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

BÁO CÁO THỰC HÀNH

VI XỬ LÝ - VI ĐIỀU KHIỂN

LAB 3: BUTTONS / SWITCHES

Lớp – Nhóm: L03 – L05
Giảng viên hướng dẫn: Lê Trọng Nhân
Cao Tiến Đạt
Sinh viên thực hiện: Ngô Quang Tùng 2213869

Thành phố Hồ Chí Minh, 10/2024

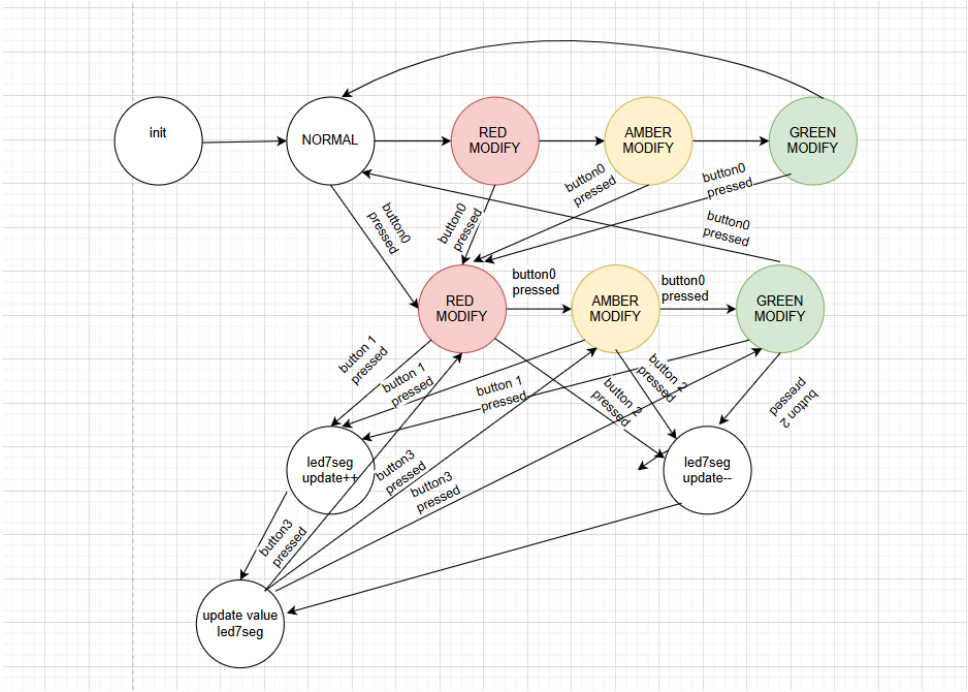
Mục lục

Chương 3

Buttons / Switches

Link GitHub

3.1 Bài tập 1: Phác thảo FSM



Hình 3.1: Sơ đồ nguyên lý

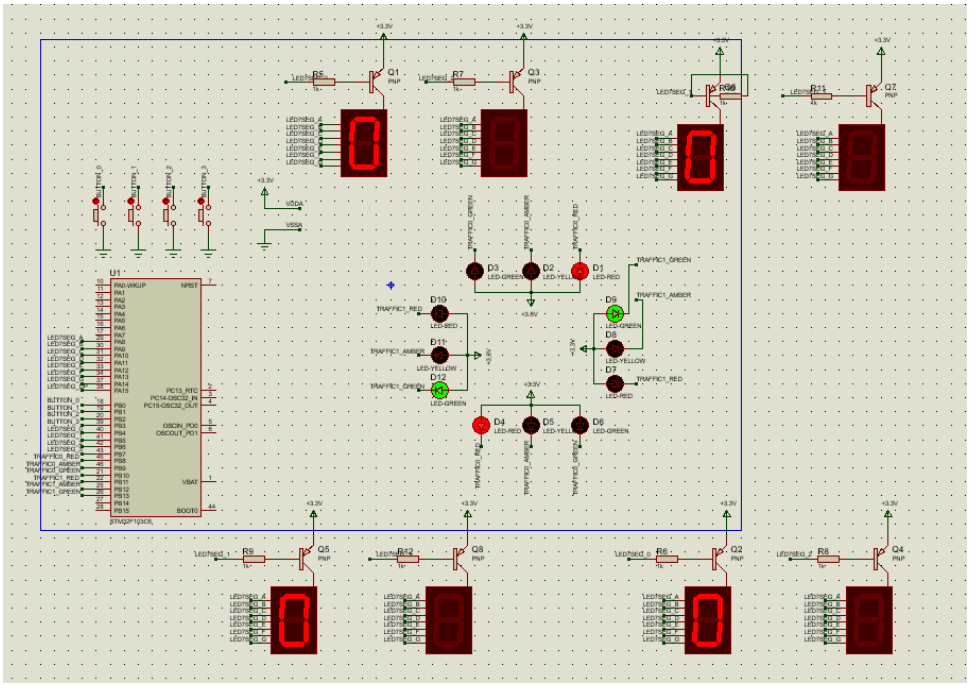
Trong vòng lặp `while(1)`, xác định có 2 máy trạng thái thực thi độc lập, thực hiện:

- Quét các LED 7 đoạn, kiểm soát bởi `timer0`. Khi `timer0Flag == 1`, theo thứ tự thay đổi tín hiệu `enable` cho các LED 7 đoạn, tạo ra hiệu ứng quét.
- Điều khiển sự chuyển đổi giữa các mode, dựa trên tín hiệu từ nút nhấn `button0`. Khi `button0` được nhấn, mode thay đổi theo thứ tự **Normal Mode (Mode 1)**, **Red Modify Mode (Mode 2)**, **Amber Modify Mode (Mode 3)**, **Green Modify Mode (Mode 4)** và quay lại **Normal Mode**.

- Bật / tắt toàn bộ các LED Red, kiểm soát bởi `timer1`. Khi `timer1Flag == 1`, thay đổi trạng thái của toàn bộ LED Red.
- Tăng giá trị tạm thời cho **Red Duration**, dựa trên tín hiệu từ nút nhấn `button1`. Khi `button1` được nhấn, tăng giá trị tạm thời của **Red Duration** lên 1 đơn vị. Đồng thời, cập nhật giá trị phù hợp cho các LED 7 đoạn.
- Giảm giá trị tạm thời cho **Red Duration**, dựa trên tín hiệu từ nút nhấn `button2`. Có cơ chế thực hiện tương tự với tăng giá trị tạm thời cho **Red Duration**. Đây là tính năng bổ sung so với yêu cầu của đề bài, làm tăng tính linh hoạt của việc cập nhật giá trị **Duration**.
- Cập nhật giá trị cho **Red Duration**, dựa trên tín hiệu từ nút nhấn `button3`. Khi `button3` được nhấn, giá trị **Red Duration** hiện tại sẽ được cập nhật bằng giá trị tạm thời.

Trong **Amber Modify Mode** và **Green Modify Mode**, xác định mỗi mode có 4 máy trạng thái thực thi độc lập, tương tự với **Red Modify Mode**. Do đó, sẽ không trình bày lại các mode nêu trên.

3.2 Bài tập 2: Sơ đồ nguyên lý Proteus

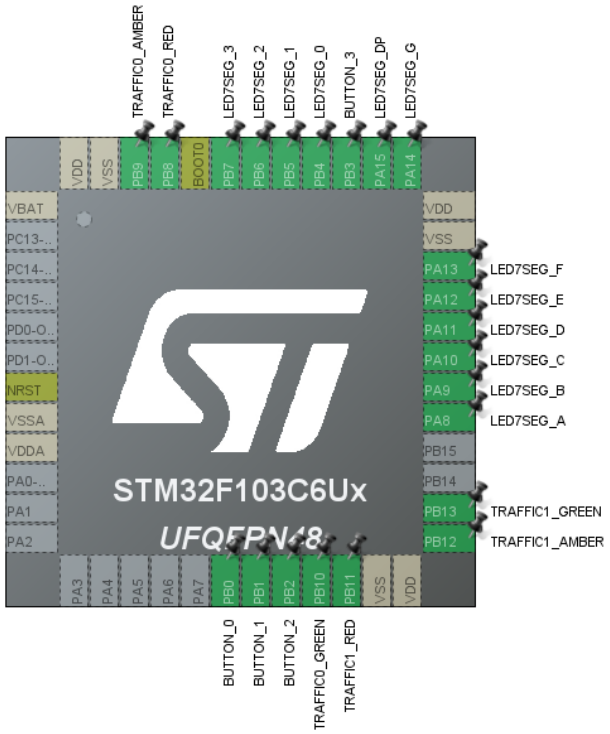


Hình 3.2: Sơ đồ nguyên lý

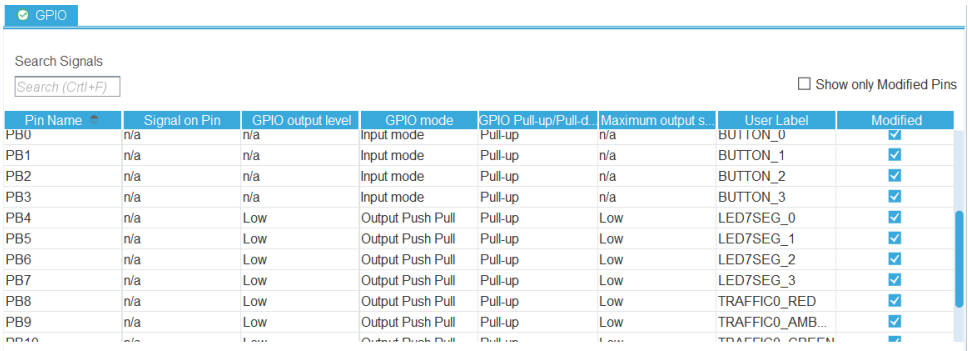
3.3 Bài tập 3: Tạo dự án STM32

Dựa theo sơ dồ nguyên lý, thiết lập các chân GPIO cho vi điều khiển:
• Đối với các tín hiệu input, thiết lập GPIO Input Mode và GPIO Pull-up Pin (Hình1.6).

• Đối với các tín hiệu output, thiết lập GPIO Output Push Pull Mode và GPIO Pull-up Pin



Hình 3.3: STM32 Project: Pinout



Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pull-d...	Maximum output s...	User Label	Modified
PB0	n/a	n/a	Input mode	Pull-up	n/a	BUTTON_0	☑
PB1	n/a	n/a	Input mode	Pull-up	n/a	BUTTON_1	☑
PB2	n/a	n/a	Input mode	Pull-up	n/a	BUTTON_2	☑
PB3	n/a	n/a	Input mode	Pull-up	n/a	BUTTON_3	☑
PB4	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_0	☑
PB5	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_1	☑
PB6	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_2	☑
PB7	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_3	☑
PB8	n/a	Low	Output Push Pull	Pull-up	Low	TRAFFIC0_RED	☑
PB9	n/a	Low	Output Push Pull	Pull-up	Low	TRAFFIC0_AMB...	☑

Hình 3.4: STM32 Project: GPIO Input

GPIO

Search Signals

Search (Ctrl+F)

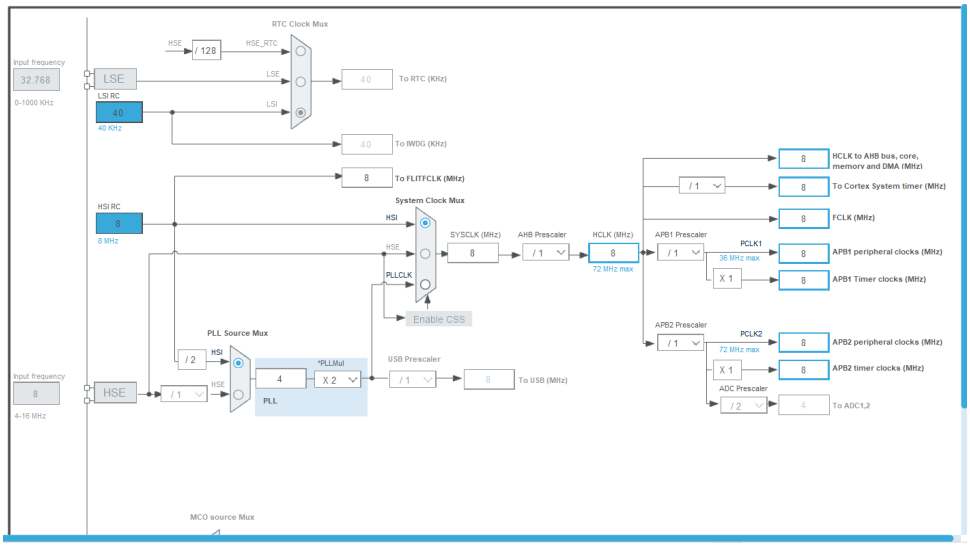
Show only Modified Pins

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pull-d..	Maximum output s..	User Label	Modified
PA8	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_A	☑
PA9	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_B	☑
PA10	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_C	☑
PA11	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_D	☑
PA12	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_E	☑
PA13	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_F	☑
PA14	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_G	☑
PA15	n/a	Low	Output Push Pull	Pull-up	Low	LED7SEG_DP	☑
PB0	n/a	n/a	Input mode	Pull-up	n/a	BUTTON_0	☑
PB1	n/a	n/a	Input mode	Pull-up	n/a	BUTTON_1	☑

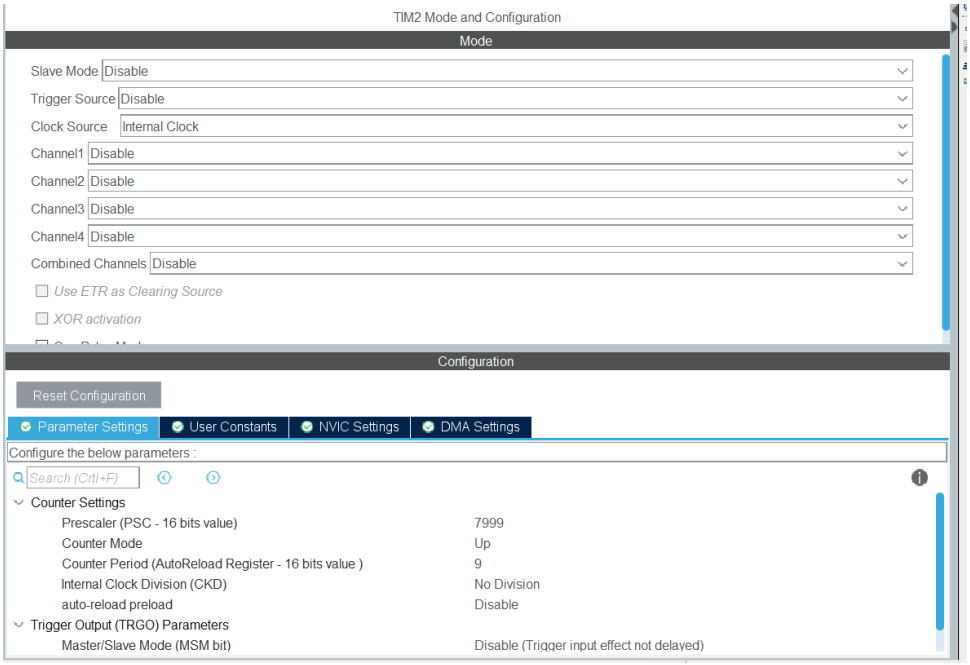
Hình 3.5: STM32 Project: GPIO Input

Để thiết lập timer interrupt bằng 10ms, thực hiện các bước:

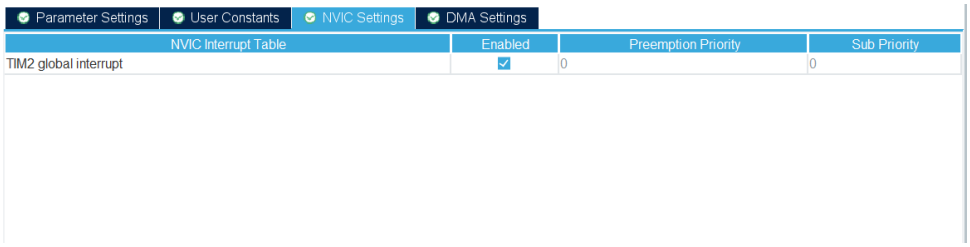
- Thiết lập tần số của Internal Clock bằng 8MHz (Hình 1.8).
- Chọn Internal Clock làm Clock Source của timer 2 (Hình 1.9).
- Thiết lập giá trị Prescaler bằng 7999, giá trị Counter Period bằng 9 (Hình 1.10).
- Chọn Enable cho TIM2 global interrupt (Hình 1.11).



Hình 3.6: STM32 Project: Pinout



Hình 3.7: STM32 Project: Clock Source và Parameter Settings



Hình 3.8: STM32 Project: NVIC Settings

3.4 Bài tập 4: Điều chỉnh các thông số thời gian

3.4.1 Đặc tả

Sử dụng header file `software_timer.h` và source file `software_timer.c` để lưu giá trị timer interrupt và quản lý toàn bộ software timer. Để đảm bảo khi thay đổi timer interrupt, có thể thay đổi ít nhất mà không ảnh hưởng đến hệ thống, xác định: Thiết

- Từ máy trạng thái, định nghĩa số lượng timer cần thiết `TIMER_NUMBER` bằng 4.
- Định nghĩa `TIMER_DURATION` lưu giá trị của timer interrupt, có giá trị bằng 10(ms).
- Sử dụng hàm `timerSet` để khởi tạo giá trị ban đầu cho các timer.
- Sử dụng hàm `timerRun` để timer hoạt động theo đúng tín hiệu từ timer interrupt.
- Quy định các giá trị Duration được đặt đúng bằng thời gian thực (đơn vị ms).
- Quy định tham số của các hàm `timerSet` là `DURATION / TIMER_DURATION`.

- Khi đó, nếu timer interrupt thay đổi, chỉ cần thay đổi giá trị của `TIMER_DURATION` đúng bằng giá trị timer interrupt.

```h
#ifndef INC_SOFTWARE_TIMER_H_
#define INC_SOFTWARE_TIMER_H_

#include "main.h"

#define TIMER_NUMBER 4
#define TIMER_DURATION 10

extern uint8_t timerFlags[TIMER_NUMBER];

void timerSet(int duration, int index);
void timerRun(void);

#endif /* INC_SOFTWARE_TIMER_H_ */
```

Chương trình 1.9: Header file software_timer,h

```c
#include "software_timer.h"

uint8_t timerFlags[TIMER_NUMBER];
static int16_t timerCounters[TIMER_NUMBER];

void timerSet(int duration, int index) {
    if (index >= 0 && index < TIMER_NUMBER) {
        timerCounters[index] = duration;
        timerFlags[index] = 0;
    }
}

void timerRun(void) {
    for (int i = 0; i < TIMER_NUMBER; i++) {
        if (timerCounters[i] > 0) {
            timerCounters[i]--;
            if (timerCounters[i] <= 0) {
                timerFlags[i] = 1;
            }
        }
    }
}
```

Chương trình 2.1: Source file software_timer,c

```c
#include "software_timer.h"

uint8_t timerFlags[TIMER_NUMBER];
static int16_t timerCounters[TIMER_NUMBER];

void timerSet(int duration, int index) {
    if (index >= 0 && index < TIMER_NUMBER) {
        timerCounters[index] = duration;
        timerFlags[index] = 0;
    }
}

void timerRun(void) {
    for (int i = 0; i < TIMER_NUMBER; i++) {
        if (timerCounters[i] > 0) {
            timerCounters[i]--;
            if (timerCounters[i] <= 0) {
                timerFlags[i] = 1;
            }
        }
    }
}
```

Chương trình 2.2: source file main.c

3.5 Bài tập 5: Hiện thực khử rung nút nhấn

Khởi tạo các header file và source file:

- `input_button`: Quản lý toàn bộ nút nhấn.
- `output_led7seg`: Điều khiển các LED 7 đoạn.
- `output_traffic`: Điều khiển các LED đơn (đèn giao thông).
- `processing_fsm`: Quản lý máy trạng thái.

Dưới đây là quản lý nút nhấn, xác định:

- Từ máy trạng thái, định nghĩa số lượng nút nhấn cần thiết `BUTTON_NUMBER` bằng 4.
- Nếu được nhấn lâu hơn 1 giây, nút nhấn sẽ vào trạng thái nhấn đè. Khi đó, trạng thái sẽ thay đổi sau mỗi 0.5 giây.
- Sử dụng hàm `buttonReading` để đọc tín hiệu từ nút nhấn, việc đọc tín hiệu từ nút nhấn được thực hiện mỗi khi timer interrupt xảy ra.

- Sử dụng hàm `buttonPressed` để xác định trạng thái hiện tại của nút nhấn. Tất cả các chương trình khác chỉ có thể gọi hàm này để xác định trạng thái của nút nhấn.

Dưới đây là quản lý máy trạng thái, xác định:

- Sử dụng hàm `fsmInit` để khởi tạo trạng thái ban đầu cho máy trạng thái.
- Sử dụng hàm `fsmReInit` để khởi tạo các trạng thái, khởi tạo giá trị của các timer hoặc khởi tạo các trạng thái khác cần thiết khi máy trạng thái chuyển tiếp.
- Sử dụng hàm `fsmProcessing` là hàm chính để quản lý máy trạng thái.

3.5.1 Hiện thực

```
#ifndef INC_INPUT_BUTTON_H_
#define INC_INPUT_BUTTON_H_

#include "main.h"
#include "software_timer.h"

#define BUTTON_NUMBER 4
#define BUTTON_PRESSED_DURATION 1000
#define BUTTON_HOLDING_DURATION 500
#define BUTTON_PRESSED GPIO_PIN_RESET
#define BUTTON_RELEASED GPIO_PIN_SET

void buttonReading(void);
int buttonPressed(int index);

#endif /* INC_INPUT_BUTTON_H_ */
```

Chương trình 2.3: Header file input_button.h

```
#include "input_button.h"

static uint16_t buttonPins[BUTTON_NUMBER] = {BUTTON_0_Pin,
   BUTTON_1_Pin, BUTTON_2_Pin, BUTTON_3_Pin};
static GPIO_PinState buttonStates[BUTTON_NUMBER];
static GPIO_PinState buttonDebounce0[BUTTON_NUMBER];
static GPIO_PinState buttonDebounce1[BUTTON_NUMBER];
static GPIO_PinState buttonDebounce2[BUTTON_NUMBER];
static uint8_t buttonFlags[BUTTON_NUMBER];
static int16_t buttonCounters[BUTTON_NUMBER];

void buttonReading(void) {
    for (int i = 0; i < BUTTON_NUMBER; i++) {
        buttonDebounce2[i] = buttonDebounce1[i];
        buttonDebounce1[i] = buttonDebounce0[i];
```

```
15          buttonDebounce0[i] = HAL_GPIO_ReadPin(GPIOB,
                buttonPins[i]);
16
17          if ((buttonDebounce0[i] == buttonDebounce1[i]) && (
                buttonDebounce0[i] == buttonDebounce2[i])) {
18              if (buttonStates[i] != buttonDebounce0[i]) {
19                  buttonStates[i] = buttonDebounce0[i];
20
21                  if (buttonStates[i] == BUTTON_PRESSED) {
22                      buttonFlags[i] = 1;
23                      buttonCounters[i] =
                            BUTTON_PRESSED_DURATION /
                            TIMER_DURATION;
24                  }
25              } else {
26                  if (buttonStates[i] == BUTTON_PRESSED) {
27                      buttonCounters[i]--;
28                      if (buttonCounters[i] <= 0) {
29                          buttonFlags[i] = 1;
30                          buttonCounters[i] =
                                BUTTON_HOLDING_DURATION /
                                TIMER_DURATION;
31                      }
32                  }
33              }
34          }
35      }
36  }
37
38  int buttonPressed(int index) {
39      if (index < 0 || index >= BUTTON_NUMBER) return 0;
40
41      if (buttonFlags[index] == 1) {
42          buttonFlags[index] = 0;
43          return 1;
44      }
45
46      return 0;
47  }
```

Chương trình 2.4: Source file input_button.c

```
1  #ifndef INC_PROCESSING_FSM_H_
2  #define INC_PROCESSING_FSM_H_
3
4  #include "input_button.h"
5
6  enum FSM_STATE {
```

```
7       FSM_NORMAL,
8       FSM_RED_MOD,
9       FSM_AMBER_MOD,
10      FSM_GREEN_MOD
11  };
12
13  void fsmInit(void);
14  void fsmReInit(enum FSM_STATE state);
15  void fsmProcessing(void);
16
17  #endif /* INC_PROCESSING_FSM_H_ */
```

Chương trình 2.5: Header file processing_fsm.h

```
1   #include "processing_fsm.h"
2
3   enum FSM_STATE fsmState = 0;
4
5   void fsmInit(void) {
6       fsmReInit(FSM_NORMAL);
7   }
8
9   void fsmReInit(enum FSM_STATE state) {
10      switch (state) {
11          case FSM_NORMAL:
12              fsmState = FSM_NORMAL;
13              break;
14          case FSM_RED_MOD:
15              fsmState = FSM_RED_MOD;
16              break;
17          case FSM_AMBER_MOD:
18              fsmState = FSM_AMBER_MOD;
19              break;
20          case FSM_GREEN_MOD:
21              fsmState = FSM_GREEN_MOD;
22              break;
23          default:
24              break;
25      }
26  }
27
28  void fsmProcessing(void) {
29      switch (fsmState) {
30          case FSM_NORMAL:
31              if (buttonPressed(0)) {
32                  fsmReInit(FSM_RED_MOD);
33              }
34              break;
```

```
35          case FSM_RED_MOD:
36              if (buttonPressed(0)) {
37                  fsmReInit(FSM_AMBER_MOD);
38              }
39              break;
40          case FSM_AMBER_MOD:
41              if (buttonPressed(0)) {
42                  fsmReInit(FSM_GREEN_MOD);
43              }
44              break;
45          case FSM_GREEN_MOD:
46              if (buttonPressed(0)) {
47                  fsmReInit(FSM_NORMAL);
48              }
49              break;
50          default:
51              break;
52      }
53  }
```

Chương trình 2.6: Source file processing_fsm.c

```
1  /* USER CODE BEGIN Includes */
2  #include "processing_fsm.h"
3  #include "software_timer.h"
4  /* USER CODE END Includes */
5
6  /* USER CODE BEGIN 2 */
7  HAL_TIM_Base_Start_IT(&htim2);
8  fsmInit();
9  /* USER CODE END 2 */
10
11 /* USER CODE BEGIN WHILE */
12 while (1) {
13     fsmProcessing();
14 }
15 /* USER CODE END WHILE */
16
17 /* USER CODE BEGIN 4 */
18 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
19     timerRun();
20     buttonReading();
21 }
22 /* USER CODE END 4 */
```

Chương trình 2.7: Source file main.c

3.6 Bài 6

3.6.1 Đặc tả

Đối với điều khiển các LED 7 đoạn, xác định:

- Trong toàn bộ project, các LED 7 đoạn được hiện thực theo từng khối, với 2 LED 7 đoạn trong mỗi khối nhằm hiển thị một số có 2 chữ số. Để thuận tiện cho việc cập nhật các giá trị, lưu các giá trị hiển thị cho từng khối LED 7 đoạn thay vì từng giá trị riêng biệt.
- Sử dụng hàm `led7segInit` để khởi tạo giá trị ban đầu cho các LED 7 đoạn.
- Sử dụng hàm `led7segScanning` để điều khiển quá trình quét LED 7 đoạn. Tần số của quá trình này được chọn bằng 1Hz, do đó thời gian chuyển đổi giữa các LED 7 đoạn là 250ms.
- Việc quét LED 7 đoạn được thực hiện hoàn toàn độc lập với máy trạng thái chính.

Đối với điều khiển các LED đơn, xác định:

- Các LED đơn không hoạt động độc lập mà hoạt động theo từng bộ đèn giao thông.
- Các trạng thái của một bộ đèn giao thông bao gồm `TRAFFIC_OFF` (tắt cả đều tắt), `TRAFFIC_RED` (đèn đỏ), `TRAFFIC_AMBER` (đèn vàng), `TRAFFIC_GREEN` (đèn xanh).
- Sử dụng hàm `trafficInit` để khởi tạo trạng thái ban đầu cho các LED đơn phù hợp với một trong số các trạng thái của bộ đèn giao thông.
- Sử dụng hàm `trafficReInit` để khởi tạo giá trị ban đầu cho các LED đơn khi bộ đèn giao thông thay đổi trạng thái.

```
1  #ifndef INC_OUTPUT_LED7SEG_H_
2  #define INC_OUTPUT_LED7SEG_H_
3
4  #include "main.h"
5  #include "math.h"
6  #include "software_timer.h"
7
8  #define LED7SEG_NUMBER 4
9  #define LED7SEG_BLOCK_NUMBER 2
10 #define LED7SEG_DIGIT_NUMBER (LED7SEG_NUMBER /
     LED7SEG_BLOCK_NUMBER)
11 #define LED7SEG_SCANNING_DURATION 250
12
13 extern int led7segNumbers[LED7SEG_BLOCK_NUMBER];
14
15 void led7segInit(void);
```

```
16 void led7segScanning(void);
17
18 #endif /* INC_OUTPUT_LED7SEG_H_ */
```

Chương trình 2.8: Header file output_led7seg.h

```
1  #include "output_led7seg.h"
2
3  int led7segNumbers[LED7SEG_BLOCK_NUMBER] = {0};
4  static int led7segIndex = 0;
5  static int led7segOffset = 0;
6  static uint16_t led7segPins[LED7SEG_NUMBER] = {LED7SEG_0_Pin,
       LED7SEG_1_Pin, LED7SEG_2_Pin, LED7SEG_3_Pin};
7
8  static uint16_t led7segNum0s[11] = {0xBF, 0x86, 0xDB, 0xCF, 0
     xE6, 0xED, 0xFD, 0x87, 0xFF, 0xEF, 0x00};
9  static uint16_t led7segNum1s[11] = {0x40, 0x79, 0x24, 0x30, 0
     x19, 0x12, 0x02, 0x78, 0x00, 0x10, 0xFF};
10
11 static void led7segDisplay(int index, int offset) {
12     if (index < 0 || index >= LED7SEG_BLOCK_NUMBER) return;
13     if (offset < 0 || offset >= LED7SEG_DIGIT_NUMBER) return;
14
15     int num = led7segNumbers[index] / (int)pow(10,
          LED7SEG_DIGIT_NUMBER - offset - 1) % 10;
16     if (num >= 0 && num <= 9) {
17         GPIOA->BSRR = led7segNum1s[num] << 8;
18         GPIOA->BSRR = led7segNum0s[num] << (8 + 16);
19     } else {
20         GPIOA->BSRR = led7segNum1s[10] << 8;
21         GPIOA->BSRR = led7segNum0s[10] << (8 + 16);
22     }
23
24     switch (index * LED7SEG_BLOCK_NUMBER + offset) {
25         case 0:
26             HAL_GPIO_WritePin(GPIOB, led7segPins[0],
                  GPIO_PIN_RESET);
27             HAL_GPIO_WritePin(GPIOB, led7segPins[1],
                  GPIO_PIN_SET);
28             HAL_GPIO_WritePin(GPIOB, led7segPins[2],
                  GPIO_PIN_SET);
29             HAL_GPIO_WritePin(GPIOB, led7segPins[3],
                  GPIO_PIN_SET);
30             break;
31         case 1:
32             HAL_GPIO_WritePin(GPIOB, led7segPins[0],
                  GPIO_PIN_SET);
```

```
33              HAL_GPIO_WritePin(GPIOB, led7segPins[1],
                    GPIO_PIN_RESET);
34              HAL_GPIO_WritePin(GPIOB, led7segPins[2],
                    GPIO_PIN_SET);
35              HAL_GPIO_WritePin(GPIOB, led7segPins[3],
                    GPIO_PIN_SET);
36              break;
37          case 2:
38              HAL_GPIO_WritePin(GPIOB, led7segPins[0],
                    GPIO_PIN_SET);
39              HAL_GPIO_WritePin(GPIOB, led7segPins[1],
                    GPIO_PIN_SET);
40              HAL_GPIO_WritePin(GPIOB, led7segPins[2],
                    GPIO_PIN_RESET);
41              HAL_GPIO_WritePin(GPIOB, led7segPins[3],
                    GPIO_PIN_SET);
42              break;
43          case 3:
44              HAL_GPIO_WritePin(GPIOB, led7segPins[0],
                    GPIO_PIN_SET);
45              HAL_GPIO_WritePin(GPIOB, led7segPins[1],
                    GPIO_PIN_SET);
46              HAL_GPIO_WritePin(GPIOB, led7segPins[2],
                    GPIO_PIN_SET);
47              HAL_GPIO_WritePin(GPIOB, led7segPins[3],
                    GPIO_PIN_RESET);
48              break;
49          default:
50              break;
51      }
52  }
53
54  void led7segInit(void) {
55      timerSet(LED7SEG_SCANNING_DURATION / TIMER_DURATION, 0);
56      HAL_GPIO_WritePin(GPIOB, led7segPins[0], GPIO_PIN_SET);
57      HAL_GPIO_WritePin(GPIOB, led7segPins[1], GPIO_PIN_SET);
58      HAL_GPIO_WritePin(GPIOB, led7segPins[2], GPIO_PIN_SET);
59      HAL_GPIO_WritePin(GPIOB, led7segPins[3], GPIO_PIN_SET);
60  }
61
62  void led7segUpdate(int num, int index) {
63      if (num < 0 || num >= pow(10, LED7SEG_DIGIT_NUMBER) ||
            index < 0 || index >= LED7SEG_BLOCK_NUMBER) return;
64      led7segNumbers[index] = num;
65  }
66
67  void led7segScanning(void) {
68      led7segDisplay(led7segIndex, led7segOffset);
```

```
69      if (timerFlags[0] == 1) {
70          timerSet(LED7SEG_SCANNING_DURATION / TIMER_DURATION,
                0);
71          led7segOffset++;
72          if (led7segOffset >= LED7SEG_DIGIT_NUMBER) {
73              led7segOffset = 0;
74              led7segIndex++;
75          }
76          if (led7segIndex >= LED7SEG_BLOCK_NUMBER) {
77              led7segIndex = 0;
78          }
79      }
80  }
```

Chương trình 2.9: Source file output_led7seg.c

```
1   #ifndef INC_OUTPUT_TRAFFIC_H_
2   #define INC_OUTPUT_TRAFFIC_H_
3
4   #include "main.h"
5
6   #define TRAFFIC_NUMBER 2
7   #define TRAFFIC_SECOND_DURATION 1000
8   #define TRAFFIC_BLINKING_DURATION 250
9
10  enum TRAFFIC_STATE {
11      TRAFFIC_OFF,
12      TRAFFIC_RED,
13      TRAFFIC_AMBER,
14      TRAFFIC_GREEN
15  };
16
17  extern int trafficRedDuration;
18  extern int trafficAmberDuration;
19  extern int trafficGreenDuration;
20  extern enum TRAFFIC_STATE trafficState[TRAFFIC_NUMBER];
21
22  void trafficInit(void);
23  void trafficReInit(enum TRAFFIC_STATE state, int index);
24
25  #endif /* INC_OUTPUT_TRAFFIC_H_ */
```

Chương trình 3.1: Header file output_traffic.h

```
1   #include "output_traffic.h"
2
3   int trafficRedDuration = 0;
```

```
4  int trafficAmberDuration = 0;
5  int trafficGreenDuration = 0;
6  enum TRAFFIC_STATE trafficState[TRAFFIC_NUMBER] = {0, 0};
7
8  static uint16_t trafficRedPins[TRAFFIC_NUMBER] = {
     TRAFFIC0_RED_Pin, TRAFFIC1_RED_Pin};
9  static uint16_t trafficAmberPins[TRAFFIC_NUMBER] = {
     TRAFFIC0_AMBER_Pin, TRAFFIC1_AMBER_Pin};
10 static uint16_t trafficGreenPins[TRAFFIC_NUMBER] = {
     TRAFFIC0_GREEN_Pin, TRAFFIC1_GREEN_Pin};
11
12 void trafficInit(void) {
13     trafficRedDuration = 5;
14     trafficAmberDuration = 2;
15     trafficGreenDuration = 3;
16     trafficReInit(TRAFFIC_OFF, 0);
17     trafficReInit(TRAFFIC_OFF, 1);
18 }
19
20 void trafficReInit(enum TRAFFIC_STATE state, int index) {
21     if (index < 0 || index >= TRAFFIC_NUMBER) return;
22
23     switch (state) {
24         case TRAFFIC_OFF:
25             HAL_GPIO_WritePin(GPIOB, trafficRedPins[index],
                 GPIO_PIN_SET);
26             HAL_GPIO_WritePin(GPIOB, trafficAmberPins[index],
                   GPIO_PIN_SET);
27             HAL_GPIO_WritePin(GPIOB, trafficGreenPins[index],
                   GPIO_PIN_SET);
28             trafficState[index] = TRAFFIC_OFF;
29             break;
30
31         case TRAFFIC_RED:
32             HAL_GPIO_WritePin(GPIOB, trafficRedPins[index],
                 GPIO_PIN_RESET);
33             HAL_GPIO_WritePin(GPIOB, trafficAmberPins[index],
                   GPIO_PIN_SET);
34             HAL_GPIO_WritePin(GPIOB, trafficGreenPins[index],
                   GPIO_PIN_SET);
35             trafficState[index] = TRAFFIC_RED;
36             break;
37
38         case TRAFFIC_AMBER:
39             HAL_GPIO_WritePin(GPIOB, trafficRedPins[index],
                 GPIO_PIN_SET);
40             HAL_GPIO_WritePin(GPIOB, trafficAmberPins[index],
                   GPIO_PIN_RESET);
```

```
41              HAL_GPIO_WritePin(GPIOB, trafficGreenPins[index],
                    GPIO_PIN_SET);
42              trafficState[index] = TRAFFIC_AMBER;
43              break;
44
45          case TRAFFIC_GREEN:
46              HAL_GPIO_WritePin(GPIOB, trafficRedPins[index],
                   GPIO_PIN_SET);
47              HAL_GPIO_WritePin(GPIOB, trafficAmberPins[index],
                    GPIO_PIN_SET);
48              HAL_GPIO_WritePin(GPIOB, trafficGreenPins[index],
                    GPIO_PIN_RESET);
49              trafficState[index] = TRAFFIC_GREEN;
50              break;
51
52          default:
53              break;
54      }
55  }
```

Chương trình 3.2: Source file output_traffic.c

```
1  #ifndef INC_PROCESSING_FSM_H_
2  #define INC_PROCESSING_FSM_H_
3
4  #include "input_button.h"
5  #include "output_traffic.h"
6  #include "output_led7seg.h"
7
8  enum FSM_STATE {
9      FSM_NORMAL,
10     FSM_RED_MOD,
11     FSM_AMBER_MOD,
12     FSM_GREEN_MOD
13 };
14
15 void fsmInit(void);
16 void fsmReInit(enum FSM_STATE state);
17 void fsmProcessing(void);
18
19 #endif /* INC_PROCESSING_FSM_H_ */
```

Chương trình 3.3: Header file proccesing_fsm.h

```
1  #include "processing_fsm.h"
2
3  enum FSM_STATE fsmState = 0;
```

```
4
5  void fsmInit(void) {
6      trafficInit();
7      fsmReInit(FSM_NORMAL);
8  }
9
10 void fsmReInit(enum FSM_STATE state) {
11     switch (state) {
12         case FSM_NORMAL:
13             timerSet(TRAFFIC_SECOND_DURATION / TIMER_DURATION
                   , 1);
14             timerSet(trafficRedDuration *
                   TRAFFIC_SECOND_DURATION / TIMER_DURATION, 2);
15             timerSet(trafficGreenDuration *
                   TRAFFIC_SECOND_DURATION / TIMER_DURATION, 3);
16             led7segNumbers[0] = trafficRedDuration;
17             led7segNumbers[1] = trafficGreenDuration;
18             trafficReInit(TRAFFIC_RED, 0);
19             trafficReInit(TRAFFIC_GREEN, 1);
20             fsmState = FSM_NORMAL;
21             break;
22
23         case FSM_RED_MOD:
24             timerSet(TRAFFIC_BLINKING_DURATION /
                   TIMER_DURATION, 1);
25             led7segNumbers[0] = 2;
26             led7segNumbers[1] = trafficRedDuration;
27             trafficReInit(TRAFFIC_OFF, 0);
28             trafficReInit(TRAFFIC_OFF, 1);
29             fsmState = FSM_RED_MOD;
30             break;
31
32         case FSM_AMBER_MOD:
33             timerSet(TRAFFIC_BLINKING_DURATION /
                   TIMER_DURATION, 1);
34             led7segNumbers[0] = 3;
35             led7segNumbers[1] = trafficAmberDuration;
36             trafficReInit(TRAFFIC_OFF, 0);
37             trafficReInit(TRAFFIC_OFF, 1);
38             fsmState = FSM_AMBER_MOD;
39             break;
40
41         case FSM_GREEN_MOD:
42             timerSet(TRAFFIC_BLINKING_DURATION /
                   TIMER_DURATION, 1);
43             led7segNumbers[0] = 4;
44             led7segNumbers[1] = trafficGreenDuration;
45             trafficReInit(TRAFFIC_OFF, 0);
```

```
46              trafficReInit(TRAFFIC_OFF, 1);
47              fsmState = FSM_GREEN_MOD;
48              break;
49
50          default:
51              break;
52      }
53  }
54
55  void fsmProcessing(void) {
56      switch (fsmState) {
57          case FSM_NORMAL:
58              if (timerFlags[1] == 1) {
59                  timerSet(TRAFFIC_SECOND_DURATION /
                        TIMER_DURATION, 1);
60                  led7segNumbers[0]--;
61                  if (led7segNumbers[0] < 0) led7segNumbers[0]
                        = 0;
62                  led7segNumbers[1]--;
63                  if (led7segNumbers[1] < 0) led7segNumbers[1]
                        = 0;
64              }
65
66              if (timerFlags[2] == 1) {
67                  switch (trafficState[0]) {
68                      case TRAFFIC_RED:
69                          timerSet(trafficGreenDuration *
                                TRAFFIC_SECOND_DURATION /
                                TIMER_DURATION, 2);
70                          led7segNumbers[0] =
                                trafficGreenDuration;
71                          trafficReInit(TRAFFIC_GREEN, 0);
72                          break;
73
74                      case TRAFFIC_AMBER:
75                          timerSet(trafficRedDuration *
                                TRAFFIC_SECOND_DURATION /
                                TIMER_DURATION, 2);
76                          led7segNumbers[0] =
                                trafficRedDuration;
77                          trafficReInit(TRAFFIC_RED, 0);
78                          break;
79
80                      case TRAFFIC_GREEN:
81                          timerSet(trafficAmberDuration *
                                TRAFFIC_SECOND_DURATION /
                                TIMER_DURATION, 2);
82                          led7segNumbers[0] =
```

```
                        trafficAmberDuration;
83                  trafficReInit(TRAFFIC_AMBER, 0);
84                  break;
85
86              default:
87                  break;
88          }
89      }
90
91      if (timerFlags[3] == 1) {
92          switch (trafficState[1]) {
93              case TRAFFIC_RED:
94                  timerSet(trafficGreenDuration *
                        TRAFFIC_SECOND_DURATION /
                        TIMER_DURATION, 3);
95                  led7segNumbers[1] =
                        trafficGreenDuration;
96                  trafficReInit(TRAFFIC_GREEN, 1);
97                  break;
98
99              case TRAFFIC_AMBER:
100                 timerSet(trafficRedDuration *
                        TRAFFIC_SECOND_DURATION /
                        TIMER_DURATION, 3);
101                 led7segNumbers[1] =
                        trafficRedDuration;
102                 trafficReInit(TRAFFIC_RED, 1);
103                 break;
104
105             case TRAFFIC_GREEN:
106                 timerSet(trafficAmberDuration *
                        TRAFFIC_SECOND_DURATION /
                        TIMER_DURATION, 3);
107                 led7segNumbers[1] =
                        trafficAmberDuration;
108                 trafficReInit(TRAFFIC_AMBER, 1);
109                 break;
110
111             default:
112                 break;
113         }
114     }
115
116     if (buttonPressed(0)) {
117         fsmReInit(FSM_RED_MOD);
118     }
119     break;
120
```

```
121         case FSM_RED_MOD:
122             if (timerFlags[1] == 1) {
123                 switch (trafficState[0]) {
124                     case TRAFFIC_OFF:
125                         timerSet(TRAFFIC_BLINKING_DURATION /
                               TIMER_DURATION, 1);
126                         trafficReInit(TRAFFIC_RED, 0);
127                         trafficReInit(TRAFFIC_RED, 1);
128                         break;
129 
130                     case TRAFFIC_RED:
131                         timerSet(TRAFFIC_BLINKING_DURATION /
                               TIMER_DURATION, 1);
132                         trafficReInit(TRAFFIC_OFF, 0);
133                         trafficReInit(TRAFFIC_OFF, 1);
134                         break;
135 
136                     default:
137                         break;
138                 }
139             }
140 
141             if (buttonPressed(0)) {
142                 fsmReInit(FSM_AMBER_MOD);
143             }
144             break;
145 
146         case FSM_AMBER_MOD:
147             if (timerFlags[1] == 1) {
148                 switch (trafficState[0]) {
149                     case TRAFFIC_OFF:
150                         timerSet(TRAFFIC_BLINKING_DURATION /
                               TIMER_DURATION, 1);
151                         trafficReInit(TRAFFIC_AMBER, 0);
152                         trafficReInit(TRAFFIC_AMBER, 1);
153                         break;
154 
155                     case TRAFFIC_AMBER:
156                         timerSet(TRAFFIC_BLINKING_DURATION /
                               TIMER_DURATION, 1);
157                         trafficReInit(TRAFFIC_OFF, 0);
158                         trafficReInit(TRAFFIC_OFF, 1);
159                         break;
160 
161                     default:
162                         break;
163                 }
164             }
```

```
165
166             if (buttonPressed(0)) {
167                 fsmReInit(FSM_GREEN_MOD);
168             }
169             break;
170
171         case FSM_GREEN_MOD:
172             if (timerFlags[1] == 1) {
173                 switch (trafficState[0]) {
174                     case TRAFFIC_OFF:
175                         timerSet(TRAFFIC_BLINKING_DURATION /
                               TIMER_DURATION, 1);
176                         trafficReInit(TRAFFIC_GREEN, 0);
177                         trafficReInit(TRAFFIC_GREEN, 1);
178                         break;
179
180                     case TRAFFIC_GREEN:
181                         timerSet(TRAFFIC_BLINKING_DURATION /
                               TIMER_DURATION, 1);
182                         trafficReInit(TRAFFIC_OFF, 0);
183                         trafficReInit(TRAFFIC_OFF, 1);
184                         break;
185
186                     default:
187                         break;
188                 }
189             }
190
191             if (buttonPressed(0)) {
192                 fsmReInit(FSM_NORMAL);
193             }
194             break;
195
196         default:
197             break;
198     }
199 }
```

Chương trình 3.4: Source file processing_fsm.c

```
1 /* USER CODE BEGIN Includes */
2 #include "processing_fsm.h"
3 #include "software_timer.h"
4 /* USER CODE END Includes */
5
6 /* USER CODE BEGIN 2 */
7 HAL_TIM_Base_Start_IT(&htim2);
8 led7segInit();
```

```
fsmInit();
/* USER CODE END 2 */

/* USER CODE BEGIN WHILE */
while (1) {
    led7segScanning();
    fsmProcessing();
/* USER CODE END WHILE */
}
```

Chương trình 3.5: Source file main.c

3.7 Bài 7

3.7.1 Đặt tả

Từ máy trạng thái tự động ta xây dựng máy trạng thái dành cho MAN, ta sẽ bổ sung thêm tính năng trên tương ứng với thêm các luồng thực thi mới vào **Red Modify Mode**. Do đó, chỉ cần bổ sung luồng thực thi đối với `FSM_RED_MAN`. trong source file `manual_fsm.c`.

3.7.2 Hiện thực

```
case FSM_RED_MAN:
    if (timerFlags[1] == 1) {
        switch (trafficState[0]) {
            case TRAFFIC_OFF:
                timerSet(TRAFFIC_BLINKING_DURATION /
                   TIMER_DURATION, 1);
                trafficReInit(TRAFFIC_RED, 0);
                trafficReInit(TRAFFIC_RED, 1);
                break;

            case TRAFFIC_RED:
                timerSet(TRAFFIC_BLINKING_DURATION /
                   TIMER_DURATION, 1);
                trafficReInit(TRAFFIC_OFF, 0);
                trafficReInit(TRAFFIC_OFF, 1);
                break;

            default:
                break;
        }
    }

    if (buttonPressed(0)) {
        fsmReInitMan(FSM_AMBER_MAN);
    }
```

```
24
25      if (buttonPressed(1)) {
26          led7segNumbers[1]++;
27          if (led7segNumbers[1] >= pow(10, LED7SEG_DIGIT_NUMBER
                )) {
28              led7segNumbers[1] = 0;
29          }
30      }
31
32      if (buttonPressed(2)) {
33          led7segNumbers[1]--;
34          if (led7segNumbers[1] < 0) {
35              led7segNumbers[1] = pow(10, LED7SEG_DIGIT_NUMBER)
                    - 1;
36          }
37      }
38
39      if (buttonPressed(3)) {
40          trafficRedDuration = led7segNumbers[1];
41      }
42      break;
```

Chương trình 3.6: FSM_RED_MAN sau khi cập nhật

3.8 Bài 8

3.8.1 Đặt tả

Từ máy trạng thái tự động ta xây dựng máy trạng thái dành cho MAN, ta sẽ bổ sung thêm tính năng trên tương ứng với thêm các luồng thực thi mới vào **Amber Modify Mode**. Do đó, chỉ cần bổ sung luồng thực thi đối với `FSM_AMBER_MAN`. trong source file `manual_fsm.c`.

3.8.2 Hiện thực

```
1  case FSM_AMBER_MAN:
2      if (timerFlags[1] == 1) {
3          switch (trafficState[0]) {
4              case TRAFFIC_OFF:
5                  timerSet(TRAFFIC_BLINKING_DURATION /
                       TIMER_DURATION, 1);
6                  trafficReInit(TRAFFIC_AMBER, 0);
7                  trafficReInit(TRAFFIC_AMBER, 1);
8                  break;
9
10             case TRAFFIC_AMBER:
```

```
11                  timerSet(TRAFFIC_BLINKING_DURATION /
                        TIMER_DURATION, 1);
12                  trafficReInit(TRAFFIC_OFF, 0);
13                  trafficReInit(TRAFFIC_OFF, 1);
14                  break;
15
16              default:
17                  break;
18          }
19      }
20
21      if (buttonPressed(0)) {
22          fsmReInitMan(FSM_GREEN_MAN);
23      }
24
25      if (buttonPressed(1)) {
26          led7segNumbers[1]++;
27          if (led7segNumbers[1] >= pow(10, LED7SEG_DIGIT_NUMBER
                )) {
28              led7segNumbers[1] = 0;
29          }
30      }
31
32      if (buttonPressed(2)) {
33          led7segNumbers[1]--;
34          if (led7segNumbers[1] < 0) {
35              led7segNumbers[1] = pow(10, LED7SEG_DIGIT_NUMBER)
                    - 1;
36          }
37      }
38
39      if (buttonPressed(3)) {
40          trafficAmberDuration = led7segNumbers[1];
41      }
42      break;
```

Chương trình 3.7: : FSM_AMBER_MAN sau khi cập nhật

3.9 Bài 9

3.9.1 Đặt tả

Từ máy trạng thái từ động ta xây dựng máy trạng thái dành cho MAN, ta sẽ bổ sung thêm tính năng trên tương ứng với thêm các luồng thực thi mới vào **Amber Modify Mode**. Do đó, chỉ cần bổ sung luồng thực thi đối với `FSM_GREEN_MAN`. trong source file `manual_fsm.c`.

3.9.2 Hiện thực

3.9.3 Source code

```
case FSM_GREEN_MAN:
    if (timerFlags[1] == 1) {
        switch (trafficState[0]) {
            case TRAFFIC_OFF:
                timerSet(TRAFFIC_BLINKING_DURATION /
                   TIMER_DURATION, 1);
                trafficReInit(TRAFFIC_GREEN, 0);
                trafficReInit(TRAFFIC_GREEN, 1);
                break;

            case TRAFFIC_GREEN:
                timerSet(TRAFFIC_BLINKING_DURATION /
                   TIMER_DURATION, 1);
                trafficReInit(TRAFFIC_OFF, 0);
                trafficReInit(TRAFFIC_OFF, 1);
                break;

            default:
                break;
        }
    }

    if (buttonPressed(0)) {
        fsmReInitMan(FSM_NORMAL_MAN);
    }

    if (buttonPressed(1)) {
        led7segNumbers[1]++;
        if (led7segNumbers[1] >= pow(10, LED7SEG_DIGIT_NUMBER
           )) {
            led7segNumbers[1] = 0;
        }
    }

    if (buttonPressed(2)) {
        led7segNumbers[1]--;
        if (led7segNumbers[1] < 0) {
            led7segNumbers[1] = pow(10, LED7SEG_DIGIT_NUMBER)
                - 1;
        }
    }

    if (buttonPressed(3)) {
        trafficGreenDuration = led7segNumbers[1];
```

```
41      }
42      break;
```

Chương trình 3.8: : FSM_GREEN_MAN sau khi cập nhật

3.10 Bài tập 10: Hoàn thành dự án