ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

VI XỬ LÝ - VI ĐIỀU KHIỂN

Báo cáo Bài tập lớn

THIẾT KẾ HỆ THỐNG ĐÈN GIAO THÔNG

GVHD: Lê Trọng Nhân

SV thực hiện: Nguyễn Trọng Anh 2110014

Lã Thị Kiều Ngân 2114149

TP. Hồ Chí Minh, tháng 12/2023

Mục lục

LỜI MỞ ĐẦU

1 Giới thiệu

Bài tập lớn chủ đề "Thiết kế hệ thống đèn giao thông" được xây dựng nhằm mô phỏng hệ thống đèn giao thông trong thực tế một cách chân thực và chi tiết nhất. Hệ thống này được phát triển với ba chế độ chính là chế độ sáng tự động, chế độ điều chỉnh bằng tay và chế độ điều chỉnh thời gian sáng. Đặc biệt, hệ thống còn tích hợp đèn hiệu dành cho người đi bộ, được sử dụng riêng trong chế độ sáng tự động.

Ở chế độ sáng tự động, các đèn giao thông tự động đếm ngược và sáng luân phiên như các đèn giao thông thông thường. Chế độ điều chỉnh bằng tay cung cấp người dùng khả năng chuyển giữa các đèn giao thông bằng cách nhấn nút. Chế độ điều chỉnh thời gian sáng của đèn giúp điều chỉnh thời gian sáng của từng đèn giao thông trong chế độ sáng tự động. Hệ thống đèn hiệu cho người đi bộ gồm một đèn, còi, và nút nhấn giúp báo hiệu thời gian sang đường cho người đi bộ. Chi tiết hệ thống sẽ được mô tả trong bài báo cáo này.

2 Mục tiêu

Để xây thực hệ thống đèn giao thông hoàn chỉnh, nhóm đặt ra các mục tiêu về thiết kế như sau:

- Thiết kế máy trạng thái 3 chế độ.
- Thiết kế các máy trạng thái nhỏ để chuyển trạng thái trong từng chế độ.
- Thiết kế hệ thống nút nhấn có phân biệt giữa nút nhấn ngắn và nút nhất dài.
- Thiết kế scheduler O(1).
- Thiết kế còi báo dành cho người đi bộ kêu càng to và càng nhanh khi gần hết thời gian.

3 Tổng quan về báo cáo

Báo cáo gồm có 3 chương:

- Chương 1 mô tả chi tiết máy trạng thái và các thiết kế được sử dụng trong bài tập lớn này.
- Chương 2 cung cấp và giải thích mã nguồn của hệ thống.
- Chương 3 cung cấp về cấu hình và mô phỏng của hệ thống.

4 Mã nguồn của hệ thống

Toàn bộ mã nguồn của hệ thống được lưu trữ tại github, repo MCU_assignment. Trong repo này, bao gồm nhiều branch tương ứng với việc phát triển nhiều tính năng cho hệ thống, và hệ thống đã hoàn thiện cuối cùng sẽ ở branch master. Repo bao gồm toàn bộ mã nguồn của hệ thống trên STM32F103C6, STM32F103RB và file mô phỏng proteus.

Chương 1: THIẾT KẾ

1 Mô tả hệ thống

Hệ thống đèn giao thông được xây dựng dựa trên 3 chế độ: AUTO MODE (chế độ tự động), TUNING MODE (chế độ điều chỉnh thời gian), MANUAL MODE (chế độ điều chỉnh bằng tay). Trong hệ thống này có sự phân biệt giữa "chế độ" và "trạng thái". Hệ thống có 3 chế độ, các hành vi của đèn giao thông ở mỗi chế độ được gọi là các trạng thái của chế độ đó. Khi được khởi tạo, hệ thống sẽ kiểm tra xem thời gian sáng của các đèn giao thông có hợp lý không. Nếu không, hệ thống sẽ chuyển sang ERROR MODE và tắt tất cả đèn. Ngược lại, hệ thống sẽ tự động chuyển sang AUTO MODE.

Hệ thống đèn giao thông cho phép di chuyển giữa chế độ khác một cách độc lập thông qua bộ 4 nút nhấn. Hệ thống nút nhấn để chuyển chế độ được thiết kế để chạy độc lập với máy trạng thái nhằm giúp giải thuật ngắn gọn hơn.

Chế độ tự động (AUTO MODE): Ở chế độ này, các đèn giao thông được tự động đếm ngược và chuyển sang trạng thái kế tiếp khi hết thời gian. Đây cũng là chế độ mặc định của hệ thống. Ngoài ra chế độ này còn hỗ trợ chức năng cho người đi bộ, được kích hoạt bằng cách nhấn thả nút nhấn 4 (nhấn giữ không có tác dụng). Khi kích hoạt chức năng này, đèn cho người đi bộ sẽ sáng, đèn này có 2 trạng thái là xanh và đỏ, hiển thị ngược với đèn giao thông ở tuyến đường chính. Khi sắp hết thời gian cho người đi bộ (đèn xanh), ở 3 giây cuối sẽ có tiếng còi báo hiệu. Tiếng còi này sẽ kêu ngày càng to và ngày càng nhanh đến khi hết thời gian đi bộ. Chức năng cho người đi bộ được tắt khi nhấn nút 4 lần nữa hoặc tự động tắt sau 2 chu kì.

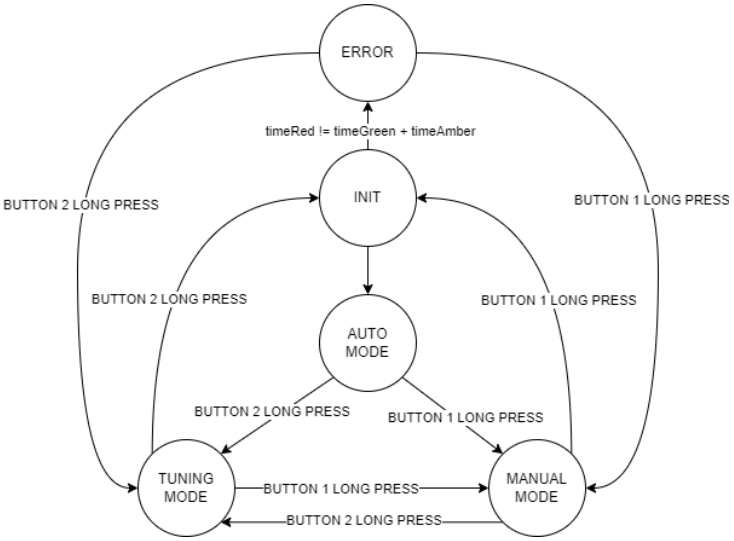
Chế độ điều chỉnh bằng tay (MANUAL MODE): Người dùng có thể chuyển sang chế độ này bằng cách nhấn giữ nút nhấn 1 dù đang ở bất kì chế độ nào. Khi ở trong chế độ này, người dùng có thể nhấn thả nút nhấn thứ 1 để chuyển màu đèn. Nếu nút nhấn 1 được nhấn giữ lần nữa thì hệ thống sẽ chuyển về INIT. Nếu hệ thống đang không ở MANUAL MODE thì nhấn thả nút nhấn 1 không có tác dụng nào.

Chế độ điều chỉnh thời gian (TUNING MODE): Người dùng có thể chuyển đến chế độ này bằng cách nhấn giữ nút nhấn 2 dù đang ở bất kì chế độ nào. Khi ở trong chế độ này, đèn đang được điều chỉnh thời gian sẽ chớp tắt với tần số 2Hz. Người dùng có thể chuyển giữa các đèn bằng cách nhấn thả nút nhấn 2. Nút nhấn 3 được dùng để điều chỉnh thời gian sáng cho đèn. Thời gian sáng của đèn sẽ tăng khi nhấn thả nút nhấn 3 và sẽ giảm khi nhấn giữ nút nhấn 3. Thời gian sáng của mỗi đèn có thể điều chỉnh trong đoạn từ 0 đến 99. Khi thời gian sáng đạt 99, nếu vẫn tiếp tục nhấn tăng thì thời gian sẽ đếm lại về 0, tương tự nếu thời gian sáng là 0 nhưng vẫn tiếp tục nhấn giảm thì thời gian sáng sẽ quay về 99. Nếu nút nhấn 2 được nhấn giữ lần nữa thì hệ thống sẽ chuyển về trạng thái INIT để kiểm tra tính hợp lệ của thời gian các đèn trước khi chuyển sang AUTO MODE.

Ngoài ra, hệ thống còn có khả năng gửi thông tin về PC thông qua UART. Khi chuyển sang chế độ nào thì tên chế độ đó sẽ được hiển thị trên PC. Ở AUTO MODE, mỗi giây PC sẽ nhận được thời gian đếm ngược của hệ thống ở cả 2 chiều của tuyến đường, nếu chức năng cho người đi bộ được kích hoạt thì PC sẽ nhận được cả thời gian đếm ngược cho người đi bộ. Ở TUNING MODE, mỗi khi thời gian sáng của đèn thay đổi thì nó sẽ được hiển thị trên PC. Khi chuyển từ TUNING MODE về INIT, nếu thời gian sáng của các đèn không hợp lệ thì trên PC sẽ hiển thị "ERROR TIMER".

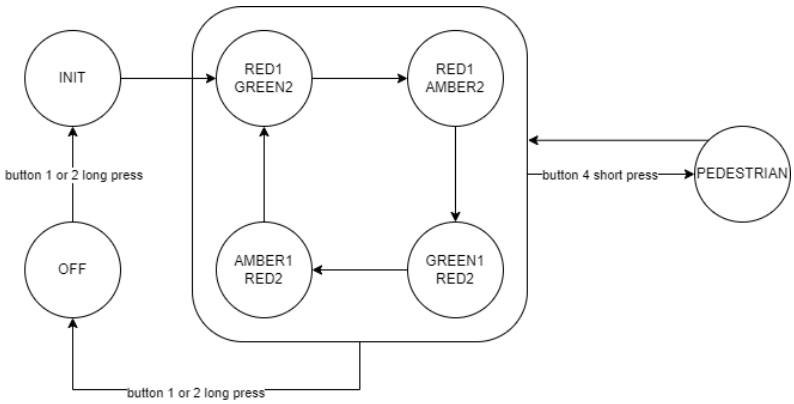
2 Máy trạng thái

Máy trạng thái cho các chế độ của hệ thống được mô tả bằng sơ đồ sau:



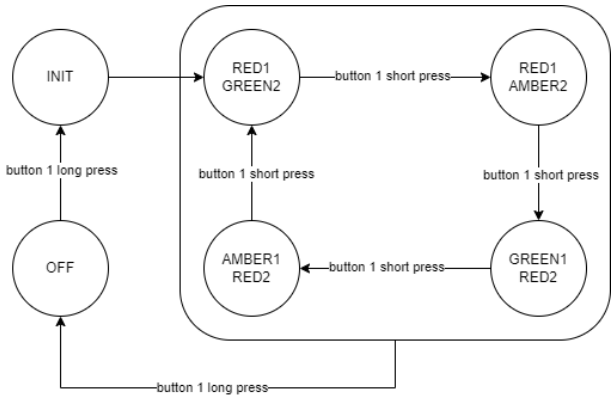
Hình 1: *Máy trạng thái*

Máy trạng thái cho các trạng thái của AUTO MODE được mô tả bằng sơ đồ sau:



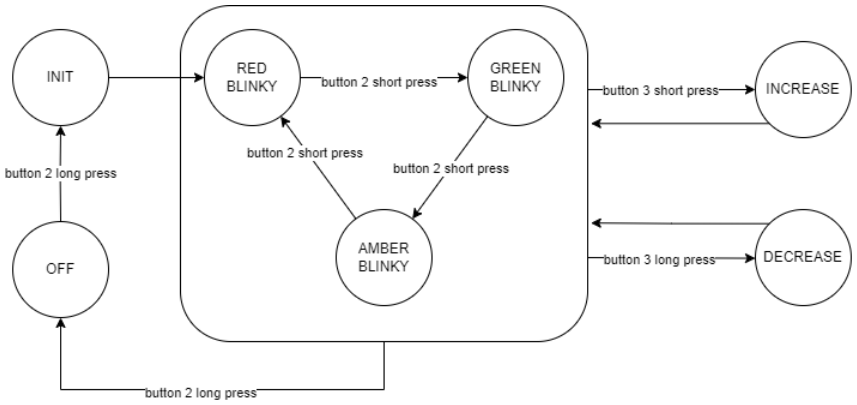
Hình 2: *AUTO MODE*

Máy trạng thái cho các trạng thái của MANUAL MODE được mô tả bằng sơ đồ sau:



Hình 3: *MANUAL MODE*

Máy trạng thái cho các trạng thái của TUNING MODE được mô tả bằng sơ đồ sau:



Hình 4: *TUNING MODE*

3 Bộ định thời (Scheduler)

Bộ định thời (scheduler) có thể xem như một hệ điều hành đơn giản cho phép các tác vụ được thực hiện lặp lại theo chu kì hoặc thực hiện 1 lần. Scheduler được thiết kế với 5 phương thức, bao gồm: khởi tạo (SCH_Init), thêm task (SCH_Add_Task), cập nhật task (SCH_Update), thực hiện task (SCH_Dispatch_Tasks) và xoá task (SCH_Delete_Task). Trong đó, hàm SCH_Update được gọi trong timer interrupt mỗi 10ms, do đó cần sử dụng cấu trúc dữ liệu và giải thuật hợp lý để tối ưu hàm này, tránh gây ra tình trạng nested interrupt. Trong bài tập lớn này, danh sách liên kết vòng (circular linked list) được sử dụng để hiện thực scheduler.

Các task của scheduler được sắp xếp trong danh sách liên kết vòng. Mỗi node (task) trong danh sách liên kết có 4 thuộc tính:

- **pTask:** con trỏ hàm trỏ đến hàm sẽ được thực thi.
- **Delay:** thời gian trì hoãn từ thời điểm đang xét đến thời điểm kế tiếp task được thực hiện.
- **Period:** thời gian giữa 2 lần task được thực thi. Nếu Period bằng 0 thì task được thực hiện 1 lần duy nhất.
- **next:** con trỏ trỏ đến task tiếp theo trong danh sách liên kết.

Khi được khởi tạo, Scheduler sẽ tạo ra một container chứa số tượng task, sỗ lượng vị trí trống và đuôi của danh sách liên kết vòng.

Khi cập nhật task, scheduler phải đến cập nhật thời gian delay của từng task, điều này có thể gây ra nested interrupt nếu có quá nhiều task cần cập nhật. Do đó, để tối ưu hàm SCH_Update, ta thiết lập thời gian delay của task sau phụ thuộc vào task liền trước, nghĩ là task 1 sẽ chạy với thời gian delay thật, task 2 sẽ chạy với thời gian delay so với task 1, task 3 sẽ chạy với thời gian delay so với task 2,... Bằng cách này, mỗi khi được gọi, hàm SCH_Update sẽ chỉ cần cập nhật task đầu tiên thay vì cập nhật tất cả task.

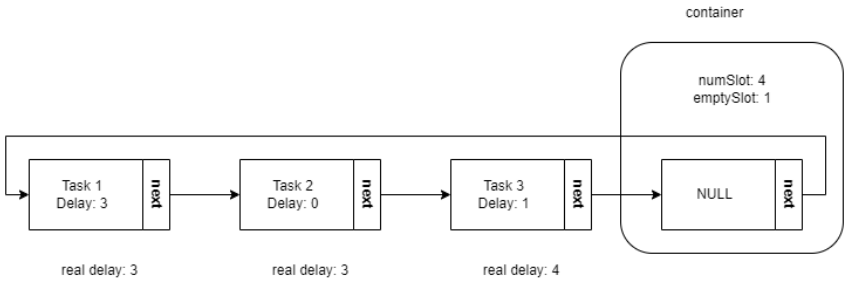
Khi thêm task, trước tiên scheduler sẽ kiểm tra xem trong danh sách liên kết có node nào đang trống không. Nếu không, scheduler sẽ yêu cầu hệ thống cấp phát bộ nhớ, ngược lại scheduler sẽ lấy node trống đầu tiên đó để lưu task. Tiếp theo scheduler sẽ kiểm tra lần lượt từ task đầu tiên đến khi tìm được task có thời gian delay thật lớn hơn task được thêm. Trong trường hợp scheduler xin cấp phát bộ nhớ, việc

thêm task chỉ đơn giản là chèn 1 node vào danh sách liên kết, trong trường hợp scheduler dùng node trống có sẵn thì thực hiện đổi chỗ node đó đến trước node đầu tiên có thời gian delay thật lớn hơn. Điều này sẽ giúp cho hệ thống tiết kiệm bộ nhớ dùng cho việc cấp phát chỗ trống. thời gian delay của node vừa được thêm sẽ được tính lại theo mốc là task liền trước.

Khi xoá task, do STM32 không thật sự giải phóng bộ nhớ mà chỉ xoá dữ liệu trong vùng nhớ đó, nên scheduler được thiết kế để xoá dữ liệu và đưa node đó về cuối danh sách liên kết để tận dụng lại khi thêm task.

Khi thực hiện task, scheduler sẽ kiểm tra và thực hiện tất cả các task có thời gian delay bằng 0. Sau khi thực hiện, nếu task đó cần được gọi lặp lại thì sẽ tiến hành xoá task và thêm lại task.

Scheduler có thể được minh hoạ như sau:



Hình 5: *cấu trúc scheduler*

Danh sách liên kết vòng được sử dụng thay vì mảng để số lượng task có thể thực hiện, đồng thời tiết kiệm bộ nhớ trong trường hợp chương trình có quá ít task. Danh sách liên kết vòng được sử dụng thay cho danh sách liên kết đơn nhằm mục đích giảm số lượng câu lệnh xong hàm xoá task.

Chương 2: HIỆN THỰC

1 Các công cụ

1.1 Các khai báo và biến toàn cục

Ta sẽ định nghĩa các hàm và biến cần sử dụng cho hệ thống, các biến và định nghĩa được sử dụng ở các file khác nhau sẽ được đặt ở khai báo toàn cục này. Ta có file global.h như sau:

```
#ifndef INC_GLOBAL_H_
#define INC_GLOBAL_H_

#include "main.h"
#include "button.h"
#include "fsm.h"
#include "physics.h"
#include "scheduler.h"
#include "software_timer.h"
#include <stdio.h>

#define TIME_CYCLE  10
#define NUM_TIMER 4

//global init and off status
#define INIT    10
#define OFF     0

//MODE OF SYSTEM
#define AUTO_MODE   1
#define MANUAL_MODE 2
#define TUNING_MODE 3
#define ERROR_MODE  4
//STATE for AUTO / MANUAL SETTING
#define RED_GREEN 1
#define RED_AMBER 2
#define GREEN_RED 3
#define AMBER_RED 4

//STATE for TUNING SETTING
#define RED_ADJ   1
#define AMBER_ADJ 2
#define GREEN_ADJ 3

extern int timeRed;
extern int timeAmber;
extern int timeGreen;

//STATE for PEDESTRIAN
#define RED_PED   1
#define GREEN_PED 2

//STATUS of buzzer
#define ON      1

```

```
//String use for UART
extern char str[];

//Interrupt use for PWM control
extern TIM_HandleTypeDef htim3;


extern UART_HandleTypeDef huart2;


//Variable use for FSM
extern int autoStatus;
extern int manualStatus;
extern int tuningStatus;
extern int pedStatus;

#endif /* INC_GLOBAL_H_ */
```

Trong hệ thống này, ta sẽ sử dụng TIME_CYCLE là 10 do interrupt ta thiết lập là 10ms, có 4 biến timer được sử dụng.

INIT là định nghĩa được sử dụng chung cho các máy trạng thái, OFF là định nghĩa được sử dụng chung cho các các máy trạng thái nhỏ của từng chế độ. Đối với máy trạng thái lớn cho các chế độ, để thuận tiện và dễ dàng trong việc định nghĩa hệ thống, các chế độ của hệ thống như AUTO_MODE, MANUAL_MODE, TUNING_MODE và ERROR_MODE được thiết lập. Đối từng chế độ, ta cũng định nghĩa các trạng thái chúng như: của AUTO MODE và MANUAL MODE là RED_GREEN, RED_AMBER, GREEN_RED, AMBER_RED tương ứng với 4 trạng thái của hệ thống đèn giao thông ở ngã tư; của TUNING MODE là RED_ADJ, AMBER_ADJ và GREEN_ADJ tương ứng với các trạng thái điều chỉnh giá trị thời gian của từng đèn. Ngoài ra, để lưu giá trị và thay đổi thời gian sáng của các đèn, các biến timeRed, timeAmber, timeGreen được khai báo extern.

Trạng thái của người đi bộ là RED_PED và GREEN_PED, trạng thái của còi dành cho người đi bộ gồm OFF đã định nghĩa ở trên và ON.

Để điều khiển PWM và truyền UART, interrupt htim3 và huart2 được khai báo extern.

Với các trạng thái của từng chế độ, ta có các biến để lưu giữ từng trạng thái của chúng bao gồm autoStatus, manualStatus, tunningStatus và pedStatus, và được khai báo extern.

1.2 Software timer

Trong phần software_timer, ta sẽ định nghĩa file software_timer.h như sau:

```
#ifndef INC_SOFTWARE_TIMER_H_
#define INC_SOFTWARE_TIMER_H_

#include "global.h"

//Variables use for timer
extern int timer_flag[];
extern int timer_duration[];
void setTimer(int duration, int index);
void timerRun();
```

Code 1: software_timer.h

Để dễ dàng quản lý mã nguồn và tránh việc tạo ra nhiều hàm có tương tự mục đích, ta sẽ thiết lập các timer_flag có cấu trúc dữ liệu là array.

Mã nguồn software_timer.c:

```c
#include "software_timer.h"

//  Array of timer variable
//  timer_counter[0]: counter for auto mode with F = 1HZ
//  timer_counter[1]: counter for blink with F = 4HZ
//  timer_counter[2]: for pedestrian count down, = 4*timeRed
//  timer_counter[3]: counter for buzzer

int timer_counter[NUM_TIMER] = {0};
int timer_flag[NUM_TIMER] = {0};
int timer_duration[NUM_TIMER] = {1000, 250, 0, 0};

// set timer flag and timer counter
void setTimer(int duration, int index){
  timer_counter[index] = duration / TIME_CYCLE;
  timer_flag[index] = 0;
}

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

Code 2: Software_timer.c

Ta sẽ khai báo tất cả các biến array cần dùng với kích thước NUM_TIMER được định nghĩa trong global. Hàm **setTimer(int duration, int index)** sẽ thiết lập giá trị cho timer_counter phù hợp, với duration là giá trị tính bằng mili giây, index và chỉ số thứ tự của biến timer. **timerRun()** sẽ cập nhật giá trị counter cho tất cả các biến timer_counter, và bật các cờ tương ứng khi giá trị counter nhỏ hơn 0.

1.3 Nút nhấn

Trong phần xử lý nút nhấn, ta sẽ định nghĩa file button.h như sau:

```c
#ifndef INC_BUTTON_H_
#define INC_BUTTON_H_

#include "global.h"
#define NUM_BUTTON  4

#define PRESSED 0
#define RELEASE 1
#define KEY_PRESS_TIME 1000


```

```
12 void getButtonValue(void);
13 int isButtonShortPress(int index);
14 int isButtonLongPress(int index);
15 #endif /* INC_BUTTON_H_ */
```

Code 3: button.h

Trong hệ thống này, ta sẽ có 4 nút nhấn nên sẽ đặc tả NUM_BUTTON là 4, và vì các nút nhấn được cấu hình là in put pull up nên trạng thái nhấn sẽ là 0, thả là 1. Ngoài ra, để phân biệt nút nhấn ngắn (nhấn thả) và nút nhấn dài (nhấn giữ), ta sẽ thiết lập thời gian KEY_PRESS_TIME là 1000ms để phân biệt 2 loại nhấn dài và ngắn.

Mã nguồn của file button.c như sau:

```
1  #include "button.h"
2
3  struct keyget {
4    int keyget1;
5    int keyget2;
6    int keyget3;
7    int oldState;
8    int shortPress;
9    int longPress;
10   int timePress;
11 };
12
13 struct keyget listButton[NUM_BUTTON] = {{RELEASE, RELEASE, RELEASE, RELEASE, 0, 0}};
14 GPIO_TypeDef * BUTTON_GPIO[NUM_BUTTON] = {BUTTON1_GPIO_Port, BUTTON2_GPIO_Port,
       BUTTON3_GPIO_Port, BUTTON_PERD_GPIO_Port};
15 uint16_t BUTTON_PIN[NUM_BUTTON] = {BUTTON1_Pin, BUTTON2_Pin, BUTTON3_Pin, BUTTON_PERD_Pin
       };
16
17 /*
18  * @brief:  get all the button value
19  *
20  * @param:  NONE
21  * @retval: NONE
22  */
23
24 void getButtonValue(void) {
25   for (int i = 0; i < NUM_BUTTON; i++){
26     listButton[i].keyget1 = listButton[i].keyget2;
27     listButton[i].keyget2 = listButton[i].keyget3;
28     listButton[i].keyget3 = HAL_GPIO_ReadPin(BUTTON_GPIO[i], BUTTON_PIN[i]);
29     if((listButton[i].keyget1 == listButton[i].keyget2) && (listButton[i].keyget2 ==
       listButton[i].keyget3)){
30       if (listButton[i].oldState != listButton[i].keyget3){
31         listButton[i].oldState = listButton[i].keyget3;
32         if (listButton[i].keyget3 == PRESSED){
33           listButton[i].shortPress = 1;
34           listButton[i].timePress = KEY_PRESS_TIME / TIME_CYCLE;
35         }
36         else if (listButton[i].keyget3 == RELEASE){
37           // if new state != old state and button is released -> Short Press is complete
38            if(listButton[i].shortPress == 1) listButton[i].shortPress = 2;
39         }
```

```
      }
      else if (listButton[i].keyget3 == PRESSED){
        listButton[i].timePress--;
        if (listButton[i].timePress == 0){
          listButton[i].longPress = 1;
          listButton[i].shortPress = 0; // When press time >= KEY_PRESS_TIME -> long press -> terminate short press
        }
      }
    }
  }
}


/*
 * @brief:  check if the button is pressed or not
 *
 * @param:  index of button [0...3]
 * @retval: 1 - button is pressed
 *      0 - button is not pressed
 */

int isButtonShortPress(int index){
  if (listButton[index].shortPress == 2){
    listButton[index].shortPress = 0;
    return 1;
  }
  else return 0;
}
/*
 * @brief:  check if the button is long pressed or not
 *
 * @param:  index of button [0...3]
 * @retval: 1 - button is long pressed
 *      0 - button is not long pressed
 */
int isButtonLongPress(int index){
  if (listButton[index].longPress == 1){
    listButton[index].longPress = 0;
    return 1;
  }
  else return 0;
}
```

Code 4: button.c

Do có nhiều nút nhấn nên ta sẽ định nghĩa một cấu trúc dữ liệu cho mỗi nút nhấn, gọi là **keyget** và tạo các array **listButton** với kích thước NUM_BUTTON để lưu các nút nhấn.

Hàm **getButtonValue(void)** có tác dụng lấy giá trị của các nút nhấn của cả hệ thống. Trong hàm này, ta sẽ xử lý nhiễu bằng cách lấy giá trị của 3 lần liên tiếp. Thêm vào đó, khi nút nhấn được nhấn, ta sẽ bật cồ shortPress lên 1 và biến timePress sẽ là biến dùng để xử lý nút nhấn dài. Khi người dùng nhấn nút nhấn hơn 1 giây, giá trị timePress sẽ giảm dần tới 0 và bật cờ longPress, đồng thời tắt cờ shortPress. Nếu người dùng thả nút nhấn khi chưa đủ một giây, cờ shortPress sẽ được cập nhật giá trị 2, tức là hoàn thành 1 lần nhấn ngắn.

Hàm **isButtonShortPress(int index)** và **isButtonLongPress(int index)** sẽ là các API để lập trình các FSM sử dụng. Hàm có tác dụng kiểm tra các nút nhấn có được nhấn dài/ngắn không, nếu có thì reset cờ phù hợp và trả về 1, ngược lại thì trả về 0.

1.4 Bật/tắt đèn

Nhằm dễ dàng quản lý hệ thống và cô lập việc điều khiển phần cứng, ta sẽ thiết lập các API để gọi từ các FSM và hệ thống cấp cao khác. Ta định nghĩa physics.h như sau:

```
#ifndef INC_PHYSICS_H_
#define INC_PHYSICS_H_

#include "global.h"

void turnOffAllLED(void);

void turnOnAllLED(void);

void clearRoadLed(void);

//AUTO / MANUAL MODE:
//turn on the led with index: 0: way 1
//                1: way 2
void turnOnRed(int index);
void turnOnAmber(int index);
void turnOnGreen(int index);

//TUNING MODE:
//Blink the suitable led
void blinkyRed(void);
void blinkyAmber(void);
void blinkyGreen(void);

/*
 * @brief: set the pedestrian led
 *
 * @param:  0 - RED
 *      1 - GREEN
 * @retval: None
 */
void setPedestrianLed(int index);
void unsetPedestrianLed(int index);
#endif /* INC_PHYSICS_H_ */
```

Code 5: physics.h

Ta sẽ định nghĩa các hàm để bật các đèn đỏ, vàng và xanh ở auto và manual mode thông qua hàm **turnOnRed(int index)**, **turnOnAmber(int index)**,**turnOnGreen(int index)** với index là 0 hoặc 1 thể hiện bật đèn ở main way hoặc side way.

Hàm **BlinkyRed(void)**, **BlinkyAmber(void)** và **BlinkyGreen(void)** sẽ phục vụ cho trạng thái tuning mode, và các hàm **setPedestrianLed(int index)** và **unsetPedestrianLed(int index)** để bật tắt đèn của người đi bộ, với index là 0 và 1 tương ứng với đỏ và xanh.

Mã nguồn file physics.c:

```c
#include "physics.h"

void turnOffAllLED(void){
  HAL_GPIO_WritePin(GPIOA, RED_LED1_Pin | RED_LED2_Pin |AMBER_LED1_Pin
      | AMBER_LED2_Pin | GREEN_LED1_Pin | GREEN_LED2_Pin | RED_PERD_Pin | GREEN_PERD_Pin,
     1);
}

void turnOnAllLED(void){
  HAL_GPIO_WritePin(GPIOA, RED_LED1_Pin | RED_LED2_Pin |AMBER_LED1_Pin
       | AMBER_LED2_Pin | GREEN_LED1_Pin | GREEN_LED2_Pin | RED_PERD_Pin |
    GREEN_PERD_Pin, 0);
}

//AUTO / MANUAL MODE:
//turn on the led with index: 0: way 1
//                1: way 2
void turnOnRed(int index){
  if (index == 0){
    HAL_GPIO_WritePin(GPIOA, RED_LED1_Pin, 0);
    HAL_GPIO_WritePin(GPIOA, AMBER_LED1_Pin | GREEN_LED1_Pin, 1);
  }
  else if (index == 1){
    HAL_GPIO_WritePin(GPIOA, RED_LED2_Pin, 0);
    HAL_GPIO_WritePin(GPIOA, AMBER_LED2_Pin | GREEN_LED2_Pin, 1);
  }
}
//turn on the led with index: 0: way 1
//                1: way 2
void turnOnAmber(int index){
  if (index == 0){
    HAL_GPIO_WritePin(GPIOA, AMBER_LED1_Pin, 0);
    HAL_GPIO_WritePin(GPIOA, RED_LED1_Pin | GREEN_LED1_Pin, 1);
  }
  else if (index == 1){
    HAL_GPIO_WritePin(GPIOA, AMBER_LED2_Pin, 0);
    HAL_GPIO_WritePin(GPIOA, RED_LED2_Pin| GREEN_LED2_Pin, 1);
  }
}

//turn on the led with index: 0: way 1
//                1: way 2
void turnOnGreen(int index){
  if (index == 0){
    HAL_GPIO_WritePin(GPIOA, GREEN_LED1_Pin, 0);
    HAL_GPIO_WritePin(GPIOA, RED_LED1_Pin | AMBER_LED1_Pin, 1);
  }
  else if (index == 1){
    HAL_GPIO_WritePin(GPIOA, GREEN_LED2_Pin, 0);
    HAL_GPIO_WritePin(GPIOA, RED_LED2_Pin| AMBER_LED2_Pin, 1);
  }
}

//TUNING MODE:
//Blink the suitable led
```

```c
void blinkyRed(void){
  if(timer_flag[1] == 1){
    HAL_GPIO_TogglePin(GPIOA, RED_LED1_Pin | RED_LED2_Pin);
    setTimer(timer_duration[1], 1);
  }
}

void blinkyAmber(void){
  if(timer_flag[1] == 1){
    HAL_GPIO_TogglePin(GPIOA, AMBER_LED1_Pin | AMBER_LED2_Pin);
    setTimer(timer_duration[1], 1);
  }
}

void blinkyGreen(void){
  if (timer_flag[1] == 1){
    HAL_GPIO_TogglePin(GPIOA, GREEN_LED1_Pin | GREEN_LED2_Pin);
    setTimer(timer_duration[1], 1);
  }
}

void clearRoadLed(void){
  HAL_GPIO_WritePin(GPIOA, RED_LED1_Pin | RED_LED2_Pin |
      AMBER_LED1_Pin | AMBER_LED2_Pin | GREEN_LED1_Pin | GREEN_LED2_Pin, 1);
}

/*
 * @brief: set the pedestrian led
 *
 * @param:  0 - RED
 *      1 - GREEN
 * @retval: None
 */
void setPedestrianLed(int index){
  if(index == 0){
    HAL_GPIO_WritePin(GPIOA, RED_PERD_Pin, 0);
    HAL_GPIO_WritePin(GPIOA, GREEN_PERD_Pin, 1);
  }
  else if (index == 1){
    HAL_GPIO_WritePin(GPIOA, RED_PERD_Pin, 1);
    HAL_GPIO_WritePin(GPIOA, GREEN_PERD_Pin, 0);
  }
}

void unsetPedestrianLed(int index) {
  if (index == 0)
    HAL_GPIO_WritePin(GPIOA, RED_PERD_Pin, 1);
  else if (index == 1)
    HAL_GPIO_WritePin(GPIOA, GREEN_PERD_Pin, 1);
}
```

Code 6: physics.c

Như đã mô tả ở trên, các hàm bật/tắt các đèn và nhấp nháy được định nghĩa. Do đèn của người đi bộ là led 3 màu nên việc bật và tắt cần đồng thời điều khiển 2 tín hiệu RED và GREEN.

1.5 Bộ định thời (Scheduler)

Các hàm có thể được gọi bởi các file khác được khai báo ở scheduler.h như sau:

```
#include "global.h"

void SCH_Init(void);
void SCH_Add_Task(void(*pFunction)(), uint32_t DELAY, uint32_t PERIOD);
void SCH_Update(void);
void SCH_Dispatch_Tasks(void);
```

Code 7: scheduler.h

Trong scheduler.c, scheduler được triển khai như sau:

Cấu trúc container và node:

```
struct sTask{
  void (* pTask)(void);
  uint32_t Delay;
  uint32_t Period;
  struct sTask *next;
};

struct container {
  struct sTask *tail;
  int numSlot;
  int emptySlot;
};

struct container* container;
```

Code 8: Container và node lưu trữ các task

Như đã nêu trong thiết kế, mỗi node sẽ có 4 thuộc tính là địa chỉ hàm cần thực thi (pTask), thời gian trì hoãn (Delay), chu kì (Period), con trỏ tới task tiếp theo (next). Container có 3 thuộc tính, dùng để lưu trữ đuôi của danh sách liên kết vòng và số node, số node rỗng trong đó.

Hàm SCH_Init(void)

```
void SCH_Init(void) {
    container = (struct container*)malloc(sizeof(struct container));
    container -> tail = NULL;
    container -> numSlot = 0;
    container -> emptySlot = 0;
}
```

Code 9: Hàm khởi tạo scheduler

Hàm SCH_Add_Task(void(*pFunction)(), uint32_t DELAY, uint32_t PERIOD)

```
struct sTask* Add_Node(struct sTask** curr, void(*pFunction)(), uint32_t DELAY, uint32_t
    PERIOD) {
    struct sTask *temp = (struct sTask*)malloc(sizeof(struct sTask));
    temp -> pTask = pFunction;
    temp -> Delay = DELAY;
    temp -> Period = PERIOD;
    if (curr == NULL || *curr == NULL) {
        temp -> next = temp;
    }
```

```
    else {
        temp -> next = (*curr) -> next;
    }
    return temp;
}

void SCH_Add_Task(void(*pFunction)(), uint32_t DELAY, uint32_t PERIOD) {
  // container empty
  if (container -> tail == NULL) {
    container -> tail = Add_Node(NULL, pFunction, DELAY, PERIOD);
    (container -> numSlot)++;
  }
  else {
    struct sTask* temp = container -> tail;
    uint32_t sumDelay = 0;
    uint32_t newDelay = 0;
    // find first task with bigger delay
    for (int i = 0; i < container -> numSlot; i++) {
      sumDelay = sumDelay + temp -> next -> Delay;
      // add head or middle
      if (sumDelay > DELAY) {
                // delay of new task
        newDelay = DELAY - (sumDelay - temp -> next -> Delay);
        temp -> next -> Delay = sumDelay - DELAY;
        // create new node if there is no empty slot
        if (container -> emptySlot == 0) {
          temp -> next = Add_Node(&temp, pFunction, newDelay, PERIOD);
          (container -> numSlot)++;
                }
        // if is, move tail to after temp and use it
        else {
          container -> tail -> pTask = pFunction;
          container -> tail -> Delay = newDelay;
          container -> tail -> Period = PERIOD;
          struct sTask *newTail = temp -> next;
          while (newTail -> next != container -> tail) {
            newTail = newTail -> next;
          }
          if (temp == container -> tail) container -> tail = newTail;
          else {
            newTail -> next = container -> tail -> next;
            container -> tail -> next = temp -> next;
            temp -> next = container -> tail;
            container -> tail = newTail;
          }
                    (container -> emptySlot)--;
        }
                break;
      } else {
        // add tail
        if (temp -> next -> pTask == 0x0000) {
             temp -> next -> pTask = pFunction;
             temp -> next -> Delay = DELAY - sumDelay;
             temp -> next -> Period = PERIOD;
                        (container -> emptySlot)--;
```

```
64            break;
65          }
66          else {
67            if (temp -> next == container -> tail) {
68              container -> tail -> next = Add_Node(&(container -> tail), pFunction, DELAY -
    sumDelay, PERIOD);
69              container -> tail = container -> tail -> next;
70              (container -> numSlot)++;
71              break;
72            }
73          }
74
75        }
76        temp = temp -> next;
77      }
78    }
79  }
```

Code 10: Hàm thêm task

Để thuận tiện cho việc cấp phát bộ nhớ, hàm **Add_Node** được khai báo để sử dụng trong hàm **SCH_Add_Task**. Hàm **SCH_Add_Task** trước tiên tìm ra vị trí thêm node (trước node có delay lớn hơn delay của node cần thêm), chia ra thành 2 trường hợp là chèn ở đầu hoặc giữa và chèn ở cuối. Ở mỗi trường hợp lại chia ra 2 trường hợp nhỏ là có node trống và không có node trống.

Hàm void SCH_Delete_Task(struct sTask preDel)**

```
void SCH_Delete_Task(struct sTask** preDel) {
    struct sTask* del = (*preDel) -> next;
    if (del != container -> tail) del -> next -> Delay += del -> Delay;
  del -> pTask = 0x0000;
  del -> Delay = 0;
  del -> Period = 0;
  if (*preDel == container -> tail)
    container -> tail = container -> tail -> next;
  else {
    if (del -> next -> pTask != 0 && del != container -> tail) {
      (*preDel) -> next = del -> next;
      del -> next = container -> tail -> next;
      container -> tail -> next = del;
      container -> tail = del;
    }
  }
  (container -> emptySlot)++;
}
```

Code 11: Hàm xoá task

Hàm SCH_Update(void)

```
void SCH_Update(void) {
  if (container -> tail) {
    if (container -> tail -> next -> Delay > 0)
      (container -> tail -> next -> Delay)--;
  }
}
```

Code 12: Hàm cập nhật task

Hàm SCH_Dispatch_Tasks(void)

```
void SCH_Dispatch_Tasks(void) {
  while (container -> tail -> next -> Delay <= 0) {
    (*(container -> tail -> next -> pTask))();
    struct sTask temp = *(container -> tail -> next);
    SCH_Delete_Task(&(container -> tail));
    if (temp.Period != 0) {
      SCH_Add_Task(temp.pTask, temp.Period, temp.Period);
    }
  }
}
```

Code 13: Hàm thực hiện task

2 Máy trạng thái

Trong hệ thống này, để tránh lặp lại code và để các chế độ có thể chuyển độc lập, ta sẽ thiết lập máy trạng thái của các chế độ chạy song song với hàm chuyển giữa các chế độ bằng nút nhấn.

Hàm fsm_traffic(void)

```
void fsm_traffic(void){
  switch(trafficMode){
  case INIT:
    if (timeRed != (timeAmber + timeGreen)){
      trafficMode = ERROR_MODE;
      HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!TIMER ERROR#\r\n"),500);
      autoStatus = OFF;
      manualStatus = OFF;
      tuningStatus = OFF;
      pedStatus = OFF;
    }
    else {
      trafficMode = AUTO_MODE;
      autoStatus = INIT;
      manualStatus = OFF;
      tuningStatus = OFF;
      pedStatus = OFF;
      setTimer(timer_duration[0], 0);
    }
    turnOffAllLED();
    break;
  case AUTO_MODE:
    if (isButtonShortPress(3)) {
      if (pedStatus == OFF) {
        pedStatus = INIT;
        setTimer(4*1000*timeRed, 2);
      }
      else pedStatus = OFF;
    }
    if (timer_flag[2] == 1) pedStatus = OFF;
    fsm_pedestrian();
    fsm_traffic_auto_mode();
    break;
```

```
  case MANUAL_MODE:
    fsm_traffic_manual_mode();
    break;
  case TUNING_MODE:
    fsm_traffic_tunning_mode();
    break;
  default:
    break;
  }
}
```

Code 14: Máy trạng thái lớn của các chế độ

Hàm fsm_switch_mode(void)

```
void fsm_switch_mode(void){
  if (isButtonLongPress(0)){
    if (trafficMode != MANUAL_MODE) {
      trafficMode = MANUAL_MODE;
      HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!SWITCH TO MANUAL MODE#\r\n")
    ,500);
      // TODO: prepare for manual mode
      manualStatus = INIT;
      autoStatus= OFF;
      tuningStatus = OFF;
      pedStatus = OFF;
      fsm_pedestrian();
    }
    else {
      trafficMode = INIT;
      HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!SWITCH TO AUTO MODE#\r\n")
    ,500);

    }
  }
  else if (isButtonLongPress(1)){
    if (trafficMode != TUNING_MODE) {
      trafficMode = TUNING_MODE;
      HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!SWITCH TO TUNING MODE#\r\n")
    ,500);
      // TODO: prepare for next mode
      tuningStatus = INIT;
      autoStatus = OFF;
      manualStatus = OFF;
      pedStatus = OFF;
      fsm_pedestrian();
    }
    else {
      trafficMode = INIT;
      HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!SWITCH TO AUTO MODE#\r\n")
    ,500);
    }
  }
}
```

Code 15: Hàm chuyển chế độ bằng nút nhấn

Khi chuyển chế độ, trạng thái của các chế độ không được gọi đều chuyển về OFF, chế độ được gọi sẽ chuyển sang trạng thái INIT. Chỉ có nút nhấn 1 và 2 được dùng để chuyển chế độ.

3 Auto mode

Chế độ này gọi một máy trạng thái nhỏ hơn để thực hiện hành vi của hệ thống:

```
void fsm_traffic_auto_mode(void){
  switch(autoStatus){
  case OFF:
    // do nothing
    break;
  case INIT:
    autoStatus = RED_GREEN;
    // prepare for red green state
    clock_counter_main = timeRed;
    clock_counter_side = timeGreen;
    clearRoadLed();
    break;
  case RED_GREEN:
    turnOnRed(0);
    turnOnGreen(1);
    if (clock_counter_side == 0){
      autoStatus = RED_AMBER;
//      clock_counter_main = timeRed - timeGreen;
      clock_counter_side = timeAmber;
      clearRoadLed();
    }
    break;
  case RED_AMBER:
    turnOnRed(0);
    turnOnAmber(1);
    if (clock_counter_side == 0){
      autoStatus = GREEN_RED;
      clock_counter_main = timeGreen;
      clock_counter_side = timeRed;
      clearRoadLed();
    }
    break;
  case GREEN_RED:
    turnOnGreen(0);
    turnOnRed(1);
    if (clock_counter_main == 0){
      autoStatus = AMBER_RED;
      clock_counter_main = timeAmber;
//      clock_counter_side = timeRed - timeGreen;
      clearRoadLed();
    }
    break;
  case AMBER_RED:
    turnOnAmber(0);
    turnOnRed(1);
    if (clock_counter_main == 0){
      autoStatus = RED_GREEN;
```

```
      clock_counter_main = timeRed;
      clock_counter_side = timeGreen;
      clearRoadLed();
    }
    break;
  default:
    break;
  }
}
```

Code 16: Máy trạng thái của AUTO MODE

Một máy trạng thái nhỏ khác được gọi cùng với hàm **fsm_traffic_auto_mode** tại AUTO MODE là **fsm_pedestrian** dùng cho người đi bộ, kích hoạt bằng cách nhấn thả nút nhấn 4:

```
void fsm_pedestrian(void){
  switch(pedStatus){
  case OFF:
    unsetPedestrianLed(0);
    unsetPedestrianLed(1);
    Buzzer.status = OFF;
    setBuzzer();
    break;
  case INIT:
    if (autoStatus == RED_GREEN || autoStatus == RED_AMBER) pedStatus = GREEN_PED;
    else if (autoStatus == GREEN_RED || autoStatus == AMBER_RED) pedStatus = RED_PED;
    Buzzer.status = INIT;
    break;
  case GREEN_PED:
    setPedestrianLed(1);
    if (clock_counter_main <= 3 && clock_counter_main > 0) {
      Buzzer.period = clock_counter_main*100;
      Buzzer.volume = 12000/clock_counter_main;
      setBuzzer();
    }
    if (clock_counter_main <= 0) {
      pedStatus = RED_PED;
      Buzzer.status = OFF;
      setBuzzer();
    }
    break;
  case RED_PED:
    setPedestrianLed(0);
    if (clock_counter_side <= 0) {
      pedStatus = GREEN_PED;
    }
    break;
  default:
    break;
  }
}
```

Code 17: Máy trạng thái của chức năng cho người đi bộ

Còi báo hiệu sắp hết giờ cho người đi bộ cũng được xây dựng như một máy trạng thái để điều chỉnh cho còi kêu ngày càng to và ngày càng nhanh:

```c
struct buzzer {
  int status;
  int volume;
  int period;
};
struct buzzer Buzzer = {OFF, 0, 0};

void setBuzzer(void){
  switch (Buzzer.status) {
  case INIT:
    setTimer(Buzzer.period, 3);
    Buzzer.status = ON;
    break;
  case OFF:
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, 0);
    if (timer_flag[3]) {
      setTimer(100, 3);
      Buzzer.status = ON;
    }
    break;
  case ON:
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, Buzzer.volume);
    if (timer_flag[3]) {
    setTimer(Buzzer.period, 3);
    Buzzer.status = OFF;
    }
    break;
  default:
    break;
  }
}
```

Code 18: Máy trạng thái của còi cho người đi bộ

Ngoài ra, để hiển thị thời gian đếm ngược của đèn giao thông sau mỗi giây, hàm **clock_counter_traffic_update** được thiết kế sử dụng set timer và gọi liên tục trong scheduler:

```c
void clock_counter_traffic_update(void){
  if((timer_flag[0] == 1) && (trafficMode == AUTO_MODE)){
    clock_counter_main--;
    clock_counter_side--;
    HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "\n!7SEG WAY1:%d#\r\n",
    clock_counter_main),500);
    HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!7SEG WAY2:%d#\r\n",
    clock_counter_side),500);
    if (pedStatus != OFF) {
      if (pedStatus == GREEN_PED)
        HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!PEDESTRIAN: %d#\r\n",
    clock_counter_main),500);
      else if (pedStatus == RED_PED)
        HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!PEDESTRIAN: %d#\r\n",
    clock_counter_side),500);
    } setTimer(timer_duration[0], 0);
    }
}
```

Code 19: Hàm in thời gian đếm ngược

4 Manunal mode

Chế độ này gọi tới máy trạng thái nhỏ hơn để thực hiện hành vi của hệ thống:

```
void fsm_traffic_manual_mode(void){
  switch(manualStatus){
  case OFF:
    // do nothing
    break;
  case INIT:
    manualStatus = RED_GREEN;
    clearRoadLed();
    break;
  case RED_GREEN:
    turnOnRed(0);
    turnOnGreen(1);
    if (isButtonShortPress(0)){
      manualStatus = RED_AMBER;
      clearRoadLed();
    }
    break;
  case RED_AMBER:
    turnOnRed(0);
    turnOnAmber(1);
    if(isButtonShortPress(0)){
      manualStatus = GREEN_RED;
      clearRoadLed();
    }
    break;
  case GREEN_RED:
    turnOnGreen(0);
    turnOnRed(1);
    if(isButtonShortPress(0)){
      manualStatus = AMBER_RED;
      clearRoadLed();
    }
    break;
  case AMBER_RED:
    turnOnAmber(0);
    turnOnRed(1);
    if(isButtonShortPress(0)){
      manualStatus = RED_GREEN;
      clearRoadLed();
    }
    break;
  default:
    break;
  }
}
```

Code 20: Máy trạng thái của MANUAL MODE

Hàm này tương tự với máy trạng thái của AUTO MODE, điểm khác biệt là máy trạng thái này không tự động chuyển trạng thái mà chuyển bằng cách nhấn nút.

5 Tuning mode

Chế độ này gọi tới máy trạng thái nhỏ hơn để thực hiện hành vi của hệ thống:

```
void fsm_traffic_tunning_mode(void){
  switch(tuningStatus){
  case OFF:
    // do nothing
  case INIT:
    tuningStatus = RED_ADJ;
    clearRoadLed();
    logNewTime();
    setTimer(timer_duration[1], 1);
    break;
  case RED_ADJ:
    blinkyRed();
    if (isButtonShortPress(2)){
      timeRed++;
      if (timeRed > 99) timeRed = 0;
      logNewTime();
    }
    else if (isButtonLongPress(2)){
      timeRed--;
      if (timeRed < 0) timeRed = 99;
      logNewTime();
    }
    else if (isButtonShortPress(1)){
      tuningStatus = GREEN_ADJ;
      clearRoadLed();
      logNewTime();
    }
    break;
  case GREEN_ADJ:
    blinkyGreen();
    if(isButtonShortPress(2)){
      timeGreen++;
      if (timeGreen > 99) timeGreen = 0;
      logNewTime();
    }
    else if(isButtonLongPress(2)){
      timeGreen--;
      if (timeGreen < 0) timeGreen = 99;
      logNewTime();
    }
    else if(isButtonShortPress(1)){
      tuningStatus = AMBER_ADJ;
      clearRoadLed();
      logNewTime();
    }
    break;
  case AMBER_ADJ:
    blinkyAmber();
    if (isButtonShortPress(2)){
      timeAmber++;
      if (timeAmber > 99) timeAmber = 0;
      logNewTime();
```

```
    }
    else if(isButtonLongPress(2)){
      timeAmber--;
      if (timeAmber < 0) timeAmber = 99;
      logNewTime();
    }
    else if (isButtonShortPress(1)){
      tuningStatus = RED_ADJ;
      clearRoadLed();
      logNewTime();
    }
    break;

  default:
    break;
  }
}
```

Code 21: Máy trạng thái của TUNING MODE

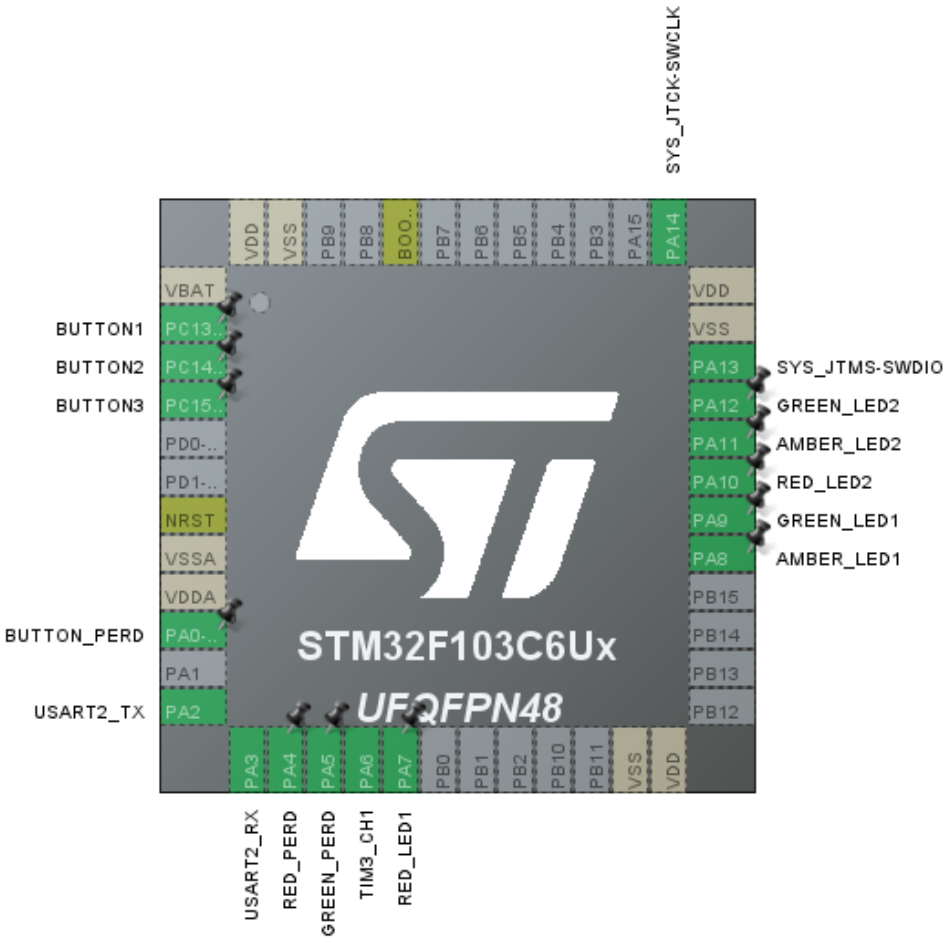
Ở chế độ này, khi nhấn thả nút nhấn 2 thì sẽ chuyển giữa các đèn, nút nhấn 3 dùng để tăng giảm thời gian. Thêm vào đó, mỗi lần thay đổi thời gian sáng của một đèn bất kì thì thời gian sẽ được in ra PC. Do đó cần tạo một hàm để in thời gian:

```
void logNewTime(){
  if (tuningStatus == RED_ADJ) HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!
    TIMER RED :%d#\r\n",timeRed),500);
  if (tuningStatus == AMBER_ADJ) HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!
    TIMER AMBER :%d#\r\n",timeAmber),500);
  if (tuningStatus == GREEN_ADJ) HAL_UART_Transmit(&huart2, (void*)str, sprintf(str, "!
    TIMER GREEN :%d#\r\n",timeGreen),500);
}
```

Code 22: Hàm in thời gian được điều chỉnh

Chương 3: MÔ PHỎNG

1 Cấu hình STM32



Hình 6: *Sơ đồ chân*

Các chân của STM32 được cấu hình như sau:

Nhóm chân PA:

- Chân PA0 dùng cho nút nhấn 4, được cấu hình là input pull up.
- Chân PA2, PA3 dùng để giao tiếp UART với PC
- Chân PA4, PA5, PA7 - PA12 dùng để điều khiển các đèn.
- Chân PA6 dùng để xuất xung PWM cho còi
- Chân PA13, PA14 dùng để debug

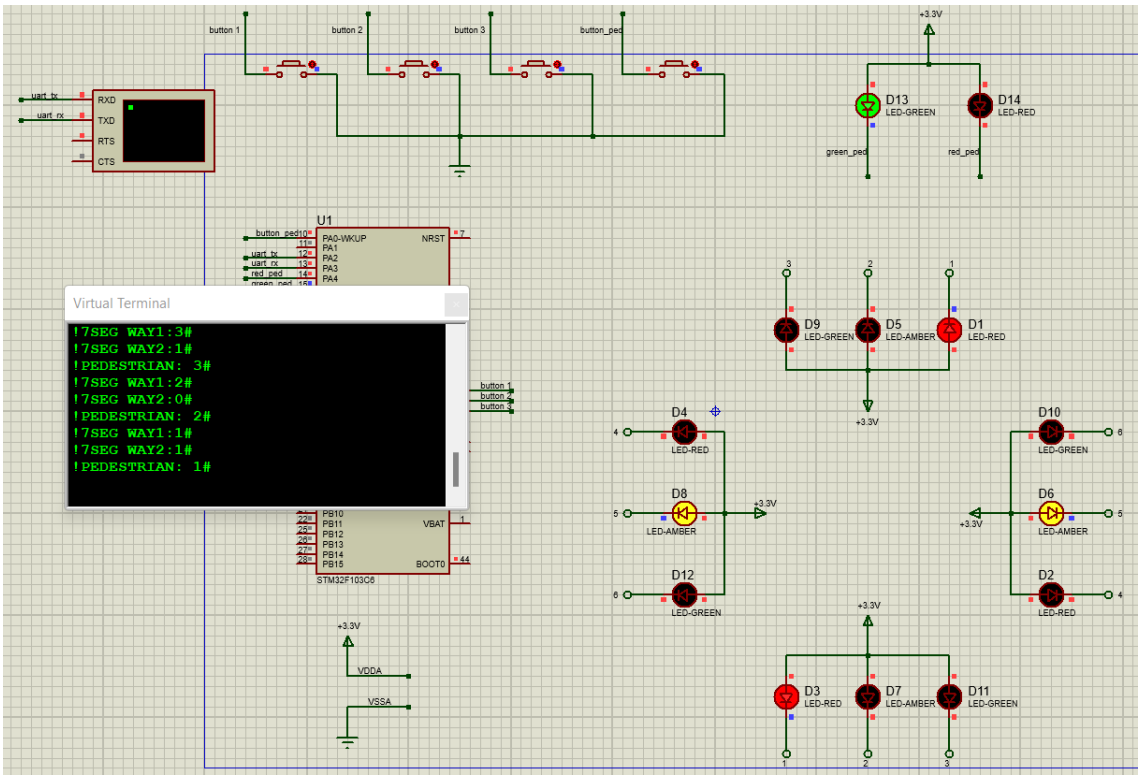
Nhóm chân PC: Chân PC13 - PC15 dùng cho các nút nhấn 1, 2, 3, được cấu hình là input pull up.

TIM2 và TIM3 được sử dụng, trong đó TIM2 để tạo ra ngắt toàn cục mỗi 10ms, TIM3 tạo xung PWM và kích hoạt channel 1 (chân PA6). UART2 được sử dụng ở chế độ bất đồng bộ, ngắt toàn cục để giao tiếp với PC.

Cấu hình chi tiết

- Interrupt TIM2: Prescaler: 7999, counter: 9 đối với STM32F103C6 sử dụng clock 8MHz, và prescaler: 9999 và counter: 63 cho STM32F103RB sử dụng clock 64MHz
- Interrupt TIM3: Prescaler: 7999, counter: 9 đối với STM32F103C6 sử dụng clock 8MHz, và prescaler: 9999 và counter: 63 cho STM32F103RB sử dụng clock 64MHz
- UART: BaudRate: 9600 Bit/s, Word length: 8 Bits, Parity Bit: None, Stop bits: 1.

2 Mô phỏng trên proteus



Hình 7: *Mô phỏng trên proteus*

Tất cả tính năng được thiết kế đều được kiểm tra và chạy chính xác trên mô phỏng lẫn trên mạch thực tế.

LỜI KẾT

Trong báo cáo này, thiết kế và triển khai hệ thống đèn giao thông đa chế độ với các tính năng tự động, điều chỉnh bằng tay, điều chỉnh thời gian và chế độ dành cho người đi bộ đã được trình bày rõ ràng. Việc phát triển và kiểm thử trên mạch thật đã mang lại những kết quả tích cực, với hệ thống chạy ổn định, hiệu quả và đã được kiểm thử trên mạch thật STM32103RB, đã demo vào 15:00 ngày 08/12/2023.

Đặc biệt, thông qua bài tập lớn này, chúng em đã hoàn thiện được những kiến thức cốt lõi của vi xử lý - Vi điều khiển, nắm rõ được các nguyên tắc thiết kế hệ thống vi xử lý - vi điều khiển. Ngoài ra, chúng em còn có cơ hội được sử dụng công cụ quản lý mã nguồn thuần thục như github, hoàn thiện các kỹ năng mềm trong dự án như quản lý tiến độ, phân chia và quản lý các task.