

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



Vì xử lí - vi điều khiển

Báo cáo project

Đèn giao thông cho người đi bộ

Advisor(s): Nguyễn Trọng Nhân

Student(s): Nguyễn Đăng Khoa 2113760

HO CHI MINH CITY, DECEMBER 2023



Contents

1 Finite State Machine	4
2 Components need for project	5
3 File Layout	6
4 Scheduler	6
4.1 SCH_Update	6
4.2 Enqueue_Task	7
4.3 SCH_Add_Task	8
4.4 SCH_Dispatch_Tasks	9
4.5 SCH_Delete_Task	9
5 Traffic Light Display	11
6 Define FSM	12
7 FSM	13
7.1 runStateFSM	13
7.2 trafficLightFSM	14
7.3 pedestrianStateFSM	15
7.4 Button	17
8 In main.c	24
9 Repository and Diagram Link	25

1 Finite State Machine

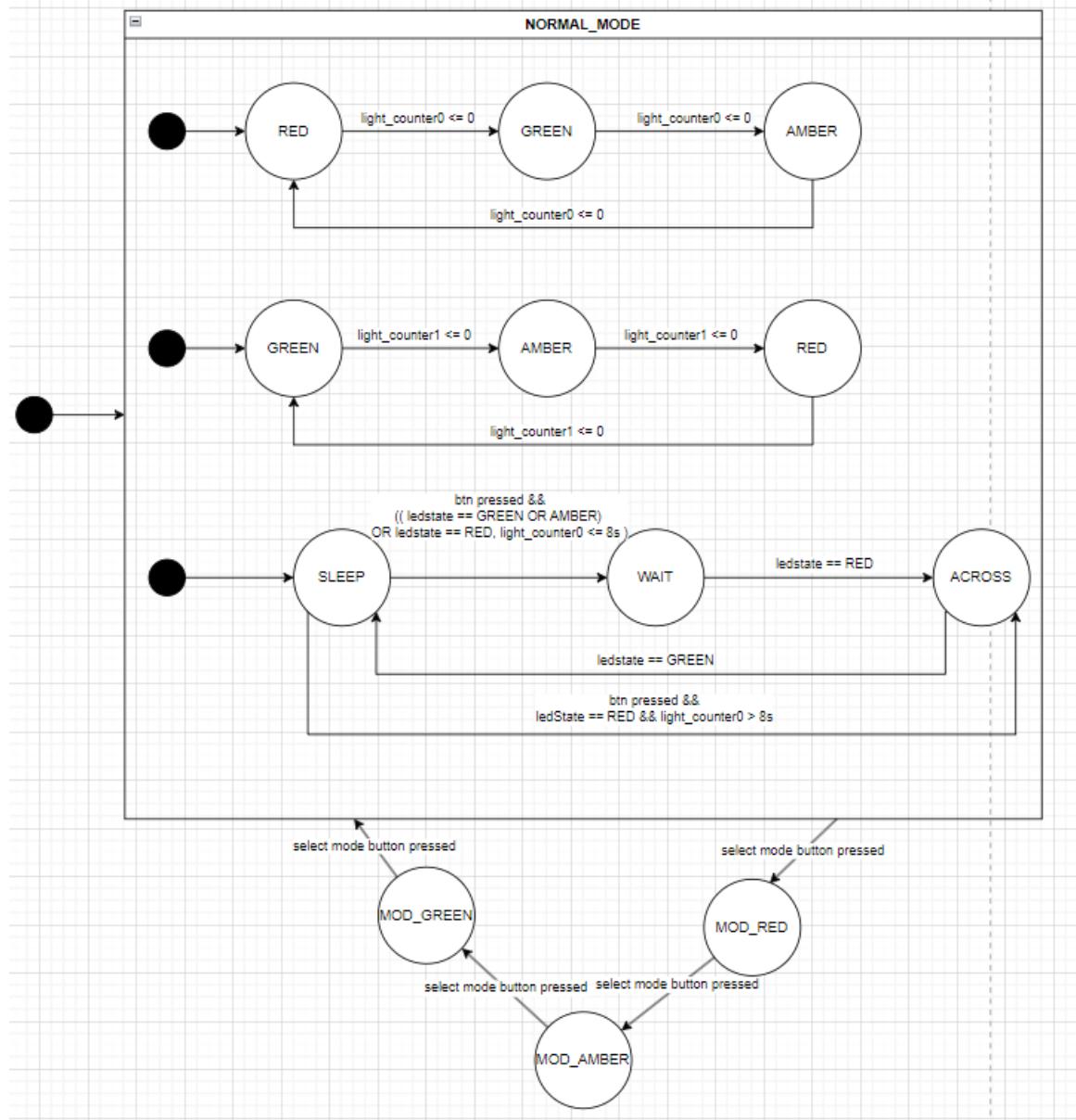


Figure 1.1: FSM RunState

Trong chế độ NORMAL MODE thì máy trạng thái ở trên sẽ chạy đồng thời ba máy trạng thái con bao gồm hai chiều của đèn giao thông và một trạng thái cho người đi bộ. Khi thoát ra trạng thái NORMAL MODE thì 3 máy trạng thái đó sẽ ngưng, và sẽ khởi động lại khi từ MOD_GREEN chuyển về lại NORMAL_MODE.

Trong NORMAL_MODE, hai máy trạng thái ở trên là cho hai chiều đường giao thông, máy trạng thái dưới cùng là dành cho người đi bộ



Máy trạng thái giao thông khá đơn giản, chỉ là đếm hết giờ rồi chuyển trạng thái.

Đối với máy trạng thái dành cho người đi bộ, chúng ta có 3 trạng thái SLEEP - khi chưa có người đi bộ nào bấm nút, WAIT - bấm nút nhưng đèn giao thông chưa chuyển sang RED, ACROSS - được phép băng qua.

Đối với máy trạng thái dành cho người đi bộ, phần lưu ý nhất là khi pState = SLEEP nếu nút được nhấn mà đèn đang GREEN hoặc AMBER hoặc (RED và counter dưới 8 giây) thì pState chuyển thành WAIT. Ngược lại nếu RED và counter trên 8 giây thì pState = ACROSS. Còn các sự kiện chuyển trạng thái còn lại khá rõ ràng.

2 Components need for project

Những linh kiện cần cho project này:

- STM32RB103
- 4 Nút nhấn
- 2 Đèn giao thông hiển thị được 3 màu. Điều khiển bằng 2 chân.
- 1 Đèn LED xanh và 1 Đèn LED đỏ báo tín hiệu cho người đi bộ.
- 1 Buzzer báo thời gian cho người đi bộ.
- 1 màn hình LCD 16x02 để trả về thời gian.

Bốn nút nhất sẽ có tên là:

- **SEL_BTN:** Chuyển MODE (NORMAL_MODE, Điều chỉnh đèn đỏ,...).
- **MOD_BTN:** Thay đổi giá trị đếm, khi nhấn đè sẽ tự động tăng 1 / 100ms.
- **SET_BTN:** Xác nhận thay đổi giá trị.
- **P_BTN:** Nút bấm xin đường cho người đi bộ.

Tín hiệu cho đèn giao thông:

- **RED:** Signal A = 1, Signal B = 0. Kết hợp thì sẽ là b'10 = 2
- **AMBER:** Signal A = 1, Signal B = 1. Kết hợp thì sẽ là b'11 = 3
- **GREEN:** Signal A = 0, Signal B = 1. Kết hợp thì sẽ là b'01 = 1



3 File Layout

- **global.h** Gồm các DEFINE và khai báo biến tạo máy trạng thái.
- **i2c-lcd.h** và **i2c-lcd.c** Interface giao tiếp với LCD 16x02 thông qua I2C.
- **input_processing.h** và **input_processing.c**: Phần xử lý chính, xử lý các máy trạng thái, in dữ liệu ra,...
- **input_reading.h** và **input_reading.c**: Interface đọc dữ liệu nút nhấn.
- **scheduler.h** và **scheduler.c**: Interface cho scheduler
- **traffic_light.h** và **traffic_light.c**: Interface cho điều khiển LED
- **main.h** và **main.c**

Chúng ta sẽ giới thiệu những phần chính, quan trọng. Nếu muốn tìm hiểu thêm về code, hãy bấm vào link github ở dưới cùng của bài viết

4 Scheduler

4.1 SCH_Update

Listing 4.1: SCH_Update

```
1 void SCH_Update(void){  
2     if (head_task == NULL) {  
3         /*  
4             * This is for situation, you just have one task run.  
5             * When task ran completely, it will dequeue, but at this  
6                 time  
7             * SCH_Update can be run, and if we not add 1 for  
8                 time_skip  
9             * instead of do nothing, your task will run wrong time.  
10            */  
11        time_skip = (count_task > 0) ? time_skip + 1 : 0;  
12        return;  
13    }  
14    time_skip++;
```



```

13     if (head_task->Delay > 0) {
14         int temp = head_task->Delay - time_skip;
15         if (temp >= 0) {
16             head_task->Delay = temp;
17             time_skip = 0;
18         } else {
19             head_task->Delay = 0;
20             time_skip = 0 - temp;
21         }
22     }
23 }
```

Đây là hàm SCH_Update O(1), điểm lưu ý ở đây là biến **time_skip**. Biến này dùng trong những trường hợp có một số task chạy quá lâu dẫn đến sai lệch. Nói cách khác thì khi delay = 0 thì task sẽ chạy nhưng nếu chạy quá lâu thì SCH_Update sẽ không trừ thêm delay được vì nó đã bằng 0 rồi.

4.2 Enqueue_Task

Listing 4.2: Enqueue_Task

```

24 sTask* Enqueue_Task(sTask* newTask){
25     //First task in queue.
26     if (head_task == NULL){
27         head_task = newTask;
28         return newTask;
29     }
30
31     //Task will add at middle of queue.
32     sTask *ini = head_task;
33     sTask *pre = NULL;
34     uint32_t sum = 0;
35     while (ini != NULL){
36         if (sum + ini->Delay > newTask->Delay){
37             if (ini == head_task){
38                 newTask->Next = head_task;
39                 head_task = newTask;
```



```
40         ini->Delay -= newTask->Delay;
41     } else {
42         newTask->Next = ini;
43         pre->Next = newTask;
44         newTask->Delay -= sum;
45         ini->Delay -= newTask->Delay;
46     }
47     return newTask;
48 }
49 sum += ini->Delay;
50 pre = ini;
51 ini = ini->Next;
52 }

//Task will add of tail
54 if (ini == NULL){
55     pre->Next = newTask;
56     newTask->Delay -= sum;
57 }
58

59
60 return newTask;
61 }
```

Hàm này sẽ sắp task vào hàng đợi, nó sẽ tính toán lại delay cho đúng với quy tắc.

4.3 SCH_Add_Task

Listing 4.3: SCH_Add_Task

```
62 sTask* SCH_Add_Task(void (*pFunc)(void), uint32_t Delay,
63                     uint32_t Period){
64     sTask* newTask = Create_Task(pFunc, Delay, Period);
65     if (newTask == NULL){
66         return NULL;
67     }
68     return Enqueue_Task(newTask);
69 }
```



Chỉ đơn giản là tạo task bằng cách dùng hàm Create_Task (chi tiết trong code trên github) và sắp task vào hàng đợi

4.4 SCH_Dispatch_Tasks

Listing 4.4: SCH_Dispatch_Task

```
69 uint8_t SCH_Dispatch_Tasks(void){  
70     if (head_task == NULL || head_task->Delay > 0) return 0;  
71  
72     // Remove head task out of queue (dequeue, not delete)  
73     // and config delay then enqueue it again.  
74     sTask* runningTask = head_task;  
75     head_task = head_task->Next;  
76  
77     runningTask->Next      = NULL;  
78     runningTask->Delay    = runningTask->Period;  
79  
80     // Run task  
81     runningTask->pFunc();  
82     // Save the task just ran  
83     TaskIdJustRun = runningTask->TaskID;  
84  
85     // Add again if a task has period value isn't equal to 0.  
86     if (runningTask->Period != 0) Enqueue_Task(runningTask);  
87     else free(runningTask);  
88  
89     return 1;  
90 }
```

4.5 SCH_Delete_Task

Listing 4.5: SCH_Delete_Task

```
91 uint8_t SCH_Delete_Task(sTask* deleted_task){  
92     if (deleted_task == NULL){  
93         errorCode = ERROR_SCH_DELETE_NULL_TASK;
```



```
94     return 0;
95 }
96 if (deleted_task == head_task){
97     count_task--;
98     head_task = head_task->Next;
99     deleted_task->Next->Delay += deleted_task->Delay;
100    free(deleted_task);
101    return 1;
102 }
103
104 // Find previous of deleted task
105 sTask* pre = NULL;
106 sTask* ini = head_task;
107 while (ini != deleted_task && ini != NULL){
108     pre = ini;
109     ini = ini->Next;
110 }
111
112 if (ini == NULL){
113     // Task not found
114     errorCode = ERROR_SCH_DELETE_NULL_TASK;
115     return 0;
116 }
117
118 if (deleted_task->Next != NULL){
119     deleted_task->Next->Delay += deleted_task->Delay;
120 }
121 if (pre != NULL){
122     pre->Next = deleted_task->Next;
123 }
124 free(deleted_task);
125 return 1;
126 }
```



5 Traffic Light Display

```

127 GPIO_TypeDef* TL_GPIO[DIMENSION * 2] = {
128     TL1_A_GPIO_Port, TL1_B_GPIO_Port,
129     TL2_A_GPIO_Port, TL2_B_GPIO_Port
130 } ;
131
132 const short TL_Pin[DIMENSION * 2] = {
133     TL1_A_Pin, TL1_B_Pin,
134     TL2_A_Pin, TL2_B_Pin
135 } ;
136
137 void resetAllLED(void){
138     HAL_GPIO_WritePin(TL_GPIO[0], TL_Pin[0], RESET);
139     HAL_GPIO_WritePin(TL_GPIO[1], TL_Pin[1], RESET);
140     HAL_GPIO_WritePin(TL_GPIO[2], TL_Pin[2], RESET);
141     HAL_GPIO_WritePin(TL_GPIO[3], TL_Pin[3], RESET);
142 }
143
144 void WritePinLED(const short index, uint8_t state){
145     HAL_GPIO_WritePin(TL_GPIO[index * 2 + 1], TL_Pin[index * 2
146         + 1], state & 1);
147     state >>= 1;
148     HAL_GPIO_WritePin(TL_GPIO[index * 2], TL_Pin[index * 2],
149         state & 1);
150 }
151
152 //Depend on `state` parameter. If RED, blinking RED, so on.
153 void TogglePinLED(uint8_t state){
154     switch(state){
155     case RED:
156         HAL_GPIO_TogglePin(TL1_A_GPIO_Port, TL1_A_Pin);
157         HAL_GPIO_TogglePin(TL2_A_GPIO_Port, TL2_A_Pin);
158         break;
159     case AMBER:
160         HAL_GPIO_TogglePin(TL1_A_GPIO_Port, TL1_A_Pin);
161     }
162 }
```



```
159     HAL_GPIO_TogglePin(TL1_B_GPIO_Port, TL1_B_Pin);  
160  
161     HAL_GPIO_TogglePin(TL2_A_GPIO_Port, TL2_A_Pin);  
162     HAL_GPIO_TogglePin(TL2_B_GPIO_Port, TL2_B_Pin);  
163     break;  
164 case GREEN:  
165     HAL_GPIO_TogglePin(TL1_B_GPIO_Port, TL1_B_Pin);  
166     HAL_GPIO_TogglePin(TL2_B_GPIO_Port, TL2_B_Pin);  
167     break;  
168 }  
169 }
```

Hàm **resetAllLED()** sẽ tắt toàn bộ LED của đèn giao thông. Hàm **WritePinLED** sẽ nhận index (chiều của đèn giao thông, 0 là dọc, 1 là ngang) và nhận STATE (RED, AMBER, GREEN) hiển thị đèn màu tương ứng với trạng thái của nó. Hàm **TogglePinLED** sẽ nhận state như trên, hàm này dùng để nhấp nháy đèn cho cả hai chiều đèn giao thông.

6 Define FSM

```
170 typedef enum {  
171     RELEASED,  
172     PRESSED,  
173     PRESSED_MORE_THAN_ONE_SECOND  
174 } ButtonState;  
175  
176 typedef enum {  
177     NORMAL_MODE = 1,  
178     MODIFY_DURATION_RED_MODE = 2,  
179     MODIFY_DURATION_AMBER_MODE = 3,  
180     MODIFY_DURATION_GREEN_MODE = 4  
181 } RunState;  
182  
183 typedef enum {  
184     SLEEP,  
185     WAIT,  
186     ACROSS
```



```
187 } PedestrianState;  
  
188  
189 typedef enum {  
190     RED = 2,  
191     AMBER = 3,  
192     GREEN = 1  
193 } LEDState;
```

7 FSM

7.1 runStateFSM

```
194 void runStateFSM(void) {  
195     switch (runState) {  
196         case NORMAL_MODE:  
197             //Run 2 traffic light FSMs.  
198             trafficLightFSM(VERTICAL);  
199             trafficLightFSM(HORIZONTAL);  
200             pedestrianStateFSM();  
201             displayPLED();  
202             break;  
203         case MODIFY_DURATION_RED_MODE:  
204             //Blinking Red LED in 0.5s  
205             blinking_counter--;  
206             if (blinking_counter <= 0) {  
207                 blinking_counter = HALF_SECOND;  
208                 TogglePinLED(RED);  
209             }  
210             break;  
211         case MODIFY_DURATION_AMBER_MODE:  
212             //Blinking Amber LED in 0.5s  
213             blinking_counter--;  
214             if (blinking_counter <= 0) {  
215                 blinking_counter = HALF_SECOND;  
216                 TogglePinLED(AMBER);
```

```
217     }
218     break;
219 case MODIFY_DURATION_GREEN_MODE:
220     //Blinking Green LED in 0.5s
221     blinking_counter--;
222     if (blinking_counter <= 0) {
223         blinking_counter = HALF_SECOND;
224         TogglePinLED(GREEN);
225     }
226     break;
227 default:
228     runState = NORMAL_MODE;
229 }
230 }
```

Ánh xạ từ FSM thiết kế ở trên, ta thấy trong NORMAL_MODE có 3 máy trạng thái chạy song song với nhau, **displayPLED()** là hàm hiển thị đèn dựa vào trạng thái dành cho người đi bộ (SLEEP thì hiển đỏ, WAIT nhấp nháy đỏ, ACROSS là màu xanh). Các trạng thái chỉnh sửa thời gian đèn RED, AMBER, GREEN thì sẽ nhấp nháy đèn mỗi 0.5 giây.

7.2 trafficLightFSM

```
231 void trafficLightFSM(const short index){
232     // Display duration via UART.
233     displayingDuration(index);
234     light_counter[index]--;
235     switch (ledState[index]){
236     case RED:
237         //Display only Red LED.
238         WritePinLED(index, RED);
239         //After amount of time, it will switch to Green.
240         if (light_counter[index] <= 0) {
241             light_counter[index] = durationGreen * ONE_SECOND;
242             ledState[index] = GREEN;
243         }
244     }
```



```

244     break;
245 case AMBER:
246     //Display only Amber LED.
247     WritePinLED(index, AMBER);
248     //After amount of time, it will switch to Red.
249     if (light_counter[index] <= 0) {
250         light_counter[index] = durationRed * ONE_SECOND;
251         ledState[index] = RED;
252     }
253
254     break;
255 case GREEN:
256     //Display only Green LED.
257     WritePinLED(index, GREEN);
258     //After amount of time, it will switch to Amber.
259     if (light_counter[index] <= 0) {
260         light_counter[index] = durationAmber * ONE_SECOND;
261         ledState[index] = AMBER;
262     }
263     break;
264 }
265 }
```

Hàm **displayingDuration()** là hàm dùng cho hiển thị lên LCD 16x02, hàm này cơ bản sẽ đợi khi đủ một giây trôi qua dựa trên **light_counter**. Phần code còn lại khá rõ ràng.

7.3 pedestrianStateFSM

```

266 uint16_t setCounterForBuzzer(){
267     uint8_t counter = light_counter[VERTICAL] / ONE_SECOND;
268     if (counter <= 3){
269         return ONE_SECOND / 20;
270     }
271     else if (counter <= 6){
272         return ONE_SECOND / 10;
```



```
273 }
274 else if (counter <= 12){
275     return ONE_SECOND / 5;
276 }
277 else {
278     return ONE_SECOND / 2;
279 }
280 }

281

282 void pedestrianStateFSM(void){
283     switch (pState){
284     case SLEEP:
285         buzzerNoBeep();
286         break;
287     case WAIT:
288         buzzerNoBeep();
289         if (ledState[VERTICAL] == RED && light_counter[VERTICAL]
290             >= 8 * ONE_SECOND){
291             pState = ACROSS;
292         }
293         break;
294     case ACROSS:
295         if (ledState[VERTICAL] == GREEN || ledState[VERTICAL] ==
296             AMBER){
297             pState = SLEEP;
298         }
299
300         //Peep peep peep
301         if (ledState[VERTICAL] == RED) {
302             buzzer_counter--;
303             if (buzzer_counter <= 0){
304                 buzzer_counter = setCounterForBuzzer();
305                 HAL_GPIO_TogglePin(BUZZER_GPIO_Port, BUZZER_Pin);
306             }
307         }
308 }
```



```
306     break;
307     default:
308         pState = SLEEP;
309     }
310 }
```

Ánh xạ từ máy trạng thái ở trên. Khi trong chế độ SLEEP và WAIT, buzzer sẽ không được kêu. Nhưng khi buzzer trong trạng thái ACROSS sẽ có counter cho mỗi lần toggle Buzzer. Hàm **setCounterForBuzzer()** dùng để tạo ra thời gian counter dựa trên số giây còn lại của người qua đường. Như trên thì ta tạo thời gian khởi tạo lại cho counter thấp dần khi thời gian cho người đi bộ càng ít.

7.4 Button

```
311 //This is abstract function. Use for those function below.
312 void inputProcessingFSM(void (*processing) (void), const
313     short index){
314     switch(buttonState[index]){
315         case RELEASED:
316             if (isButtonPressed(index)){
317                 buttonState[index] = PRESSED;
318                 (*processing)();
319             }
320             break;
321         case PRESSED:
322             if (!isButtonPressed(index)){
323                 buttonState[index] = RELEASED;
324             } else if (isButtonPressedOneSec(index)){
325                 buttonState[index] = PRESSED_MORE_THAN_ONE_SECOND;
326             }
327             break;
328         case PRESSED_MORE_THAN_ONE_SECOND:
329             if (!isButtonPressedOneSec(index)){
330                 buttonState[index] = RELEASED;
331             }
332             break;
333     }
334 }
```



```
332     }
333 }
334 void handleSetValueButton(void){
335     buttonReading(SET_BTN);
336     inputProcessingFSM(setValue, SET_BTN);
337 }
338 void handleModifyButton(void){
339     buttonReading(MOD_BTN);
340     inputProcessingFSM(modifyingValue, MOD_BTN);
341     //Handle when button hold more than one second
342     //increase `adjusting value` after INCREASING_PERIOD ms
343     if (buttonState[MOD_BTN] == PRESSED_MORE_THAN_ONE_SECOND) {
344         increasing_counter--;
345         if (increasing_counter <= 0) {
346             increasing_counter = INCREASING_PERIOD;
347             modifyingValue();
348         }
349     } else {
350         increasing_counter = INCREASING_PERIOD;
351     }
352 }
353 void handleSelectModeButton(void){
354     buttonReading(SEL_BTN);
355     inputProcessingFSM(changingMode, SEL_BTN);
356 }
357
358 void handlePedestrianButton(void){
359     buttonReading(P_BTN);
360     inputProcessingFSM(handlePedestrianPressedEvent, P_BTN);
361 }
```

Chúng ta sẽ tạo ra ‘abstract function’ **inputProcessingFSM**. Mỗi lần nút được nhấn thì sẽ đi thực hiện (*processing)(), ta chỉ cần truyền con trỏ hàm tương ứng. Các con trỏ hàm mà các nút nhấn sử dụng sẽ ở dưới đây.

```
362 void resetTrafficLight(void){
```



```
363 pState = SLEEP;
364 ledState[VERTICAL] = GREEN;
365 ledState[HORIZONTAL] = RED;
366 light_counter[VERTICAL] = durationGreen * ONE_SECOND;
367 light_counter[HORIZONTAL] = durationRed * ONE_SECOND;
368 buzzerNoBeep();
369 }
370
371 void setValue(void){
372 switch(runState){
373 case NORMAL_MODE:
374 break;
375 case MODIFY_DURATION_RED_MODE:
376 durationRed = adjust_duRed;
377
378 lcd_clear();
379 lcd_put_cur(0, 0);
380 sprintf(message, "CUR_VALUE: %ds", durationRed);
381 lcd_send_string(message);
382 lcd_put_cur(1, 0);
383 sprintf(message, "MOD_RED: %ds", adjust_duRed);
384 lcd_send_string(message);
385
386 break;
387 case MODIFY_DURATION_AMBER_MODE:
388 durationAmber = adjust_duAmber;
389
390 lcd_clear();
391 lcd_put_cur(0, 0);
392 sprintf(message, "CUR_VALUE: %ds", durationAmber);
393 lcd_send_string(message);
394 lcd_put_cur(1, 0);
395 sprintf(message, "MOD_AMBER: %ds", adjust_duAmber);
396 lcd_send_string(message);
397
```



```
398     break;
399 case MODIFY_DURATION_GREEN_MODE:
400     durationGreen = adjust_duGreen;
401
402     lcd_clear();
403     lcd_put_cur(0, 0);
404     sprintf(message, "CUR_VALUE: %ds", durationGreen);
405     lcd_send_string(message);
406     lcd_put_cur(1, 0);
407     sprintf(message, "MOD_GREEN: %ds", adjust_duGreen);
408     lcd_send_string(message);
409
410     break;
411 }
412 }
413
414 void changingMode(void){
415     //Turn off all LEDs.
416     resetAllLED();
417
418     //Changing state and initial new value for new mode.
419     switch(runState){
420     case NORMAL_MODE:
421         runState = MODIFY_DURATION_RED_MODE;
422         blinking_counter = HALF_SECOND;
423         adjust_duRed = durationRed;
424         resetTrafficLight();
425
426         lcd_clear();
427         lcd_put_cur(0, 0);
428         sprintf(message, "CUR_VALUE: %ds", durationRed);
429         lcd_send_string(message);
430         lcd_put_cur(1, 0);
431         sprintf(message, "MOD_RED: %ds", adjust_duRed);
432         lcd_send_string(message);
```



```
433
434     break;
435 case MODIFY_DURATION_RED_MODE:
436     runState = MODIFY_DURATION_AMBER_MODE;
437     blinking_counter = HALF_SECOND;
438     adjust_duAmber = durationAmber;
439     resetTrafficLight();
440
441     lcd_clear();
442     lcd_put_cur(0, 0);
443     sprintf(message, "CUR_VALUE: %ds", durationAmber);
444     lcd_send_string(message);
445     lcd_put_cur(1, 0);
446     sprintf(message, "MOD_AMBER: %ds", adjust_duAmber);
447     lcd_send_string(message);
448
449     break;
450 case MODIFY_DURATION_AMBER_MODE:
451     runState = MODIFY_DURATION_GREEN_MODE;
452     blinking_counter = HALF_SECOND;
453     adjust_duGreen = durationGreen;
454     resetTrafficLight();
455
456     lcd_clear();
457     lcd_put_cur(0, 0);
458     sprintf(message, "CUR_VALUE: %ds", durationGreen);
459     lcd_send_string(message);
460     lcd_put_cur(1, 0);
461     sprintf(message, "MOD_GREEN: %ds", adjust_duGreen);
462     lcd_send_string(message);
463
464     break;
465 case MODIFY_DURATION_GREEN_MODE:
466     runState = NORMAL_MODE;
467     resetTrafficLight();
```



```
468     break;
469 }
470 }
471
472 void modifyingValue(void){
473     switch(runState){
474         case NORMAL_MODE:
475             break;
476         case MODIFY_DURATION_RED_MODE:
477             increaseOne(&adjust_duRed);
478
479             lcd_clear();
480             lcd_put_cur(0, 0);
481             sprintf(message, "CUR_VALUE: %ds", durationRed);
482             lcd_send_string(message);
483             lcd_put_cur(1, 0);
484             sprintf(message, "MOD_RED: %ds", adjust_duRed);
485             lcd_send_string(message);
486
487             break;
488         case MODIFY_DURATION_AMBER_MODE:
489             increaseOne(&adjust_duAmber);
490
491             lcd_clear();
492             lcd_put_cur(0, 0);
493             sprintf(message, "CUR_VALUE: %ds", durationAmber);
494             lcd_send_string(message);
495             lcd_put_cur(1, 0);
496             sprintf(message, "MOD_AMBER: %ds", adjust_duAmber);
497             lcd_send_string(message);
498
499             break;
500         case MODIFY_DURATION_GREEN_MODE:
501             increaseOne(&adjust_duGreen);
502 }
```



```

503     lcd_clear();
504     lcd_put_cur(0, 0);
505     sprintf(message, "CUR_VALUE: %ds", durationGreen);
506     lcd_send_string(message);
507     lcd_put_cur(1, 0);
508     sprintf(message, "MOD_GREEN: %ds", adjust_duGreen);
509     lcd_send_string(message);
510     break;
511 }
512 }
513
514 void handlePedestrianPressedEvent(){
515     if (runState != NORMAL_MODE) return;
516     switch(pState){
517         case SLEEP:
518             if (ledState[VERTICAL] == GREEN || ledState[VERTICAL] ==
519                 AMBER
520                 || (ledState[VERTICAL] == RED && light_counter[
521                     VERTICAL] < 8 * ONE_SECOND)
522             ) {
523                 pState = WAIT;
524             } else {
525                 pState = ACROSS;
526             }
527             break;
528         case WAIT:
529             break;
530         case ACROSS:
531             break;
532         default:
533             pState = SLEEP;
534     }
535 }
```

Thực chất các hàm rất đơn giản, nhưng do truyền dữ liệu ra màn hình LCD 16x02 nên code trông có vẻ khá dài và phức tạp. Để ý hàm **changingMode**, như đã nói ở trên,



khi bắt đầu chuyển sang NORMAL_MODE ta dùng hàm **resetTrafficLight** để reset lại FSM của 2 chiều đèn giao thông.

8 In main.c

Listing 8.1: Các hàm trong main.c

```
534 void readInputAndProcessing(void){  
535     handlePedestrianButton();  
536     handleSelectModeButton();  
537     handleModifyButton();  
538     handleSetValueButton();  
539 }  
540 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){  
541     SCH_Update();  
542 }
```

Listing 8.2: main.c

```
543 lcd_init();  
544 SCH_Init();  
545  
546 inputReadingInit();  
547 inputProcessingInit(&huart2);  
548  
549 SCH_Add_Task(readInputAndProcessing, 0, 10);  
550 SCH_Add_Task(runStateFSM, 10, 10);  
551  
552 HAL_TIM_Base_Start_IT(&htim2);  
553  
554 while (1)  
555 {  
556     SCH_Dispatch_Tasks();  
557 }
```

Theo như đoạn code trên thì 4 hàm đầu tiên sẽ khởi tạo lần lượt là: giao tiếp lcd, scheduler, đọc nút nhấn, xử lý đầu vào.



Hàm xử lí và đọc giá trị đầu, runStateFSM sẽ được thêm vào scheduler với chu kì là 10ms.

Hàm **SCH_Dispatch** sẽ được chạy trong vòng while.

9 Repository and Diagram Link

Họ và tên: Nguyễn Đăng Khoa

MSSV: 2113760

Github: https://github.com/NgKoaz/STM32_TrafficLightForPestrian

Diagram: <https://drive.google.com/file/d/1WQi3pvziPSDsy-x04GGYc7gwKt5vIKrg/view?usp=sharing>