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

VI XỬ LÝ - VI ĐIỀU KHIỂN (CO3009)

Báo cáo Bài tập lớn
THIẾT KẾ HỆ THỐNG ĐÈN GIAO THÔNG

GVHD: Lê Trọng Nhân

Lớp: L01
Sinh viên thực hiện: Vũ Hoàng Thiên An 2110717
Trịnh Hoàng Anh 2110023
Trần Quốc Phong 2110446

TP. Hồ Chí Minh, Tháng 12/2023

Mục lục

1 Phân công

STT	Tên	Nhiệm vụ	Hoàn Thiện	Điểm
1	Vũ Hoàng Thiên An	Hiện thực Automatic Mode	100%	33%
2	Trịnh Hoàng Anh	Hiện thực Manual Mode và Tunning Mode	100%	33%
3	Trần Quốc Phong	Hiện thực pedestrian Mode	100%	33%

2 Github repository

Github repository URL cho final project: https://github.com/thammgh37/Assignment_VXL.git

3 Project Description

3.1 Mô tả nguyên lí hoạt động

Trong Bài tập lớn này, STM32F103RB được sử dụng để hiện thực một hệ thống đèn giao thông ở một giao lộ với những chức năng sau :

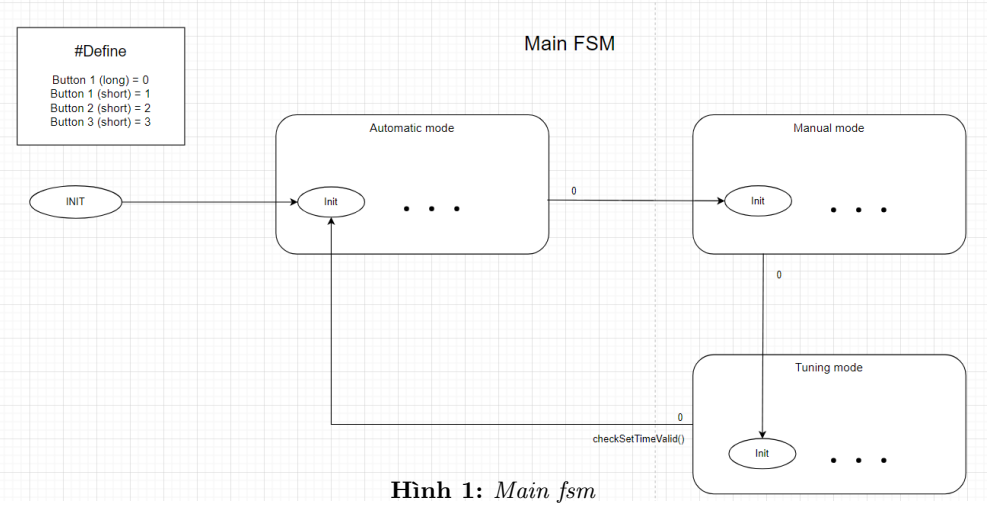
- Automatic mode : Hệ thống đèn giao thông hoạt động bình thường với 3 loại đèn : đỏ, vàng , xanh
- Mannual mode : Hệ thống đèn giao thông sẽ được điều khiển bởi người dùng, 1 nút nhấn được sử dụng để chuyển trạng thái đèn giao thông
- Tuning mode : Trạng thái này cho phép người dùng điều chỉnh thời gian sáng của các đèn
- Pedestrian scramble : 1 nút nhấn, 2 đèn led và 1 loa dành cho người đi bộ được thêm vào. Khi nút nhấn cho người đi bộ được nhấn, đèn led và loa được bật và hoạt động theo mô tả sau :
  - Đèn led dành cho người đi bộ sẽ hoạt động ngược lại với đèn đường. Khi đèn đường là đỏ và vàng, đèn dành cho người đi bộ sẽ là màu xanh và người đi bộ được phép qua đường. Khi đèn đường là xanh, đèn dành cho người đi bộ sẽ là màu đỏ và người đi bộ không được phép qua đường.
  - Loa dành cho người đi bộ sẽ bắt đầu kêu khi đèn dành cho người đi bộ đã ở trạng thái xanh được một nửa thời gian và tắt khi đèn dành cho người đi bộ chuyển sang trạng thái đỏ. Loa sẽ kêu ngày càng nhanh và to.
  - Đèn và loa dành cho người đi bộ sẽ hoạt động trong vòng 2 chu kỳ của đèn đường từ khi nút nhấn cho người đi bộ được nhấn. Nếu vẫn trong thời gian hoạt động và nút nhấn cho người đi bộ được nhấn một lần nữa, thời gian hoạt động 2 chu kỳ sẽ được tính lại từ thời điểm nhấn nút

3.2 Phần cứng được sử dụng để hiện thực project

- Đèn :
  - 2 đèn (đỏ và xanh) cho đường chính
  - 2 đèn (đỏ và xanh) cho đường phụ
  - 2 đền (đỏ và xanh) cho người đi bộ ở đường chính
- Nút nhấn :
  - Button 1 : button dùng để chỉnh và chuyển mode
  - Button 2 : button dùng để tăng giá trị thời gian trong Tunning mode
  - Button 3 : button dùng để save lại giá trị thời gian được setting trong Tunning mode
  - Button 4 : button dùng cho người đi bộ
- Buzzer : Dành cho người đi bộ để thông báo khi thời gian qua đường sắp hết
- UART : Hiện thị thời gian ở các chế độ.

4 System Design

4.1 Main FSM Design



Hình 1: *Main fsm*

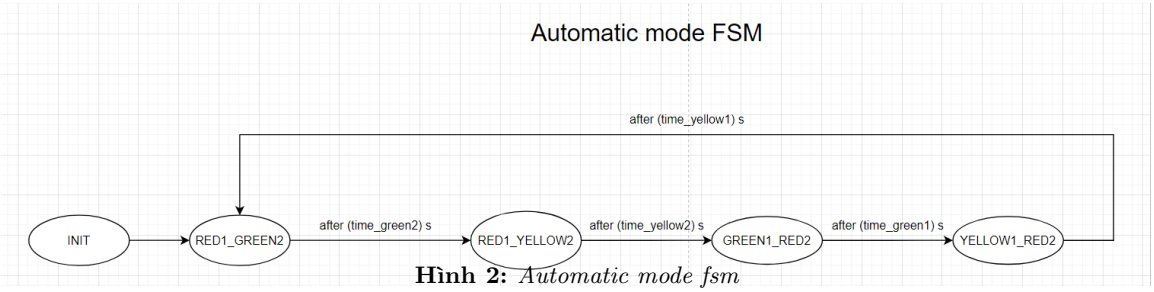
Brief :Đây là máy trạng thái chính của hệ thống có bao gồm các 4 trạng thái : **INIT**, **AUTOMATIC_MODE**, **TUNING_MODE** và **MANUAL_MODE**.

- **INIT**: Trạng thái khởi tạo, khi chương trình bắt đầu chạy sẽ bắt đầu tại đây.
- **AUTOMATIC_MODE** : Ở trạng thái này, một hàm trạng thái con khác sẽ được gọi liên tục để hệ thống có thể hoạt động trong AUTOMATIC_MODE.
- **TUNING_MODE** : Ở trạng thái này, một hàm trạng thái con khác sẽ được gọi liên tục để hệ thống có thể hoạt động trong TUNING_MODE.
- **MANUAL_MODE** : Ở trạng thái này, một hàm trạng thái con khác sẽ được gọi liên tục để hệ thống có thể hoạt động trong MANUAL_MODE.

State	Nhiệm vụ
INIT	• Set biến automatic_state cho automatic FSM • Set main state cho main FSM chuyển sang automatic mode • Set môi trường cho automatic mode
AUTOMATIC_MODE	• Bật đèn sáng tương ứng với trạng thái của automatic state ở 2 đường • Gọi hàm automatic fsm • Check button 1 để chuyển mode và set môi trường cho manual mode nếu có
MANUAL_MODE	• Bật đèn sáng tương ứng với trạng thái của manual state ở 2 đường • Gọi hàm manual fsm • Check button 1 để chuyển mode và set môi trường cho tunning mode nếu có
TUNING_MODE	• Bật đèn sáng tương ứng với trạng thái của tunning state ở 2 đường • Gọi hàm tunning fsm • Check button 1 để chuyển mode và set môi trường cho automatic mode nếu có

Bảng 1: *Thực thi trong máy trạng thái chính*

4.2 Automatic mode FSM Design



Hình 2: *Automatic mode fsm*

Brief :Đây là máy trạng thái được gọi trong Automatic mode của hệ thống có bao gồm 5 trạng thái : **INIT**, **RED1_GREEN2**, **RED1_YELLOW2**, **GREEN1_RED2** VÀ **YELLOW1_RED2**.

- **INIT**: Trạng thái khởi tạo, khi máy trạng thái Automatic mode bắt đầu chạy sẽ bắt đầu tại đây.
- **Các trạng thái còn lại** : Ở những trạng thái này, đèn đường sẽ được bật sáng tương ứng, thời gian sáng còn lại của đèn đường sẽ được cập nhật mỗi 1 giây và việc chuyển trạng thái sẽ phụ thuộc vào thời gian sáng còn lại của đèn đường

State	Nhiệm vụ
INIT	• Set biến automatic_state thành RED1_GREEN2 • Set môi trường cho trạng thái RED1_GREEN2 và các biến liên quan đến thời gian đèn sáng ở các đường
RED1_GREEN2	• Đèn đỏ ở đường 1 và đèn xanh ở đường 2 được bật sáng • Thời gian sáng còn lại của đèn ở 2 đường được cập nhật mỗi 1 giây • Khi thời gian sáng của đèn ở 1 trong 2 đường bằng 0, việc chuyển trạng thái sẽ diễn ra • Chuyển trạng thái : set biến automatic_state thành RED1_YELLOW2, set môi trường cho trạng thái RED1_YELLOW2 và các biến liên quan đến thời gian đèn sáng ở các đường
RED1_YELLOW2	• Đèn đỏ ở đường 1 và đèn vàng (đèn đỏ và xanh cùng bật) ở đường 2 được bật sáng • Thời gian sáng còn lại của đèn ở 2 đường được cập nhật mỗi 1 giây • Khi thời gian sáng của đèn ở 1 trong 2 đường bằng 0, việc chuyển trạng thái sẽ diễn ra • Chuyển trạng thái : set biến automatic_state thành GREEN1_RED2, set môi trường cho trạng thái GREEN1_RED2 và các biến liên quan đến thời gian đèn sáng ở các đường
GREEN1_RED2	• Đèn xanh ở đường 1 và đèn đỏ ở đường 2 được bật sáng • Thời gian sáng còn lại của đèn ở 2 đường được cập nhật mỗi 1 giây • Khi thời gian sáng của đèn ở 1 trong 2 đường bằng 0, việc chuyển trạng thái sẽ diễn ra • Chuyển trạng thái : set biến automatic_state thành YELLOW1_RED2, set môi trường cho trạng thái YELLOW1_RED2 và các biến liên quan đến thời gian đèn sáng ở các đường
YELLOW1_RED2	• Đèn vàng (đèn đỏ và xanh cùng bật) ở đường 1 và đèn đỏ ở đường 2 được bật sáng • Thời gian sáng còn lại của đèn ở 2 đường được cập nhật mỗi 1 giây • Khi thời gian sáng của đèn ở 1 trong 2 đường bằng 0, việc chuyển trạng thái sẽ diễn ra • Chuyển trạng thái : set biến automatic_state thành RED1_GREEN2, set môi trường cho trạng thái RED1_GREEN2 và các biến liên quan đến thời gian đèn sáng ở các đường

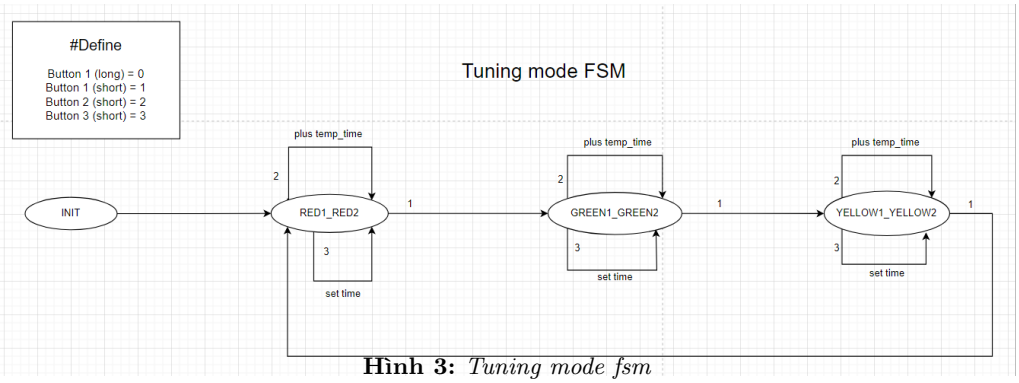
Bảng 2: *Thực thi trong máy trạng thái*

4.3 Manual mode Design

Brief : Do đặc thù thiết kế của manual mode, nên sẽ không có máy trạng thái dành riêng cho manual mode mà chỉ là hàm được tạo ra để đồng bộ code với các mode khác. Ở trong hàm manual FSM các nhiệm vụ sau được thực thi:

- Check button 1 nếu có: chuyển manual state sang RED1_GREEN2
- Check button 2 nếu có: chuyển manual state sang GREEN1_RED2
- Check button 2 nếu có: chuyển manual state sang YELLOW1_RED2

4.4 Tuning mode FSM Design



Hình 3: *Tuning mode fsm*

Brief :Đây là máy trạng thái được gọi trong mode tunning của hệ thống có bao gồm các 4 trạng thái : **INIT**, **RED1_RED2**, **GREEN1_GREEN2** và **YELLOW1_YELLOW2**.

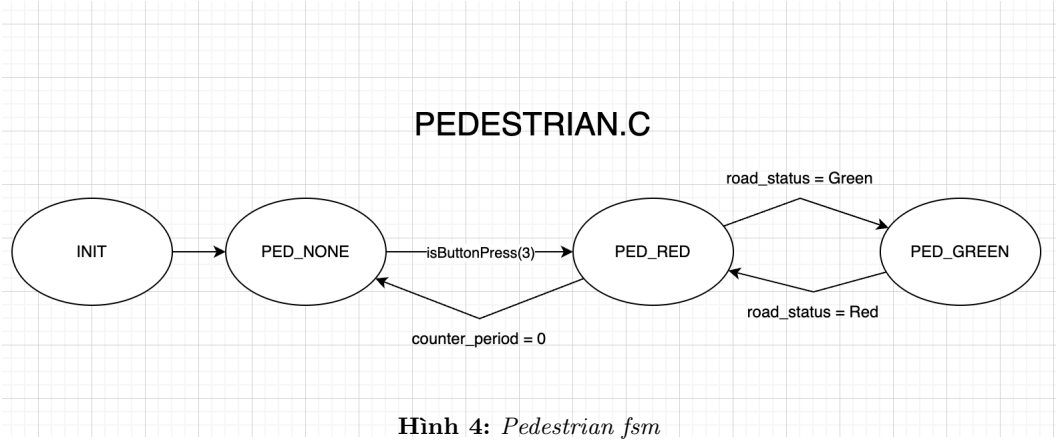
- **INIT**: Trạng thái khởi tạo, khi máy trạng thái tunning mode bắt đầu chạy sẽ bắt đầu tại đây.
- **RED1_RED2** : Ở trạng thái này, tiến hành đọc nút nhấn để điều chỉnh red periob time của đèn đỏ ở cả 2 đường.
- **GREEN1_GREEN2** :Ở trạng thái này, tiến hành đọc nút nhấn để điều chỉnh green periob time của đèn xanh ở cả 2 đường.
- **YELLOW1_YELLOW2** : Ở trạng thái này, tiến hành đọc nút nhấn để điều chỉnh green periob time của đèn vàng ở cả 2 đường.

State	Nhiệm vụ
INIT	• Set biến tunning_state cho thành RED1_RED2 • Set môi trường cho trạng thái RED1_RED2 và các biến liên quan đến setting thời gian
RED1_RED2	• check button 1 nếu có : chuyển sang tunning_state tiếp theo(GREEN1_GREEN2). • check button 2 nếu có : cộng thời gian cho red period time • Check button 3 nếu có : save lại thời gian được setting
GREEN1_GREEN2	• check button 1 nếu có : chuyển sang tunning_state tiếp theo(YELLOW1_YELLOW2). • check button 2 nếu có : cộng thời gian cho green period time • Check button 3 nếu có : save lại thời gian được setting
YELLOW1_YELLOW2	• check button 1 nếu có : chuyển sang tunning_state tiếp theo(RED1_RED2). • check button 2 nếu có : cộng thời gian cho yellow period time • Check button 3 nếu có : save lại thời gian được setting

Bảng 3: *Thực thi trong máy trạng thái chính*

4.5 Pedestrian FSM Design

Mô tả logic hoạt động: Mỗi khi nút nhấn thứ 4 được nhấn, dựa vào trạng thái đèn đường hiện tại mà đèn sang đường sẽ được tuỳ chỉnh. Đèn của người sang đường sẽ được set đối lập với đèn giao thông, nếu đèn giao thông là đỏ thì đèn người sang đường sẽ là xanh (PED_GREEN). Nếu đèn giao thông là xanh hoặc vàng , thì đèn sang đường sẽ là đỏ (PED_RED). Ngoài ra, khi đèn giao thông đang ở trạng thái tự động (AUTO), mỗi khi đèn sang đường chuyển qua đèn xanh, thì loa sẽ kêu càng ngày càng to và nhanh, khi đèn qua đường chuyển sang đỏ thì loa sẽ tắt.



Hình 4: *Pedestrian fsm*

Trạng thái	Mô tả
INIT	• Set trạng thái ban đầu của fsm là PED_NONE • Set trạng thái 2 đèn của người đi đường là trạng thái tắt
PED_NONE	Nếu button4 được nhấn và: • Nếu chế độ của đèn giao thông đang là AUTO, set đèn giao thông theo như logic đã mô tả và sau 2 chu kì, đèn sang đường sẽ tắt. • Nếu chế độ của đèn giao thông đang là MANUAL, set đèn giao thông theo như logic đã mô tả và đèn sang đường sẽ không bao giờ tắt.
PED_RED	• Khi đèn đường đổi sang đỏ thì sẽ chuyển trạng thái sang PED_GREEN. • Nếu như đèn đường đang ở trạng thái AUTO, thì sẽ bật thêm loa.
PED_GREEN	• Thay đổi độ lớn và tốc độ của phát âm thanh của loa theo thời gian. • Nếu như đèn đường là đèn xanh hoặc vàng thì chuyển trạng thái về PED_RED và tắt loa.

Bảng 4: *Bảng mô tả fsm pedestrian*

5 Mô tả hoạt động của hệ thống nút nhấn trong project

5.1 Button 1:

Mode	button state	function
Automatic	long press	chuyển sang manual mode
Manual	long press	chuyển sang tunning mode
Manual	short press	bật đèn đỏ sáng ở đường một và đèn xanh ở đường 2
Tunning	long press	chuyển sang automatic mode
Tunning	short press	Bật sáng các đèn để chỉnh thời gian đèn đỏ xanh và vàng

Bảng 5: *Table with Line Breaks in a Cell*

5.2 Button 2:

Mode	button state	function
Manual	short press	bật đèn xanh sáng ở đường một và đèn đỏ ở đường 2
Tunning	short press	Tăng period time của đèn đang sáng

Bảng 6: *Table with Line Breaks in a Cell*

5.3 Button 3:

Mode	button state	function
Manual	short press	bật đèn vàng sáng ở đường một và đèn đỏ ở đường 2
Tunning	short press	Lưu period time của đèn đang sáng

Bảng 7: *Table with Line Breaks in a Cell*

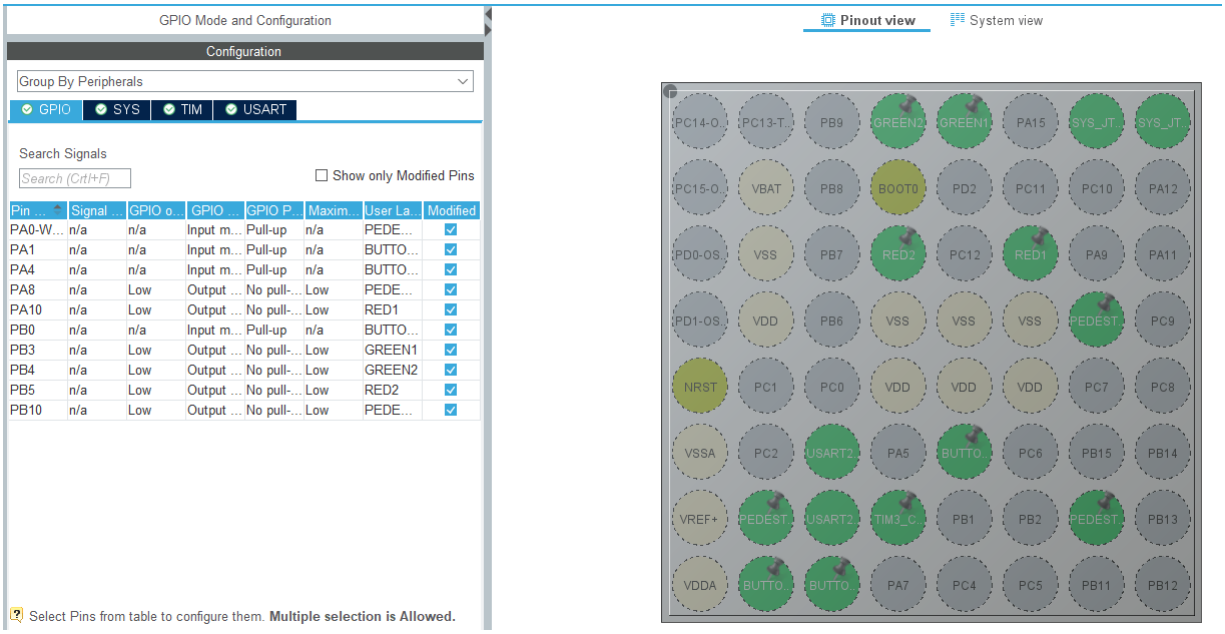
5.4 Pedestrian button :

Mode	button state	function
Automatic	short press	Bật hệ thống đèn cho người đi đường và kích hoạt buzzer để thông báo
Manual	short press	Bật hệ thống đèn cho người đi đường

Bảng 8: *Table with Line Breaks in a Cell*

6 System Configuration

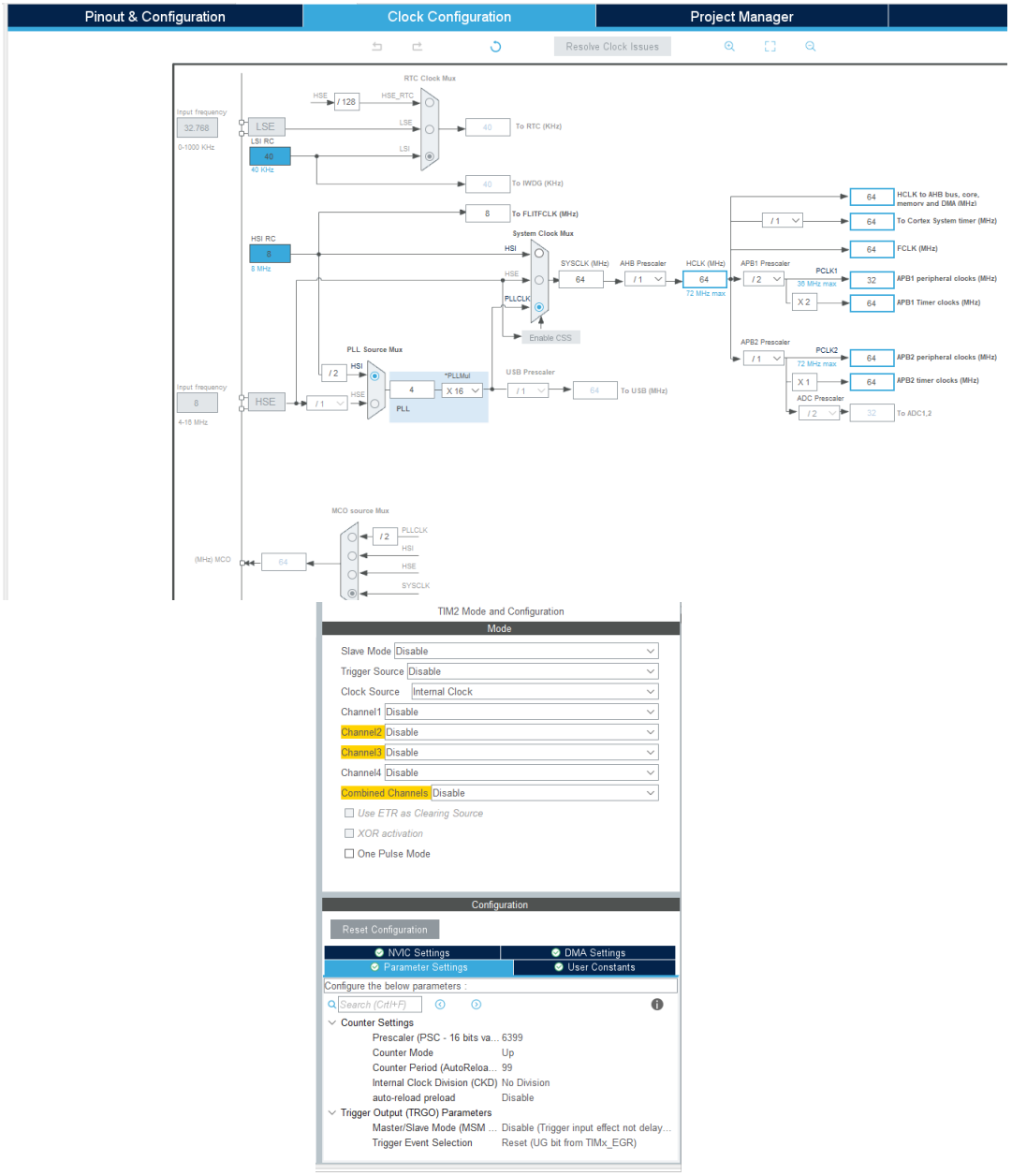
6.1 GPIO



Chân	Chức năng
PA10	Chân tín hiệu output cho đèn đỏ ở đường 1
PB5	Chân tín hiệu output cho đèn đỏ ở đường 2
PB3	Chân tín hiệu output cho đèn xanh ở đường 1
PB4	Chân tín hiệu output cho đèn xanh ở đường 2
PB10	Chân tín hiệu output cho đèn đỏ của người đi bộ
PA8	Chân tín hiệu output cho đèn xanh của người đi bộ
PA1	Chân tín hiệu input của nút nhấn 1
PA4	Chân tín hiệu input của nút nhấn 2
PB0	Chân tín hiệu input của nút nhấn 3
PA0	Chân tín hiệu input của nút nhấn cho người đi bộ
PA2	Chân truyền tín hiệu trong giao tiếp UART
PA3	Chân nhận tín hiệu trong giao tiếp UART
PA6	Chân tạo PWM cho Buzzer

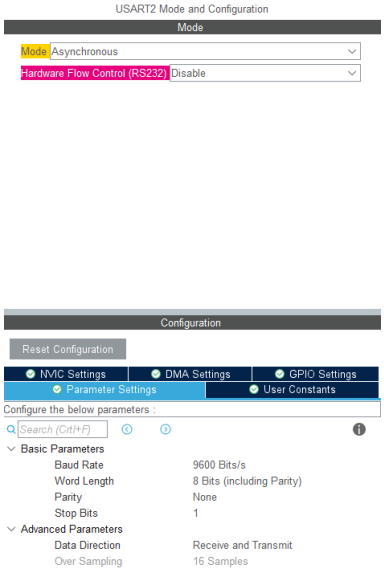
Bảng 9: *Bảng mô tả GPIO*

6.2 Timer interrupt



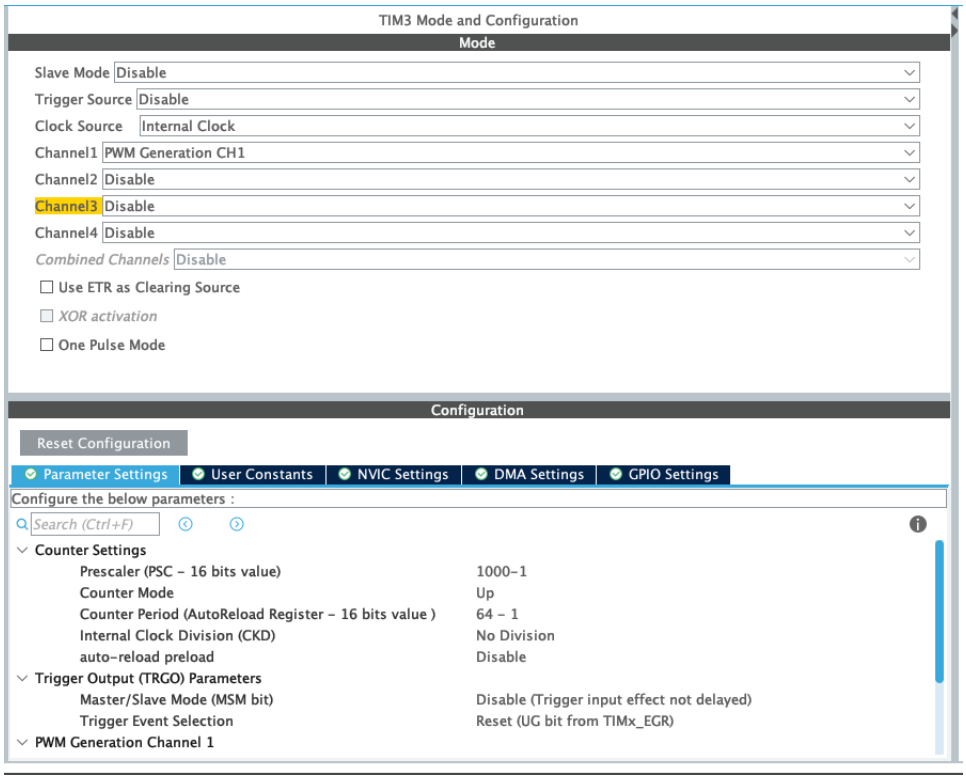
Với giá trị Clock bằng 64MHz, các giá trị Prescaler và Couter Period được điều chỉnh phù hợp để Timer Interrupt của hệ thống bằng 10 ms

6.3 UART Communication



2 chân được sử dụng trong giao tiếp UART của hệ thống là PA2 và PA3 đã được nhắc tới trong phần GPIO, các thông số cơ bản được cố định như trên

6.4 Generate PWM for Buzzer



7 System Implementation

7.1 File and Global variable Description

Tên file	Chức năng
button.h và button.c	Đọc và xử lý tín hiệu nút nhấn đầu vào (chống rung, phân biệt nhấn giữ và nhấn 1 lần)
scheduler.h và scheduler.c	Hiện thực scheduler sử dụng trong chương trình
driver.h và driver.c	Hiện thực các hàm với các câu lệnh điều khiển tín hiệu ở các chân output
global.h và global.c	Khai báo các global variable sử dụng trong chương trình
fsm.h và fsm.c	Hiện thực main_fsm và những fsm có trong main_fsm (và những hàm cần thiết đi kèm nếu có)
fsm_pedestrian.h và fsm_pedestrian.c	Hiện thực pedestrian_fsm (và những hàm cần thiết đi kèm nếu có)

Bảng 10: *Bảng mô tả các file*

Tên biến	Đặc tả
status	Đại diện cho trạng thái hiện tại của đèn đường (phân biệt các mode với nhau), vd : AUTO_RED_GREEN, MAN_RED_GREEN, ...
pedestrianStatus	Đại diện cho trạng thái hiện tại của chức năng cho người đi bộ
periodTime[3]	1 mảng gồm 3 phần tử đại diện cho thời gian sáng của đèn đường với 3 loại đèn : • Phần tử 0 : đèn đỏ • Phần tử 1 : đèn xành • Phần tử 2 : đèn vàng
setTimeRoad[3]	1 mảng gồm 3 phần tử đại diện cho thời gian sáng đã điều chỉnh trong Tuning mode (cần kiểm tra tính khả thi trước khi gán vào periodTime) của đèn đường với 3 loại đèn : • Phần tử 0 : đèn đỏ • Phần tử 1 : đèn xành • Phần tử 2 : đèn vàng
timeSetting	Biến đại diện cho thời gian sáng của loại đèn đang điều chỉnh trong Tuning mode (cần nhấn button3 để gán giá trị vào setTimeRoad)
timeRoad1	Đại diện cho thời gian sáng còn lại của loại đèn hiện tại ở đường 1
timeRoad2	Đại diện cho thời gian sáng còn lại của loại đèn hiện tại ở đường 2

Bảng 11: *Bảng mô tả global variables*

7.2 Scheduler

7.2.1 Update :

```
void SCH_Update(void){
  count_SCH_Update ++;
  if (SCH_tasks_G[0].pTask && SCH_tasks_G[0].RunMe == 0){
    if ( SCH_tasks_G[0].Delay >0){
      SCH_tasks_G[0].Delay--;
    }
    if (SCH_tasks_G[0].Delay == 0){
```

```
      SCH_tasks_G[0].RunMe = 1;
    }
  }
}
```

7.2.2 Add task :

```
   uint32_t SCH_Add_Task(void (* pFunction)(), uint32_t DELAY, uint32_t PERIOD)
   {
  uint8_t newTaskIndex = 0;
  uint32_t sumDelay = 0;
  uint32_t newDelay = 0;
  for (newTaskIndex = 0; newTaskIndex < SCH_MAX_TASKS ; newTaskIndex++){
    sumDelay += SCH_tasks_G[newTaskIndex].Delay;
    if (sumDelay < DELAY){
      newDelay = DELAY - ( sumDelay - SCH_tasks_G[newTaskIndex].Delay);
      SCH_tasks_G[newTaskIndex].Delay = sumDelay - DELAY;
      for (uint8_t i = SCH_MAX_TASKS; i > newTaskIndex ; i--){
        SCH_tasks_G[i].Delay = SCH_tasks_G[i-1].Delay;
        SCH_tasks_G[i].Period = SCH_tasks_G[i-1].Period;
        SCH_tasks_G[i].TaskID = SCH_tasks_G[i-1].TaskID;
        SCH_tasks_G[i].pTask = SCH_tasks_G[i-1].pTask;
      }
      SCH_tasks_G[newTaskIndex].Period = PERIOD/TIME_CYCLE;
      SCH_tasks_G[newTaskIndex].TaskID = Get_New_Task_ID();
      SCH_tasks_G[newTaskIndex].pTask = pFunction;
      SCH_tasks_G[newTaskIndex].Delay = newDelay;
      if (newDelay == 0){
        SCH_tasks_G[newTaskIndex].RunMe = 1;
      }
      else {
        SCH_tasks_G[newTaskIndex].RunMe = 0;
      }
      return SCH_tasks_G[newTaskIndex].TaskID;
    }
    else{
      if (SCH_tasks_G[newTaskIndex].pTask == 0x0000){
        SCH_tasks_G[newTaskIndex].Delay = DELAY - sumDelay;
        SCH_tasks_G[newTaskIndex].Period = PERIOD/TIME_CYCLE;
        SCH_tasks_G[newTaskIndex].TaskID = Get_New_Task_ID();
        SCH_tasks_G[newTaskIndex].pTask = pFunction;
        if (SCH_tasks_G[newTaskIndex].Delay == 0){
          SCH_tasks_G[newTaskIndex].RunMe = 1;
        }
        else{
          SCH_tasks_G[newTaskIndex].RunMe = 0;
        }
        return SCH_tasks_G[newTaskIndex].TaskID;
      }
    }
  }
  return SCH_tasks_G[newTaskIndex].TaskID;
}
```

7.2.3 Remove task :

```
   uint8_t SCH_Delete_Task(uint32_t taskID){
  uint8_t Return_code  = 0;
  uint8_t taskIndex;
  uint8_t j;
```

```
  if (taskID != NO_TASK_ID){
    for (taskIndex = 0 ; taskIndex < SCH_MAX_TASKS; taskIndex++){
      if (SCH_tasks_G[taskIndex].TaskID == taskID){
        Return_code = 1;
        if (taskIndex != 0 && taskIndex < SCH_MAX_TASKS - 1){
          if ( SCH_tasks_G[taskIndex+1].pTask != 0x0000){
            SCH_tasks_G[taskIndex + 1].Delay += SCH_tasks_G[taskIndex].Delay;
          }
        }
        for (j = taskIndex ; j < SCH_MAX_TASKS - 1 ; j ++){
          SCH_tasks_G[j].Delay = SCH_tasks_G[j+1].Delay;
          SCH_tasks_G[j].Period = SCH_tasks_G[j+1].Period;
          SCH_tasks_G[j].RunMe = SCH_tasks_G[j+1].RunMe;
          SCH_tasks_G[j].TaskID = SCH_tasks_G[j+1].TaskID;
          SCH_tasks_G[j].pTask = SCH_tasks_G[j+1].pTask;
        }
        SCH_tasks_G[j].pTask = 0;
        SCH_tasks_G[j].Period = 0;
        SCH_tasks_G[j].Delay = 0;
        SCH_tasks_G[j].RunMe = 0;
        SCH_tasks_G[j].TaskID = 0;
        return Return_code;
      }
    }
  }
  return Return_code;
}
```

7.2.4 Dispatch :

```
void SCH_Update(void){
  count_SCH_Update ++;
  if (SCH_tasks_G[0].pTask && SCH_tasks_G[0].RunMe == 0){
    if ( SCH_tasks_G[0].Delay >0){
      SCH_tasks_G[0].Delay--;
    }
    if (SCH_tasks_G[0].Delay == 0){
      SCH_tasks_G[0].RunMe = 1;
    }
  }
}
```

7.3 Important Functions

7.3.1 Automatic mode fsm

```
void automatic_mode_fsm(){
  switch(automatic_state){
    case INIT :
      automatic_state = RED1_GREEN2;
      timeRoad1 = periodTime[0];
      timeRoad2 = periodTime[1];
      SCH_Add_Task(show_time, 1, 0);
      break;
    case RED1_GREEN2:
      status = AUTO_RED_GREEN;
      if(timeRoad2 <= 0){
        automatic_state = RED1_YELLOW2;
        timeRoad2 = periodTime[2];
        SCH_Add_Task(show_time, 1, 0);
```

```
      }
      break;
    case RED1_YELLOW2 :
      status = AUTO_RED_YELLOW;
      if(timeRoad2 <= 0){
        automatic_state = GREEN1_RED2;
        timeRoad1 = periodTime[1];
        timeRoad2 = periodTime[0];
        SCH_Add_Task(show_time, 1, 0);
      }
      break;
    case GREEN1_RED2:
      status = AUTO_GREEN_RED;
      if(timeRoad1 <= 0){
        automatic_state = YELLOW1_RED2;
        timeRoad1 = periodTime[2];
        SCH_Add_Task(show_time, 1, 0);
      }
      break;
    case YELLOW1_RED2 :
      status = AUTO_YELLOW_RED;
      if(timeRoad1 <= 0){
        automatic_state = RED1_GREEN2;
        timeRoad1 = periodTime[0];
        timeRoad2 = periodTime[1];
        SCH_Add_Task(show_time, 1, 0);
      }
      break;
    default:
      break;
  }
}
```

7.3.2 Manual mode fsm

```
void manual_mode_FSM(){
  if (isButtonPress(0) == 1){
    status = MAN_RED_GREEN;
    manual_state = RED1_GREEN2;
  }
  if (isButtonPress(1) == 1){
    status = MAN_GREEN_RED;
    manual_state = GREEN1_RED2;
  }
  if (isButtonPress(2) == 1){
    status = MAN_YELLOW_RED;
    manual_state = YELLOW1_RED2;
  }
}
```

7.3.3 Tuning mode fsm

```
void tuningFSM(int * state){
  switch (*state){
  case INIT:
    displaySettingTask = SCH_Add_Task(displayTimeSetting, 10, 10);
    for (int i = 0 ; i < 3; i ++){
      setTimeRoad[i] = periodTime[i];
    }
    timeSetting = setTimeRoad[0];
```

```
    *state = RED1_RED2;
    break;
  case RED1_RED2:
    if(isButtonPress(0) == 1){
      timeSetting = setTimeRoad[1];
      *state = GREEN1_GREEN2;
    }
    if (isButtonPress(1) == 1){
      plusTimeRoad();
    }
    if (isButtonPress(2) == 1){
      updateTimeRoad(RED1_RED2);
    }
    break;
  case GREEN1_GREEN2:
    if(isButtonPress(0) == 1){
      timeSetting = setTimeRoad[2];
      *state = YELLOW1_YELLOW2;
    }
    if (isButtonPress(1) == 1){
      plusTimeRoad();
    }
    if (isButtonPress(2) == 1){
      updateTimeRoad(GREEN1_GREEN2);
    }
    break;
  case YELLOW1_YELLOW2:
    if(isButtonPress(0) == 1){
      timeSetting = setTimeRoad[0];
      *state = RED1_RED2;
    }
    if (isButtonPress(1) == 1){
      plusTimeRoad();
    }
    if (isButtonPress(2) == 1){
      updateTimeRoad(YELLOW1_YELLOW2);
    }
    break;
  }
}
```

7.3.4 Main fsm

```
void mainFSM(){
  switch(main_state){
  case INIT:
    //todo
    automatic_state = INIT;
    main_state = AUTOMATIC_MODE;
    displayAutoTask = SCH_Add_Task(decrease_and_show_time, 102, 100);
    break;
  case AUTOMATIC_MODE:
    setStateTrafficSystem(automatic_state);
    automatic_mode_fsm();
    if(isButtonPressLong(0) == 1){
      manual_state = INIT;
      main_state = MANUAL_MODE;
      SCH_Delete_Task(displayAutoTask);
    }
    //todo
    break;
```

```
  case MANUAL_MODE:
    manual_mode_FSM();
    setStateTrafficSystem(manual_state);
    if ( isButtonPressLong(0) == 1){
      /*set up enviroment for tuning mode*/
      tuning_state = INIT;

      main_state = TUNING_MODE;
    }
    //todo
    break;
  case TUNING_MODE:
    setStateTrafficSystem(tuning_state);
    tuningFSM(&tuning_state);
    if ( isButtonPressLong(0) == 1){
      SCH_Delete_Task(displaySettingTask);
      checkSetTimeValid();
      main_state = AUTOMATIC_MODE;
      displayAutoTask = SCH_Add_Task(decrease_and_show_time, 102, 100);
      automatic_state = INIT;
    }
    //todo
    break;
  }
}
```

7.3.5 Pedestrian fsm

```
// ======== SUB FUNCTION ============
/*
    Function: Turn on red led of pedestrian light and turn off green led of
    pedestrian light
*/
void setPedestrianRed(){
  HAL_GPIO_WritePin(PEDESTRIAN_RED_GPIO_Port, PEDESTRIAN_RED_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(PEDESTRIAN_GREEN_GPIO_Port, PEDESTRIAN_GREEN_Pin,
    GPIO_PIN_RESET);
}
/*
    Function: Turn on green led of pedestrian light and turn off red led of
    pedestrian light
*/
void setPedestrianGreen(){
  HAL_GPIO_WritePin(PEDESTRIAN_RED_GPIO_Port, PEDESTRIAN_RED_Pin, GPIO_PIN_RESET
    );
  HAL_GPIO_WritePin(PEDESTRIAN_GREEN_GPIO_Port, PEDESTRIAN_GREEN_Pin,
    GPIO_PIN_SET);
}
/*
    Function: Turn on red led of pedestrian light and turn on green led of
    pedestrian light
*/
void setPedestrianYellow(){
  HAL_GPIO_WritePin(PEDESTRIAN_RED_GPIO_Port, PEDESTRIAN_RED_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(PEDESTRIAN_GREEN_GPIO_Port, PEDESTRIAN_GREEN_Pin,
    GPIO_PIN_SET);
}
/*
    Function: Turn off red led of pedestrian light and turn off green led of
    pedestrian light
*/
```

```
void clearPedestrian(){
  HAL_GPIO_WritePin(PEDESTRIAN_RED_GPIO_Port, PEDESTRIAN_RED_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(PEDESTRIAN_GREEN_GPIO_Port, PEDESTRIAN_GREEN_Pin, GPIO_PIN_RESET);
}
/*
    Function: Setting PWM to change the duty cycle of the output signal.
    @PWM: use to set the duty cycle of the signal
    @FREQ: use to set the frequency of the signal
*/
void setPWM(int PWM, int FREQ){
  if(FREQ != -1)
    HAL_TIM_SET_PRESCALER(&htim3, (uint32_t)FREQ);
  HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1,PWM);
}
//  ======= FSM ========
/*
    Function: Run the pedestrian light state follow the described fsm.
*/
void fsm_pedestrian(){
    switch(pedestrianStatus) {
        case INIT:
            clearPedestrian();
            pedestrianStatus = PED_NONE;
            break;
        case PED_NONE:
            if(isButtonPress(3) == 1){
                if(status == AUTO_RED_GREEN || status == AUTO_RED_YELLOW){
                    pedestrianStatus = PED_GREEN;
                    counterPed = 2;
                    setPedestrianGreen();
                    // Turn on timer to setting PWM to turn on the buzzer and change the duty cycle of the signal.
                    setTimer2(25);
                } else if(status == MAN_RED_GREEN || status == MAN_RED_YELLOW ){
                    pedestrianStatus = PED_GREEN;
                    setPedestrianGreen();
                    // Not turn on the buzzer
                    setTimer2(0);
                }
                else if(status == AUTO_GREEN_RED || status == MAN_GREEN_RED || status == AUTO_YELLOW_RED || status == MAN_YELLOW_RED){
                    pedestrianStatus = PED_RED;
                    counterPed = 2;
                    setPedestrianRed();
                }
            }
            break;
        case PED_RED:
            if(timer2_flag){
                timer2_flag = 0;
                // After 2 period, return to initial state and turn off all led
                if(counterPed == 0){
                    pedestrianStatus = PED_NONE;
                    clearPedestrian();
                    break;
                }
            }
            else if(status == AUTO_RED_GREEN){
                pedestrianStatus = PED_GREEN;
                setPedestrianGreen();
```

```
                // Turn on timer to setting PWM to turn on the buzzer and change the duty cycle of the signal.
                setTimer2(25);
            }
            else if (status == MAN_RED_GREEN){
                pedestrianStatus = PED_GREEN;
                setPedestrianGreen();
                // Not turn on the buzzer
                setTimer2(0);
            }
            break;
        case PED_GREEN:
            if(timer2_flag){
                // change the duty cycle and the frequency of the output signal base on current time.
                if(PWM == 0){
                    PWM = 63-63*timeRoad1/(double)periodTime[0] + 10;
                    FREQ = 1000 + 2000 *((double)periodTime[0]- timeRoad1);
                    setPWM(PWM, FREQ);
                } else {
                    PWM = 0;
                    setPWM(PWM, -1);
                }
                setTimer2(5 + 15 * timeRoad1/(double)periodTime[0]);
            }
            // Change the state to PED_RED
            if(status == AUTO_GREEN_RED || status == MAN_GREEN_RED || status == AUTO_YELLOW_RED || status == MAN_YELLOW_RED){
                counterPed--;
                pedestrianStatus = PED_RED;
                setTimer2(200);
                setPedestrianRed();
                setPWM(0, -1);
            }
            break;
        default:
            break;
    }
}
```

7.3.6 Put it all :

```
int main(void)
{
  HAL_Init();

  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_USART2_UART_Init();
  MX_TIM3_Init();


  SCH_Init();
  HAL_TIM_Base_Start_IT(&htim2);
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);

  status = INIT;
  pedestrianStatus = INIT;
  SCH_Add_Task(mainFSM, 1,1);
  SCH_Add_Task(fsm_pedestrian, 1, 1);
```

```
  SCH_Add_Task(readPin, 1, 1);
  SCH_Add_Task(timerRun,1,1);
  while (1)
  {
    SCH_Dispatch_Tasks();
  }
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  SCH_Update();
}
```