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

VI XỬ LÝ - VI ĐIỀU KHIỂN (CO3009)

Báo cáo nghiệm thu

DỰ ÁN

HỆ THỐNG ĐÈN GIAO THÔNG SỬ DỤNG STM32F103RB

Giảng viên hướng dẫn: Lê Trọng Nhân
Huỳnh Phúc Nghị
Sinh viên thực hiện: Nguyễn Nhật Khải (2111506)
Nguyễn Thanh Hiền (2111203)

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 12 2023

Mục lục

Danh sách hình vẽ

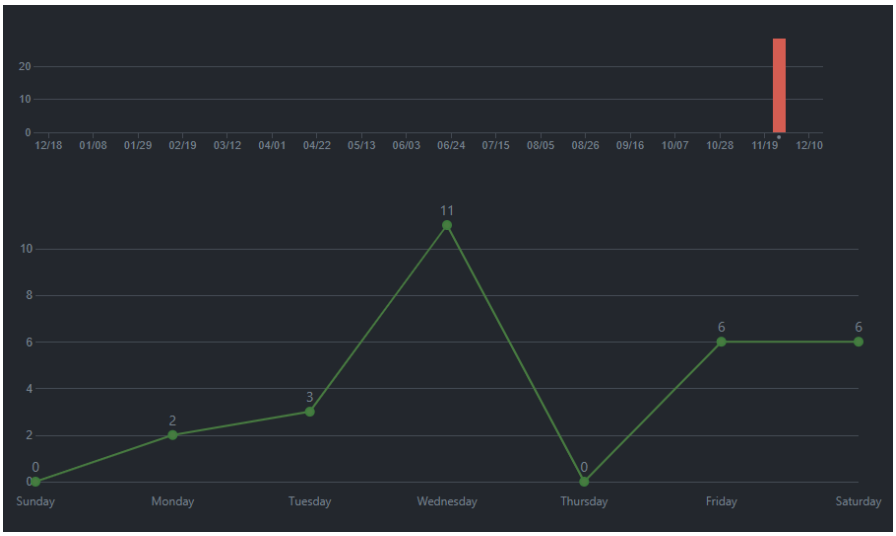
Danh sách bảng

1 GIỚI THIỆU

1.1 Thông tin chung

- GitHub Repository: CO3009 Mini Project
- Thời gian thực hiện: 27/11/2023 - 02/12/2023 (6 ngày)

Hình 1: Biểu đồ tần suất chỉnh sửa (thêm/xóa) của hai thành viên



Hình 2: Thống kê số lượng commit

1.2 Tổng quan dự án

Trong dự án này, sinh viên sẽ sử dụng Board STM32F103RB để mô phỏng cơ chế của đèn giao thông hai chiều.

1.2.1 Đối tượng sử dụng

- Người quản lý giao thông: Điều khiển màu đèn, thời gian sáng các đèn giao thông của các phía.
- Người qua đường: Theo dõi trạng thái các đèn giao thông, yêu cầu được qua đường.
- Người điều khiển phương tiện: Theo dõi trạng thái các đèn giao thông.

1.2.2 Cơ chế hoạt động

Hệ thống được phát triển với 3 chế độ: tự động, thủ công và điều chỉnh thời lượng sáng đèn. Ngoài ra, hệ thống còn được tích hợp đèn và loa báo hiệu dành riêng cho người đi bộ để xin sang đường.

Ở chế độ "Tự động", đèn giao thông sẽ đếm ngược, luân phiên sáng đèn như bình thường và người đi bộ có thể xin được sang đường. Trong chế độ "Thủ công", màu đèn của ở hai phía sẽ được điều chỉnh độc lập bởi người quản lý giao thông và đèn/loa cho người đi bộ cũng hoạt động dựa trên đèn của phương tiện. Để điều chỉnh thời lượng sáng của các đèn, ta chuyển sang chế độ "Điều chỉnh". Chi tiết phân tích và đặc tả hệ thống sẽ được đề cập trong sâu hơn ở các phần sau.

2 ĐẶC TẢ YÊU CẦU

2.1 Yêu cầu chức năng

2.1.1 Yêu cầu hệ thống

Hệ thống đèn giao thông hỗ trợ cơ chế điều khiển hai chiều của ngã tư, sử dụng board STM32F103RB.

2.1.2 Yêu cầu người dùng

Người điều khiển giao thông sẽ được hỗ trợ các chức năng:

- **Tự động:** Hệ thống tự động hoạt động với các đèn của hai phía giao thông tự động thay đổi: xanh - vàng - đỏ, theo thời gian đã được thiết lập.
- **Thủ công:** Lựa chọn màu đèn được phép sáng của các phía đèn giao thông, đèn sẽ chỉ thay đổi màu khi người điều khiển giao thông cho phép
- **Điều chỉnh:** Thay đổi thời gian sáng cảu các đèn của chế độ **Tự động**, với cơ chế tăng/giảm thời lượng bằng nút bấm (hỗ trợ tự động tăng/giam liên tục khi nhấn đè).

Người qua đường sẽ được hỗ trợ chức năng:

- **Xin được sang đường:** Cảnh báo cho người qua đường thông thường, người gặp khó khăn về thị giác hoặc thính giác. Khi xin được sang đường, hệ thống sẽ căn cứ vào đèn của các phương tiện giao thông để hiển thị màu phù hợp, đồng thời, cảnh báo âm thanh để gợi ý: được sang đường, sắp hết thời gian được di chuyển.

2.2 Yêu cầu kỹ thuật

2.2.1 Nút nhấn

- Hiện thực cơ chế nhấn đè để: tự động tăng/giảm giá trị khi ở chế độ điều chỉnh thời lượng sáng các đèn, hoặc tự động nhảy sang trạng thái con/chế độ tiếp theo.
- Đối với nút nhấn xin sang đường, chỉ nhận duy nhất một lần trong một chu kỳ đèn (cho phép người đi bộ qua đường thành công).

2.2.2 Đèn tín hiệu

	Phương tiện	Người đi bộ
Tự động	Hoạt động theo thời gian đã thiết lập	Sáng khi xin sang đường và nhấp nháy theo nhịp của loa phát thanh
Thủ công	Chỉ sáng đèn theo màu đang được lựa chọn	Tự động nhảy theo màu đèn của phương tiện (cơ chế ngược lại với đèn cho xe)
Điều chỉnh	Chỉ sáng đèn theo màu đang được điều chỉnh	Không sáng

Bảng 1: Cơ chế hoạt động của đèn tín hiệu ở các chế độ

2.2.3 Loa phát thanh

- Chỉ bật khi người đi bộ xin sang đường
- Điều chỉnh bằng giá trị: duty cycle (để tăng âm lượng) và thời gian bật/tắt (để phát tín hiệu theo nhịp điệu)
- Khi mới cho phép người đi bộ sang đường, loa sẽ báo động với âm lượng nhỏ và chậm.
- Khi gần hết thời gian sang đường, loa sẽ báo động với âm lượng to và nhanh.

2.2.4 UART

- Baudrate: 9600 bps
- Data bit: 8-bit
- Parity bit: Không
- Mục đích: Truyền dữ liệu thời gian của đèn cho phương tiện 1 phía với cú pháp **!7SEG:xx#** *(với xx là thời gian với 2 chữ số)*

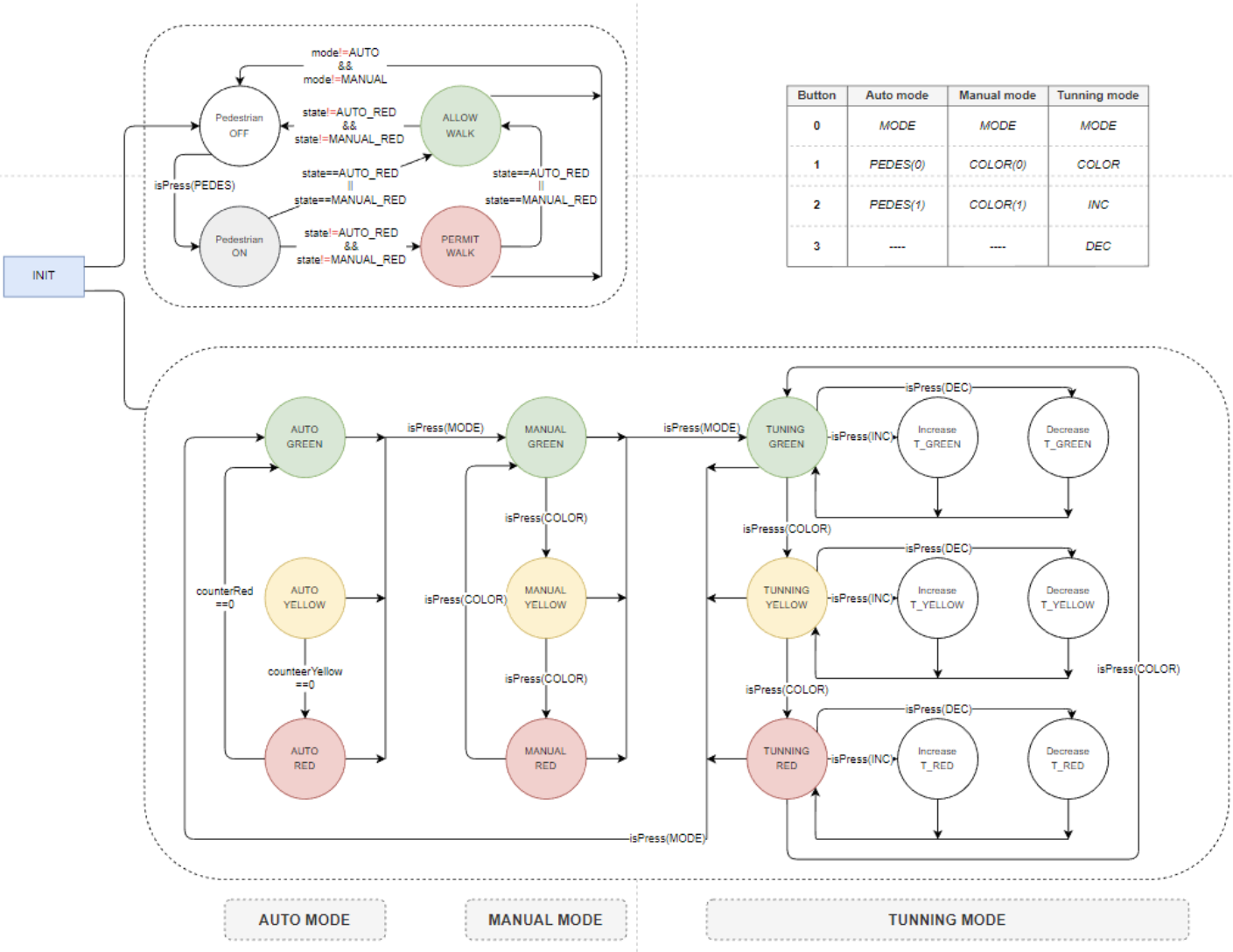
2.3 Danh sách trang thiết bị

Thiết bị	Số lượng	Mục đích sử dụng
Đèn LED đơn xanh lá	4	Đèn phương tiện (2 phía), người đi bộ (2 phía)
Đèn LED đơn màu đỏ	4	Đèn phương tiện (2 phía), người đi bộ (2 phía)
Đèn LED đơn màu vàng	4	Đèn phương tiện (2 phía), người đi bộ (2 phía)
Nút nhấn	4	Điều khiển các chế độ đèn của phương tiện và người đi bộ 2 phía xin sang đường
Dây bus: đực-đực, đực-cái, cái-cái	Mỗi loại 2 bó	Kết nối các linh kiện và mạch STM32
Loa tín hiệu	1	Mô phỏng loa xin sang đường của 1 phía
Board NUCLEO STM32F103RB	1	Điều khiển hệ thống đèn giao thông
Mạch chuyển UART-USB	1	Đọc thời gian hiển thị đèn cho phương tiện của 1 phía
Máy tính với phần mềm đọc UART	1	Đọc và kiểm tra dữ liệu UART

Bảng 2: Danh sách trang thiết bị sử dụng

3 THIẾT KẾ

3.1 Thiết kế



Hình 3: Máy trạng thái hệ thống đèn giao thông

Sơ đồ máy trạng thái được tách ra làm hai phần dành cho phương tiện giao thông và người đi bộ (với cơ chế của người đi bộ khi được kích hoạt sẽ hoạt động dựa trên trạng thái của đèn cho phương tiện giao thông)

3.1.1 FSM: Phương tiện giao thông

Ban đầu, đèn sẽ chạy **tự động**, nút nhấn là tín hiệu chuyển sang chế độ tiếp **thủ công**, **điều chỉnh** và trở lại tự động. Cơ chế hoạt động của từng chế độ sẽ như sau:

1. **Tự động:** Đèn của mỗi phía sẽ được khởi tạo với màu đối lập nhau (một bên xanh, một bên đỏ) và bắt đầu chạy theo vòng lặp: xanh → vàng → đỏ, rồi quay ngược về xanh với thời gian sáng mỗi màu theo như thời gian đã khởi tạo (có thể thay đổi trong cơ chế **điều chỉnh**).

2. **Thủ công:** Ở chế độ này, màu đèn phương tiện của mỗi phía sẽ hoàn toàn do người điều khiển giao thông điều chỉnh thủ công, vì vậy, sẽ phải **cực kỳ cẩn trọng**. Khi chuyển sang chế độ này, hai phía sẽ được khởi tạo với màu đối lập nhau (một bên đỏ, một bên xanh). Mỗi lượt nhấn nút, màu của đèn sẽ chuyển từ: xanh → vàng → đỏ, và quay ngược lại ban đầu

3. **Điều chỉnh:** Mỗi màu đèn sẽ có thời gian sáng khác nhau, chế độ này được dùng để điều chỉnh thời gian. Một nút sẽ để chọn màu muốn chỉnh thời gian, hai nút dùng để tăng/giảm.

3.1.2 FSM: Người đi bộ

Ban đầu, đèn và loa sẽ luôn tắt. Khi được người đi bộ kích hoạt trong chế độ **tự động**, đèn và loa sẽ hoạt động dựa trên màu đèn phương tiện:

- Đèn phương tiện màu xanh hoặc vàng: Đèn người đi bộ màu đỏ.
- Đèn phương tiện màu đỏ: Đèn người đi bộ chuyển sang màu xanh. Ban đầu, loa kêu với âm lương nhỏ, tốc độ chậm. Gần hết thời gian, loa tăng âm lượng và tốc độ.

Chế độ **thủ công**, đèn của người đi bộ sẽ hoạt động căn cứ đèn phương tiện, nhưng loa không chạy vì không có yếu tố thời gian để điều chỉnh âm lượng và tốc độ.

3.1.3 Công dụng của nút nhấn

Nút	Tự động	Thủ công	Điều chỉnh
0	Chuyển chế độ	Chuyển chế độ	Chuyển chế độ
1	Xin sang đường phía A	Đổi màu đèn phía A	Đổi màu đèn muốn điều chỉnh
2	Xin sang đường phía B	Đổi màu đèn phía B	Tăng thời lượng
3	—	—	Giảm thời lượng

Bảng 3: Công dụng của nút nhấn trong các chế độ

3.2 Cấu hình hệ thống

3.2.1 Sơ đồ nguyên lý mô phỏng

Pin name	GPIO mode	User label	Purpose
PA2	—	USART2_TX	Truyền UART
PA3	—	USART2_RX	Nhận UART
PA5	Output	LED_RED	Đèn hệ thống (1s)
PA6	—	TIM3_CH1	Loa phía A
PA7	—	TIM3_CH2	Lóa phía B
PA8	Input	BUTTON0	Nút 0
PA9	Input	BUTTON1	Nút 1
PA10	Input	BUTTON2	Nút 2
PA11	Input	BUTTON3	Nút 3
PB0	Output	PEDESTRIAN0_RED	Đèn đỏ (người đi bộ) phía A
PB1	Output	PEDESTRIAN0_GREEN	Đèn xanh (người đi bộ) phía A
PB4	Output	TRAFFIC0_RED	Đèn đỏ (phương tiện) phía A
PB5	Output	TRAFFIC0_YELLOW	Đèn vàng (phương tiện) phía A
PB6	Output	TRAFFIC0_GREEN	Đèn xanh (phương tiện) phía A
PB8	Output	PEDESTRIAN1_RED	Đèn đỏ (người đi bộ) phía B
PB9	Output	PEDESTRIAN1_GREEN	Đèn xanh (người đi bộ) phía B
PB12	Output	TRAFFIC1_RED	Đèn đỏ (phương tiện) phía B
PB13	Output	TRAFFIC1_YELLOW	Đèn vàng (phương tiện) phía B
PB14	Output	TRAFFIC1_GREEN	Đèn xanh (phương tiện) phía B

Bảng 4: Danh sách các pinout

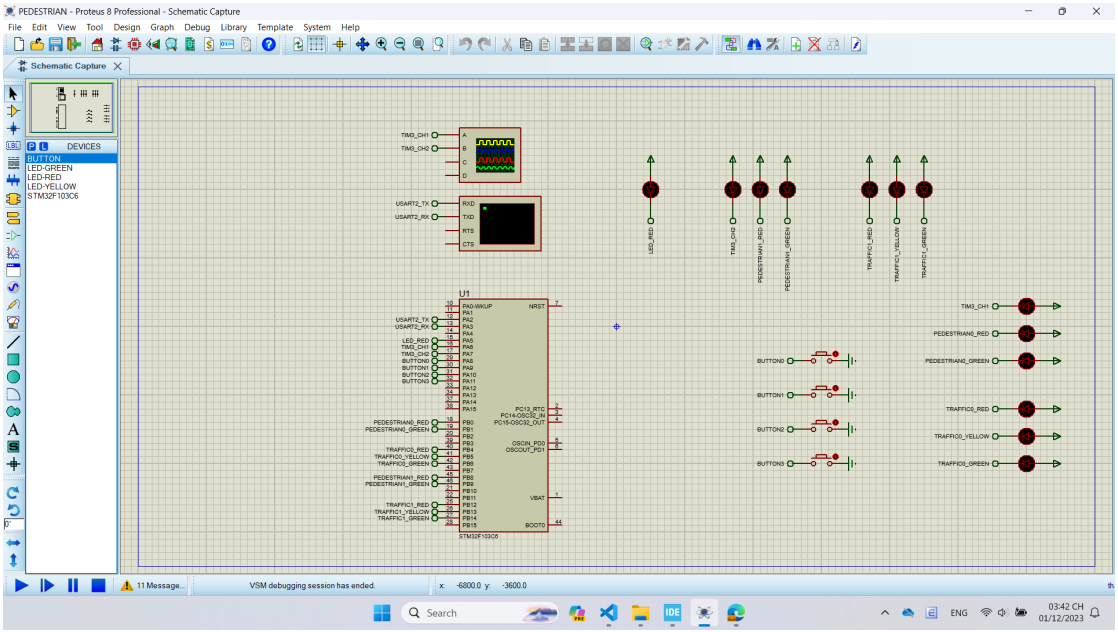
Thực hiện sắp xếp các linh kiện phục vụ cho việc mô phỏng lên không gian làm việc chung với: vi điều khiển STM32F103C6, Virtual Terminal (hiển thị UART), 1 Oscilloscope (kiểm tra xung PWM), 4 nút nhất và 13 đèn LED đơn:

- Giả sử, dòng điện do vi điều khiển cung cấp thỏa mãn điều kiện hoạt động của LED và không vượt quá mức cho phép. Ta kết nối chân các LED đơn với vi điều khiển trung tấm
  - Cathode của các LED nối tương ứng với cách chân của vi điều khiển theo bảng 4.
  - Anode của LED đơn nối với nguồn 3.3V
- Kết nối nút nhấn với vi điều khiển:

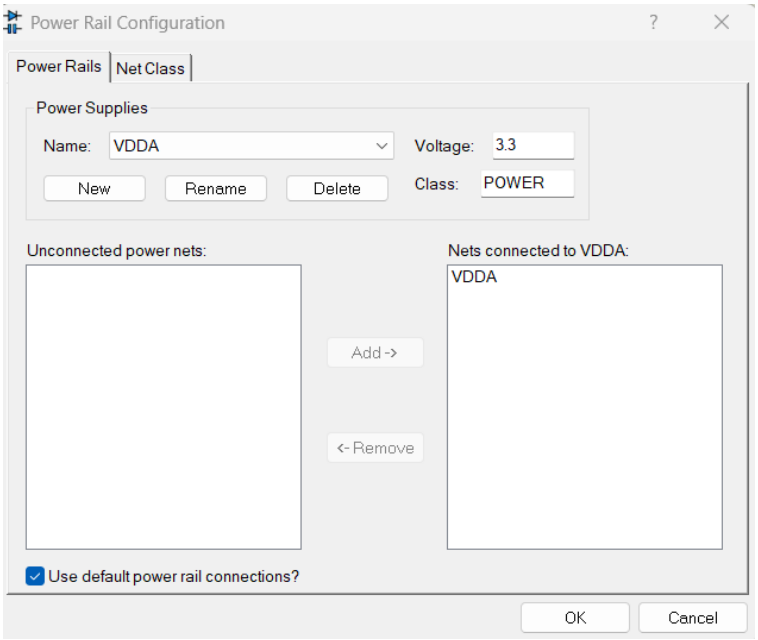
  - Một đầu nút nhấn nối xuống đất.
  - Đầu còn lại, ta nối lần lượt tới các chân nhận tín hiệu input của vi điều khiển.

- Kết nối Virtual Terminal với vi điều khiển chung tâm:
  - Nối chân RXD của Virtual Terminal với chân TXD (PA2) của vi điều khiển.
  - Nối chân TXD của Virtual Terminal với chân RXD (PA3) của vi điều khiển.

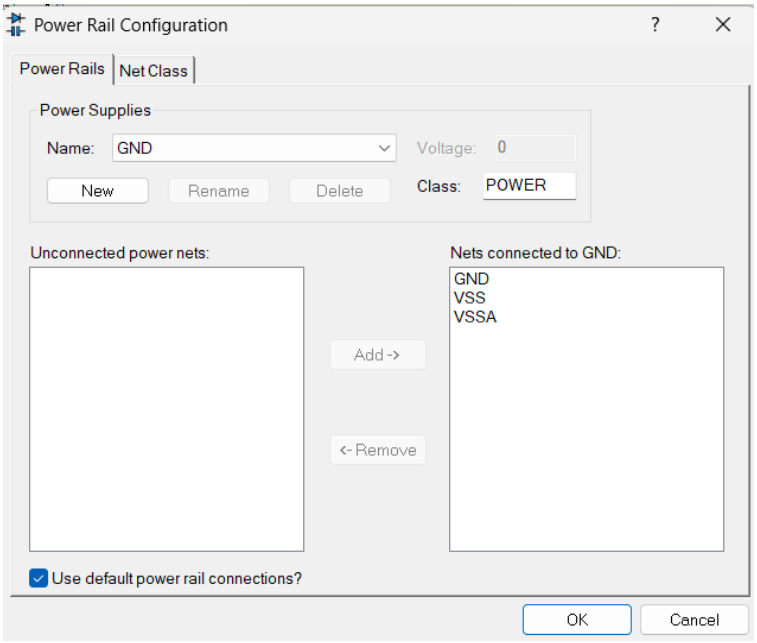
- Kết nối Oscilloscope với đầu ra điều khiển loa phát tín hiệu của vi điều khiển:
  - Nối chân ngõ A của Oscilloscope với chân PWM kênh 1 (PA6) của vi điều khiển.
  - Nối chân ngõ B của Oscilloscope với chân PWM kênh 2 (PA7) của vi điều khiển.



Hình 4: Sơ đồ nguyên lý mô phỏng với Proteus



(a) VDDA



(b) VSSA

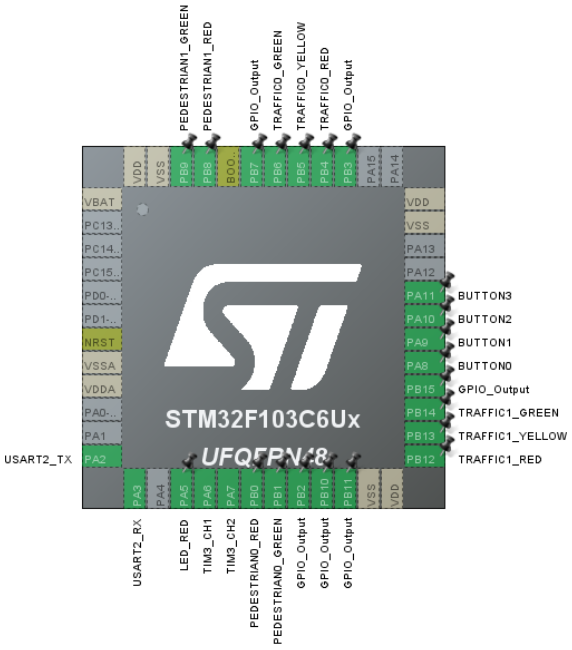
Hình 5: Cấu hình đường nối nguồn/đất

Ta thiết lập đồng bộ **VSSA** với đất chung của hệ thống và **VDDA** là nguồn chung của hệ thống ở mức 3.3V.

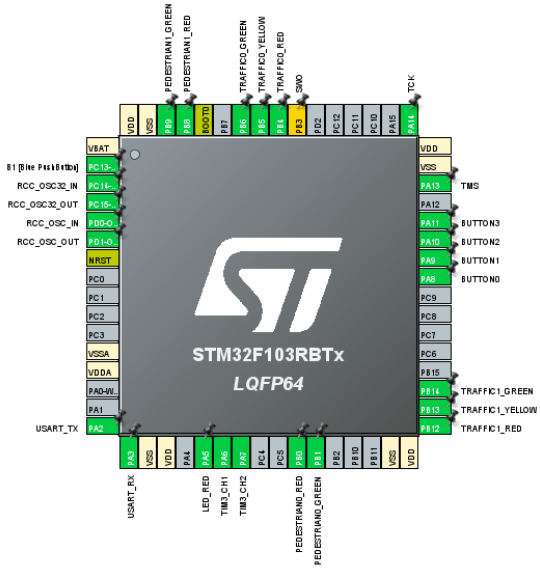
3.2.2 Cấu hình vi điều khiển

3.2.2.a Cấu hình GPIO

Yêu cầu: Các chân input/output đều được kéo lên và tuân theo Sơ đồ nguyên lý mô phỏng.



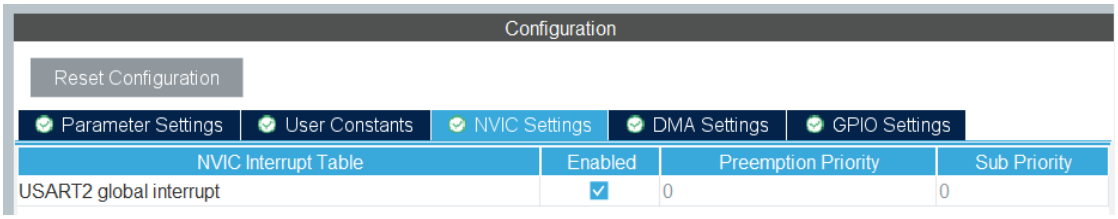
Hình 6: Cấu hình GPIO: STM32F103C6



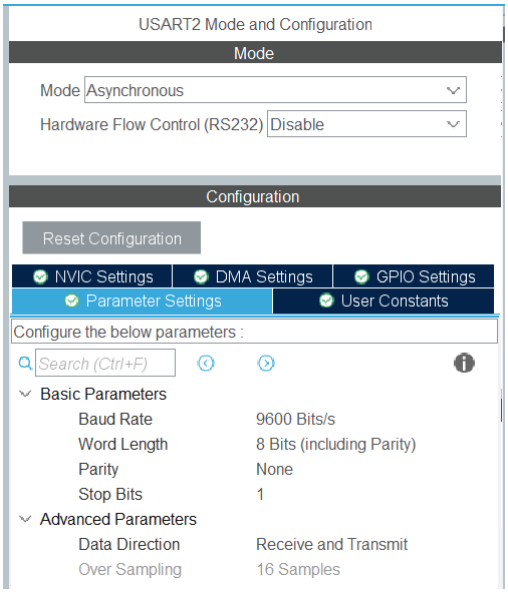
Hình 7: Cấu hình GPIO: STM32F103RB

3.2.2.b Cấu hình giao tiếp

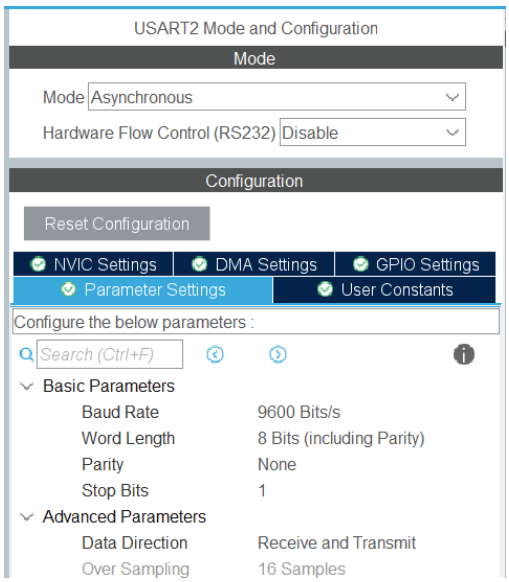
Yêu cầu: Hệ thống cần sử dụng UART với baudrate 9600 bits/s, để truyền gói dữ liệu 8 bit, không có parity và chỉ 1 stop bit.



Hình 8: Cài đặt NVIC USART2



(a) STM32F103C6



(b) STM32F103RB

Hình 9: Cấu hình USART2

3.2.2.c Cấu hình Timer

Yêu cầu: Hệ thống cần sử dụng TIM2 ($f_{TICK} = 2kHz$), TIM3 ($f_{TICK} = 100Hz$) với channel 1 và 2 dùng để băm xung PWM cho hai loa phát tín hiệu.

Trong quá trình mô phỏng với Proteus, nhóm sử dụng STM32F103C6 ($f_{TIM} = 8MHz$). Quá trình hiện thực chính thức, nhóm chuyển sang sử dụng board NUCLEO STM32F103RB ($f_{TIM} = 64MHz$). Vì vậy, timer sẽ được cấu hình khác

nhau đối với mỗi file **.ioc**. Timer được cấu hình dựa trên công thức:

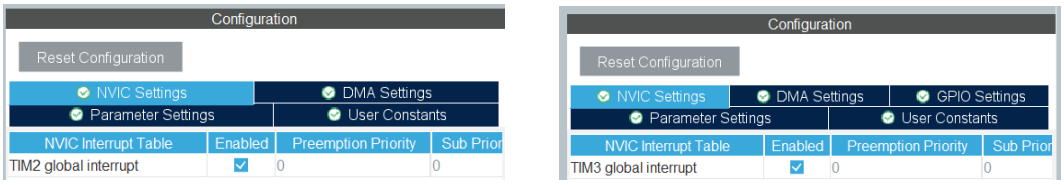
$$ARR(\text{Auto Reload Register}) = f_{TIM} \div f_{TICK} \quad (1)$$

$$PSR(\text{Prescaler}) = ARR \div 65535 \quad (2)$$

$$\text{Counter Period} = f_{TIM} \div (f_{TICK} \times (PSC + 1)) - 1 \quad (3)$$

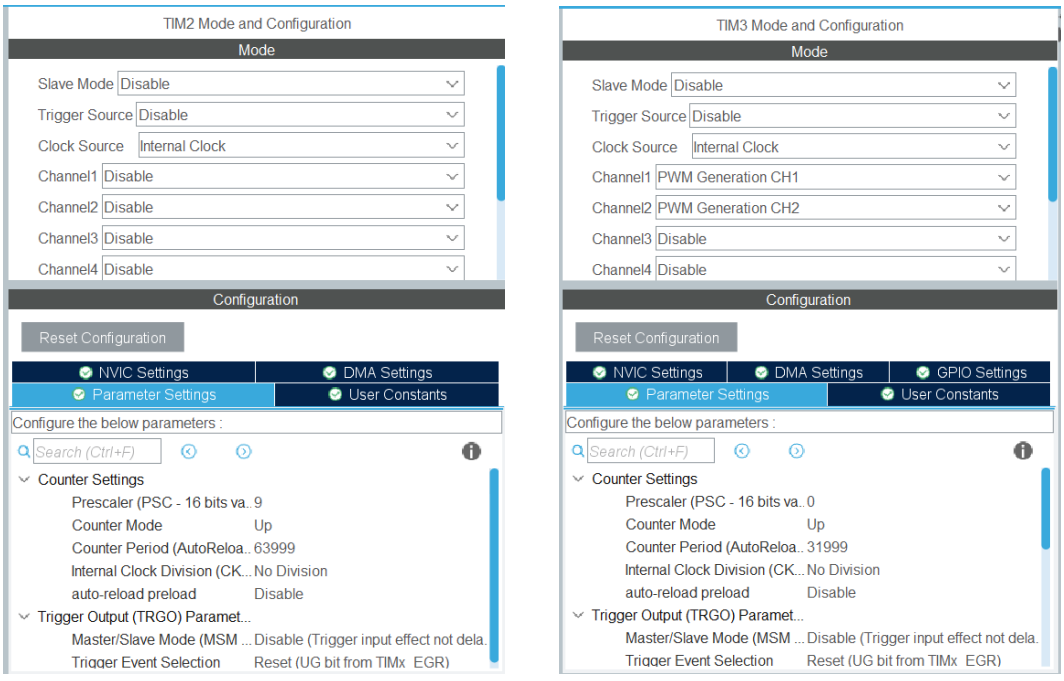
	STM32F103**C6** ($f_{TIM} = 8MHz$)		STM32F103**RB** ($f_{TIM} = 64MHz$)	
Timer	TIM2 ($f_{TICK} = 2kHz$)	TIM3 ($f_{TICK} = 100Hz$)	TIM2 ($f_{TICK} = 2kHz$)	TIM3 ($f_{TICK} = 100Hz$)
Prescaler	1	0	9	0
Counter Mode	Up	Up	Up	Up
Counter Period	39999	3999	63999	31999

Bảng 5: Cấu hình timer cho mô phỏng và hiện thực



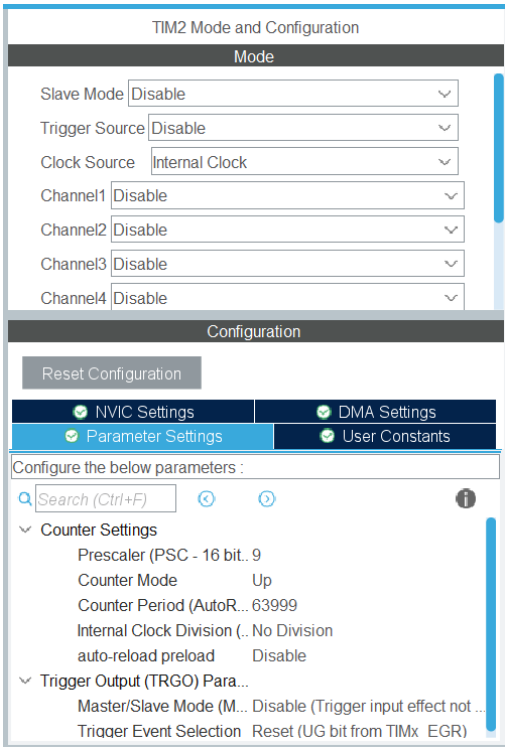
(a) TIM2 (b) TIM3

Hình 10: Cài đặt NVIC cho timer

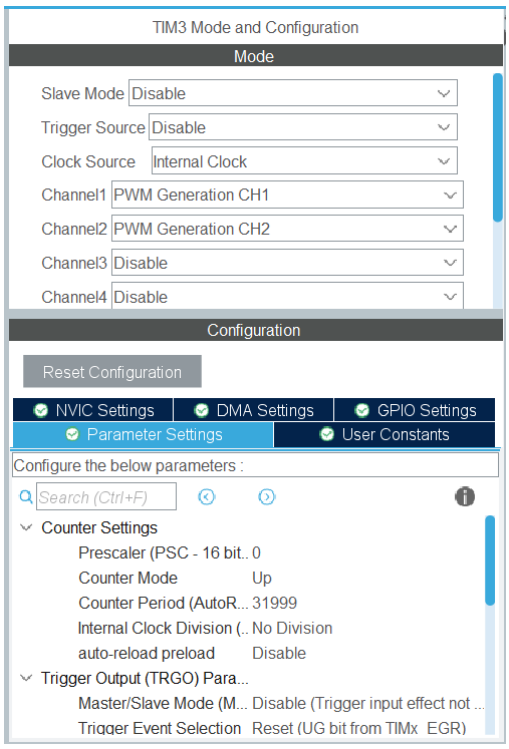


(a) TIM2 (b) TIM3

Hình 11: Cấu hình timer mô phỏng: **STM32F103C6**



(a) TIM2



(b) TIM3

Hình 12: Cấu hình timer hiện thực: **STM32F103RB**

4 HIỆN THỰC

4.1 Bộ định thời

4.1.1 Tổng quan ý tưởng

```
#define SCH_TASKNUMBER 20
SCH_Task tasks[SCH_TASKNUMBER];
```

Program 1: Mảng `task` lưu thông tin các tác vụ

Sử dụng một mảng với số phần tử tối đa `SCH_TASKNUMBER` để lưu trữ thông tin các tác vụ.

4.1.2 Cấu trúc dữ liệu

```
typedef struct {
    void (*functionPointer)(void);
    uint8_t id;
    uint32_t delay;
    uint32_t period;
    unsigned char flag;
} SCH_Task;
```

Program 2: Cấu trúc dữ liệu SCH_Task

Đặc tả cấu trúc dữ liệu của một tác vụ được lưu trữ:

- `functionPointer`: Con trỏ hàm của tác vụ (bắt buộc có kiểu hàm `void(void)`). Nếu `functionPointer == 0` (trỏ đến null), tác vụ chưa được thiết lập và ngược lại.
- `id`: Mã số định danh của mỗi tác vụ. Trong suốt thời gian tồn tại, mỗi tác vụ có một mã số định danh duy nhất.
- `delay`: Khoảng thời gian (ms) cho đến lần thực thi kế tiếp của tác vụ.
- `period`: Khoảng thời gian (ms) giữa hai lần chạy liên tiếp của tác vụ. Nếu `period == 0`, tác vụ chỉ được thực hiện một lần duy nhất.
- `flag`: Cờ thông báo tác vụ có sẵn sàng được thi hay chưa. Nếu `flag == 1`, tác vụ đã sẵn sàng thực thi và ngược lại.

4.1.3 Danh sách hàm bộ định thời

Tên hàm	Tham số	Ý nghĩa	Độ phức tạp
SCH_Init	void	Khởi tạo giá trị cho mảng, với thông số id được gán giá trị cụ thể, đảm bảo tính duy nhất để thuận tiện cho các giải thuật thêm/xóa.	$O(n)$
SCH_Update	void	Kiểm tra phần tử đầu tiên mỗi lần ngắt timer và cập nhật giá trị delay và flag (nếu tác vụ đã được khởi tạo).	$O(1)$
SCH_Dispatch	void	Kiểm tra phần tử đầu tiên nếu sẵn sàng sẽ thực thi, xóa khỏi mảng và bổ sung lại tác vụ vào mảng(nếu là task lặp lại).	$O(n)$
SCH_AddTask	void (*functionPointer) (void), uint32_t delay, uint32_t period	Kiểm tra dung lượng mảng lưu trữ, tìm kiếm vị trí thích hợp để thêm tác vụ vào mảng bằng cách duyệt từng phần tử để dời về sau và chèn tác vụ vào vị trí mong muốn.	$O(n)$
SCH_DeleteTask	uint8_t id	Duyệt từng phần tử để tìm kiếm và xóa tác vụ với id tương ứng. Giá trị id được tái sử dụng cho lần sau.	$O(n)$
SCH_RefreshTask	void	Làm mới tác vụ thay cho việc lần lượt xóa và thêm một tác vụ khi cập nhật trong hàm SCH_Dispatch nhằm đảm bảo ID của tác vụ không bị thay đổi.	$O(n)$

Bảng 6: Danh sách các hàm bộ định thời

4.2 Các thành phần của hệ thống

4.2.1 Nút nhấn

Nút nhấn được hiện thực hỗ trợ cơ chế: **nhấn thả và nhấn đè**. Trong đó, người dùng bắt buộc phải nhấn đè trong thời gian `BUTTON_PRESSED_DURATION` thì hệ thống mới chuyển sang cơ chế tự động tăng/giảm/phản hồi.

```
#define BUTTON_NUMBER 4
#define BUTTON_PRESSED_DURATION 2000
#define BUTTON_HOLDING_DURATION 200
#define BUTTON_PRESSED GPIO_PIN_RESET
#define BUTTON_RELEASED GPIO_PIN_SET
```

Program 3: Giá trị được định nghĩa trong `button.h`

4.2.2 Đèn tín hiệu

4.2.2.a Đèn giao thông

Để thời gian mô phỏng/kiểm thử không kéo dài quá lâu, nhóm đặt thời lượng sáng tối đa mà người dùng được phép điều chỉnh của các đèn là **20s** (tối thiểu là **1s**).

Hệ thống bao gồm 3 màu đèn (xanh, vàng, đỏ) nên sẽ định nghĩa các biến:

- `trafficXDuration`: Thời gian sáng của đèn màu **X**
- `trafficStates`: Mảng lưu trữ trạng thái màu đèn của 2 phía
- `trafficCounters`: Mảng lưu trữ thời gian đếm ngược sáng đèn của 2 phía

```
#define TRAFFIC_NUMBER 2
#define TRAFFIC_DURARION_MIN 1000
#define TRAFFIC_DURARION_MAX 20000
#define TRAFFIC_DURARION_AUTO 1000

enum TRAFFIC_STATE {TRAFFIC_OFF, TRAFFIC_RED,
   TRAFFIC_GREEN, TRAFFIC_YELLOW};

extern uint32_t trafficRedDuration;
extern uint32_t trafficGreenDuration;
extern uint32_t trafficYellowDuration;

extern enum TRAFFIC_STATE trafficStates[TRAFFIC_NUMBER];
extern uint32_t trafficCounters[TRAFFIC_NUMBER];
```

Program 4: Giá trị được định nghĩa trong `traffic.h`

Các hàm được định nghĩa trong file `traffic.c` chủ yếu để bật/tắt các màu đèn, đếm ngược và chuyển trạng thái hệ thống đèn.

```
/* PRIVATE FUNCTION */
void trafficToggle(uint8_t index, enum TRAFFIC_STATE
    state);
/* PUBLIC FUNCTION: from traffic.h */
void traffic0Off(void);
void traffic0Red(void);
void traffic0Green(void);
void traffic0Yellow(void);

void traffic1Off(void);
void traffic1Red(void);
void traffic1Green(void);
void traffic1Yellow(void);
```

Program 5: Các hàm được định nghĩa trong `traffic.c`

4.2.2.b Đèn người đi bộ

Tương tự, đèn cho người đi bộ cũng sẽ hoạt động với các biến để: lưu trữ trạng thái màu đèn và thời gian đếm ngược của hai phía. Đồng thời, các hàm được định nghĩa cũng chỉ phục vụ mục đích bật/tắt đèn, đếm ngược và chuyển trạng thái hệ thống.

```
#define PEDESTRIAN_NUMBER 2

enum PEDESTRIAN_STATE {PEDESTRIAN_OFF, PEDESTRIAN_RED,
    PEDESTRIAN_GREEN};
extern enum PEDESTRIAN_STATE pedestrianStates[
    PEDESTRIAN_NUMBER];
extern uint32_t pedestrianCounters[PEDESTRIAN_NUMBER];
```

Program 6: Giá trị được định nghĩa trong `pedestrian.h`

```
/* PRIVATE FUNCTION */
void pedestrianToggle(uint8_t index, enum
    PEDESTRIAN_STATE state);
/* PUBLIC FUNCTION: from pedestrian.h */
void pedestrian0On(void);
void pedestrian1On(void);
void pedestrian0Off(void);
void pedestrian1Off(void);
```

Program 7: Các hàm được định nghĩa trong `pedestrian.c`

4.2.3 Loa phát thanh

Âm lượng của loa phát thanh được quy đổi về khoảng giá trị **[0 - 100]** để người dùng dễ hiểu và sử dụng.

```
#define BUZZER_NUMBER 2
#define BUZZER_RATIO 20 / 100
#define BUZZER_MIN 0
#define BUZZER_MAX htim3.Init.Period
#define BUZZER_TIMEOUT 2000

#define BUZZER_VOLUME_MIN 0   // Min volume: virtual
#define BUZZER_VOLUME_MAX 100 // Max volume: virtual
#define BUZZER_VOLUME_AUTO 10 // Auto volume: virtual

extern uint8_t buzzerVolumes[BUZZER_NUMBER];
```

Program 8: Giá trị được định nghĩa trong `buzzer.h`

Các hàm của loa phát thanh được hiện thực để: bật/tắt và tăng âm lượng của loa.

```
void buzzer0On(void);
void buzzer0Off(void);
void buzzer1On(void);
void buzzer1Off(void);
```

Program 9: Các hàm được định nghĩa trong `buzzer.c`

4.2.4 Giao tiếp UART

Các hàm trong file này sử dụng cho việc truyền dữ liệu với cú pháp **!7SEG:xx** *(với xx là thời gian có hai chữ số)* thể hiện:

- `uartXduration`: Thời lượng sáng của đèn màu **X**.
- `uartCounter`: Thời gian đếm ngược của đèn giao thông phía A (phục vụ cho việc mô phỏng).

```
void uartRedDuration(void);
void uartGreenDuration(void);
void uartYellowDuration(void);
void uartCounter(void);
```

Program 10: Các hàm được định nghĩa trong `uart.c`

4.3 Máy trạng thái

4.3.1 Tổng quan

Hệ thống được vận hành với 3 trạng thái (chưa bao gồm trạng thái khởi tạo) và được chuyển bằng BUTTON_0. Vì vậy, trong hàm `while(1)`, ta sẽ chạy hàm `fsmProcessing(void)` để điều khiển toàn bộ hệ thống giao thông. Mỗi chế độ sẽ được hoạt động trên cơ chế của bộ định thời.

```
void fsmInit(void) {
  fsmState = FSM_INIT;
  fsmIDs[0] = SCH_AddTask(testButton, 0, TIMER_TICK);
  SCH_AddTask(testBuzzer, 0, 0);
  SCH_AddTask(testLED, 0, 0);
}
void fsmProcessing(void) {
  if (buttonPressed(0)) {
    switch (fsmState) {
    case FSM_INIT:
      for (uint8_t i = 0; i < FSM_TASK; i ++) {
        SCH_DeleteTask(fsmIDs[i]);
      }
      SCH_AddTask(fsmAuto, 10, 0);
      fsmState = FSM_AUTO;
      break;
    case FSM_AUTO:
      for (uint8_t i = 0; i < FSM_AUTO_TASK; i ++) {
        SCH_DeleteTask(fsmAutoIDs[i]);
      }
      SCH_AddTask(fsmAutoStop, 0, 0);
      SCH_AddTask(fsmManual, 10, 0);
      fsmState = FSM_MANUAL;
      break;
    case FSM_MANUAL:
      for (uint8_t i = 0; i < FSM_MANUAL_TASK; i ++) {
        SCH_DeleteTask(fsmManualIDs[i]);
      }
      SCH_AddTask(fsmManualStop, 0, 0);
      SCH_AddTask(fsmTunning, 10, 0);
      fsmState = FSM_TUNNING;
      break;
    case FSM_TUNNING:
```

```
34        for (uint8_t i = 0; i < FSM_TUNNING_TASK; i ++) {
35          SCH_DeleteTask(fsmTunningIDs[i]);
36        }
37        SCH_AddTask(fsmTunningStop, 0, 0);
38        SCH_AddTask(fsmAuto, 10, 0);
39        fsmState = FSM_AUTO;
40        break;
41      default:
42        break;
43      }
44    }
45  }
```

Program 11: Hàm khởi tạo và thực thi FSM của toàn bộ hệ thống

5 KIỂM THỬ

5.1 Kịch bản kiểm thử

Kịch bản được xây dựng để kiểm tra tính đúng đắn 3 chế độ hoạt động của hệ thống đèn giao thông, với kết đính kèm là kết quả mô phỏng bằng Proteus.

BUTTON0 là nút nhấn chuyên biệt để chuyển giữa các chế độ hoạt động của hệ thống: Tự động → Thủ công → Điều chỉnh (và quay trở về "Tự động").

5.1.1 Chế độ: Tự động

Mã	Mục đích	Cách thực hiện	Kỳ vọng	Kết quả
KD-01	Kiểm thử kết nối: đèn/loa/nút nhấn	Nạp, chạy chương trình và nhấn các BUTTON1, BUTTON2, BUTTON3	- Ban đầu, đèn/loa đều bật, đèn hệ thống nháy mỗi 1s - Mỗi lần nhấn/thả thì các đèn sẽ chớp tắt để kiểm tra các thiết bị đã kết nối thành công	Thành công
KD-02	Khởi động hệ thống	Sau khi kiểm tra ở bước **KD-01**, nhấn BUTTON0 để bắt đầu	Hệ thống chuyển sang chế độ "Tự động"	Thành công
TD-01	Kiểm thử thời gian hoạt động đèn cho phương tiện	Theo dõi thời gian thông qua Virtual Terminal và màu đèn	Đèn đổi màu: xanh → vàng → đỏ khi hết thời gian	Thành công
TD-02	Kiểm thử xin sang đường	Nhấn BUTTON1 xin sang đường (khi đèn xanh/vàng)	- Đèn người đi bộ sáng: Đỏ (chưa được đi) - Đèn người đi bộ chuyển sang: Xanh (loa bật: nhỏ, chậm) khi được phép sang đường - Loa tăng âm lượng và nhanh dần khi gần hết giờ	Thành công

Bảng 7: Kiểm thử: Chế độ Tự động

Trường hợp **TD-02**, người đi bộ nhấn nút xin sang đường nhiều lần thì hệ thống cũng chỉ nhận duy nhất một lần trong mỗi chu kỳ đèn đi bộ (không cộng dồn số lần nhấn).

5.1.2 Chế độ: Thủ công

Từ chế độ "Tự động", người dùng nhấn BUTTON0 để chuyển sang chế độ "Thủ công" để chỉnh tay màu của từng phía đèn giao thông.

Mã	Mục đích	Cách thực hiện	Kỳ vọng	Kết quả
TC-01	Đổi màu đèn phía A	Nhấn thả/đè BUTTON1	Mỗi lần nhấn, màu của đèn sẽ chuyển theo thứ tự: xanh → vàng → đỏ	Thành công
TC-02	Đổi màu đèn phía B	Nhấn thả/đè BUTTON2	Mỗi lần nhấn, màu của đèn sẽ chuyển theo thứ tự như **TC-01**	Thành công

Bảng 8: Kiểm thử: Chế độ Thủ công

5.1.3 Chế độ: Điều chỉnh

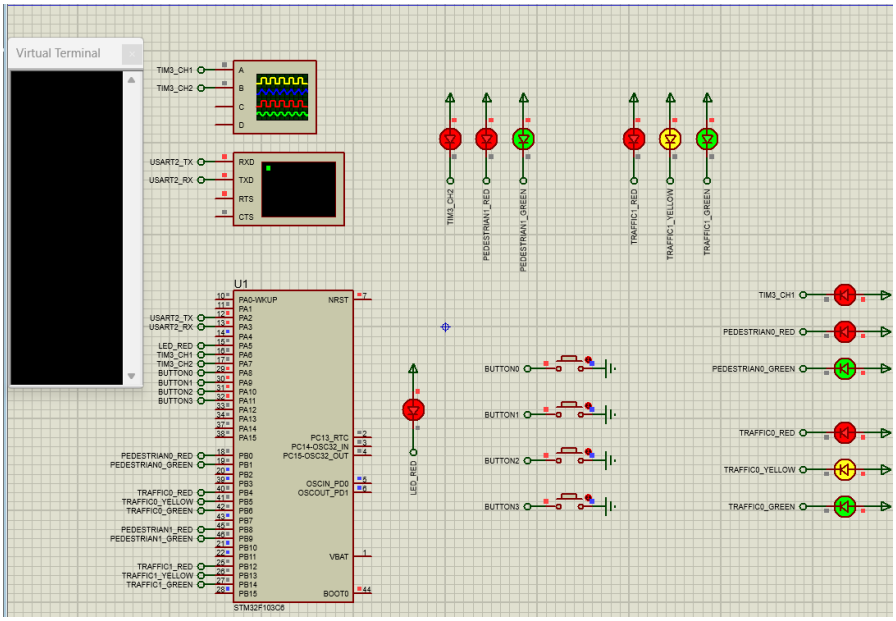
Từ chế độ "Thủ công", người dùng nahans BUTTON0 để chuyển sang chế độ "Điều chỉnh" để thay đổi thời gian sáng của các màu đèn trong chế độ "Tự động".

Mã	Mục đích	Cách thực hiện	Kỳ vọng	Kết quả
DC-01	Chọn màu đèn để chỉnh thời gian (nhấn thả)	Nhấn thả/đè BUTTON1	Đèn của hai phía sẽ sáng màu đang điều chỉnh: xanh → vàng → đỏ	Thành công
DC-02	Tăng thời gian sáng đèn	Nhấn thả/đè BUTTON2, theo dõi Virtual Terminal	Thời gian sáng đèn của màu đang điều chỉnh sẽ tăng dần	Thành công
DC-03	Giảm thời gian sáng đèn	Nhấn thả/đè BUTTON3, theo dõi Virtual Terminal	Thời gian sáng đèn của màu đang điều chỉnh sẽ giảm dần	Thành công

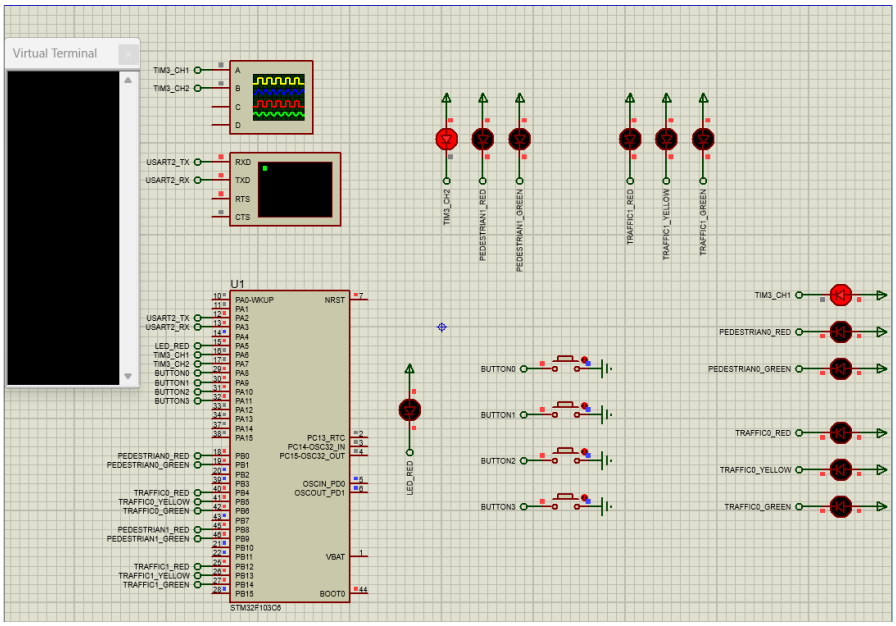
Bảng 9: Kiểm thử: Chế độ Điều chỉnh

5.2 Kết quả kiểm thử

5.2.1 Chế độ: Tự động

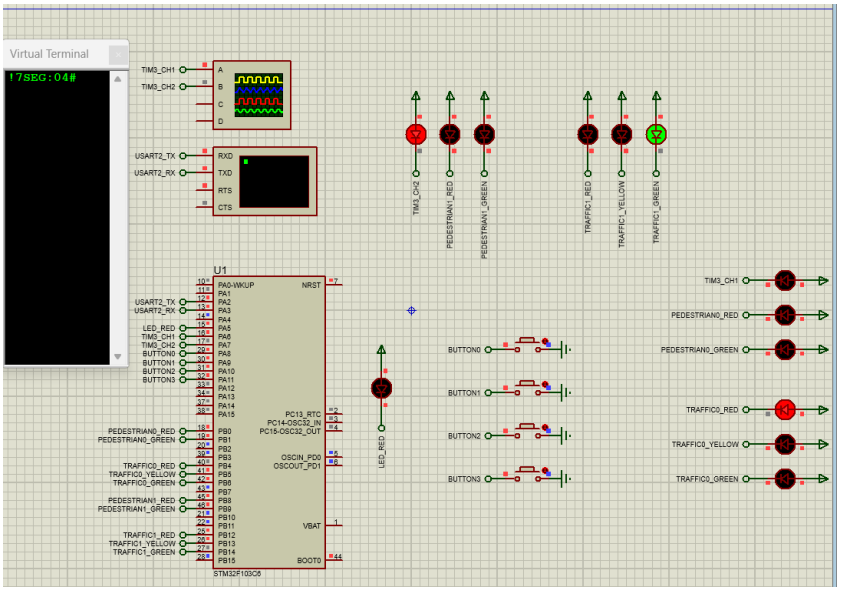


(a) Trước khi nhấn BUTTON1 hoặc BUTTON2 hoặc BUTTON3

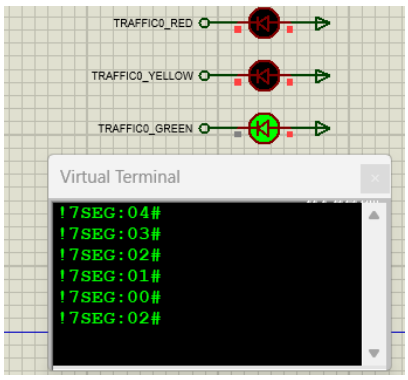


(b) Khi nhấn BUTTON1, hoặc BUTTON2, hoặc BUTTON3

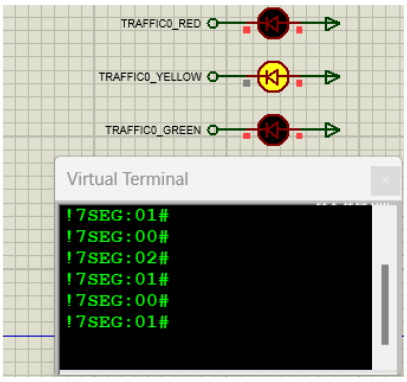
Hình 13: **KD-01**: Kiểm thử kết nối đèn/loa/nút nhấn



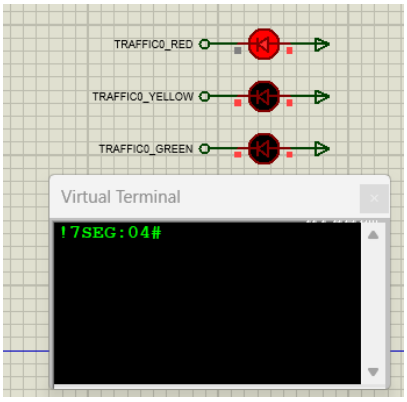
Hình 14: **KD-02:** Khởi động hệ thống



(a) Đèn xanh (mặc định: 2s)

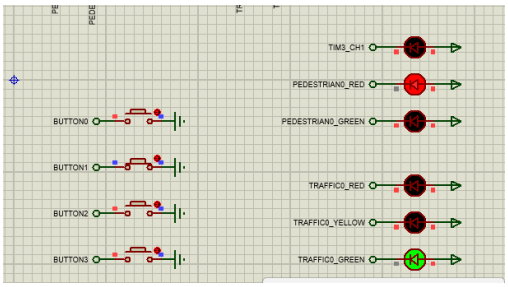


(b) Đèn vàng (mặc định: 1s)

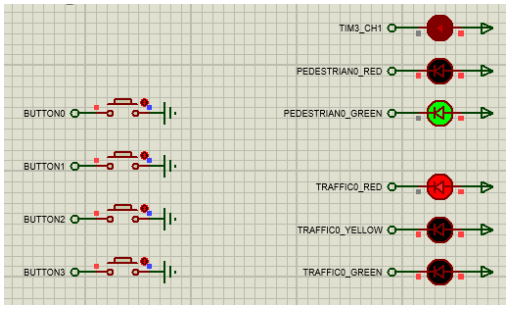


(c) Đèn đỏ (mặc định: 4s)

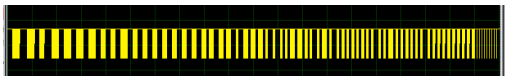
Hình 15: **TD-01:** Kiểm thử thời gian hoạt động đèn cho phương tiện



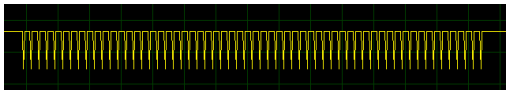
(a) Đèn đỏ: Chưa cho sang đường



(b) Đèn xanh: Cho sang đường (loa bật)



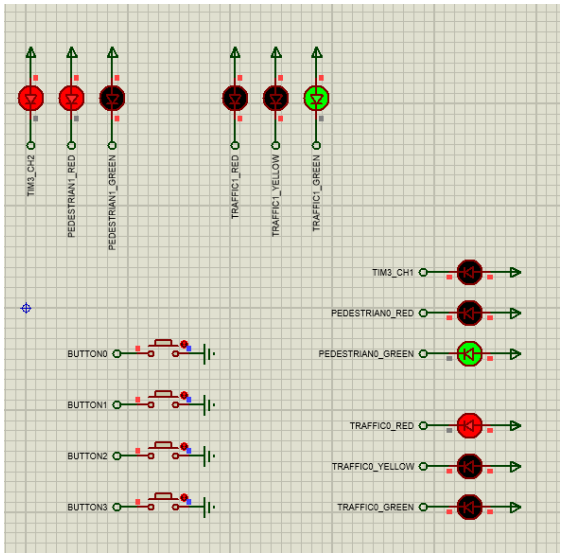
(c) Xung PWM: Toàn bộ quá trình



(d) Xung PWM: Xung mỗi đợt băm

Hình 16: **TD-02:** Kiểm thử xin sang đường

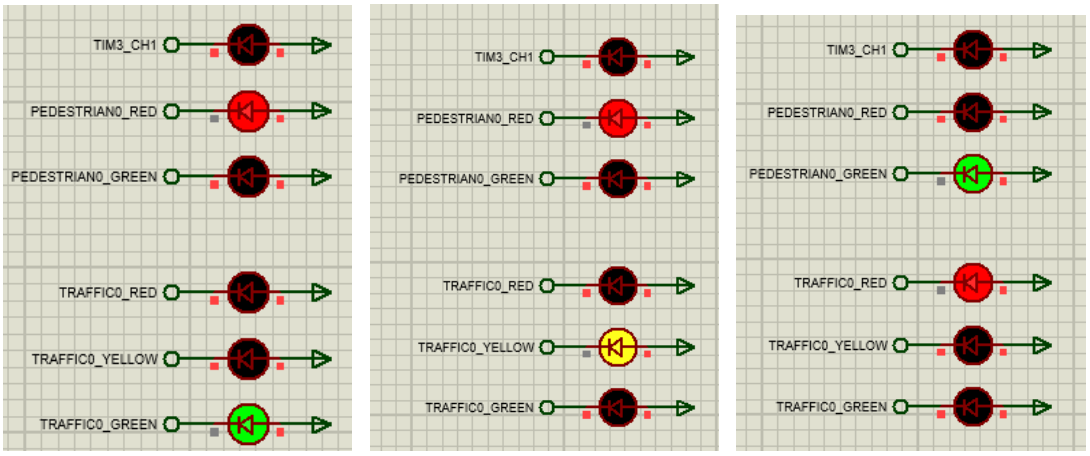
5.2.2 Chế độ: Thủ công



Hình 17: Trạng thái ban đầu: Chế độ Thủ công

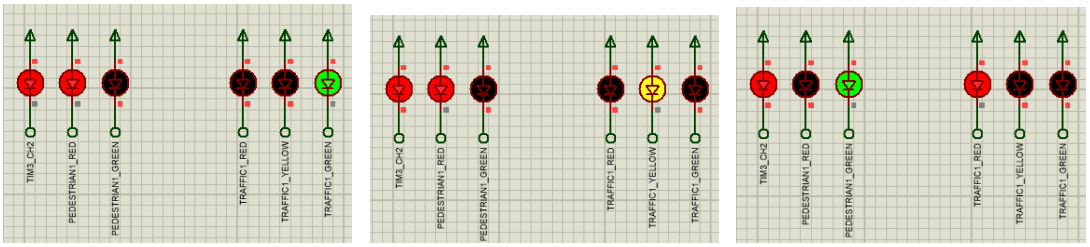
Ở trạng thái ban đầu của chế độ Thủ công:

- **Đèn phía A** *(bên phải)*: Đèn giao thông được đặt màu đỏ (tương ứng với đèn màu xanh cho người đi bộ). Đèn được đổi màu bởi **BUTTON1** theo thứ tự: đỏ → xanh → vàng, và lặp lại.
- **Đèn phía B** *(bên trên)*: Đèn giao thông được đặt màu xanh (tương ứng với đèn màu đỏ cho người đi bộ). Đèn được đổi màu bởi **BUTTON2** theo thứ tự: xanh → vàng → đỏ, và lặp lại.



(a) Xe: xanh; Đi bộ: đỏ (b) Xe: vàng; Đi bộ: đỏ (c) Xe: đỏ; Đi bộ: xanh

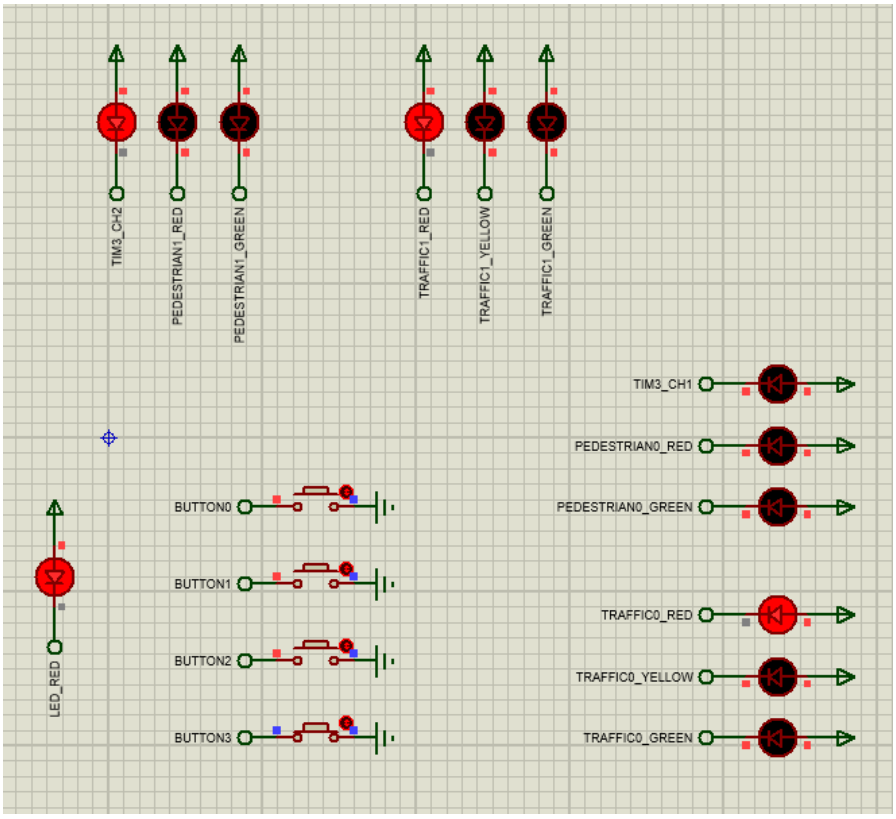
Hình 18: **TC-01:** Đổi màu đèn phía A *(bên phải)*



(a) Xe: xanh; Đi bộ: đỏ (b) Xe: vàng; Đi bộ: đỏ (c) Xe: đỏ; Đi bộ: xanh

Hình 19: **TC-02:** Đổi màu đèn phía B *(bên trên)*

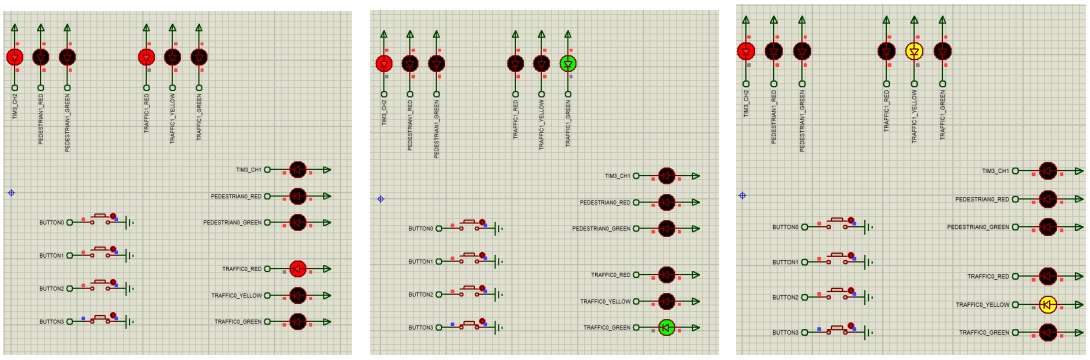
5.2.3 Chế độ: Điều chỉnh



Hình 20: Trạng thái ban đầu: Chế độ Điều chỉnh

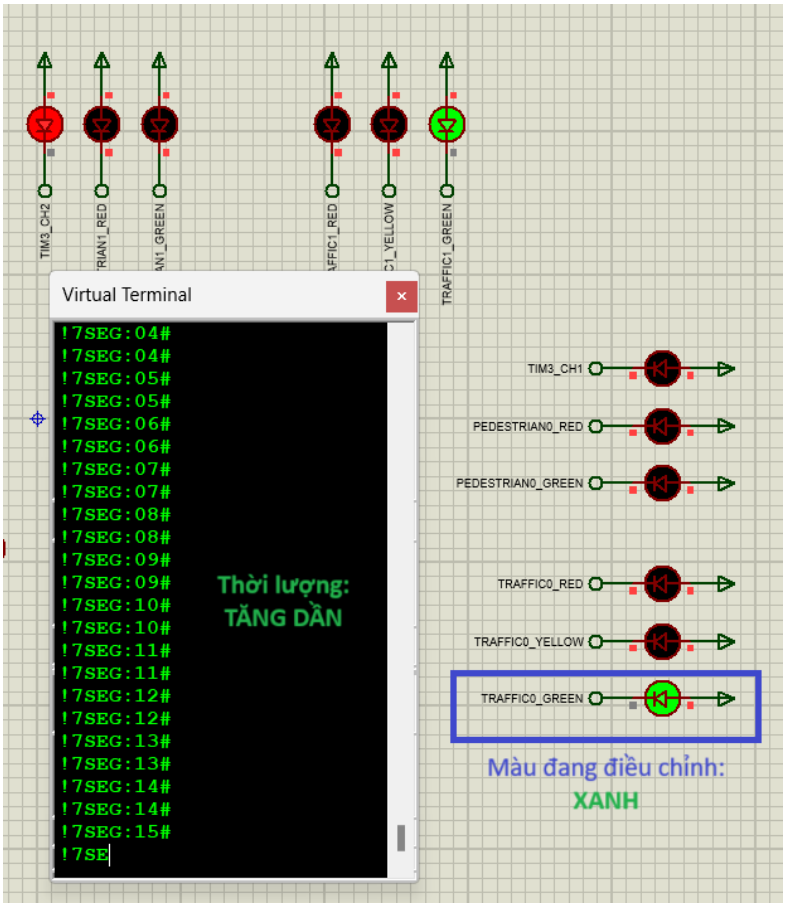
Ở trạng thái ban đầu của chế độ Điều chỉnh, đèn cho phương tiện nhấp nháy với màu đỏ - màu đèn đang được điều chỉnh. Người dùng có thể:

- **Chọn màu đèn khác:** nhấn BUTTON1 để đổi màu đèn muốn điều chỉnh thời lượng sáng. Đèn sẽ thay đổi theo thứ tự: đỏ → xanh → vàng, lặp lại.
- **Tăng thời lượng:** nhấn BUTTON2 để tăng thời lượng sáng của màu đang điều chỉnh. Thời lượng sẽ được cài đặt để chạy trong chế độ Tự động, hiển thị cụ thể lên Virtual Terminal.
- **Giảm thời lượng:** nhấn BUTTON3 để giảm thời lượng sáng của màu đang điều chỉnh. Thời lượng sẽ được cài đặt để chạy trong chế độ Tự động, hiển thị cụ thể lên Virtual Terminal.

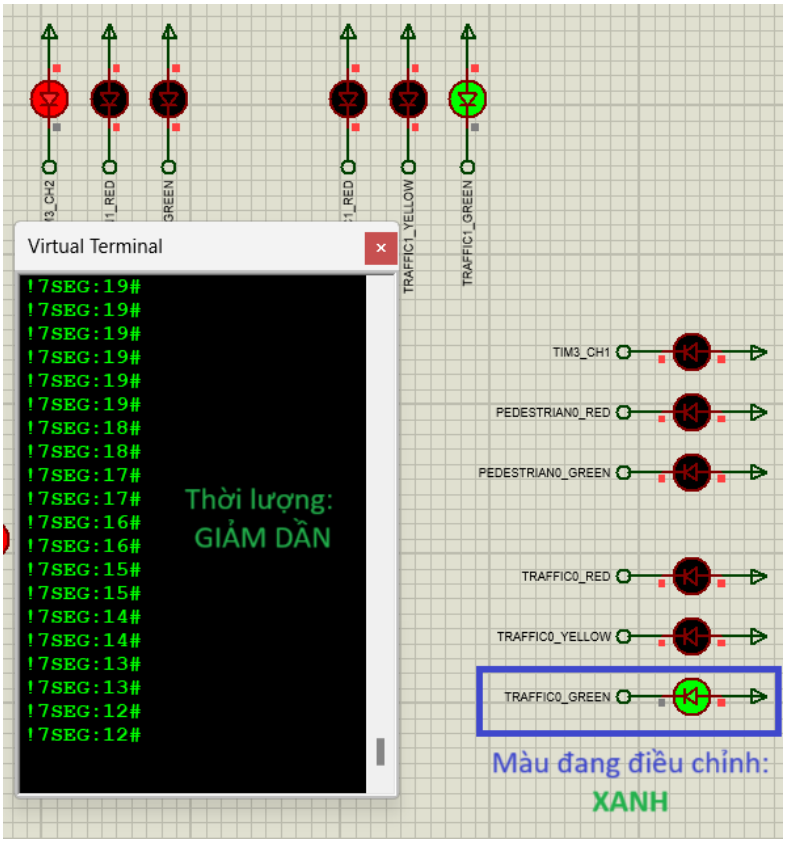


(a) Màu điều chỉnh: đỏ (b) Màu điều chỉnh: xanh (c) Màu điều chỉnh: vàng

Hình 21: **DC-01:** Chọn màu muốn điều chỉnh



Hình 22: **DC-02:** Tăng thời lượng sáng đèn



Hình 23: **DC-03:** Giảm thời lượng sáng đèn

6 PHỤ LỤC

6.1 Bộ định thời

6.1.1 Hàm khởi tạo

```
void SCH_Init(void) {
    for (uint8_t i = 0; i < SCH_TASKNUMBER; i ++) {
        tasks[i].functionPointer = 0;
        tasks[i].id = SCH_TASKNUMBER - i;
        tasks[i].delay = 0;
        tasks[i].period = 0;
        tasks[i].flag = 0;
    }
}
```

Program 12: Hàm khởi tạo: `SCH_Init`

6.1.2 Hàm cập nhật

```
void SCH_Update(void) {
    if (tasks[0].functionPointer == 0) return;
  if (tasks[0].delay > 0) {
    if (tasks[0].delay > TIMER_TICK) {
      tasks[0].delay -= TIMER_TICK;
    }
    else {
      tasks[0].delay = 0;
    }
  }
  if (tasks[0].delay == 0) {
    tasks[0].flag = 1;
  }
}
```

Program 13: Hàm cập nhật: `SCH_Upadte`

6.1.3 Hàm thực thi

```
void SCH_Dispatch(void) {
    if (tasks[0].flag == 0) return;
    (*tasks[0].functionPointer)();
    if (tasks[0].period > 0) {
        SCH_RefreshTask();
    }
    else {
        SCH_DeleteTask(tasks[0].id);
    }
}
```

Program 14: Hàm thực thi: SCH_Dispatch

6.1.4 Hàm thêm tác vụ

```
uint8_t SCH_AddTask(void (*functionPointer)(void),
   uint32_t delay, uint32_t period) {
    if (tasks[SCH_TASKNUMBER - 1].functionPointer != 0)
   return 0;
    uint8_t currentID = tasks[SCH_TASKNUMBER - 1].id;
    uint32_t currentDelay = 0;
    for (uint8_t i = 0; i < SCH_TASKNUMBER; i ++) {
        currentDelay += tasks[i].delay;
        if (currentDelay > delay || tasks[i].
   functionPointer == 0) {
            for (uint8_t j = SCH_TASKNUMBER - 1; j > i;
   j --) {
                tasks[j] = tasks[j - 1];
            }
            tasks[i].functionPointer = functionPointer;
            tasks[i].id = currentID;
            tasks[i].period = period;
            tasks[i].flag = 0;
            if (currentDelay > delay) {
                int newDelay = currentDelay - delay;
                tasks[i].delay = tasks[i + 1].delay -
   newDelay;
                if (tasks[i].delay == 0) {
                    tasks[i].flag = 1;
                }
                tasks[i + 1].delay = newDelay;
```

```
                if (tasks[i + 1].delay == 0) {
                    tasks[i + 1].flag = 1;
                }
            }
            else {
                tasks[i].delay = delay - currentDelay;
                if (tasks[i].delay == 0) {
                    tasks[i].flag = 1;
                }
            }
            return tasks[i].id;
        }
    }
    return 0;
}
```

Program 15: Hàm thêm tác vụ: `SCH_AddTask`

6.1.5 Hàm xóa tác vụ

```
unsigned char SCH_DeleteTask(uint8_t id) {
    for (uint8_t i = 0; i < SCH_TASKNUMBER; i ++) {
      if (tasks[i].functionPointer == 0) return 0;
        if (tasks[i].id == id) {
            uint8_t currentID = tasks[i].id;
            if (tasks[i + 1].functionPointer != 0) {
                tasks[i + 1].delay += tasks[i].delay;
            }
            for (uint8_t j = i; j < SCH_TASKNUMBER - 1;
   j ++) {
                tasks[j] = tasks[j + 1];
            }
            tasks[SCH_TASKNUMBER - 1].functionPointer =
   0;
            tasks[SCH_TASKNUMBER - 1].id = currentID;
            tasks[SCH_TASKNUMBER - 1].delay = 0;
            tasks[SCH_TASKNUMBER - 1].period = 0;
            tasks[SCH_TASKNUMBER - 1].flag = 0;
            return tasks[SCH_TASKNUMBER - 1].
   functionPointer == 0;
        }
```

```
    }
    return 0;
}
```

Program 16: Hàm xóa tác vụ: SCH$_D$elete

6.1.6 Hàm làm mới tác vụ

```
unsigned char SCH_RefreshTask(void) {
    if (tasks[0].functionPointer == 0) return 0;
    SCH_Task currentTask = tasks[0];
    uint32_t currentDelay = 0;
    for (uint8_t i = 0; i < SCH_TASKNUMBER; i ++) {
        if (i + 1 == SCH_TASKNUMBER || tasks[i + 1].
   functionPointer == NULL) {
            tasks[i].functionPointer = currentTask.
   functionPointer;
            tasks[i].id = currentTask.id;
            tasks[i].period = currentTask.period;
            tasks[i].flag = 0;
            tasks[i].delay = currentTask.period -
   currentDelay;
            if (tasks[i].delay == 0) {
                tasks[i].flag = 1;
            }
            return 1;
        }
        currentDelay += tasks[i + 1].delay;
        if (currentDelay > currentTask.period) {
            tasks[i].functionPointer = currentTask.
   functionPointer;
            tasks[i].id = currentTask.id;
            tasks[i].period = currentTask.period;
            tasks[i].flag = 0;
            int newDelay = currentDelay - currentTask.
   period;
            tasks[i].delay = tasks[i + 1].delay -
   newDelay;
            if (tasks[i].delay == 0) {
                tasks[i].flag = 1;
            }
```

```
28                 tasks[i + 1].delay -= tasks[i].delay;
29                 if (tasks[i + 1].delay == 0) {
30                     tasks[i + 1].flag = 1;
31                 }
32                 return 1;
33             }
34             else {
35                 tasks[i] = tasks[i + 1];
36             }
37         }
38         return 0;
39 }
```

Program 17: Hàm làm mới tác vụ: SCH_RefreshTask