VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

VI XỬ LÍ-VI ĐIỀU KHIỂN(CO3009)

Final Project

Traffic Light using STM32F103RB

Advisor: Lê Trọng Nhân
Students: Nguyễn Xuân Triều - 2110610
Lã Minh Đức - 2110132
Nguyễn Thanh Liêm - 2111637

HO CHI MINH CITY, DECEMBER 2023

Contents

1 Danh sách thành viên & Nhiệm vụ được phân công

STT	Họ & Tên	MSSV	Nhiệm vụ được phân công	Tỷ lệ hoàn thành
1	Nguyễn Xuân Triều	2110610	- VIẾT CODE VÀ THIẾT KẾ	100%
2	Lã Minh Đức	2110132	- VIẾT CODE VÀ THIẾT KẾ	100%
3	Nguyễn Thanh Liêm	2111637	- VIẾT CODE VÀ VIẾT BÁO CÁO	100%

2 Đường dẫn đến thư mục lưu trữ github

Nhóm chúng em đã hoạt động tương tác với code trên github. Đường dẫn đến github.
https://github.com/laduc11/project_final

3 Yêu cầu bài tập lớn

Trong bài tập lớn này, nhóm sử dụng vi điều khiển STM32F103RB để triển khai một hệ thống đèn giao thông tại một giao lộ. Hệ thống này có các chức năng chính sau:

- Hoạt động bình thường: Hệ thống hoạt động bình thường theo ba màu đèn giao thông theo màu đỏ, vàng, xanh.
- Chế độ cài đặt: Người dùng có thể thay đổi thời gian giữa các đèn dựa vào việc tương tác với nút nhấn.
- Chế độ người đi bộ: Khi người dùng nhấn vào nút nhấn, đèn người đi bộ sẽ sáng lên, đèn này sẽ tắt sau hai chu kỳ người đi bộ di chuyển

4 Thiết bị phần cứng

Để thực hiện bài tập lớn, nhóm chúng em đã có các thiết bị phần cứng:

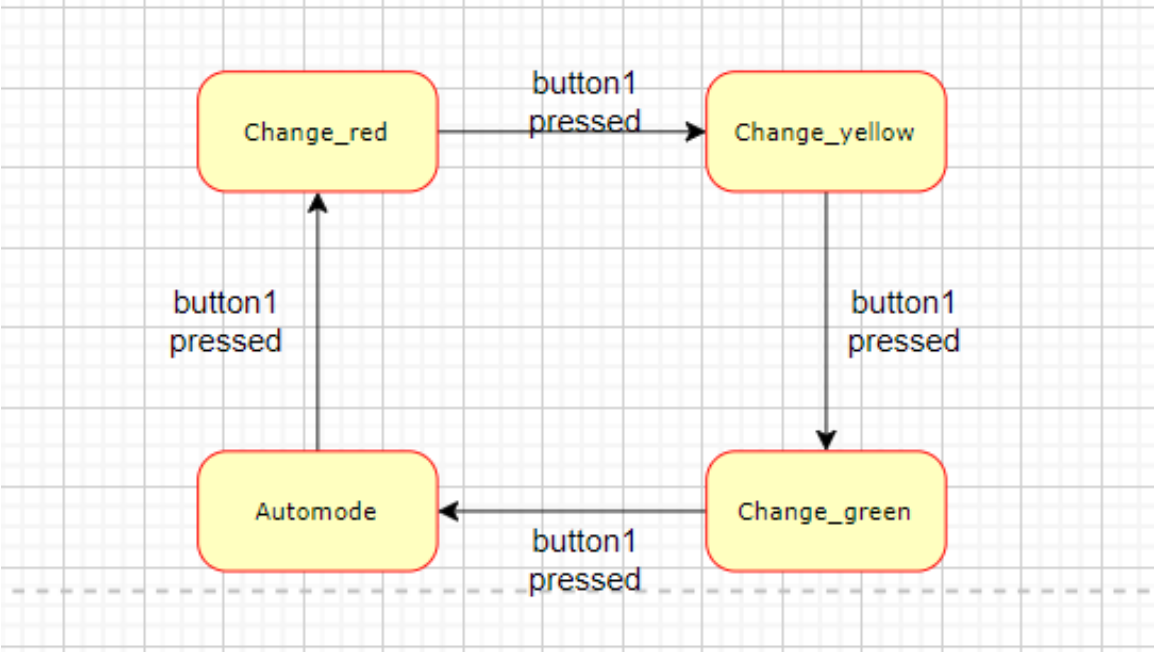
- 2 đèn có ba màu đỏ xanh vàng phục vụ cho việc di chuyển của các phương tiện tham gia giao thông.
- 1 đèn dành cho người đi bộ có hai màu đỏ và xanh.
- 4 nút nhấn. 1 nút nhấn dùng để chuyển giữa các trạng thái, 1 nút nhấn dùng khi vào trạng thái tùy chỉnh để tăng thời gian, 1 nút nhấn xác nhận thời gian đã chỉnh và 1 nút nhấn dùng để khởi động chế độ người đi bộ.
- Loa: Khi chế độ người đi bộ còn thời gian đếm ngược 5 giây, loa sẽ kêu lên càng lúc càng nhanh dần và to hơn
- UART: Gửi thông tin hiển thị thời gian các đèn giao thông, trạng thái tùy chỉnh và

5 Máy trạng thái

Các chức năng còn lại được cài đặt theo biến cờ, nên không thêm vào máy trạng thái (người đi bộ và loa).

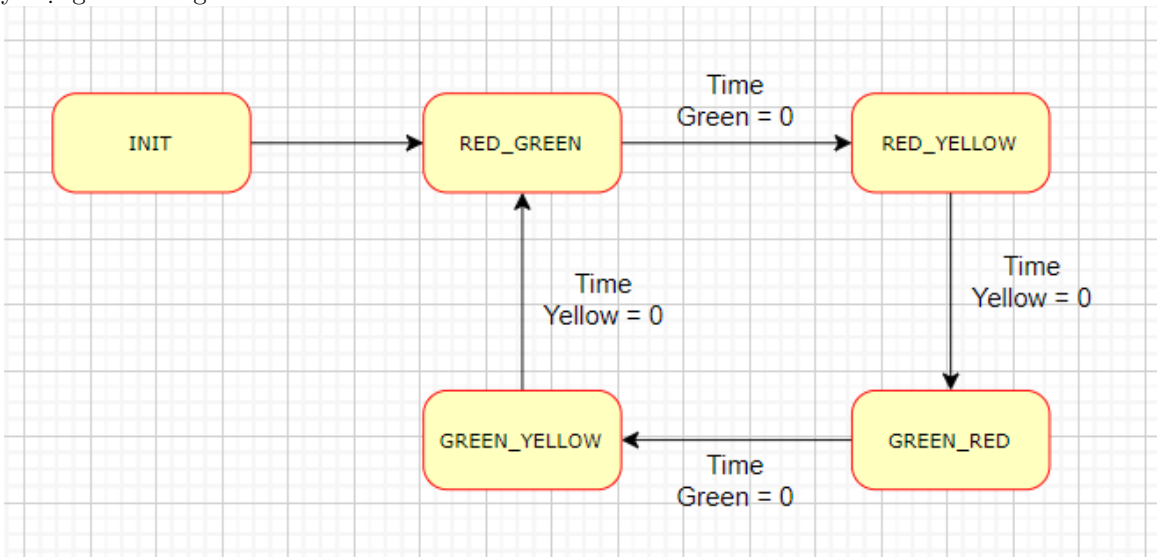
5.1 Máy trạng thái chính của luồng hoạt động

Máy trạng thái chính hoạt động, chuyển giữa các trạng thái automode và các setting



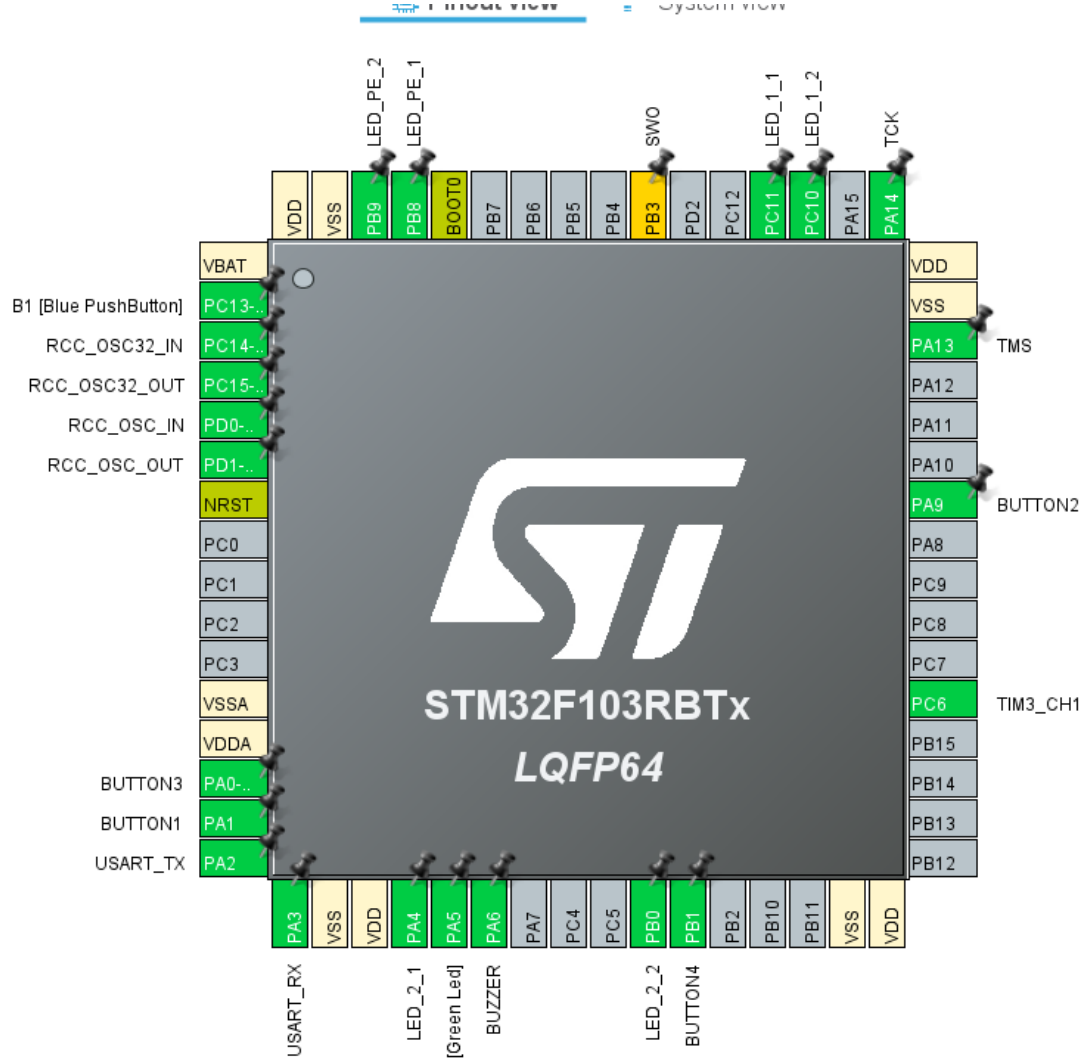
5.2 Máy trạng thái hoạt động riêng chế độ automode

Máy trạng thái riêng của automode



6 Cài đặt phần cứng

6.1 Thiết lập ioc



Traffic light

- First line: LED_1_1 - PC11, LED_1_2 - PC10
- Second line LED_2_1 - PA4, LED_2_2 - PB0

Pedestrian

- BUZZER - PA6
- LED_PE_1 - PB8
- LED_PE_2 - PB9

UART

- UART_TX - PA2
- UART_RX - PA3

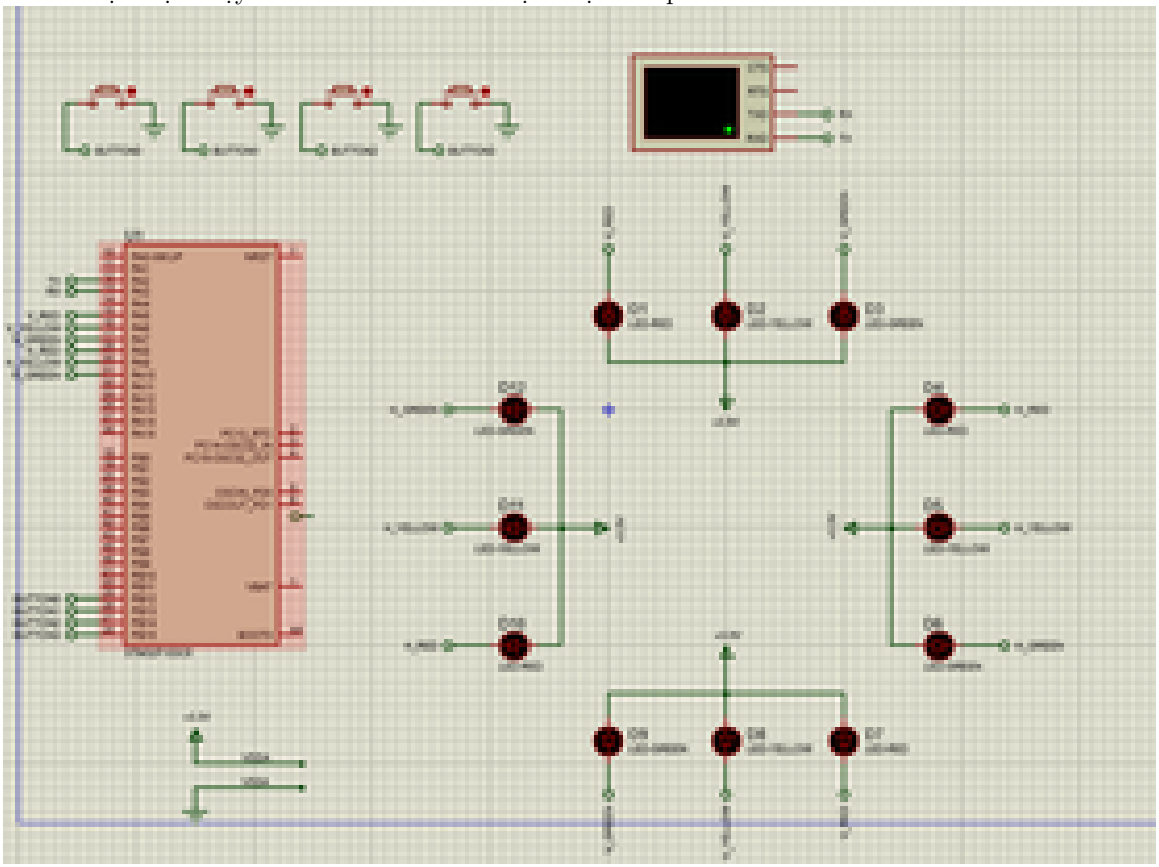
6.2 Các thiết lập cài đặt khác

Pi...	Signal ...	GPIO ...	GPIO ...	GPIO ...	Maxim...	User L...	Modifi...
PA0-...	n/a	n/a	Input ...	Pull-up	n/a	BUTT...	☑
PA1	n/a	n/a	Input ...	Pull-up	n/a	BUTT...	☑
PA4	n/a	Low	Output...	No pull...	Low	LED_...	☑
PA5	n/a	Low	Output...	No pull...	Low	LD2 [...	☑
PA6	n/a	Low	Output...	No pull...	Low	BUZZ...	☑
PA9	n/a	n/a	Input ...	Pull-up	n/a	BUTT...	☑
PB0	n/a	Low	Output...	No pull...	Low	LED_...	☑
PB1	n/a	n/a	Input ...	Pull-up	n/a	BUTT...	☑
PB8	n/a	Low	Output...	No pull...	Low	LED_...	☑
PB9	n/a	Low	Output...	No pull...	Low	LED	☑

Pi...	Signal ...	GPIO ...	GPIO ...	GPIO ...	Maxim...	User L...	Modifi...
PA5	n/a	Low	Output...	No pull...	Low	LD2 [...	☑
PA6	n/a	Low	Output...	No pull...	Low	BUZZ...	☑
PA9	n/a	n/a	Input ...	Pull-up	n/a	BUTT...	☑
PB0	n/a	Low	Output...	No pull...	Low	LED_...	☑
PB1	n/a	n/a	Input ...	Pull-up	n/a	BUTT...	☑
PB8	n/a	Low	Output...	No pull...	Low	LED_...	☑
PB9	n/a	Low	Output...	No pull...	Low	LED_...	☑
PC10	n/a	Low	Output...	No pull...	Low	LED_...	☑
PC11	n/a	Low	Output...	No pull...	Low	LED_...	☑
PC13...	n/a	n/a	Extern...	No pull...	n/a	B1 [Bl...	☑

7 Sơ đồ mạch

Nhóm đã thực hiện chạy kiểm thử trên sơ đồ mạch dựa vào proteus.



8 Mã nguồn các thiết bị phần cứng

8.1 button

8.1.1 Mã nguồn

```
#include "button.h"

// Time cycle = 10ms
#define WAITING_TIME 200          // press and hold in 2 seconds

typedef struct
{
        GPIO_TypeDef *port;
        uint16_t pin;
} listButton;

// List of button
```

```
listButton button[MAX_BUTTON];
statusButton buttonFlag[MAX_BUTTON];
statusButton previousStatus[MAX_BUTTON][3];
uint16_t counter[MAX_BUTTON];
uint8_t isPressed;

/*
 * Initial start value for all attribute
 * Input: none
 * Output: none
 * */
void init_button()
{
        button[0].port = BUTTON1_GPIO_Port;
        button[0].pin = BUTTON1_Pin;
        button[1].port = BUTTON2_GPIO_Port;
        button[1].pin = BUTTON2_Pin;

        isPressed = 0;

        for (int i = 0; i < MAX_BUTTON; i++)
        {
                counter[i] = WAITING_TIME;
                buttonFlag[i] = NORMAL;
                for (int j = 0; j < 3; j++)
                {
                        previousStatus[i][j] = NORMAL;
                }
        }
}

/*
 * Get input signal from all button
 * Input: none
 * Output: none
 * */
void get_key()
{
        for (int i = 0; i < MAX_BUTTON; i++)
        {
                previousStatus[i][0] = previousStatus[i][1];
                previousStatus[i][1] = previousStatus[i][2];
                previousStatus[i][2] = (HAL_GPIO_ReadPin(button[i].port,
                button[i].pin) == GPIO_PIN_RESET) ? PRESSED : NORMAL;
        }

        // Process key
        for (int i = 0; i < MAX_BUTTON; i++)
        {
```

```
        if ((previousStatus[i][0] == PRESSED) &&
            (previousStatus[i][1] == PRESSED) &&
            (previousStatus[i][2] == PRESSED))
        {
            if (isPressed)
            {
                counter[i]--;
                if (counter[i] == 0)
                {
                        counter[i] = WAITING_TIME;
                        buttonFlag[i] = LONG_PRESSED;
                }
            }
            else
            {
                isPressed = 1;
                buttonFlag[i] = PRESSED;
            }
        }
        else
        {
                isPressed = 0;
                counter[i] = WAITING_TIME;
                buttonFlag[i] = NORMAL;
        }
        }
}

/*
 * Process 4 nearest input read from button
 * Input: none
 * Output: none
 * */
void processKey()
{
        for (int i = 0; i < MAX_BUTTON; i++)
        {
        if ((previousStatus[i][0] == PRESSED) &&
            (previousStatus[i][1] == PRESSED) &&
            (previousStatus[i][2] == PRESSED))
        {
            if (isPressed)
            {
                counter[i]--;
                if (counter[i] == 0)
                {
                        counter[i] = WAITING_TIME;
                        buttonFlag[i] = LONG_PRESSED;
                }
```

```
            }
            else
            {
                isPressed = 1;
                buttonFlag[i] = PRESSED;
            }
        }
        else
        {
                isPressed = 0;
                counter[i] = WAITING_TIME;
                buttonFlag[i] = NORMAL;
        }
        }
}

/*
 * Get status of chosen button
 * Input: Index of button
 * Output: Status of button
 * */
statusButton get_button_flag(uint8_t buttonIdx)
{
        statusButton temp = buttonFlag[buttonIdx];
        if (temp != NORMAL)
        {
                isPressed = 1;
        }
        buttonFlag[buttonIdx] = NORMAL;
        return temp;
}
```

8.2 software_timer

8.2.1 Mã nguồn

```
#include "software_timer.h"

int flag[MAX_TIMER] = {0};
int timer_counter[MAX_TIMER] = {0};

/*
Get flag of timer
input: ID of timer
output: flag of this timer
*/
int get_flag(int idx_timer)
{
```

```
    return flag[idx_timer];
}

/*
Count down and trigger the flag when counter = 0
input: ID of timer
output: none
*/
void run_timer(int idx)
{
    if (timer_counter[idx] > 0)
    {
        timer_counter[idx]--;
        if (timer_counter[idx] == 0)
        {
            flag[idx] = 1;
        }
    }
}

/*
Set the time for timer
input: ID of timer and duration of this timer
output: none
*/
void set_timer(int idx_timer, int duration)
{
    flag[idx_timer] = 0;
    timer_counter[idx_timer] = duration;
}
```

8.3 Scheduler

8.3.1 Mã nguồn

```
/*
 * scheduler.c
 *
 *  Created on: Dec 8, 2023
 *      Author: DELL
 */

#include "scheduler.h"

#define TIME_CYCLE 10    // 10 miliseconds

// Timestamps
uint32_t timestamps;
```

```
// array of tasks
sTask SCH_Tasks_G[SCH_MAX_TASK];
uint8_t visited[SCH_MAX_TASK] = {0};
uint32_t NumsOfTask;

// Initial the scheduler
void SCH_Init()
{
    for (int i = 0; i < SCH_MAX_TASK; i++)
    {
        visited[i] = 0;
    }
    NumsOfTask = 0;
    timestamps = 0;
}

// Increase 1 time unit
void SCH_Update()
{
    timestamps = timestamps + 1;
    if (visited[0] != 0)
    {
        SCH_Tasks_G[0].Delay = SCH_Tasks_G[0].Delay - 1 * TIME_CYCLE;
        if (SCH_Tasks_G[0].Delay == 0)
        {
            SCH_Tasks_G[0].RunMe = 1;
        }
    }
}

// Add new task into array of task
uint8_t SCH_Add_Task(void (*pFunction)(), uint32_t Delay, uint32_t Period)
{
    // Check array is full
    if (NumsOfTask == SCH_MAX_TASK)
        return 0;

    // Add new task
    uint8_t success = 0;
    uint32_t curTime = 0;
    uint32_t task;
    for (task = 0; task < NumsOfTask; task++)
    {
        if (curTime <= Delay)
        {
            curTime = curTime + SCH_Tasks_G[task].Delay;
        }

        if (curTime > Delay)
```

```
            {
                Delay = Delay - (curTime - SCH_Tasks_G[task].Delay);
                success = 1;
                break;
            }
        }
        if (!success)
        {
            Delay = Delay - curTime;
            success = 1;
        }

        // Shift right the task 1 unit
        sTask curTask = SCH_Tasks_G[task];
        curTask.Delay = curTask.Delay - Delay;
        for (uint32_t i = task; i < NumsOfTask; i++)
        {
            sTask tempTask = SCH_Tasks_G[i + 1];
            SCH_Tasks_G[i + 1] = curTask;
            SCH_Tasks_G[i + 1].TaskID = SCH_Tasks_G[i + 1].TaskID + 1; // update TaskID
            curTask = tempTask;
        }

        // Add new task with new Delay into the list
        SCH_Tasks_G[task].pTask = pFunction;
        SCH_Tasks_G[task].Delay = Delay;
        SCH_Tasks_G[task].Period = Period;
        SCH_Tasks_G[task].RunMe = 0;
        SCH_Tasks_G[task].TaskID = task;

        // Update status of the array of task
        NumsOfTask = NumsOfTask + 1;
        success = 1;
        visited[NumsOfTask - 1] = 1;

        return success;
}

// Delete task from array of task
uint8_t SCH_Delete(uint32_t TaskID)
{
        uint8_t success = 0;
        // Check TaskID is correct, array of task is empty
        if (TaskID >= NumsOfTask || NumsOfTask == 0)
            return 0;

        // Calculate Delay for all tasks after that and shift left them
        uint32_t Delay = SCH_Tasks_G[TaskID].Delay;
        for (uint32_t task = TaskID; task < NumsOfTask - 1; task++)
```

```
    {
        SCH_Tasks_G[task] = SCH_Tasks_G[task + 1];
        SCH_Tasks_G[task].TaskID = SCH_Tasks_G[task].TaskID - 1;
        SCH_Tasks_G[task].Delay = SCH_Tasks_G[task].Delay + Delay;
    }
    success = 1;
    NumsOfTask = NumsOfTask - 1;
    visited[NumsOfTask] = 0;

    return success;
}

// Run the task in the array of task
void SCH_Dispatch_Tasks()
{
        if (NumsOfTask == 0)
                return;
    while (SCH_Tasks_G[0].RunMe == 1 || SCH_Tasks_G[0].Delay == 0)
    {
        // Run the task
        (*SCH_Tasks_G[0].pTask)();
        sTask tempTask = SCH_Tasks_G[0];
        SCH_Delete(tempTask.TaskID);
        if (tempTask.Period != 0)
        {
            // Add this task into array again
            SCH_Add_Task(tempTask.pTask, tempTask.Period, tempTask.Period);
        }
    }
}

// Get timestamps
uint32_t get_time()
{
        return timestamps;
}
```

9 Mã nguồn quá trình thực thi

Trong phần giải thích này, nhóm xin được giải thích các thành phần chính trong bài code, các phần cứng button, timer, scheduler đã được hiện thực ở trên.

9.1 main.c

9.1.1 Giải thích mã nguồn

Trong file main, sẽ có những phần gọi timer để gọi scheduler_update. Trong đó nhóm hiện thực bài tập lớn theo phương thức scheduler.

9.1.2 Mã nguồn

```
    // Add tasks to scheduler
  SCH_Add_Task(get_key, 10, 10);
  SCH_Add_Task(manager_state, 10, 100);
  SCH_Add_Task(stateUI, 10, 1000);

  while (1)
  {
          SCH_Dispatch_Tasks();
    /* USER CODE END WHILE */

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

9.2 manager.c

9.2.1 Giải thích mã nguồn

clearScreen():Xóa màn hình console bằng cách gửi một chuỗi điều khiển đến UART để xóa màn hình.

printString(const char *str):In một chuỗi ký tự đến cổng UART để hiển thị trên console.

printNumber(int val): Chuyển đổi một số nguyên thành chuỗi ký tự và sau đó gửi chuỗi đó đến cổng UART để hiển thị.

buzzerReset():Đặt lại trạng thái của buzzer về 0.

BuzzerLouder(): Tăng độ sáng của buzzer nếu buzzerFlag được đặt thành 1, ngược lại giảm độ sáng.

pedesRun(): Kiểm tra trạng thái buzzerFlag để quyết định xem có nên chạy chức năng buzzer hay không.

Dựa vào trạng thái của state_auto_mode và buzzerCounter, quyết định xem có nên tăng độ sáng của buzzer hay không. stateUI():

Hiển thị giao diện người dùng tương ứng với trạng thái hiện tại của hệ thống (được xác định bởi state_mode). Có thể hiển thị giao diện đèn giao thông tự động hoặc các chế độ thay đổi đèn đỏ, vàng, xanh. manager_state():

Máy trạng thái hữu hạn quản lý chế độ của đèn giao thông:

- Các chế độ bao gồm AUTO_MODE, RED_CHANGING, YELLOW_CHANGING, GREEN_CHANGING.
- Dựa vào sự kiện nhấn nút và trạng thái hiện tại của hệ thống để quyết định chuyển đổi giữa các chế độ và thực hiện các hành động tương ứng (tăng thời gian đèn đỏ, vàng, xanh).

9.2.2 Mã nguồn

```
#include "manager.h"

STATE_MODE state_mode = AUTO_MODE;

int buzzerDuty = 0;
int flagPedes = 0;
int buzzerCounter = 0;

void clearScreen()
{
        char clr[] = "\033[2J";
        HAL_UART_Transmit(&huart2, (uint8_t *)clr,
 sizeof(clr) - 1, HAL_MAX_DELAY);
}
void printString(const char *str)
{
        HAL_UART_Transmit(&huart2, (uint8_t *)str,
 strlen(str), HAL_MAX_DELAY);
}
void printNumber(int val)
{
        char str[100];
        HAL_UART_Transmit(&huart2, (void *)str, sprintf(str,
 "%d\r\n", val), HAL_MAX_DELAY);
}

void buzzerReset()
{
        buzzerDuty = 0;
}

void BuzzerLouder()
{
        if (buzzerFlag == 1)
        {
                if (buzzerDuty < 100)
                {
                        buzzerDuty += 10;
                }
                __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, buzzerDuty);
                buzzerFlag = 0;
```

```
        }

        if (buzzerFlag == 0)
        {
                __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, 0);
                buzzerFlag = 1;
        }
}

void pedesRun()
{
        if (buzzerFlag == 0){
                return;
        }
        else{
                switch (state_auto_mode)
                {
                case INIT:

                case RED_GREEN:
                        if (buzzerCounter >= hor_countdown ) {
                                buzzerCounter = 0;
                                BuzzerLouder();
                        }
                        else {
                                buzzerCounter++;
                        }

                        break;
                case RED_YELLOW:
                        if (buzzerCounter >= hor_countdown ) {
                                buzzerCounter = 0;
                                BuzzerLouder();
                        }
                        else {
                                buzzerCounter++;
                        }
                        break;
                case GREEN_RED:
                        buzzerCounter = 0;
                        buzzerDuty = 0;
                        __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, 0);
                        break;
                case YELLOW_RED:
                        buzzerCounter = 0;
                        buzzerDuty = 0;
                        __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, 0);
                        break;
                }
```

```
        }


}

void stateUI()
{
        switch (state_mode)
        {
        case AUTO_MODE:
                traffic_light();
                break;
        case RED_CHANGING:
                red_changing_UI();
                break;
        case YELLOW_CHANGING:
                yellow_changing_UI();
                break;
        case GREEN_CHANGING:
                green_changing_UI();
                break;
        }
}

/*
 * Finite state machine to change mode of traffic light
 * Input: none
 * Output: none
 * */
void manager_state()
{
        switch (state_mode)
        {
        case AUTO_MODE:

                if (get_button_flag(0) == PRESSED ||
  get_button_flag(0) == LONG_PRESSED)
                {
                        state_mode = RED_CHANGING;
                        init_manual();
                        printString("MODE ");
                        printValue(2);
                        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
                }

                if (get_button_flag(1) == PRESSED ||
  get_button_flag(1) == LONG_PRESSED)
                {
                        printString("PRESSED BUTTON ");
```

```
                        printValue(2);
                }
                break;
        case RED_CHANGING:
                if (get_button_flag(0) == PRESSED ||
get_button_flag(0) == LONG_PRESSED)
                {
                        state_mode = YELLOW_CHANGING;
                        init_manual();
                        printString("MODE ");
                        printValue(3);
                        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
                }

                if (get_button_flag(1) == PRESSED ||
get_button_flag(1) == LONG_PRESSED)
                {
                        increase_led_red();
                        printString("INCREASE RED TIMER : ");
                        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
                }
                break;
        case YELLOW_CHANGING:
                if (get_button_flag(0) == PRESSED ||
get_button_flag(0) == LONG_PRESSED)
                {
                        state_mode = GREEN_CHANGING;
                        init_manual();
                        printString("MODE ");
                        printValue(4);
                        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
                }

                if (get_button_flag(1) == PRESSED ||
get_button_flag(1) == LONG_PRESSED)
                {
                        increase_led_yellow();
                        printString("INCREASE YELLOW TIMER : ");
                        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
                }
                break;
        case GREEN_CHANGING:
                if (get_button_flag(0) == PRESSED ||
get_button_flag(0) == LONG_PRESSED)
                {
                        state_mode = AUTO_MODE;
                        state_auto_mode = INIT;
                        if (led_time[RED] != led_time[YELLOW] + led_time[GREEN])
                        {
```

```
                        led_time[RED] = led_time[YELLOW] + led_time[GREEN];
                    }

                    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
                }

                if (get_button_flag(1) == PRESSED ||
  get_button_flag(1) == LONG_PRESSED)
                {
                    increase_led_green();
                    printString("INCREASE GREEN TIMER : ");
                    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
                }
                break;
        default:
                break;
        }
}
```

9.3 auto_mode.c

9.3.1 Giải thích mã nguồn

turn_off_light(): Hàm này tắt tất cả đèn giao thông bằng cách đặt các chân GPIO tương ứng về trạng thái RESET.

V_Green(), V_Yellow(), V_Red(), H_Green(), H_Yellow(), H_Red(): Những hàm này điều khiển các đèn LED riêng lẻ cho đèn giao thông dọc và ngang, đặt chúng ở trạng thái phù hợp (SET hoặc RESET).

pedesOff(), pedesGreen(), pedesRed(): Những hàm này điều khiển các đèn LED cho người đi bộ, đặt chúng ở trạng thái phù hợp. traffic_light():

Đây là hàm chính thực hiện logic cho bộ điều khiển đèn giao thông.

- Sử dụng một máy trạng thái hữu hạn (state_auto_mode) để chuyển đổi giữa các trạng thái khác nhau: INIT, RED_GREEN, RED_YELLOW, GREEN_RED, và YELLOW_RED.
- Hàm cập nhật các bộ đếm ngược (hor_countdown và ver_countdown) dựa trên thời gian được cấu hình cho mỗi trạng thái.
- Nó gọi các hàm điều khiển LED tương ứng để hiển thị đèn phù hợp cho mỗi trạng thái.
- Nó cũng xử lý đèn người đi bộ và một cờ báo hiệu âm thanh dựa trên một số điều kiện nhất định.

9.3.2 Mã nguồn

```
/*
 * auto_mode.c
 *
 *  Created on: Dec 8, 2023
 *      Author: DELL
```

```
 */

#include "auto_mode.h"

uint8_t led_time[3] = {10, 4, 6};
uint8_t hor_countdown = 0;
uint8_t ver_countdown = 0;
StateNormal state_auto_mode = INIT;
int onPedes = 0;
int buzzerFlag = 0;

/*
 * Turn off all traffic light
 * Input: none
 * Output: none
 * */
void turn_off_light()
{
        HAL_GPIO_WritePin(LED_1_1_GPIO_Port, LED_1_1_Pin, RESET);
        HAL_GPIO_WritePin(LED_1_2_GPIO_Port, LED_1_2_Pin, RESET);
        HAL_GPIO_WritePin(LED_2_1_GPIO_Port, LED_2_1_Pin, RESET);
        HAL_GPIO_WritePin(LED_2_2_GPIO_Port, LED_2_2_Pin, RESET);
}

/*
 * Turn on led red in a vertical direction
 * Input: none
 * Output: none
 * */
void V_Green()
{
        HAL_GPIO_WritePin(LED_1_1_GPIO_Port, LED_1_1_Pin, SET);
        HAL_GPIO_WritePin(LED_1_2_GPIO_Port, LED_1_2_Pin, RESET);
}

/*
 * Turn on led yellow in a vertical direction
 * Input: none
 * Output: none
 * */
void V_Yellow()
{
        HAL_GPIO_WritePin(LED_1_1_GPIO_Port, LED_1_1_Pin, SET);
        HAL_GPIO_WritePin(LED_1_2_GPIO_Port, LED_1_2_Pin, SET);
}

/*
 * Turn on led green in a vertical direction
 * Input: none
```

```
 * Output: none
 * */
void V_Red()
{
        HAL_GPIO_WritePin(LED_1_1_GPIO_Port, LED_1_1_Pin, RESET);
        HAL_GPIO_WritePin(LED_1_2_GPIO_Port, LED_1_2_Pin, SET);
}

/*
 * Turn on led red in a horizontal direction
 * Input: none
 * Output: none
 * */
void H_Green()
{
        HAL_GPIO_WritePin(LED_2_1_GPIO_Port, LED_2_1_Pin, SET);
        HAL_GPIO_WritePin(LED_2_2_GPIO_Port, LED_2_2_Pin, RESET);
}

/*
 * Turn on led yellow in a horizontal direction
 * Input: none
 * Output: none
 * */
void H_Yellow()
{
        HAL_GPIO_WritePin(LED_2_1_GPIO_Port, LED_2_1_Pin, SET);
        HAL_GPIO_WritePin(LED_2_2_GPIO_Port, LED_2_2_Pin, SET);
}

/*
 * Turn on led green in a horizontal direction
 * Input: none
 * Output: none
 * */
void H_Red()
{
        HAL_GPIO_WritePin(LED_2_1_GPIO_Port, LED_2_1_Pin, RESET);
        HAL_GPIO_WritePin(LED_2_2_GPIO_Port, LED_2_2_Pin, SET);
}

void pedesOff()
{
        HAL_GPIO_WritePin(LED_PE_1_GPIO_Port, LED_PE_1_Pin, RESET);
        HAL_GPIO_WritePin(LED_PE_2_GPIO_Port, LED_PE_2_Pin, RESET);
}

void pedesGreen()
{
```

```
        HAL_GPIO_WritePin(LED_PE_1_GPIO_Port, LED_PE_1_Pin, SET);
        HAL_GPIO_WritePin(LED_PE_2_GPIO_Port, LED_PE_2_Pin, RESET);
}

void pedesRed()
{
        HAL_GPIO_WritePin(LED_PE_1_GPIO_Port, LED_PE_1_Pin, RESET);
        HAL_GPIO_WritePin(LED_PE_2_GPIO_Port, LED_PE_2_Pin, SET);
}



/*
 * Activate current traffic light state
 * Input: none
 * Output: none
 * */

void traffic_light(void)
{
        switch (state_auto_mode)
        {
        case INIT:
                turn_off_light();
                hor_countdown = led_time[RED];
                ver_countdown = led_time[GREEN];
                state_auto_mode = RED_GREEN;
                onPedes = 0;
                break;
        case RED_GREEN:
                hor_countdown--;
                ver_countdown--;
                V_Red();
                H_Green();
                if (ver_countdown <= 0)
                {
                        state_auto_mode = RED_YELLOW;
                        ver_countdown = led_time[YELLOW];
                }
                if (onPedes == 1){
                        pedesGreen();
                        if (hor_countdown <= 10) buzzerFlag = 1;
                }
                else{
                        pedesOff();
                }
                clearScreen();
                printString("RED: ");
                printNumber(hor_countdown);
```

```
            printString("GREEN: ");
            printNumber(ver_countdown);
            break;
    case RED_YELLOW:
            hor_countdown--;
            ver_countdown--;
            V_Red();
            H_Yellow();
            if (hor_countdown <= 0)
            {
                    state_auto_mode = GREEN_RED;
                    hor_countdown = led_time[GREEN];
                    ver_countdown = led_time[RED];
            }
            if (onPedes == 1){
                    pedesGreen();
                    if (hor_countdown <= 10) buzzerFlag = 1;
            }
            else{
                    pedesOff();
            }
            clearScreen();
            printString("RED: ");
            printNumber(hor_countdown);
            printString("YELLOW: ");
            printNumber(ver_countdown);
            break;
    case GREEN_RED:
            hor_countdown--;
            ver_countdown--;
            V_Green();
            H_Red();
            if (hor_countdown <= 0)
            {
                    state_auto_mode = YELLOW_RED;
                    hor_countdown = led_time[YELLOW];
            }
            if (onPedes == 1){
                    pedesRed();
                    buzzerFlag = 0;
            }
            else{
                    pedesOff();
            }
            clearScreen();
            printString("GREEN: ");
            printNumber(hor_countdown);
            printString("RED: ");
            printNumber(ver_countdown);
```

```
            break;
        case YELLOW_RED:
            hor_countdown--;
            ver_countdown--;
            V_Yellow();
            H_Red();
            if (hor_countdown <= 0)
            {
                state_auto_mode = RED_GREEN;
                hor_countdown = led_time[RED];
                ver_countdown = led_time[GREEN];
            }
            if (onPedes == 1){
                pedesRed();
                buzzerFlag = 0;
            }
            else{
                pedesOff();
            }
            clearScreen();
            printString("YELLOW: ");
            printNumber(hor_countdown);
            printString("RED: ");
            printNumber(ver_countdown);
            break;
        default:
            break;
        }
}
```

9.4 manual_mode

9.4.1 Giải thích mã nguồn

init_manual(): Khởi tạo tất cả các tham số cần thiết khi chuyển sang chế độ điều khiển thủ công.Thiết lập toggleLED về 0 và tắt tất cả các đèn giao thông.

red_changing_UI(): Hiển thị giao diện người dùng khi đang thay đổi thời gian đèn đỏ. Nếu toggleLED bằng 1, thì bật tất cả các đèn đỏ. Nếu toggleLED bằng 0, thì tắt tất cả các đèn.

yellow_changing_UI(): Hiển thị giao diện người dùng khi đang thay đổi thời gian đèn vàng. Nếu toggleLED bằng 1, thì bật tất cả các đèn vàng. Nếu toggleLED bằng 0, thì tắt tất cả các đèn.

green_changing_UI(): Hiển thị giao diện người dùng khi đang thay đổi thời gian đèn xanh. Nếu toggleLED bằng 1, thì bật đèn xanh. Nếu toggleLED bằng 0, thì tắt tất cả các đèn.

increase_led_red(): Tăng thời gian của đèn đỏ lên 1 đơn vị. Giới hạn tối đa thời gian đèn đỏ là 40 giây.

increase_led_yellow():Tăng thời gian của đèn vàng lên 1 đơn vị.Giới hạn tối đa thời gian đèn vàng là thời gian tổng của đèn xanh và đèn vàng không vượt quá thời gian đèn đỏ.

increase_led_green():Tăng thời gian của đèn xanh lên 1 đơn vị. Giới hạn tối đa thời gian đèn xanh là thời gian tổng của đèn xanh và đèn vàng không vượt quá thời gian đèn đỏ.

9.4.2 Mã nguồn

```
#include "manual_mode.h"
// Led blink with frequency 2Hz

uint8_t toggleLED = 0;

/*
 * Initial all start parameter
 * Input: none
 * Output: none
 * */
void init_manual()
{
        toggleLED = 0;
        turn_off_light();
}

/*
 * Display modified led red
 * Input: none
 * Output: none
 * */
void red_changing_UI()
{
    if (toggleLED == 1)
    {
        // turn on all led red
        V_Red();
        H_Red();
    }
    else
    {
        // turn off all led
        turn_off_light();
    }
    toggleLED = 1 - toggleLED;
}

/*
 * Display modified led yellow
 * Input: none
 * Output: none
 * */
void yellow_changing_UI()
{
    if (toggleLED == 1)
    {
        // turn on all led yellow
```

```
            V_Yellow();
            H_Yellow();
        }
        else
        {
            // turn off all led
            turn_off_light();
        }
        toggleLED = 1 - toggleLED;
}

/*
 * Display modified led green
 * Input: none
 * Output: none
 * */
void green_changing_UI()
{
        if (toggleLED == 1)
        {
            // turn on led green
            V_Green();
            H_Green();
        }
        else
        {
            // turn off all led
            turn_off_light();
        }
        toggleLED = 1 - toggleLED;
}

/*
 * Increase the period of led red one time unit
 * Input: none
 * Output: none
 * */
void increase_led_red()
{
            led_time[RED] = led_time[RED] + 1;
            if(led_time[RED] == 40) led_time[RED] = 1;
            clearScreen();
            printString("RED TIME: ");
            printNumber(led_time[RED]);
}

/*
 * Increase the period of led yellow one time unit
 * Input: none
```

```
 * Output: none
 * */
void increase_led_yellow()
{
        if (led_time[YELLOW] + led_time[GREEN] == led_time[RED])
                return;
        led_time[YELLOW] = led_time[YELLOW] + 1;
        clearScreen();
        printString("YELLOW TIME: ");
        printNumber(led_time[RED]);
}

/*
 * Increase the period of led red one time unit
 * Input: none
 * Output: none
 * */
void increase_led_green()
{
        if (led_time[YELLOW] + led_time[GREEN] == led_time[RED])
                return;
        led_time[GREEN] = led_time[GREEN] + 1;
        clearScreen();
        printString("GREEN TIME: ");
        printNumber(led_time[RED]);
}
```