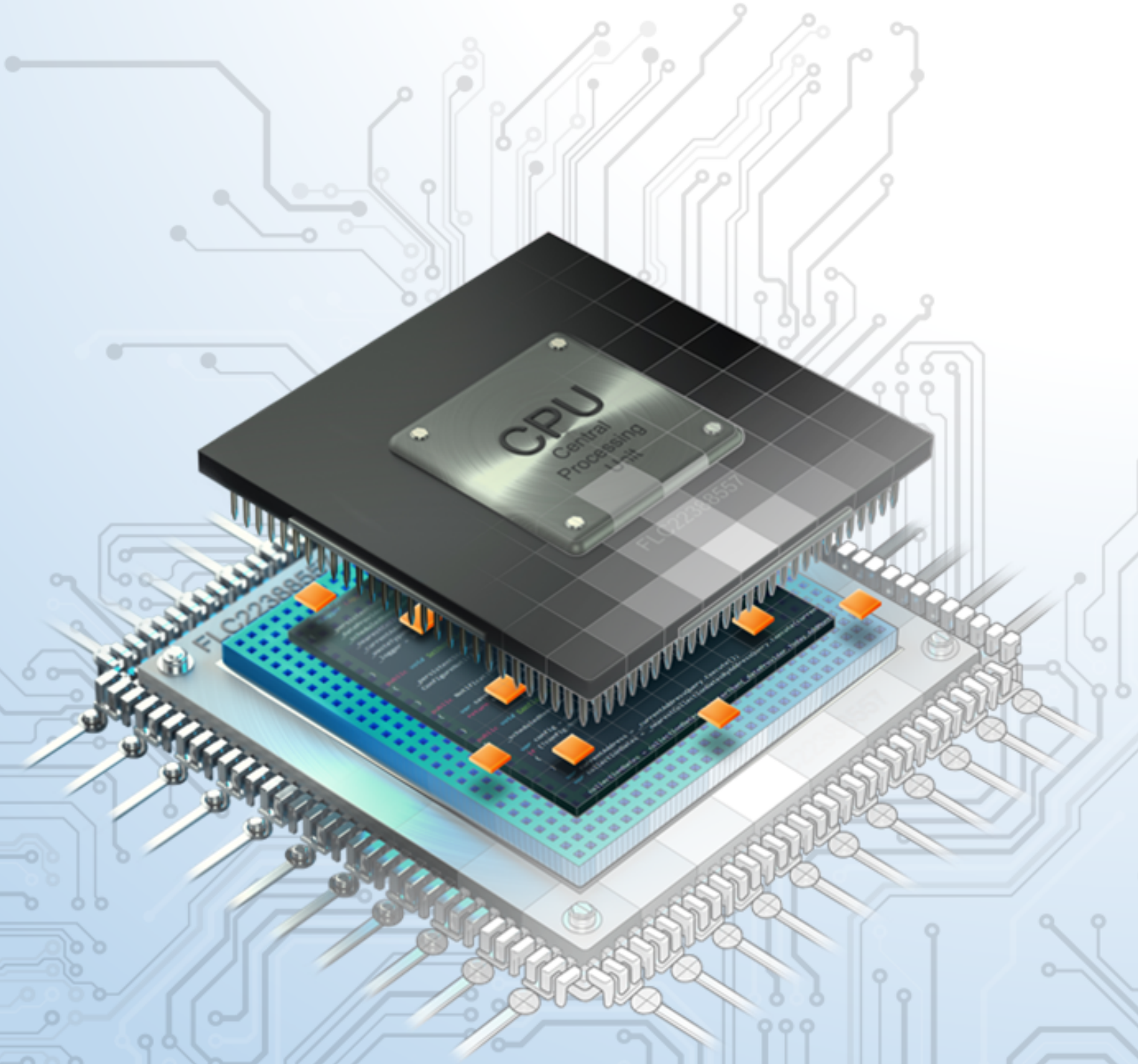
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller



Dr. Le Trong Nhan

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KĨ THUẬT MÁY TÍNH

VI XỬ LÍ (CO3010)

FINAL REPORT

Giáo viên hướng dẫn: Lê Trọng Nhân

Sinh viên thực hiện: Võ Hoài Nam - 2013834
Hà Thị Huyền -2013344
Tô Dịu Quang -2014251

Github: `https://github.com/namyouwon/Project-MCU/tree/mint812`

Tp. Hồ Chí Minh,Tháng 12/2023

Mục lục

1 Yêu cầu 4
1.1 Mô tả đề tài 4
1.2 Yêu cầu đề bài 4
2 Máy trạng thái (FSM) 5
2.1 Hệ thống 5
2.2 Mô tả chi tiết 5
3 Cấu trúc file 7
4 Hiện thực 8
4.1 Buzzer 8
4.2 Button 8
4.3 UART 10
4.4 FSM 11
4.5 Main 19

1 Yêu cầu

1.1 Mô tả đề tài

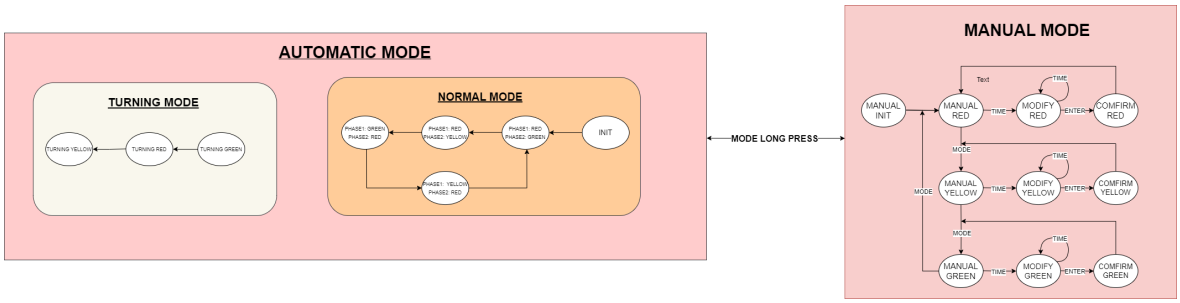
Sử dụng STM32F103RB để xây dựng hệ thống đèn giao thông, có chế độ bình thường (normal mode), chế độ người đi bộ (perdestrian mode) và chế độ thiết lập thủ công(manual mode).

1.2 Yêu cầu đề bài

- Chế độ bình thường (normal mode): hoạt động bình thường với bộ đếm được tự đọng thiết lập.
- Chế độ điều khiển thủ công (manual mode): khi chuyển sang chế độ này, người điều khiển có thể kích hoạt các đèn giao thông sáng theo ý của mình dựa vào việc tương tác mới nút nhấn.
- Chế độ người đi bộ (turning mode): đây là trạng thái dành cho người đi bộ khi muốn đi qua đường.

2 Máy trạng thái (FSM)

2.1 Hệ thống



Hình 1: Máy trạng thái hệ thống)

- Mỗi đường sẽ được thiết kế gồm có:
  - 1 đèn đỏ (RED LED), 1 đèn xanh (GREEN LED), 1 đèn vàng(YELLOW LED) hiển thị trạng thái cho các phương tiện di chuyển trên đường
  - 1 đèn đỏ (RED LED), 1 đèn xanh (GREEN LED), 1 đèn vàng(YELLOW LED) hiển thị trạng thái của người đi bộ.
  - 1 chuông (buzzer) cảnh báo tính trạng của người đi bộ.
  - 1 nút nhấn (**PERDES**) để cho người đi bộ có thể nhấn khi muốn xin qua đường.
- Một tủ điều khiển trung tâm dùng để điều khiển chung cho ngã tư gồm có các nút nhấn:
  - **MODE:** dùng để chuyển đổi giữa các chế độ hệ thống hoặc các trạng thái đèn với nhau.
  - **TIME:** dùng để chọn đèn hiện đang được chỉ định để thiết lập thời gian
  - **ENTER:** dùng để lưu thay đổi khi điều chỉnh thời gian của đèn.

2.2 Mô tả chi tiết

- **CHẾ ĐỘ BÌNH THƯỜNG (NORMAL MODE)**
  - Ở chế độ này, các đèn cho phương tiện tại mỗi đường hoạt động tự động.
  - Thời gian sáng của các đèn có thể điều chỉnh trong **MANUAL MODE)**
  - Ở chế độ, nếu nhấn giữ nút **MODE** trong 2s thì sẽ chuyển sang chế độ **MANUAL MODE**
- **CHẾ ĐỘ ĐIỀU KHIỂN THỦ CÔNG (MANUAL MODE)**
  - Ở chế độ này, người điều khiển có thể điều khiển trạng thái đèn hoặc thiết lập thời gian sáng của đèn.

- Khi người điều khiển muốn điều khiển đèn thủ công không theo thời gian, có thể nhấn nút **MODE** đèn dành cho phương tiện sẽ chuyển sang trạng thái tiếp theo.
- Khi người điều khiển muốn thay đổi thời gian sáng của đèn ở **NORMAL MODE**, chọn đèn cần thay đổi và nhấn nút **TIME**, đèn được chọn sẽ vào trạng thái **MODIFY**, người điều khiển có thể tăng thời gian sáng bằng việc nhấn nút **TIME** tương ứng với quy ước 1 lần nhấn = 1s. Sau khi hài lòng với thời gian thiết lập mới, có thể nhấn nút **ENTER** để lưu lại thời gian đã thiết lập.
- Nhấn giữ nút **MODE** trong 2s để trở về chế độ bình thường **NORMAL MODE**.

- **CHẾ ĐỘ CHO NGƯỜI ĐI BỘ (TURNING MODE**
  - Chế đợ này chỉ được kích hoạt khi đang cùng ở chế độ bình thường **NORMAL MODE**.
  - Khi người đi bộ nhấn nút **PERDES** trên đường mình đang di chuyển, thì đèn cho người đi bộ sẽ sáng ngược lại với đèn giao thông cho các phương tiện. Đồng thời chuông báo cũng sẽ được kích hoạt nếu thời gian cho người đi bộ qua đường chỉ còn 3s, và chuông báo sẽ kêu ngày càng nhanh tương ứng với việc thời gian sắp hết.

3 Cấu trúc file

- **button.c**: khai báo và xử lí nút nhấn
- **buzzer.c**: khai báo và xử lí chuông báo
- **software_timer.c**: khai báo và xử lí timer
- **gpio.c**: file config GPIO do IDE sinh code
- **traffic_light.c**: hàm xử lí fsm trong hệ thóng
- **uart.c**: hiện thực các hàm send uart để gửi giá trị
- **usart.c**: file config uart do IDE sinh code

4 Hiện thực

4.1 Buzzer

Hiện thực các hàm cơ bản của buzzer

```
uint8_t duty_cycle = 0;

void buzzer_init(){
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
}

void buzzer_SetVolume(uint8_t _duty_cycle){
  duty_cycle = _duty_cycle;
  __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1,duty_cycle);
}
```

Program 1: Source code hiện thực hàm buzzer.c

4.2 Button

```
#include "button.h"

int Keyred0[20] = {NORMAL_STATE};
int Keyred1[20] = {NORMAL_STATE};
int Keyred2[20] = {NORMAL_STATE};
int Keyred3[20] = {NORMAL_STATE};

int Key[20] = {NORMAL_STATE};
int MAX_KEY = 4;
int TimerForKeypress[20] = {200};
int flag_short[20] = {0};
int flag_long[20] = {0};

void readKey(){
  Key[0] = HAL_GPIO_ReadPin(A0_GPIO_Port, A0_Pin); //
   pedestrian
  Key[1] = HAL_GPIO_ReadPin(A1_GPIO_Port, A1_Pin); // mode
  Key[2] = HAL_GPIO_ReadPin(A2_GPIO_Port, A2_Pin); //time
  Key[3] = HAL_GPIO_ReadPin(A3_GPIO_Port, A3_Pin); //enter
}
void getKeyInput(){
  readKey();
  for(int i = 0; i < MAX_KEY; i++){
    Keyred0[i] = Keyred1[i];
    Keyred1[i] = Keyred2[i];
    Keyred2[i] = Key[i];
```

```
26      if((Keyred0[i] == Keyred1[i]) && (Keyred1[i] == Keyred2
    [i])){
27        if(Keyred3[i] != Keyred2[i]){
28          Keyred3[i] = Keyred2[i];
29          if(Keyred2[i] == PRESS_STATE){
30            //TODO
31            flag_short[i] = 1;
32            TimerForKeypress[i] = 200;
33          }
34        }else{
35          TimerForKeypress[i]--;
36          if(TimerForKeypress[i] == 0){
37            if(Keyred2[i] == PRESS_STATE){
38              //TODO
39              flag_long[i] = 1;
40            }
41            TimerForKeypress[i] = 200;
42          }
43        }
44      }else{
45        flag_long[i] = 0;
46        flag_short[i] = 0;
47      }
48    }
49  }
50
51  int isButtonPedes(){
52    if(flag_short[0]){
53      flag_short[0] = 0;
54      return 1;
55    }
56      return 0;
57  }
58
59  int isButtonMode(){
60    if(flag_short[1]){
61      flag_short[1] = 0;
62      return 1;
63    }
64      return 0;
65  }
66
67  int isModeLongPress(){
68    if(flag_long[1]){
69      flag_long[1] = 0;
70      return 1;
71    }
72      return 0;
73  }
```

```
74
75 int isButtonTime(){
76   if(flag_short[2]){
77     flag_short[2] = 0;
78     return 1;
79   }
80     return 0;
81 }
82
83 int isButtonEnter(){
84   if(flag_short[3]){
85     flag_short[3] = 0;
86     return 1;
87   }
88     return 0;
89 }
```

Program 2: Source code hiện thực button.c

4.3 UART

Trong phần này nhóm sẽ hiện thực để hệ thống gửi các hàm cơ bản của gửi uart

```
1  #include "uart.h"
2
3  //UART_HandleTypeDef huart2;
4
5  unsigned char temp = 0;
6  uint8_t msg[100];
7
8  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
9    if (huart->Instance == USART2){
10     HAL_UART_Transmit(&huart2, &temp, 1, 200);
11     HAL_UART_Receive_IT(&huart2, &temp, 1);
12   }
13 }
14
15 void sendString(uint8_t* str){
16   HAL_UART_Transmit(&huart2, (void*)msg, sprintf((void*)msg
     ,"%s",str), 10);
17 }
18
19 void sendNum(uint8_t num) {
20     char number[3];
21     sprintf(number, "%.2d", num);
22     HAL_UART_Transmit(&huart2, (uint8_t*)number, strlen(
     number), 10);
23 }
24
```

```c
void send7seg(uint8_t phase1, uint8_t phase2){
  sendString("!Phase 1: ");
  sendNum(phase1);
  sendString("; Phase 2: ");
  sendNum(phase2);
  sendString("#\r\n");
//  HAL_UART_Transmit(&huart2, (uint8_t*) "#\r\n", 3, 100);
}

void sendManual(uint8_t temp){
  sendString("!Temporary value: ");
  sendNum(temp);
  sendString("#\r\n");
}
```

Program 3: Source code hiện thực uart.c

4.4 FSM

Các máy trạng thái xử lý đều được nhóm đặt trong traffic_light.c

- **DEFINE**: Phần này nhóm sẽ define các trạng thái, các biến status, flag

```c
#define MANUAL_INIT     1
#define MANUAL_RED      2
#define MANUAL_GREEN    4
#define MANUAL_YELLOW   3

#define   INIT      100
#define   PHASE_2_GREEN   101 //Phase 1 red
#define   PHASE_2_YELLOW  102 //Phase 1 red
#define   PHASE_1_GREEN   103 //Phase 2 red
#define   PHASE_1_YELLOW  104 //Phase 2 red

#define TUNING_RED      10
#define TUNING_GREEN    11
#define TUNING_YELLOW   12

#define AUTOMATIC     15
#define MANUAL        16

uint8_t status_automation = INIT;
uint8_t count_red = 5;
uint8_t count_yellow = 2;
uint8_t count_green = 3;
uint8_t counter_normal = 0;
uint8_t num1 = 0, num2 = 0;

uint8_t status_manual = MANUAL_INIT;
```

```
uint8_t count_buffer = 0;

uint8_t flag_tuning = 0;
uint8_t count_tuning = 20;
uint8_t flag_color = 0;
uint8_t tog = 0;

uint8_t status_traffic  = AUTOMATIC;

```

Program 4: Hiện thực máy trạng thái trong hàm traffic_light.c

- **HANDLER**: Phần này sẽ hiện thực các hàm hiển thị trên led với mong muốn đơn giản hóa trong máy trạng thái

```
void Phase1_GreenOn(){
    HAL_GPIO_WritePin(D2_GPIO_Port, D2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_SET);
}
void Phase1_RedOn(){
    HAL_GPIO_WritePin(D2_GPIO_Port, D2_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_RESET);
}
void Phase1_YellowOn(){
    HAL_GPIO_WritePin(D2_GPIO_Port, D2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, GPIO_PIN_RESET);
}

void Phase2_GreenOn(){
    HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(D7_GPIO_Port, D7_Pin, GPIO_PIN_SET);
}
void Phase2_RedOn(){
    HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin, GPIO_PIN_SET);
```

```
24      HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin,
    GPIO_PIN_RESET);
25      HAL_GPIO_WritePin(D7_GPIO_Port, D7_Pin,
    GPIO_PIN_RESET);
26  }
27  void Phase2_YellowOn(){
28      HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin,
    GPIO_PIN_RESET);
29      HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin,
    GPIO_PIN_SET);
30      HAL_GPIO_WritePin(D7_GPIO_Port, D7_Pin,
    GPIO_PIN_RESET);
31  }
32
33  void Pedes_GreenOn(){
34      HAL_GPIO_WritePin(D8_GPIO_Port, D8_Pin,
    GPIO_PIN_RESET);
35      HAL_GPIO_WritePin(D9_GPIO_Port, D9_Pin,
    GPIO_PIN_RESET);
36      HAL_GPIO_WritePin(D10_GPIO_Port, D10_Pin,
    GPIO_PIN_SET);
37  }
38  void Pedes_RedOn(){
39      HAL_GPIO_WritePin(D8_GPIO_Port, D8_Pin,
    GPIO_PIN_SET);
40      HAL_GPIO_WritePin(D9_GPIO_Port, D9_Pin,
    GPIO_PIN_RESET);
41      HAL_GPIO_WritePin(D10_GPIO_Port, D10_Pin,
    GPIO_PIN_RESET);
42  }
43  void Pedes_YellowOn(){
44      HAL_GPIO_WritePin(D8_GPIO_Port, D8_Pin,
    GPIO_PIN_RESET);
45      HAL_GPIO_WritePin(D9_GPIO_Port, D9_Pin,
    GPIO_PIN_SET);
46      HAL_GPIO_WritePin(D10_GPIO_Port, D10_Pin,
    GPIO_PIN_RESET);
47  }
48
49  void Pedes_Off(){
50      HAL_GPIO_WritePin(D8_GPIO_Port, D8_Pin,
    GPIO_PIN_RESET);
51      HAL_GPIO_WritePin(D9_GPIO_Port, D9_Pin,
    GPIO_PIN_RESET);
52      HAL_GPIO_WritePin(D10_GPIO_Port, D10_Pin,
    GPIO_PIN_RESET);
53  }
54  void Blink_Green(){
55      HAL_GPIO_WritePin(D2_GPIO_Port, D2_Pin,
```

```
    GPIO_PIN_RESET);
56    HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin,
    GPIO_PIN_RESET);
57    HAL_GPIO_TogglePin(D4_GPIO_Port, D4_Pin);
58    HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin,
    GPIO_PIN_RESET);
59    HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin,
    GPIO_PIN_RESET);
60    HAL_GPIO_TogglePin(D7_GPIO_Port, D7_Pin);
61 }
62 void Blink_Red(){
63    HAL_GPIO_TogglePin(D2_GPIO_Port, D2_Pin);
64    HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin,
    GPIO_PIN_RESET);
65    HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin,
    GPIO_PIN_RESET);
66    HAL_GPIO_TogglePin(D5_GPIO_Port, D5_Pin);
67    HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin,
    GPIO_PIN_RESET);
68    HAL_GPIO_WritePin(D7_GPIO_Port, D7_Pin,
    GPIO_PIN_RESET);
69 }
70 void Blink_Yellow(){
71    HAL_GPIO_WritePin(D2_GPIO_Port, D2_Pin,
    GPIO_PIN_RESET);
72    HAL_GPIO_TogglePin(D3_GPIO_Port, D3_Pin);
73    HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin,
    GPIO_PIN_RESET);
74    HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin,
    GPIO_PIN_RESET);
75    HAL_GPIO_TogglePin(D6_GPIO_Port, D6_Pin);
76    HAL_GPIO_WritePin(D7_GPIO_Port, D7_Pin,
    GPIO_PIN_RESET);
77 }
```

- **NORMAL MODE**: Hiện thực máy trạng thái của trạng thái hiển thị bình thường

```
1  void normal_trafic(){
2    switch (status_automation){
3    case INIT:
4      counter_normal = count_green;
5      num1 = count_red;
6      num2 = count_green;
7      status_automation = PHASE_2_GREEN;
8      break;
9    case PHASE_2_GREEN:
10     counter_normal--;
11     Phase2_GreenOn();
12     Phase1_RedOn();
13     flag_color = TUNING_GREEN;
```

```
14      if (counter_normal <= 0){
15        counter_normal = count_yellow;
16        num1--;
17        num2 = count_yellow;
18        status_automation = PHASE_2_YELLOW;
19        Phase2_YellowOn();
20        Phase1_RedOn();
21        flag_color = TUNING_YELLOW;
22      }
23      else{
24        num1--;
25        num2--;
26      }
27      break;
28    case PHASE_2_YELLOW:
29      counter_normal--;
30      Phase2_YellowOn();
31      Phase1_RedOn();
32      flag_color = TUNING_YELLOW;
33      if (counter_normal <= 0) {
34        counter_normal = count_green;
35        status_automation = PHASE_1_GREEN;
36        num1 = count_green;
37        num2 = count_red;
38        Phase1_GreenOn();
39        Phase2_RedOn();
40        flag_color = TUNING_RED;
41      }
42      else{
43        num1--;
44        num2--;
45      }
46    break;
47    case PHASE_1_GREEN:
48      counter_normal--;
49      Phase1_GreenOn();
50      Phase2_RedOn();
51      flag_color = TUNING_RED;
52      if (counter_normal <= 0) {
53        counter_normal = count_yellow;
54        status_automation = PHASE_1_YELLOW;
55        num1 = count_yellow;
56        num2--;
57        Phase1_YellowOn();
58        Phase2_RedOn();
59      }
60      else{
61        num1--;
62        num2--;
```

```
    }
  break;
  case PHASE_1_YELLOW:
    counter_normal--;
    Phase1_YellowOn();
    Phase2_RedOn();
    flag_color = TUNING_RED;
    if (counter_normal <= 0) {
      counter_normal = count_green;
      status_automation = PHASE_2_GREEN;
      num1 = count_red;
      num2 = count_green;
      Phase2_GreenOn();
      Phase1_RedOn();
      flag_color = TUNING_GREEN;
    }
    else{
      num1--;
      num2--;
    }
  break;
  default:
    status_automation = INIT;
    break;
  }
  if(flag_tuning){
    count_tuning--;
  }
}


```

- **MANUAL MODE**: Hiện thực máy trạng thái của chế độ chỉnh thời gian

```
void manual_trafic(){
    switch (status_manual){
    case MANUAL_INIT:
        status_manual = MANUAL_RED;
        count_buffer = count_red;
    case MANUAL_RED:
        if(timer2_flag){
            setTimer2(50);
            Blink_Red();
        }
        if(isButtonMode()){
            status_manual = MANUAL_YELLOW;
            count_buffer = count_yellow;
        }
        if(isButtonTime()){
            count_buffer++;
```

```
17          }
18          if(isButtonEnter()){
19              count_red = count_buffer;
20              sendString("!Save count red#");
21              if(count_red < count_green)
22                  count_yellow = count_green - count_red;
23              else{
24                  count_green = count_red - count_yellow;
25              }
26              count_tuning = 2*(count_green + count_red +
    count_yellow);
27          }
28
29          break;
30      case MANUAL_YELLOW:
31          if(timer2_flag){
32              setTimer2(50);
33              Blink_Yellow();
34          }
35          if(isButtonMode()){
36              status_manual = MANUAL_GREEN;
37              count_buffer = count_green;
38          }
39          if(isButtonTime()){
40              count_buffer++;
41          }
42          if(isButtonEnter()){
43              count_yellow = count_buffer;
44              sendString("!Save count yellow#");
45              if(count_red < count_green)
46                  count_red = count_green - count_yellow;
47              else{
48                  count_green = count_red - count_yellow;
49              }
50              count_tuning = 2*(count_green + count_red +
    count_yellow);
51          }
52          break;
53      case MANUAL_GREEN:
54          if(timer2_flag){
55              setTimer2(50);
56              Blink_Green();
57          }
58          if(isButtonMode()){
59              status_manual = MANUAL_RED;
60              count_buffer = count_red;
61          }
62          if(isButtonTime()){
63              count_buffer++;
```

```
        }
        if(isButtonEnter()){
            count_green = count_buffer;
            sendString("!Save count green#");
            if(count_red < count_green)
                count_red = count_green - count_yellow;
            else{
                count_yellow = count_red - count_green;
            }
            count_tuning = 2*(count_green + count_red +
    count_yellow);
        }
        break;
    default:
        status_manual = MANUAL_INIT;
    }
}

```

- **TURNING MODE**: Hiển thực máy trạng thái khi có người đi bộ bấm nút ở Phase 1

```
void tuning_traffic(){ //person walking on phase1
  if(isButtonPedes()){
    flag_tuning = 1;
    count_tuning = 2*(count_green + count_red +
   count_yellow);
  }
  if(count_tuning <= 0){
    flag_tuning = 0;
    Pedes_Off();
    buzzer_SetVolume(0);
    count_tuning = 2*(count_green + count_red +
   count_yellow);
  }
  if(flag_tuning){
    switch(flag_color){
      case TUNING_GREEN:
        Pedes_GreenOn();
        if(counter_normal <= 3){
          if(timer3_flag){
            setTimer1(10*counter_normal);
            tog = 1 - tog;
            buzzer_SetVolume(50*tog);
          }
        }
        break;
      case TUNING_RED:
        Pedes_RedOn();
        buzzer_SetVolume(0);
```

```
            break;
        case TUNING_YELLOW:
            Pedes_YellowOn();
            buzzer_SetVolume(0);
            break;
    }
  }
}
```

- **FSM**: Trong phần này, nhóm sẽ xử lý toàn bộ các trạng thái đã được yêu cầu. Trong phần này nhóm sẽ chỉ cho người đi bộ nhấn nút nhấn khi hệ thống đèn đang ở normal, còn khi ở trạng thái manual(chỉnh thời gian) thì không.

```
void traffic_light(){
    switch(status_traffic){
    case AUTOMATIC:
        if(timer1_flag){
            setTimer1(100);
            normal_trafic();
            send7seg(num1, num2);
        }
        if(isModeLongPress()){
            status_traffic = MANUAL;
        }
        tuning_traffic();
        //send uart 2 phase counter
        break;
    case MANUAL:
        manual_trafic();
        if(timer1_flag){
            setTimer1(100);
            sendManual(count_buffer);
        }
        if(isModeLongPress()){
            status_traffic = AUTOMATIC;
        }
        //send temp or ?????
        break;
    default:
        status_traffic = AUTOMATIC;
    }
}

```

Program 5: Máy trạng thái xử lý toàn bộ project

4.5 Main

```
int main(void)
```

```
    {
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_TIM2_Init();
    MX_TIM3_Init();
    MX_USART2_UART_Init();
    buzzer_init();
    HAL_TIM_Base_Start_IT(&htim2);
    HAL_UART_Receive_IT(&huart2, &temp_start, 1);
    setTimer1(100);
    setTimer2(50);
    setTimer3(30);
    while (1)
    {
      traffic_light();
    }
    }
```

Program 6: Hiện thực hàm main.c