_buzzer_time -= 200;
180             }
181             if (hor_pul + 100 <= 950)
182             {
183               hor_pul += 100;
184             }
185             __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1,
    hor_pul);
186             break;
187           default:
188             break;
189         }
190         setTimer(2, hor_buzzer_time);
191       }
192       break;
193     default:
194       break;
195   }
196 }
```

Program 7: Hàm **fsmPedestrianModeRun()** trong file **fsm_pedestrian_mode.c**

Hàm này sẽ cấu hình trạng thái của hệ thống cho người đi bộ tại mỗi hướng đường cho phù hợp với từng trạng thái nêu tại phần 2:

- Nếu hệ thống đang ở trạng thái **PEDESTRIAN ON** hoặc **PEDESTRIAN OFF**, hệ thống sẽ tắt các đèn và loa.
- Nếu hệ thống ở trạng thái **WALK STOP**, đèn đỏ sẽ sáng để cảnh báo người đi bộ không được qua đường. Đồng thời, loa sẽ được kêu theo chu kỳ với một nửa thời gian kêu và một nửa thời gian tắt, chu kỳ này sẽ được hiện thực bởi một software timer. Bên cạnh đó, hàm **__HAL_TIM_SetCompare()** được sử dụng để điều chỉnh duty cycle cho từng chân PWM với 3 tham số là handler của timer, kênh của timer cho tín hiệu PWM và giá trị của counter để chân PWM chuyển mức logic. (theo cài đặt ở phần 3, chu kỳ counter của timer 3 là 999 nên tham số nên nằm trong khoảng từ 0 tới 999).
- Nếu hệ thống ở trạng thái **WALK ALLOW**, đèn xanh sẽ sáng để thông báo người đi bộ có thể qua đường, và loa cũng sẽ kêu theo cơ chế tương tự trạng thái **WALK STOP**, nhưng với chu kỳ và cường độ khác.
- Nếu hệ thống ở trạng thái **BUZZER LOUDER**, đèn xanh sẽ nhấp nháy (hiện thực bằng một software timer) và loa sẽ kêu càng ngày càng lớn và càng ngày càng nhanh để cảnh báo người đi bộ thời gian qua đường sắp kết thúc. Cụ thể, sau mỗi chu kỳ của buzzer, giá trị counter truyền vào hàm **__HAL_TIM_SetCompare()** được tăng lên để tăng cường độ của loa và chu kỳ của software timer cho buzzer được giảm đi để giảm chu kỳ của loa.

Trước khi cấu hình trạng thái của hệ thống, hàm **fsmPedestrianModeRun()** gọi lần lượt 2 hàm **fsmVerPedRun()** và **fsmHorPedRun()** để kiểm tra các điều kiện chuyển trạng thái và cấu hình các điều kiện ban đầu của trạng thái được chuyển (nếu có) trong hệ thống dành cho người đi bộ tại hướng đường dọc và ngang theo FSM nêu tại phần 2.

```
// set up the initial environment of each state (if state transitions occur)
void fsmVerPedRun()
{
  switch (ver_ped)
  {

    case PED_OFF:
      if (isButtonPressed(BUTTON_PED_VER) && mode == AUTO_MODE)
      {
        ver_ped = PED_ON;
      }
      break;

    case PED_ON:
      switch (vertical_state)
      {
        case AUTO_GREEN:
        case AUTO_YELLOW:
          setTimer(3, BUZZER_time_stop);
          __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2, BUZZER_pul_stop);
          ver_buzzer_state = BUZZER_ON;
```

```
22          ver_ped = WALK_STOP;
23          break;
24        case AUTO_RED:
25          setTimer(3, BUZZER_time_allow);
26          __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2,
    BUZZER_pul_allow);
27          ver_buzzer_state = BUZZER_ON;
28          ver_ped = WALK_ALLOW;
29          break;
30        default:
31          break;
32      }
33      if (mode != AUTO_MODE)
34      {
35        ver_ped = PED_OFF;
36      }
37      break;
38
39    case WALK_STOP:
40      if (vertical_state == AUTO_RED)
41      {
42        setTimer(3, BUZZER_time_allow);
43        __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_2,
    BUZZER_pul_allow);
44        ver_buzzer_state = BUZZER_ON;
45        ver_ped = WALK_ALLOW;
46      }
47      if (mode != AUTO_MODE)
48      {
49        ver_ped = PED_OFF;
50      }
51      break;
52
53    case WALK_ALLOW:
54      if (currentCounter(1) <= TIME_LEFT_FOR_LOUDER && mode ==
     AUTO_MODE)
55      {
56        setTimer(5, PED_LED_TOGGLE_DURATION);
57        ver_buzzer_time = BUZZER_time_allow - 200;
58        setTimer(3, ver_buzzer_time);
59        ver_pul = BUZZER_pul_allow + 100;
60        __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, ver_pul);
61        ver_buzzer_state = BUZZER_ON;
62        ver_ped = BUZZER_LOUDER;
63      }
64      if (mode != AUTO_MODE)
65      {
66        ver_ped = PED_OFF;
67      }
```

```
68        break;
69
70      case BUZZER_LOUDER:
71        if (vertical_state == AUTO_GREEN || mode != AUTO_MODE)
72        {
73          ver_ped = PED_OFF;
74        }
75        break;
76
77      default:
78        break;
79    }
80  }
81
```

Program 8: Hàm **fsmVerPedRun()** trong file **fsm_pedestrian_mode.c**

```
1  // set up the initial environment of each state (if state
       transitions occur)
2  void fsmHorPedRun()
3  {
4    switch (hor_ped)
5    {
6      case PED_OFF:
7        if (isButtonPressed(BUTTON_PED_HOR) && mode == AUTO_MODE
       )
8        {
9          hor_ped = PED_ON;
10       }
11       break;
12     case PED_ON:
13       switch (horizontal_state)
14       {
15         case AUTO_GREEN:
16         case AUTO_YELLOW:
17           setTimer(2, BUZZER_time_stop);
18           __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1,
       BUZZER_pul_stop);
19           hor_buzzer_state = BUZZER_ON;
20           hor_ped = WALK_STOP;
21           break;
22         case AUTO_RED:
23           setTimer(2, BUZZER_time_allow);
24           __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1,
       BUZZER_pul_allow);
25           hor_buzzer_state = BUZZER_ON;
26           hor_ped = WALK_ALLOW;
27           break;
28         default:
29           break;
```

```
30        }
31        if (mode != AUTO_MODE)
32        {
33          hor_ped = PED_OFF;
34        }
35        break;
36      case WALK_STOP:
37        if (horizontal_state == AUTO_RED)
38        {
39          setTimer(2, BUZZER_time_allow);
40          __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1,
    BUZZER_pul_allow);
41          hor_buzzer_state = BUZZER_ON;
42          hor_ped = WALK_ALLOW;
43        }
44        if (mode != AUTO_MODE)
45        {
46          hor_ped = PED_OFF;
47        }
48        break;
49      case WALK_ALLOW:
50        if (currentCounter(0) <= TIME_LEFT_FOR_LOUDER && (mode
    == AUTO_MODE))
51        {
52          setTimer(4, PED_LED_TOGGLE_DURATION);
53          hor_buzzer_time = BUZZER_time_allow - 200;
54          setTimer(2, hor_buzzer_time);
55          hor_pul = BUZZER_pul_allow + 100;
56          __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, hor_pul);
57          hor_buzzer_state = BUZZER_ON;
58          hor_ped = BUZZER_LOUDER;
59        }
60        if (mode != AUTO_MODE)
61        {
62          hor_ped = PED_OFF;
63        }
64        break;
65      case BUZZER_LOUDER:
66        if (mode != AUTO_MODE || horizontal_state == AUTO_GREEN)
67        {
68          hor_ped = PED_OFF;
69        }
70        break;
71      default:
72        break;
73    }
74  }
```

Program 9: Hàm **fsmHorPedRun()** trong file **fsm_pedestrian_mode.c**

- Hàm **uartProcessing()** được hiện thực như sau:

```
#include "f_uart.h"

int save_time_count = -1;

void uartTransmit7SEG(uint8_t counter)
{
  char str[50];
  sprintf(str, "!7SEG:%02d#", counter);
  uint8_t str_len = (uint8_t)strlen(str);
  for (uint8_t i = 0; i < str_len; ++i)
  {
    HAL_UART_Transmit(&huart2, (uint8_t*)(str + i), 1, 50);
  }
}
void uartProcessing()
{
  // The receiving application only accept message of the form
    "!7SEG:xx#"
  switch (mode)
  {
    case AUTO_MODE:
      // Only transmit horizontal counter value
      if (isTimerUp(6))
      {
        uint8_t formatted_counter = (currentCounter(0) / 1000)
    % 100;
        uartTransmit7SEG(formatted_counter);
        setTimer(6, UART_TIME_STEP);
      }
      break;
    case INC_RED:
    case INC_GREEN:
    case INC_YELLOW:
      // Print TUNING value only there's a change
      if (time_count != save_time_count)
      {
        uartTransmit7SEG(time_count);
        save_time_count = time_count;
      }
      break;
    default:
      // Reset for future printing of TUNING value
      save_time_count = -1;
      break;
  }
}
```

Program 10: Module **f_uart**

Nếu hệ thống đèn giao thông đang ở chế độ **AUTO MODE**, hệ thống sẽ truyền bộ đếm xuống tại hướng đường ngang (truy xuất từ hàm **currentCounter()** của module **f_timer**) mỗi 1 giây (hiện thực bằng một software timer) qua kết nối UART. Mặt khác, nếu hệ thống đang ở chế độ **TUNING MODE** (một trong 3 trạng thái **INC RED**, **INC GREEN** hoặc **INC YELLOW**, hệ thống sẽ truyền qua kết nối UART giá trị "TUNING" khi nó có sự thay đổi. Thao tác truyền qua kết nối UART được hiện thực bằng hàm **uartTransmit7SEG()**.

- Hàm **buttonProcessing()** được hiện thực ở chương trình 6 đã được đề cập ở trên.

Bên cạnh đó, module **f_timer** được sử dụng để hiện thực các software timer như sau:

```c
#include "f_timer.h"

int timer_counter[NUM_OF_TIMER];
int timer_flag[NUM_OF_TIMER];

int currentCounter(int timer)
{
  return timer_counter[timer];
}

int isTimerUp(int timer)
{
  return (timer_flag[timer] == 1);
}

void setTimer(int timer, int duration)
{
  timer_counter[timer] = duration / timer_cycle;
  timer_flag[timer] = 0;
}

void timerRun()
{
  for (int timer = 0; timer < NUM_OF_TIMER; timer++)
  {
    if (timer_counter[timer] > 0)
    {
      timer_counter[timer] --;
      if (timer_counter[timer] == 0)
      {
        timer_flag[timer] = 1;
      }
    }
  }
}
```

Program 11: Module **f_timer**

Cuối cùng, module **f_global** được sử dụng để khai báo và định nghĩa các biến toàn cục và hằng số của chương trình.

```
#ifndef INC_F_GLOBAL_H_
#define INC_F_GLOBAL_H_

#include "main.h"

/*
 * Define global Variables
 */

extern int timer_cycle;
extern int timer_for_key_pressed;
extern int mode;

extern int RED_time;
extern int YELLOW_time;
extern int GREEN_time;

/*
 * Define for AUTO MODE
 */

#define INIT       0
#define AUTO_MODE    1
#define AUTO_RED     2
#define AUTO_YELLOW    3
#define AUTO_GREEN     4

/*
 * Define for MANUAL MODE
 */
#define MANU_MODE    5
#define MANU_RED     6
#define MANU_YELLOW    7
#define MANU_GREEN     8

/*
 * Define for TUNING MODE
 */
#define INC_RED      9
#define INC_YELLOW     10
#define INC_GREEN    11

/*
 * Define for PEDESTRIAN MODE
 */
#define PED_OFF          12
#define PED_ON           13
```

```
48 #define WALK_ALLOW         14
49 #define BUZZER_LOUDER      15
50 #define WALK_STOP        16
51 #define PED_LED_TOGGLE_DURATION 500
52
53 /*
54  * Define for LEDs
55  */
56 #define LED_ON     GPIO_PIN_RESET
57 #define LED_OFF    GPIO_PIN_SET
58
59 /*
60  * Define for Buzzer
61  */
62 #define BUZZER_time_allow    1000
63 #define BUZZER_time_stop     500
64 #define BUZZER_pul_allow     300
65 #define BUZZER_pul_stop      900
66 #define BUZZER_OFF         0
67 #define BUZZER_ON        1
68 #define TIME_LEFT_FOR_LOUDER  5000
69
70 /*
71  * Define for Timer
72  * Timer0 for horizontal LEDs, Timer1 for vertical LEDs,
73  * Timer2 for horizontal buzzer, Timer3 for vertical buzzer
74  * Timer4 for pedestrian horizontal LED toggling
75  * Timer5 for pedestrian horizontal LED toggling
76  * Timer6 for UART transmitting
77  * Timer7 for buttons reading
78  * Timer8 for button auto increment
79  */
80 #define NUM_OF_TIMER  9
81
82 /*
83  * Define for Button
84  */
85 #define NUM_OF_BUTTONS    5
86
87 #define BUTTON_MODE      0
88 #define BUTTON_SET       1
89 #define BUTTON_INC       2
90 #define BUTTON_PED_VER     3
91 #define BUTTON_PED_HOR     4
92 #define BUTTON_TIME_STEP   10
93
94 #define NORMAL_STATE   GPIO_PIN_SET
95 #define PRESSED_STATE GPIO_PIN_RESET
96
```

```
97  #define BUTTON_PRESSED    0
98  #define BUTTON_LONG_PRESSED 1
99  #define BUTTON_RELEASED   2
100 #define BUTTON_INIT     4
101
102 #define BUTTON_AUTO_INC_TIME  500
103
104 /*
105  * Define for UART
106  */
107 #define UART_TIME_STEP 1000
108
109 #endif /* INC_F_GLOBAL_H_ */
```

Program 12: File **f_global.h**

```
1  #include "f_global.h"
2
3  int timer_cycle = 1;
4  int timer_for_key_pressed = 1000;
5
6  int mode = INIT;
7
8  int RED_time = 10000;
9  int GREEN_time = 7000;
10 int YELLOW_time = 3000;
```

Program 13: File **f_global.c**

Source code được lưu trữ tại GitHub: Github_Final Project.