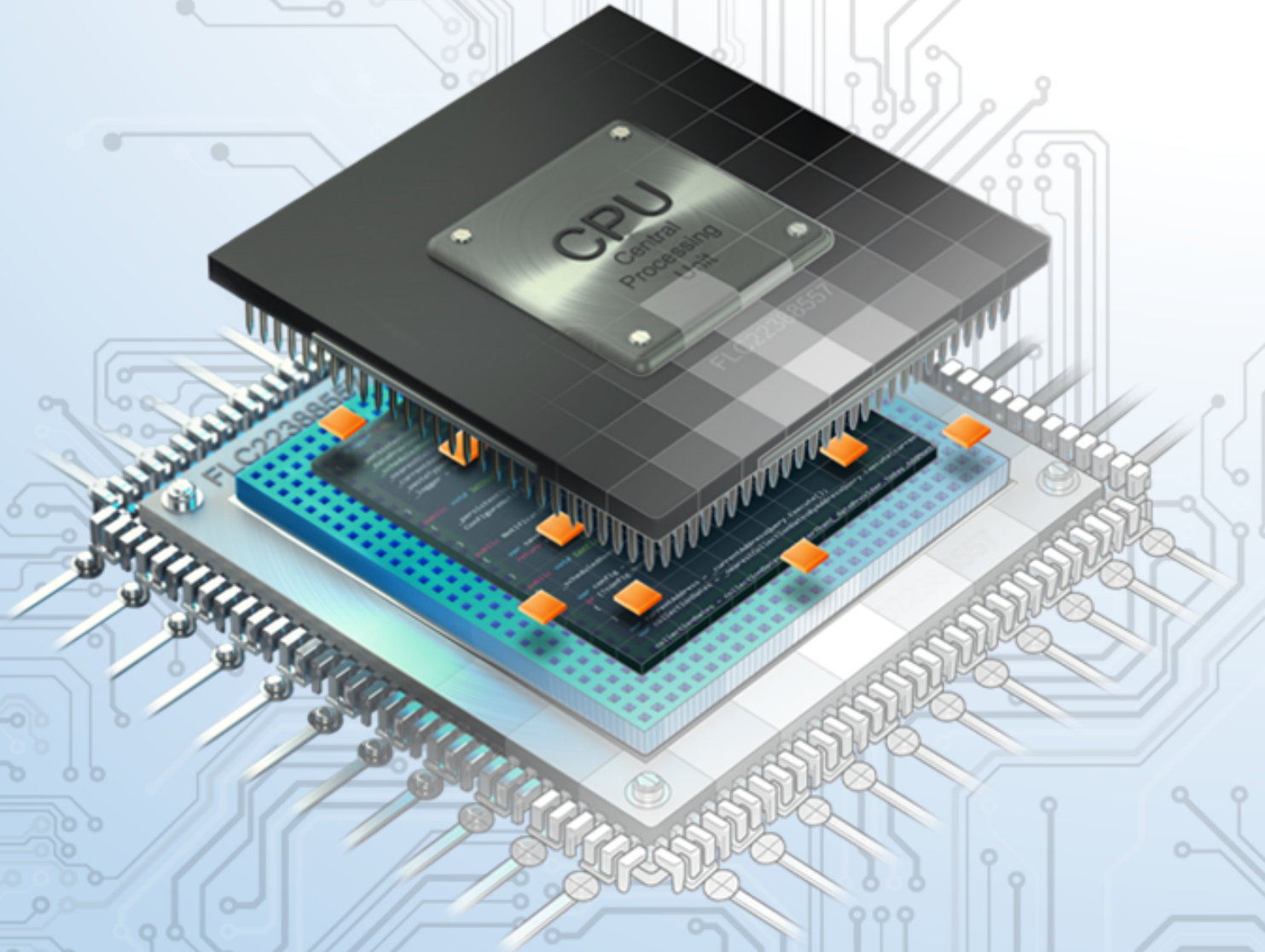
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

LAB 2: TIMER

Sinh viên thực hiện: Nguyễn Thanh Hiền (MSSV: 2111203)
Lớp: L03 - HK231



Dr. Le Trong Nhan

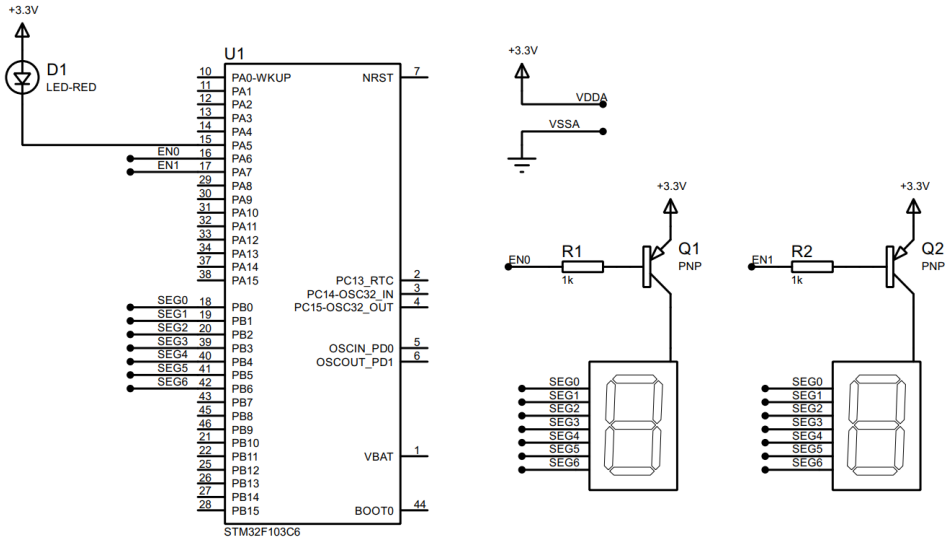
Mục lục

Github: CO3009-Lab2

1 Exercise 1

1.1 Yêu cầu

Mô phỏng mạch trong Proteus với hai đèn LED 7 đoạn. Sinh viên được sử dụng lại hàm `display7SEG(int num)` để hiện thực hàm ngắt hiển thị số "1" cho đèn thứ nhất và số "2" cho đèn thứ hai. Thời gian quét giữa 2 đèn là nửa giây (500ms). Trình bày sơ đồ nguyên lý, source code trong hàm `HAL_TIM_PeriodElapsedCallback` và trả lời câu hỏi: tần số quét LED là bao nhiêu?



Hình 1: Simulation schematic in Proteus

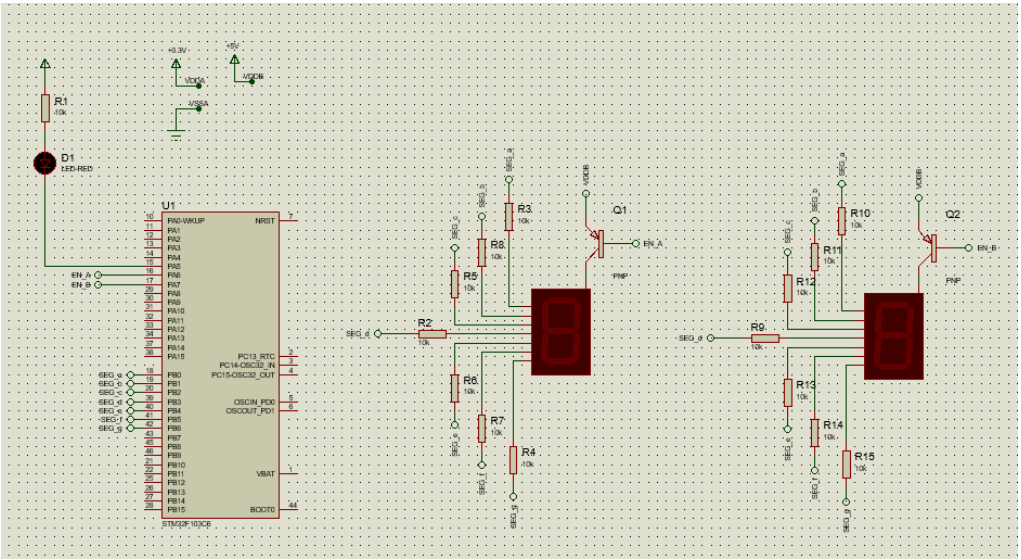
1.2 Bài làm

Một cấu trúc timer được em thêm vào với các thông tin:

- `int limit`: Giá trị được thiết lập và giữ nguyên để thể hiện cận trên để cài đặt lại `timer_counter` sau mỗi lần tràn.
- `int counter`: Biến đếm lên của timer
- `int flag`: Cờ đánh dấu timer đã tràn, sẽ được bắt và xử lý bởi vòng `while(1)`

Ngoài ra, ta sử dụng thêm mảng `int timer_limit[NUM_TIMER]` để lưu tất cả cận trên của các timer đang được sử dụng, dùng để thiết lập giá trị ban đầu của biến `int limit` trong `struct timer`.

```
#define NUM_TIMER 1
#define SEG_TIMER_DURATION 50
```



Hình 2: Sơ đồ nguyên lý với 2 đèn LED 7 đoạn

```
#define SEG_TIMER_INDEX 0
typedef struct{
    int limit;
    int counter;
    int flag;
} timer_config;
timer_config timer[NUM_TIMER];
int timer_limit[NUM_TIMER] = {SEG_TIMER_DURATION};
```

Program 1: Cấu trúc lưu trữ dữ liệu 1 timer

Vì hàm ngắt chỉ nên được thực hiện trong một khoảng thời gian cực kỳ ngắn, nên hàm này chỉ dùng để đếm lên và dựng cờ đơn giản. Các thao tác phức tạp được chuyển ra bên ngoài hàm.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  for(int index = 0; index < NUM_TIMER; index++){
    if(timer[index].counter > 0){
      timer[index].counter--;
      if(timer[index].counter <= 0){
        timer[index].flag = 1;
      }
    }
  }
}
```

Program 2: Hiện thực hàm ngắt HAL_TIM_PeriodElapsedCallback

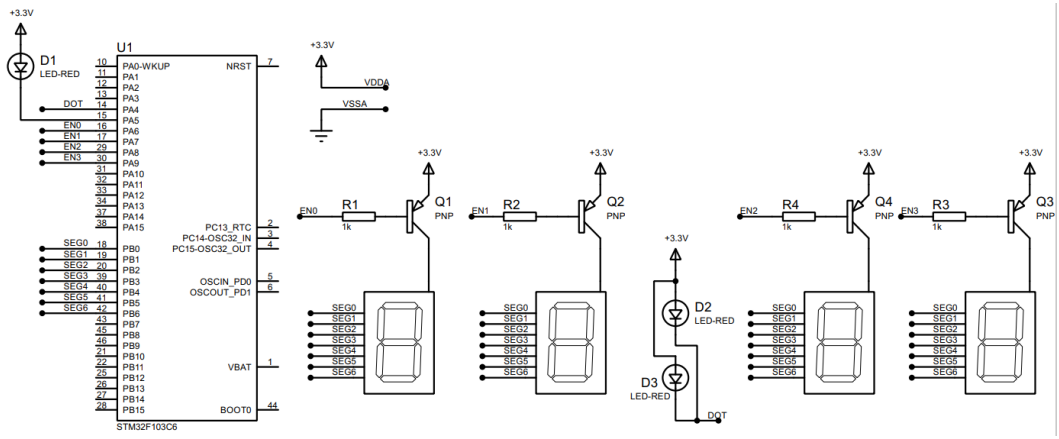
Trả lời câu hỏi: Các LED được quét sau mỗi nửa giây (500ms) và ta có 2 LED. Vì vậy, thời gian của một chu kỳ quét LED là: $T = 500 * 2 = 1000(ms) = 1s$, nên tần số quét là:

$$f = \frac{1}{T} = \frac{1}{1} = 1Hz \tag{1}$$

2 Exercise 2

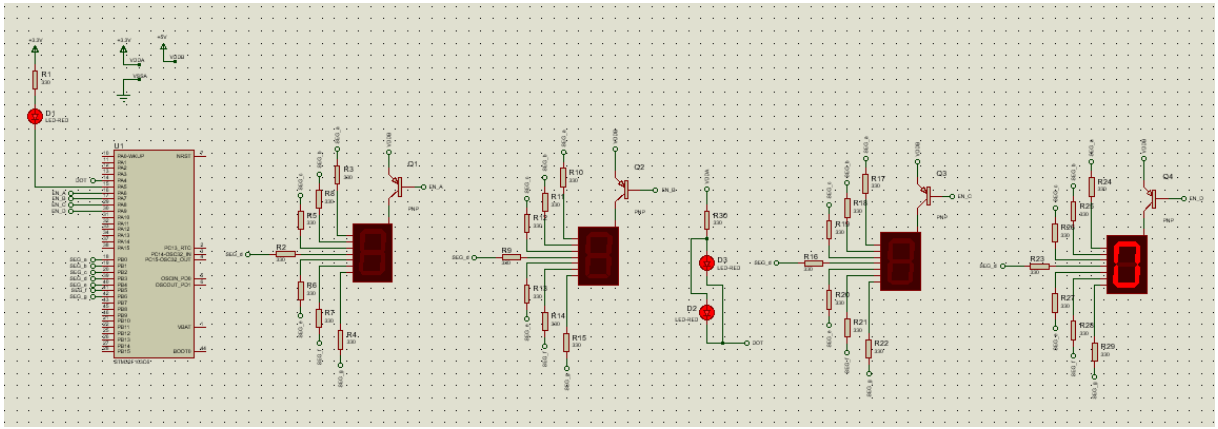
2.1 Yêu cầu

Mở rộng cho 4 đèn LED 7 đoạn và 2 đèn LED đơn (cùng kết nối tới chân **PA4**, được đặt tên là `DOT`. Mô phỏng để LED đơn nhấp nháy mỗi giây. Ngoài ra, hiển thị số "3" tại LED 7 đoạn thứ ba, số "0" tại LED 7 đoạn cuối cùng (để hiển thị thời gian là 12h30). Thời gian quét đèn LED 7 đoạn vẫn là nửa giây (500ms). Hiện thực trong hàm interrupt, sau đó, trình bày sơ đồ nguyên lý và sourcecode của hàm `HAL_TIM_PeriodElapsedCallback` và trả lời câu hỏi: tần số của quá trình quét LED là bao nhiêu?



Hình 3: Simulation schematic in Proteus

2.2 Bài làm



Hình 4: Sơ đồ nguyên lý với 4 LEDs bảy đoạn

Đối với bài này, **phần source code được tái sử dụng từ bài trước**. Vì em đã sử dụng các biến để thac tác trên software timer khi số lượng timer tăng.

Trả lời câu hỏi:

- ***Đối với LED 7 đoạn,*** mỗi được quét sau mỗi nửa giây (500ms) và ta có 4 LED. Vì vậy, thời gian của một chu kỳ quét LED là: $T = 500 * 4 = 2000(ms) = 2s$, nên tần số quét là:

$$f = \frac{1}{T} = \frac{1}{2} = 0.5Hz \tag{2}$$

- ***Đối với LED đơn,*** cả hai LED sẽ cùng chớp tắt sau mỗi một giây (1000ms) và để LED hoàn thành 1 lần chớp tắt thì ta có chu kỳ là: $T = 1000 * 2 = 2000(ms) = 2(s)$, nên tần số của LED đơn là:

$$f = \frac{1}{T} = \frac{1}{2} = 0.5Hz \tag{3}$$

3 Exercise 3

3.1 Yêu cầu

Hiên thực hàm `update7SEG(int index)` với một mảng gồm 4 số nguyên, đánh dấu thứ tự của LEDs 7 đoạn được hiển thị. Hàm này cần được gọi bởi timer interrupt. Biến `index_led` cần được cập nhất để giữ trong khoảng hợp lệ [0; 3].

Trình bày source code của hàm `update7SEG(int index)` và `HAL_TIM_PeriodElapsedCallback` . Sinh viên cần thay đổi giá trị trong `led_buffer` để kiểm tra hiển thị.

3.2 Bài làm

Sử dụng các hàm hỗ trợ: `enable_led_seg(int index)` và `disable_led_seg(int index )` để bật/tắt đèn LED 7 đoạn tại vị trí tương ứng.

```
void enable_led_seg(int index){
    HAL_GPIO_WritePin(seg_enable_pin[index].port,
   seg_enable_pin[index].pin, GPIO_PIN_RESET);
}
void disable_led_seg(int index){
    HAL_GPIO_WritePin(seg_enable_pin[index].port,
   seg_enable_pin[index].pin, GPIO_PIN_SET);
}
```

Program 3: Hàm hỗ trợ bật/tắt LED 7 đoạn tại vị trí tương ứng

Theo yêu cầu đề bài, sinh viên phải hiện thực hàm `update7SEG(int index)` sử dụng switch case.

```
#define NUM_LED 4
int led_index = 0;
int led_buffer[4] = {1, 2, 3, 4};
void update7SEG(int index){
```

```
  switch(index){
  case 0:
    enable_led_seg(0);
    disable_led_seg(1);
    disable_led_seg(2);
    disable_led_seg(3);
    display7SEG(led_buffer[0], GPIOB);
    break;
  case 1:
    disable_led_seg(0);
    enable_led_seg(1);
    disable_led_seg(2);
    disable_led_seg(3);
    display7SEG(led_buffer[1], GPIOB);
    break;
  case 2:
    disable_led_seg(0);
    disable_led_seg(1);
    enable_led_seg(2);
    disable_led_seg(3);
    display7SEG(led_buffer[2], GPIOB);
    break;
  case 3:
    disable_led_seg(0);
    disable_led_seg(1);
    disable_led_seg(2);
    enable_led_seg(3);
    display7SEG(led_buffer[3], GPIOB);
    break;
  default:
    disable_led_seg(0);
    disable_led_seg(0);
    disable_led_seg(0);
    disable_led_seg(0);
    display7SEG(MAX_NUM + 1, GPIOB);
    break;
  }
}
```

Program 4: Hàm update7SEG(int index)

Tuy nhiên, nếu có thể sử dụng các biến để linh động thay đổi khi số lượng đèn thay đổi, giá trị hiện thực cũng thay đổi và để source code gọn hơn thì hàm `update7SEG(int index)` có thể được hiện thực như sau:

```
#define NUM_LED 4
int led_index = 0;
int led_buffer[4] = {1, 2, 3, 4};
void update7SEG(int index){
    for(uint16_t i = 0; i < NUM_LED; i++){
        if(i == index)
```

```
            enable_led_seg(i);
        if(i != index)
            disable_led_seg(i);
    }
    display7SEG(led_buffer[index], GPIOB);
}
```

Program 5: Hàm update7SEG(int index) sau khi rút gọn

Nội dung hàm `HAL_TIM_PeriodElapsedCallback` **tương tự các bài trước đó** ***(tham khảo phần: Hiện thực hàm ngắt HAL_TIM_PeriodElapsedCallback)***

4 Exercise 4

4.1 Yêu cầu

Thay đổi chu kỳ kích hoạt hàm `update7SEG` để thay đổi tần số của 4 LED 7 đoạn thành 1Hz. Dấu chấm vẫn chớp tắt sau mỗi giây. Trình bày source code của hàm `HAL_TIM_PeriodElapsedCallback`.

4.2 Bài làm

Để tần số của 4 LED 7 đoạn là 1Hz thì chu kỳ để chớp tắt tất cả các LED là: $T = \frac{1}{f} = \frac{1}{1} = 1(s)$. Như vậy, thời gian nhấp nháy của mỗi LED là:

$$t_{7SEG} = \frac{T}{NUM\_LED} = \frac{1}{4} = 0.25(s) = 250(ms) \tag{4}$$

Timer được cài đặt sẽ gọi interrupt sau mỗi **10ms**, như vậy, tổng số chu kỳ cho mỗi LED được sáng là:

$$N_{count} = \frac{t_{7SEG}}{T_{timer}} = \frac{250}{10} = 25(T_{timer}) \tag{5}$$

Vậy, ta không cần thay đổi nội dung của hàm `HAL_TIM_PeriodElapsedCallback` *(tham khảo phần: Hiện thực hàm ngắt HAL_TIM_PeriodElapsedCallback)* mà chỉ cần thay đổi thông số `#define SEG_TIMER_DURATION 25` vì hàm này chỉ dùng để dựng cờ, các thao tác phức tạp khác được xử lý bởi hàm `main`.

```
while (1)
{
  // LED scanning
  update7SEG(led_index);
  if(dot_status == 1)
      enable_led_seg(NUM_LED);
  if(dot_status == 0)
      disable_led_seg(NUM_LED);
  // Timer controller for next state
  if(timer[SEG_TIMER_INDEX].flag == 1){
```

```
      set_timer(SEG_TIMER_INDEX);
      led_index = (led_index + 1) % NUM_LED;// Switch to next
    LEDs
  }
  if(timer[DOT_TIMER_INDEX].flag == 1){
      set_timer(DOT_TIMER_INDEX);
      dot_status = !dot_status;
  }
}
```

5 Exercise 5

5.1 Yêu cầu

Hiện thực hàm `updateClockBuffer` hiển thị với thông tin về **giờ** và **phút** được hiển thị bởi mỗi cặp LED 7 đoạn. Trường hợp giờ/phút chỉ có 1 số thì bổ sung số 0 ở trước. Trình bày source code của vòng `while(1)` và hàm `updateClockBuffer`.

5.2 Bài làm

Thiết lập biến *hour*, *minute* và *second* trong chương trình `main.c` với các giá trị tương đương 15 giờ 08 phút và 50 giây. Đồng thời, cài đặt biến $int\ led\_buffer[NUM\_LED] = \{1,5,0,8\}$ trong chương trình `led_seg.c`.

```
int hour = 15, minute = 8, second = 50;

while(1){
    // LED scanning
    update7SEG(led_index);
    displayDot(dot_status);
    // Timer controller for next state
    if(timer[SEG_TIMER_INDEX].flag == 1){
      set_timer(SEG_TIMER_INDEX);
      led_index = (led_index + 1) % NUM_LED;// Switch to next
    LEDs
    }
    if(timer[DOT_TIMER_INDEX].flag == 1){
      set_timer(DOT_TIMER_INDEX);
      dot_status = !dot_status;
    }
    if(timer[CLOCK_TIMER_INDEX].flag == 1){
      set_timer(CLOCK_TIMER_INDEX);
      second++;
      if(second >= 60){
          second = 0;
          minute++;
```

```
        }
        if(minute >= 60){
            hour++;
            minute = 0;
        }
        if(hour >= 24){
            hour = 0;
        }
        updateClockBuffer(hour, minute);
    }
}
```

Program 6: Nội dung hàm while(1)

```
void updateClockBuffer(int hour, int minute){
    led_buffer[0] = hour / 10;
    led_buffer[1] = hour % 10;
    led_buffer[2] = minute / 10;
    led_buffer[3] = minute % 10;
}
```

Program 7: Nội dung hàm updateClockBuffer(int hour, int minute)

6 Exercise 6

6.1 Yêu cầu

Tham khảo đoạn chương trình sau:

Bước 1: Khai báo biến và hàm cho software timer, như sau:

```
/* USER CODE BEGIN 0 */
int timer0_counter = 0;
int timer0_flag = 0;
int TIMER_CYCLE = 10;
void setTimer0(int duration){
  timer0_counter = duration /TIMER_CYCLE;
  timer0_flag = 0;
}
void timer_run(){
  if(timer0_counter > 0){
    timer0_counter--;
    if(timer0_counter == 0) timer0_flag = 1;
  }
}
/* USER CODE END 0 */
```

Program 8: Software timer based timer interrupt

Bước 2 2: Hàm `timer_run()` được gọi trong timer interrupt như sau:

```c
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){

  timer_run();

  //YOUR OTHER CODE
}
```

Program 9: Software timer based timer interrupt

Bước 3: Dùng timer trong `main` bằng cách gọi hàm `setTimer0`, sau đó kiểm tra cờ $timer0\_flag$.

```c
setTimer0(1000);
while (1){
    if(timer0_flag == 1){
        HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
        setTimer0(2000);
    }
}
```

Program 10: Software timer is used in main fuction to blink the LED

Trả lời các câu hỏi:

1. Điều gì sẽ xảy ra nếu đoạn code tại dòng thứ 1 của chương trình trên bị xóa?
2. Điều gì sẽ xảy ra tiếp theo nếu đoạn code tại dòng thứ 1 được đổi thành `setTimer0(1)`, tại sao?
3. Khi thay đổi dòng thứ 1 thành `setTimer0(10)` thì sẽ khác gì so với 2 câu hỏi trước đó, tại sao?

6.2 Bài làm

1. Nếu xóa `void setTimer0(1000)` thì các biến sẽ được cài đặt ở trạng thái ban đầu $timer0\_counter = 0$, $timer0\_flag = 0$. Khi ISR được gọi, hàm `void timer_run()` nhưng vì biến $timer0\_flag = 0$ không thỏa điều kiện là số dương lớn hơn 0 nên **đèn sẽ luôn ở trạng thái đó, không thay đổi**. Vì timer không thể được thiết lập giá trị đếm, không thể thay đổi cờ để chuyển trạng thái LED.
2. Nếu đổi thành `void setTimer0(1)` thì hàm `void run_timer()` sẽ đếm xuống 1 lần, trước khi cài đặt lại đồng hồ với `void setTimer0(2000)`. **Như vậy, tại lần gọi interrupt đầu tiên (10ms), chương trình sẽ chớp LED 1 lần trước khi quay trở lại chu kỳ thay đổi trạng thái thông thường với LED thay đổi sau mỗi:** $200 * 10 = 2000(ms) = 2(s)$**.**
3. Nếu thay đổi thành `void setTimer0(10) void run_timer()` thì chương trình sẽ thực hiện tương tự câu 2 với điểm khác là **tại lần gọi interrupt thứ 10 (100ms), chương trình sẽ chớp LED 1 lần trước khi quay trở lại chu kỳ thay đổi trạng thái thông thường như câu 2.**

7 Exercise 7

7.1 Yêu cầu

Thay đổi source code của bài 5 phần cập nhật giá trị giờ, phút và giây sử dụng timer để bỏ hàm Delay. Đồng thời, hàm điều khiển DOT cũng cần di chuyển tới hàm `main`

7.2 Bài làm

Vì trước đó em đã hiện thực software timer để loại bỏ hoàn toàn hàm delay, cũng như đưa các hàm vào `main` nên phần này em sẽ trình bày lại source code timer đã hiện thực. Cụ thể, vì mỗi phần tử: LED 7 đoạn, DOT và đồng hồ đếm theo giây sẽ chạy với tần suất khác nhau nên sẽ cần số lượng timer tương ứng. Từ đó, biến và macro trong phần *Cấu trúc lưu trữ dữ liệu 1 timer* sẽ được thay đổi thành:

```
#define NUM_TIMER 3
#define SEG_TIMER_DURATION 25
#define SEG_TIMER_INDEX 0
#define DOT_TIMER_DURATION 50
#define DOT_TIMER_INDEX 1
#define CLOCK_TIMER_DURATION 100
#define CLOCK_TIMER_INDEX 2
typedef struct{
    int limit;
    int counter;
    int flag;
} timer_config;
timer_config timer[NUM_TIMER];
extern int timer_limit[NUM_TIMER];
```

Program 11: Cấu trúc lưu trữ 1 timer thay đổi theo số lượng timer trong global.h

Sau đó, ta thiết lập các hàm cho timer trong `software_timer.c`

```
int timer_limit[NUM_TIMER] = {SEG_TIMER_DURATION,
   DOT_TIMER_DURATION, CLOCK_TIMER_DURATION};
void init_software_timer(){
    for(uint16_t index = 0; index < NUM_TIMER; index++){
        timer[index] = (timer_config){timer_limit[index],
   timer_limit[index], 0};
    }
}
void set_timer(int index){
    timer[index].counter = timer[index].limit;
    timer[index].flag = 0;
}
```

Program 12: Các hàm thiết lập timer trong software_timer.c

Cuối cùng, nội dung của hàm `void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)` (2) và vòng lặp `while(1)` (6) vẫn giữ nguyên.

8 Exercise 8

8.1 Yêu cầu

Di chuyển hàm `void update7SEG()` vào `main`, chỉ sử dụng timer interrupt để xử lý software timer. Tất cả các thao tác xử lý (tính toán phức tạp) sẽ được di chuyển vào vòng lặp vô hạn, tối ưu độ phức tạp của hàm interrupt.

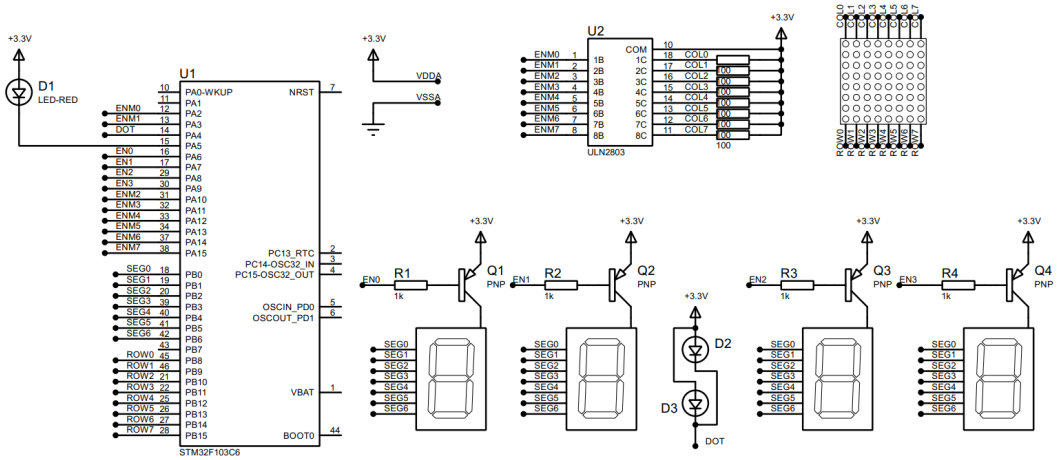
8.2 Bài làm

Từ các bài trước, em đã di chuyển các hàm tính toán phức tạp vào main và tối ưu timer interrupt.

9 Exercise 9

9.1 Yêu cầu

Sinh viên bổ sung thêm đèn LED ma trận **MATRIX-8X8-RED** và **ULN2803** (NPN transistor để enable một cột LED). Chương trình có thể thay đổi tín hiệu (từ chân ENM0 đến ENM7) khi cần thiết, và dữ từ ROW0 đến ROW7 cần được nối tương ứng với PB8 - PB15.

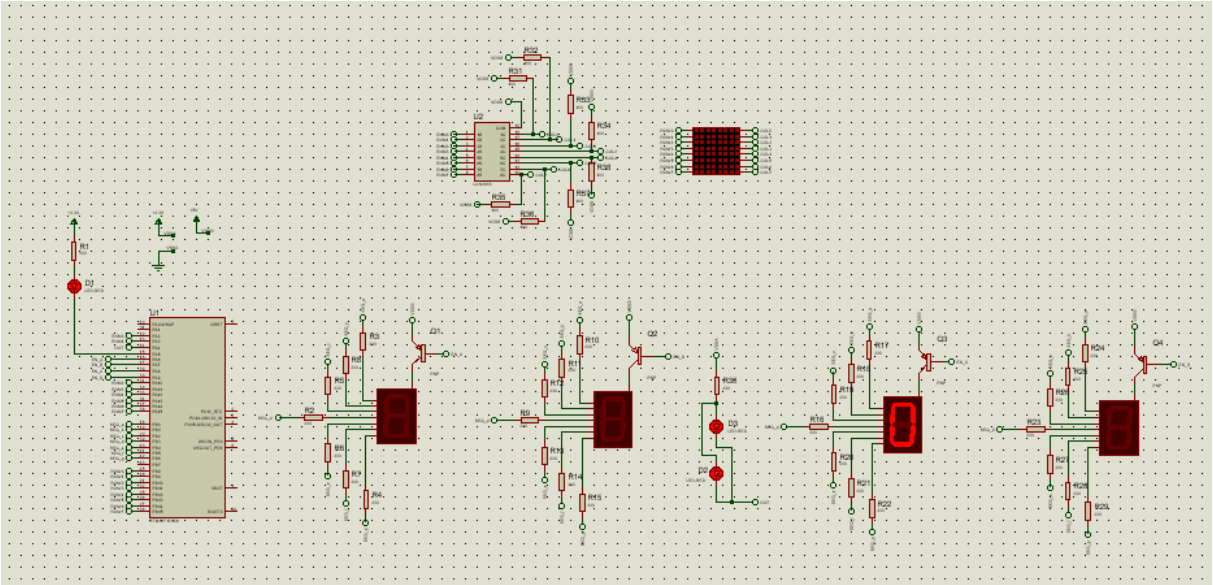


Hình 5: Sơ đồ nguyên lý mẫu với LED ma trận được thêm vào

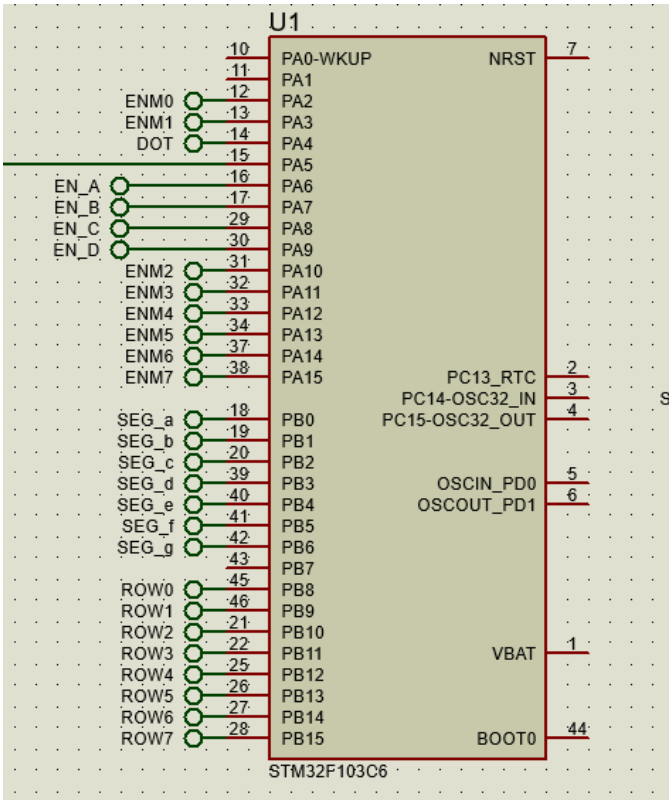
Trình bày sơ đồ nguyên lý trong Proteus và hiện thực hàm `updateLEDMatrix(int index)` tương tự như 4 đèn LED 7 đoạn. Sinh viên được tự chọn tần số thiết lập cho

hàm này và hàm phải được gọi trong `main`. Đồng thời, thay đổi biến *matrix_buffer* để hiển thị ký tự "A".

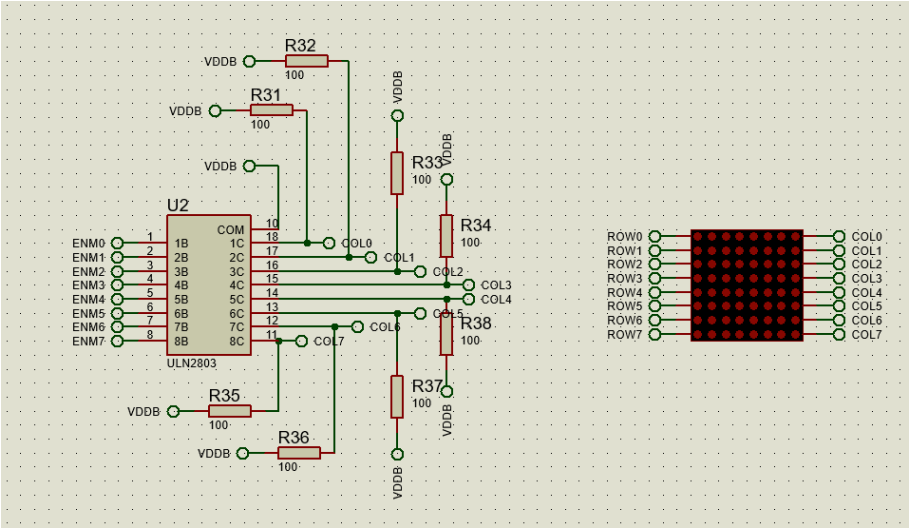
9.2 Bài làm



Hình 6: Sơ đồ nguyên lý mạch với LED ma trận



Hình 7: Các chân tín hiệu của STM32

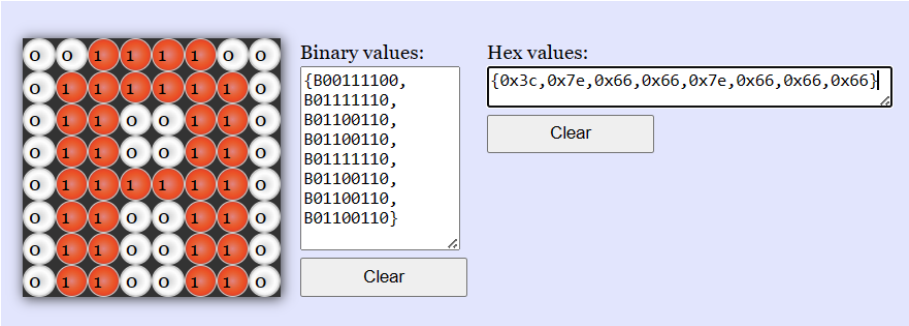


Hình 8: Các chân tín hiệu nối tới LED ma trận

Đối với bài này, STM32 sẽ sử dụng 16 bit dữ liệu thay vì sử dụng tổng cộng 64 chân (8x8) để điều khiển đèn LED ma trận. Ta có thể hiện thực bằng cách sử dụng timer để tuần tự kích hoạt một dòng LED sáng, và truyền dữ liệu theo dạng chuỗi để hiển thị nội dung của dòng đó. Vì vậy, ta sẽ cần:

- 8 chân kích hoạt (ENMx): Lựa chọn dòng sẽ kích hoạt trong mỗi lần ngắt timer.
- 8 chân truyền dữ liệu (ROWx): Giá trị HIGH/LOW tương ứng từng bit của dòng.

Ví dụ: Đối với chữ cái "A", tương ứng với từng dòng sẽ có một mã HEX để biểu diễn trạng thái HIGH/LOW của chân điều khiển tại dòng tương ứng. Trường hợp này là chuỗi: {0x3c, 0x7e, 0x66, 0x66, 0x7e, 0x66, 0x66, 0x66}



Hình 9: Tạo mã HEX cho chữ cái A

Vì để hiển thị được chữ cái, mỗi dòng của đèn LED sẽ được quét với tần số cao. Trong bài này, em thiết lập số lần đếm cho timer là 9 (tức sẽ đếm 10 giá trị từ 0 đến 9) với mỗi lần đếm là 10ms. Như vậy, ta có thời gian sáng của mỗi dòng là:

$$t_{row} = N_{count} * T_{timer} = 10 * 10 = 100(ms) \tag{6}$$

Như vậy, để quét hết toàn bộ 8 dòng thì LED ma trận sẽ thực hiện trong chu kỳ và tần số là:

$$T_{matrix} = 100 * 8 = 800(ms) = 0.8(s) f_{matrix} = \frac{1}{T_{matrix}} = \frac{1}{0.8} = 1,25(Hz) \tag{7}$$

Giá trị	HEX
A	{0x3c, 0x7e, 0x66, 0x66, 0x7e, 0x66, 0x66, 0x66}
B	{0x7c, 0x62, 0x62, 0x7c, 0x7c, 0x62, 0x62, 0x7c}
C	{0x3e, 0x7e, 0x60, 0x60, 0x60, 0x60, 0x7e, 0x3e}
D	{0x7c, 0x7e, 0x66, 0x66, 0x66, 0x66, 0x7e, 0x7c}
E	{0x7e, 0x7e, 0x60, 0x7c, 0x7c, 0x60, 0x7e, 0x7e}
F	{0x7e, 0x7e, 0x60, 0x7c, 0x7c, 0x60, 0x60, 0x60}
G	{0x3e, 0x7e, 0x60, 0x6e, 0x6e, 0x66, 0x7e, 0x3e}
H	{0x66, 0x66, 0x66, 0x7e, 0x7e, 0x66, 0x66, 0x66}
I	{0x7e, 0x7e, 0x18, 0x18, 0x18, 0x18, 0x7e, 0x7e}
J	{0x7e, 0x7e, 0x0c, 0x0c, 0x0c, 0x6c, 0x6c, 0x3c}
K	{0x62, 0x66, 0x6c, 0x78, 0x78, 0x6c, 0x66, 0x62}
L	{0x60, 0x60, 0x60, 0x60, 0x60, 0x60, 0x7e, 0x7e}
M	{0x63, 0x77, 0x7f, 0x6b, 0x63, 0x63, 0x63, 0x63}
N	{0x63, 0x73, 0x7b, 0x7b, 0x6f, 0x67, 0x67, 0x63}
O	{0x3c, 0x7e, 0x66, 0x66, 0x66, 0x66, 0x7e, 0x3c}
P	{0x7c, 0x7e, 0x62, 0x7e, 0x7c, 0x60, 0x60, 0x60}
Q	{0x3c, 0x42, 0x42, 0x42, 0x42, 0x4a, 0x46, 0x3f}
R	{0x7c, 0x7e, 0x62, 0x7e, 0x7c, 0x66, 0x66, 0x66}
S	{0x3e, 0x7e, 0x60, 0x7c, 0x7e, 0x06, 0x7e, 0x7c}
T	{0x7e, 0x7e, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18}
U	{0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x7e, 0x3c}
V	{0x66, 0x66, 0x66, 0x66, 0x66, 0x26, 0x3c, 0x18}
W	{0x63, 0x63, 0x63, 0x6b, 0x6b, 0x7f, 0x77, 0x22}
X	{0x66, 0x66, 0x3c, 0x18, 0x18, 0x3c, 0x66, 0x66}
Y	{0xc3, 0x66, 0x3c, 0x18, 0x18, 0x18, 0x18, 0x18}
Z	{0x7e, 0x7e, 0x06, 0x0c, 0x18, 0x30, 0x7e, 0x7e}
0	{0x3c, 0x7e, 0x66, 0x66, 0x66, 0x66, 0x7e, 0x3c}
1	{0x18, 0x38, 0x78, 0x18, 0x18, 0x18, 0x18, 0x7e}
2	{0x7c, 0x7e, 0x06, 0x1c, 0x38, 0x60, 0x7e, 0x7e}
3	{0x7c, 0x7e, 0x06, 0x3c, 0x3c, 0x06, 0x7e, 0x7c}
4	{0x66, 0x66, 0x66, 0x7e, 0x3e, 0x06, 0x06, 0x06}
5	{0x7e, 0x7e, 0x60, 0x7c, 0x7e, 0x02, 0x7e, 0x7c}
6	{0x3e, 0x7e, 0x60, 0x7c, 0x7e, 0x62, 0x7e, 0x3c}
7	{0x7e, 0x7e, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06}
8	{0x3c, 0x7e, 0x66, 0x3c, 0x3c, 0x66, 0x7e, 0x3c}
9	{0x3c, 0x7e, 0x66, 0x7e, 0x3e, 0x06, 0x7e, 0x7c}

Bảng 1: Bảng chuyển đổi giá trị HEX

Thực hiện tương tự bài LED 7 đoạn, ta sẽ thao tác trực tiếp trên GPIOB để truyền chuỗi dữ liệu. Chương trình tại các hàm hỗ trợ như sau:

```
void enable_matrix_col(int col_index){
  HAL_GPIO_WritePin(matrix_enable_pin[col_index].port,
   matrix_enable_pin[col_index].pin, GPIO_PIN_RESET);
}
void disable_matrix_col(int col_index){
  HAL_GPIO_WritePin(matrix_enable_pin[col_index].port,
   matrix_enable_pin[col_index].pin, GPIO_PIN_SET);
}
void display_matrix_character(int row_index){
  GPIOB->ODR = (GPIOB->ODR & (0xFF)) | (~led_matrix_character
   [led_matrix_value - 65][row_index] << 8);
}
void display_matrix_number(int row_index){
  GPIOB->ODR = (GPIOB->ODR & (0xFF)) | (~led_matrix_number[
   led_matrix_value][row_index] << 8);
}
```

Program 13: Hàm private trong chương trình led_matrix.c

```
void updateMatrixBuffer(char value){
  led_matrix_value = (int)value;
}
void updateMatrixRow(int row_index){
  for(uint16_t i = 0; i < NUM_ROW; i++){
    if(i == row_index)
      enable_matrix_col(i);
    if(i != row_index)
      disable_matrix_col(i);
  }
  if(led_matrix_value < 10)
    display_matrix_number(row_index);
  if(led_matrix_value <= 90 && led_matrix_value >= 65)
    display_matrix_character(row_index);
}
```

Program 14: Hàm public trong chương trình led_matrix.c

```
char matrix_value = 'A';
int matrix_row_index = 0;
```

Program 15: Các biến hỗ trợ trong chương trình main.c

```
updateMatrixRow(matrix_row_index);
updateMatrixBuffer(matrix_value);

if(timer[MATRIX_TIMER_INDEX].flag == 1){
  set_timer(MATRIX_TIMER_INDEX);
  matrix_row_index = (matrix_row_index + 1) % NUM_ROW;
}
```

Program 16: Nội dung bổ sung vào vòng while

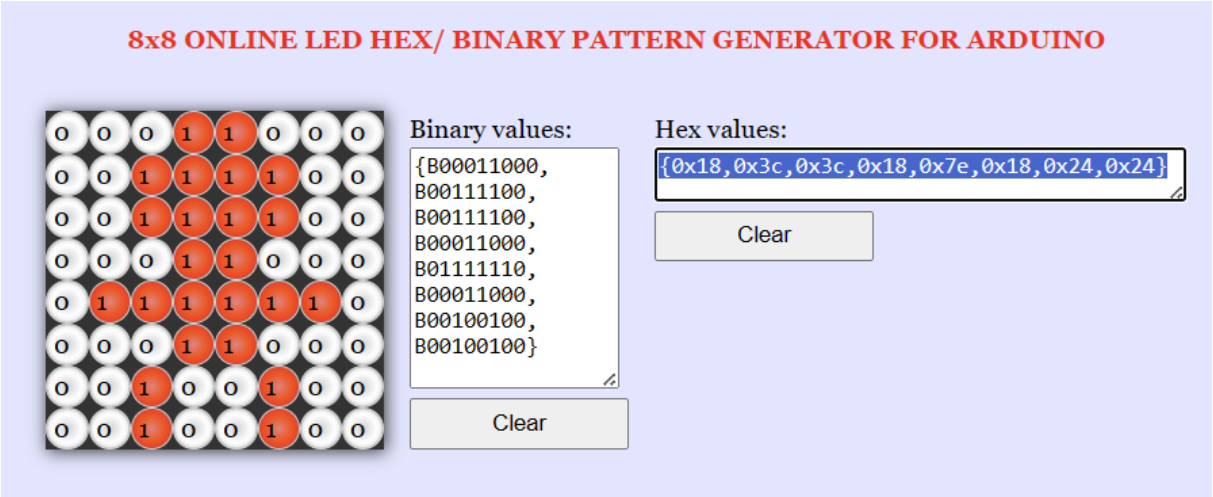
10 Exercise 10

10.1 Yêu cầu

Hiển thị hoạt ảnh trên đèn LED ma trận, ví dụ, một nhân vật di chuyển sang trái. Trình bày hướng giải quyết và source code.

10.2 Bài làm

Đối với chuỗi dữ liệu trên một dòng tương tự bài trước, nhân vật sẽ nằm ở giữa mà hình LED 7 đoạn. Bởi vì một dòng có kích thước 8 bit, nên để di chuyển từ phải sang trái thì tọa độ dịch chuyển tối đa (từ vị trí trung tâm) là: sang phải 7 vị trí, sang trái 7 vị trí hoặc ngữ nguyên tại vị trí giữa.



Hình 10: Hình ảnh nhân vật

Như vậy, để di chuyển nhân vật (với chu kỳ quét LED ma trận 800ms), em sẽ cho nhân vật dịch chuyển 1 tọa độ sau mỗi 1s. Giá trị đầu vào của dữ liệu hiển thị sẽ là tọa độ dịch chuyển.

```
void display_matrix_shape(int row_index){
  int updatedValue = led_matrix_shape[row_index];
  if(led_matrix_value < 7)
    updatedValue = updatedValue >> (7 - led_matrix_value);
  if(led_matrix_value > 7)
    updatedValue = updatedValue << (led_matrix_value - 7);
  GPIOB->ODR = (GPIOB->ODR & (0xFF)) | (~updatedValue << 8);
}
```

Program 17: Cập nhật hàm hiển thị và dịch chuyển nhân vật trong chương trình led_matrix.c