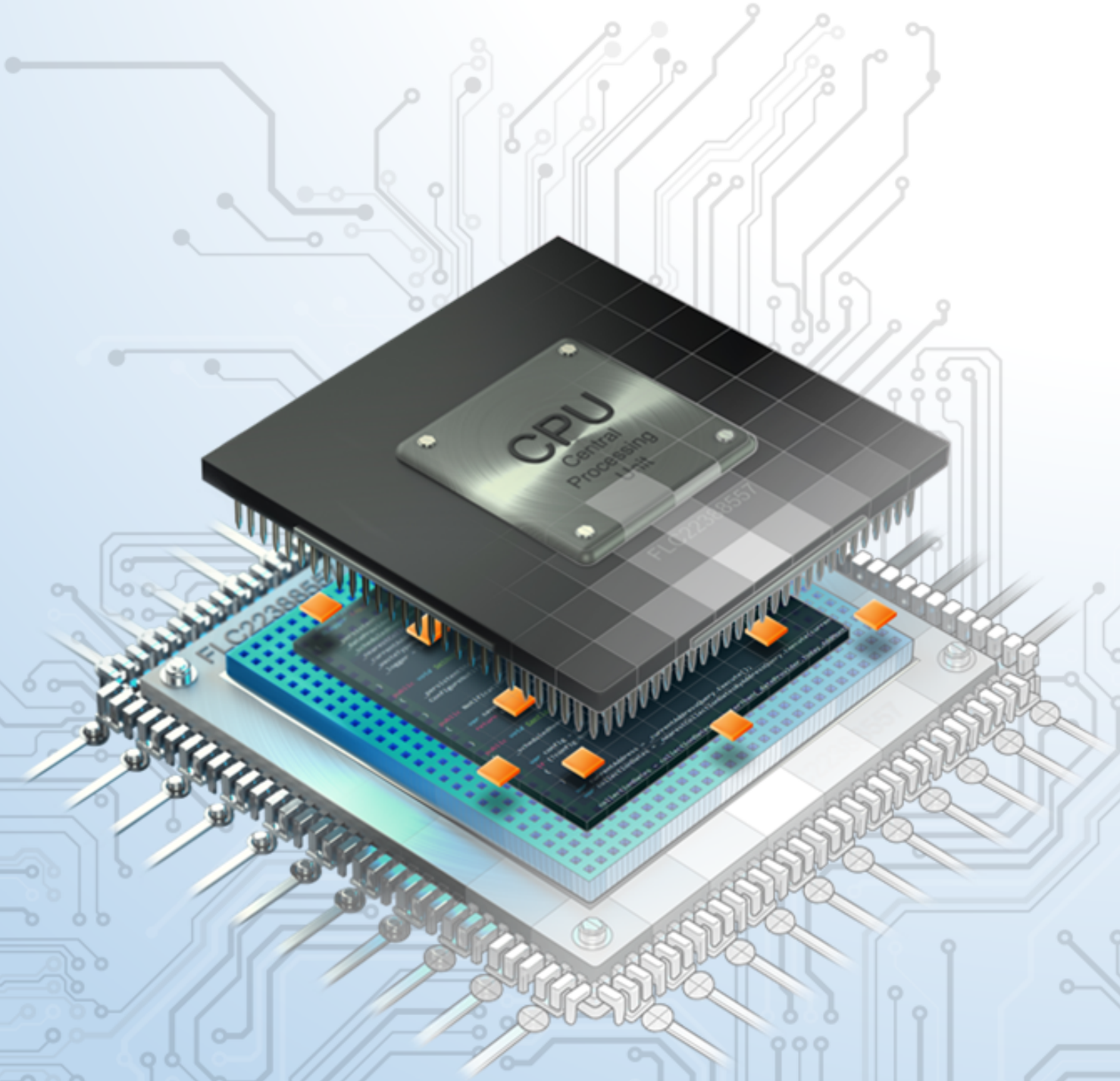
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

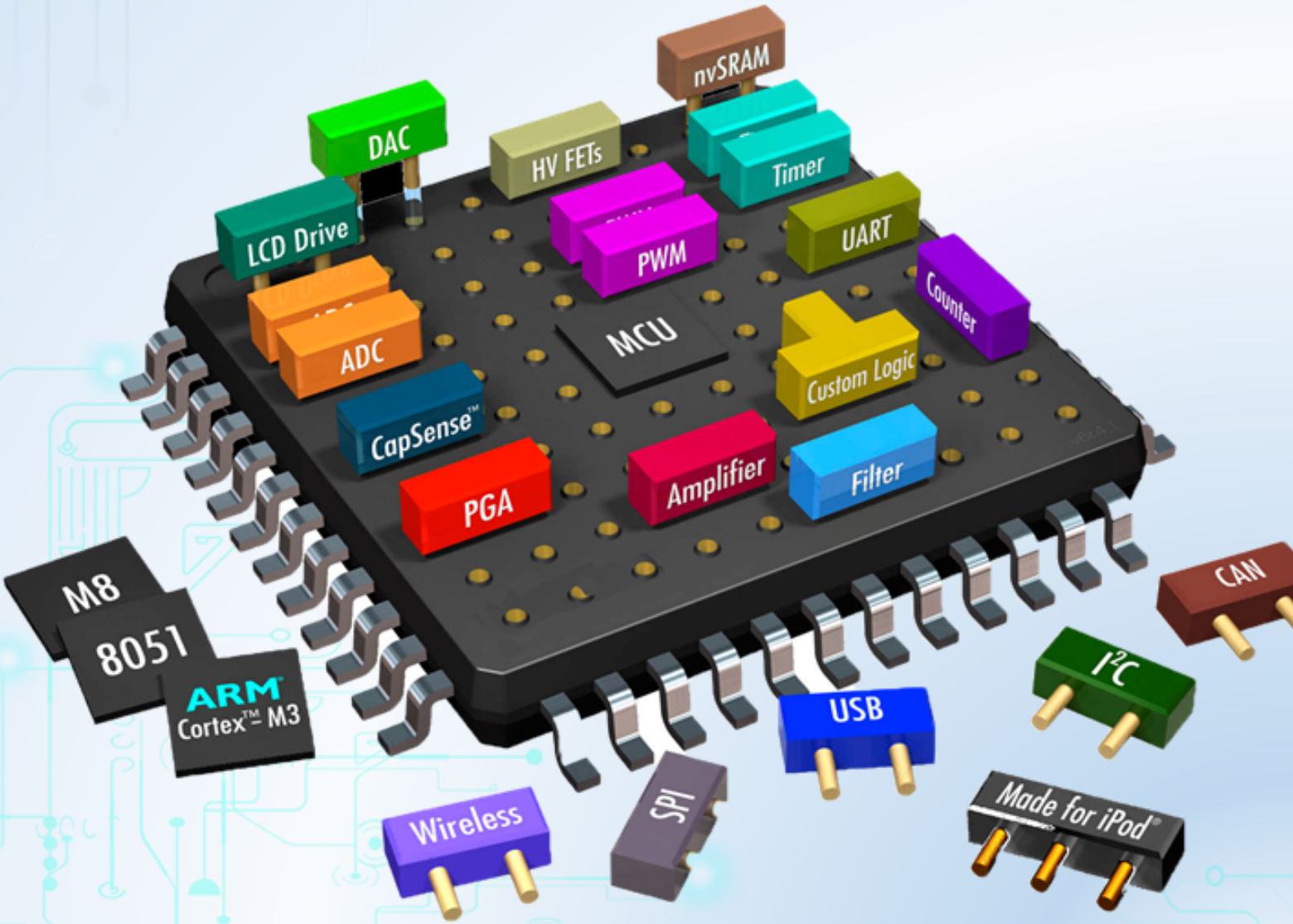


Dr. Le Trong Nhan

Mục lục

CHƯƠNG 1

LED Animations



1 Information

Student name: Lê Hoàng Minh Tú
Student_Id: 1915812

Topic: **Micro-controller lap 1**
Teaching Assistant: Huynh Phuc Nghi
Tutorial professor: Le Trong Nhan
Public_day: 13-09-2021, 3:23 p.m.
Class: L03-L05
Company: Ho Chi Minh University of Technology

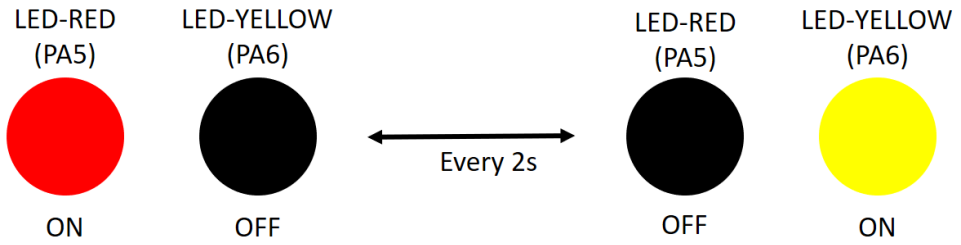
For more information, please contact via email: tu.le_handspro@hcmut.edu.vn

2 Exercise and Report

2.1 Exercise 1

From the simulation on Proteus, one more LED is connected to pin **PA6** of the STM32 (negative pin of the LED is connected to PA6). The component suggested in this exercise is **LED-YELLOW**, which can be found from the device list.

In this exercise, the status of two LEDs are switched every 2 seconds, as demonstrated in the figure bellow.



Hình 1.1: State transitions for 2 LEDs

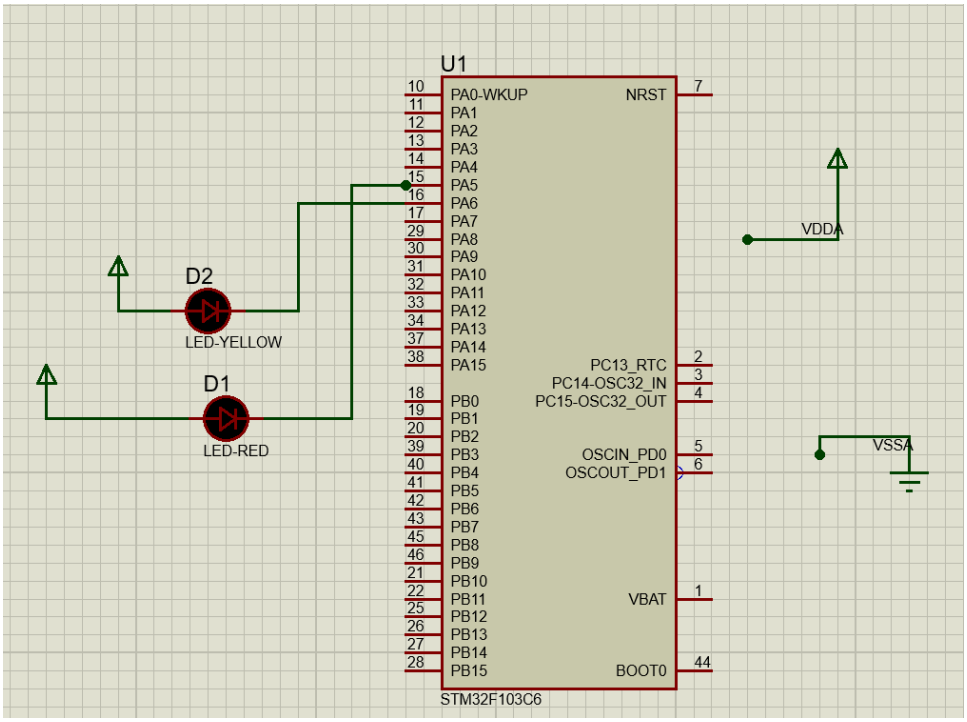
Report 1: Depict the schematic from Proteus simulation in this report. The caption of the figure is a downloadable link to the Proteus project file (e.g. a github link).

Report 2: Present the source code in the infinite loop while of your project. If a user-defined functions is used, it is required to present in this part. A brief description can be added for this function (e.g. using comments). A template to present your source code is presented bellow.

Solution:

```
// A5: led red
// A6: led yellow
// 0 is led on, 1 is led off
////////////////////////////
// A5 = 0, A6 = 1 (2s)
// A5 = 1, A6 = 0 (2s)
// set the begin case
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, SET);
while (1)
{
    /* USER CODE END WHILE */
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_6);
    HAL_Delay(2000);
    /* USER CODE BEGIN 3 */
}
```

Program 1.1: Toggle Pin



Hình 1.2: *Link GitHub*

2.2 Exercise 2

Extend the first exercise to simulate the behavior of a traffic light. A third LED, named **LED-GREEN** is added to the system, which is connected to **PA7**. A cycle in this traffic light is 5 seconds for the RED, 2 seconds for the YELLOW and 3 seconds for the GREEN. The LED-GREEN is also controlled by its negative pin.

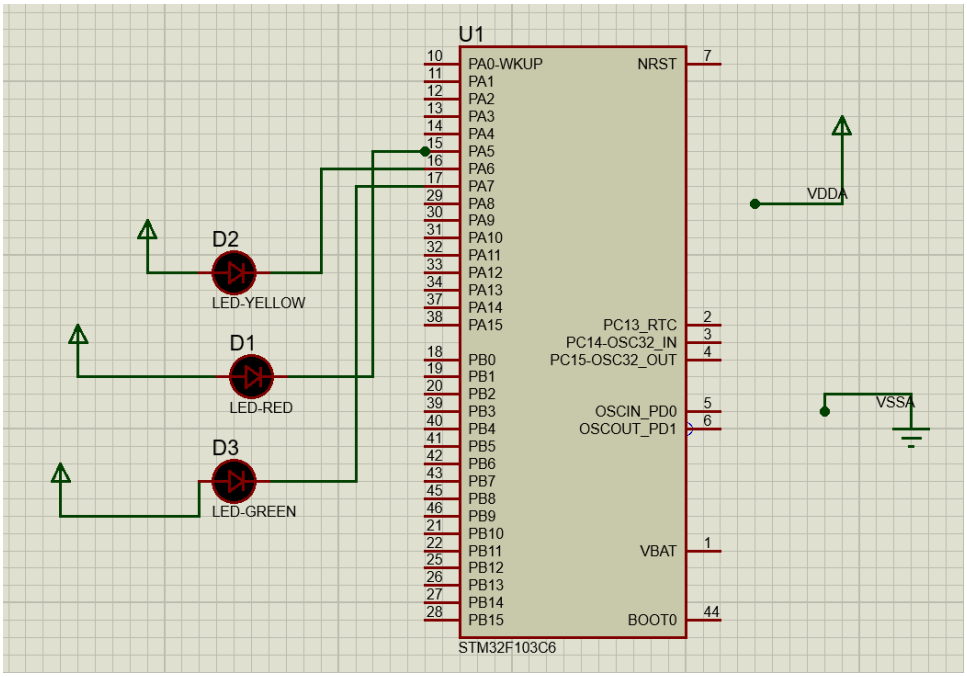
Similarly, the report in this exercise includes the schematic of your circuit and a your source code in the while loop.
Report 1: Present the schematic.

Report 2: Present the source code in while.
Solution:

```
// A5: led red
// A6: led yellow
// A7: led green
// 0 is led on, 1 is led off
////////////////////////////
// A5 = 0, A6 = 1, A7 = 1 (5s)
// A5 = 1, A6 = 0, A7 = 1 (2s)
// A5 = 1, A6 = 1, A7 = 0 (3s)
// set the begin case
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
while (1)
{
    /* USER CODE END WHILE */
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_7);
```

```
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    HAL_Delay(5000);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5, SET);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_6, SET);
    HAL_Delay(2000);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_6, SET);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_7, SET);
    HAL_Delay(3000);
    /* USER CODE BEGIN 3 */
}
```

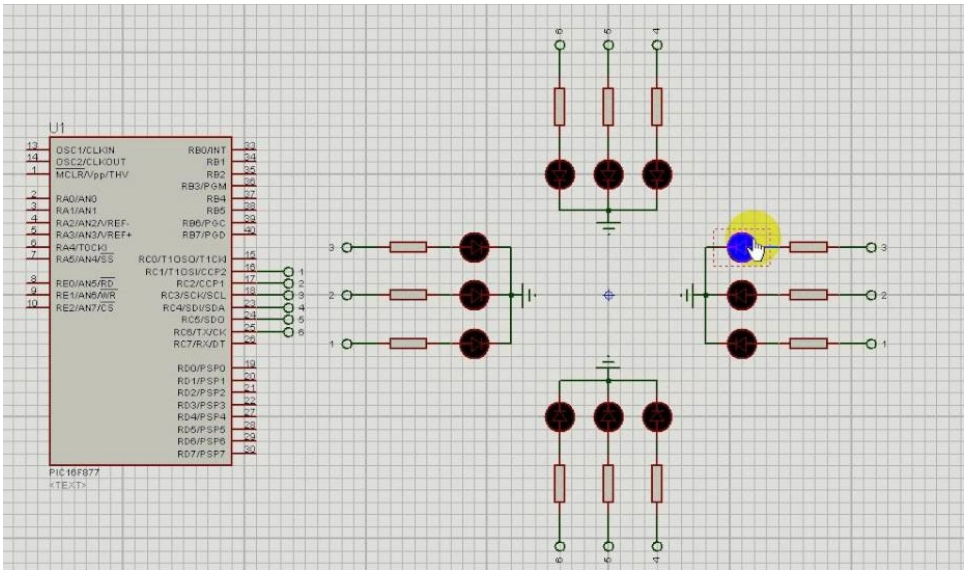
Program 1.2: Controll 1 traffic light



Hình 1.3: *Link GitHub*

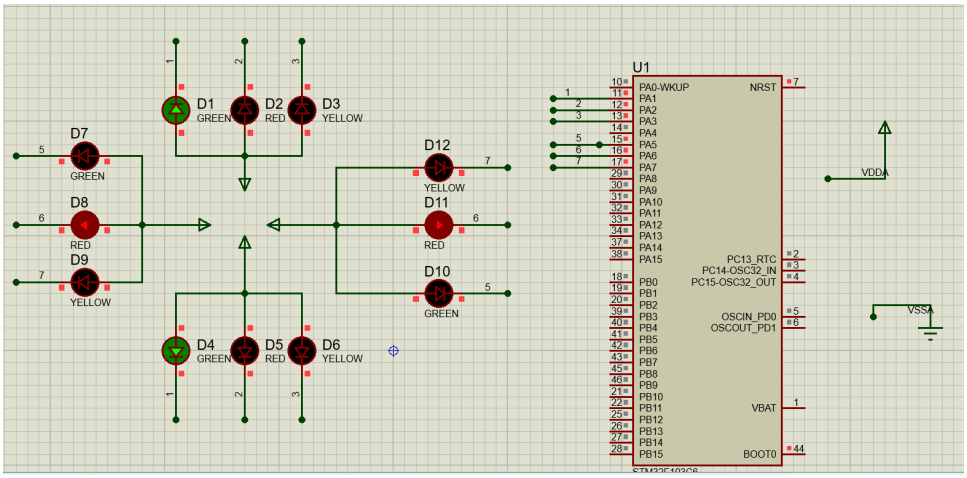
2.3 Exercise 3

Extend to the 4-way traffic light. Arrange 12 LEDs in a nice shape to simulate the behaviors of a traffic light. A reference design can be found in the figure bellow.



Hình 1.4: *Reference design for a 4 way traffic light*

Solution: To simulizing the 4-way traffic. the traffic light with the one at the opposite side share the same light color. So, we just need 6 pins for controlling 12 leds.



Hình 1.5: *Link GitHub*

Here is a source code. We have A5 to A7 are used for 2 vertical-traffic lights. And A1 to A3 are used for 2 horizontal lights.

```
// A5: led green (3s)
// A6: led red (5s)
// A7: led yellow (2s)
//////////////////////////////////
// A1: green led
// A2: red led
```

```
// A3: yellow led
// 0 is led on, 1 is led off
//////////////////////
//A5 = 1, A6 = 0, A7 = 1 (5s) red led | green   A1 = 0, A2 = 1, A3 = 1 (3s)
//A5 = 0, A6 = 1, A7 = 1 (2s) green   | yellow  A1 = 1, A2 = 1, A3 = 0 (2s)
//A5 = 1, A6 = 1, A7 = 0 (3s) yellow  | red   A1 = 1, A2 = 0, A3 = 1 (5s)

//set first state
//left
HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_7);
//right
HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_2);
HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_3);
int sec = 1;
while (1)
{
/* USER CODE END WHILE */
  if(sec==3){
    //right
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_1);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_3);
  }
  else if(sec==5){
    //left
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_6);
    //right
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_2);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_3);
  }
  else if(sec==7){
    //left
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_7);
  }
  else if(sec==10){
    //left
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_6);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_7);
    //right
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_1);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_2);
    sec = 0;
  }
  sec = sec + 1;
```

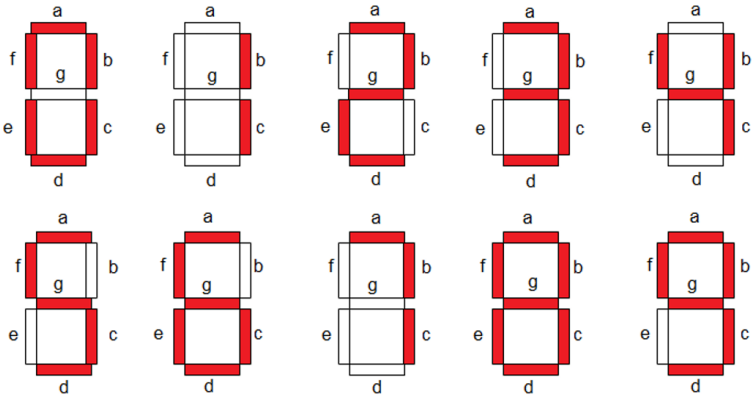
```
53   HAL_Delay(1000);
54 /* USER CODE BEGIN 3 */
55 }
```

Program 1.3: 4-way traffic lights

2.4 Exercise 4

Add **only one 7 led segment** to the schematic in Exercise 3. This component can be found in Proteus by the keyword **7SEG-COM-ANODE**. For this device, the common pin should be connected to the power supply and other pins are supposed to connected to PB0 to PB6. Therefore, to turn-on a segment in this 7SEG, the STM32 pin should be in logic 0 (0V).

Implement a function named **display7SEG(int num)**. The input for this function is from 0 to 9 and the outputs are listed as following:



Hình 1.6: Display a number on 7 segment LED

This function is invoked in the while loop for testing as following:

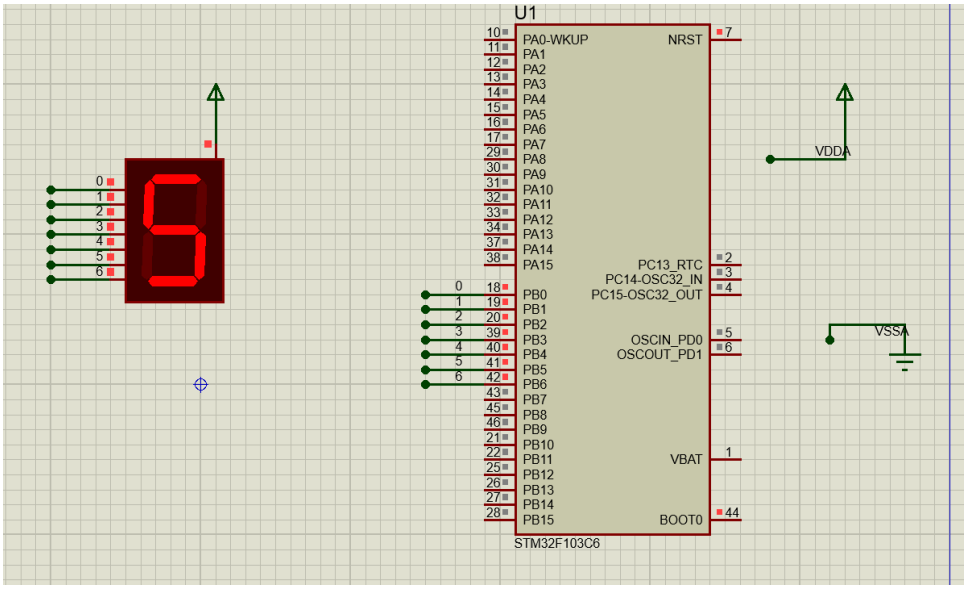
```
int counter = 0;
while (1){
    if(counter >= 10) counter = 0;
    display7SEG(counter++);
    HAL_Delay(1000);

}
```

Program 1.4: An example for your source code

Report 1: Present the schematic.

Report 2: Present the source code for display7SEG function.

Solution: 7SEG_COM_ANODE is controled by 7 pins from a to g, if one of these pins is in logic 0, that part will light up. So depending on that details, we can prepresent number by passing the right logic of a to g pin.



Hình 1.7: *Link GitHub*

With type is a declaration of GPIO type (which is GPIOA, or GPIOB, ...), and uint_16 from A to G are the GPIO_PIN_NUMBER.

```
void display7SEG (int num, GPIO_TypeDef* type, uint16_t A,
   uint16_t B, uint16_t C, uint16_t D, uint16_t E, uint16_t
    F, uint16_t G){
  if(num == 0){
    HAL_GPIO_WritePin(type, A, RESET);
    HAL_GPIO_WritePin(type, B, RESET);
    HAL_GPIO_WritePin(type, C, RESET);
    HAL_GPIO_WritePin(type, D, RESET);
    HAL_GPIO_WritePin(type, E, RESET);
    HAL_GPIO_WritePin(type, F, RESET);
    HAL_GPIO_WritePin(type, G, SET);
```

```
  }
  else if(num == 1){
    HAL_GPIO_WritePin(type, A, SET);
    HAL_GPIO_WritePin(type, B, RESET);
    HAL_GPIO_WritePin(type, C, RESET);
    HAL_GPIO_WritePin(type, D, SET);
    HAL_GPIO_WritePin(type, E, SET);
    HAL_GPIO_WritePin(type, F, SET);
    HAL_GPIO_WritePin(type, G, SET);
  }
  else if(num == 2){
    HAL_GPIO_WritePin(type, A, RESET);
    HAL_GPIO_WritePin(type, B, RESET);
    HAL_GPIO_WritePin(type, C, SET);
    HAL_GPIO_WritePin(type, D, RESET);
    HAL_GPIO_WritePin(type, E, RESET);
    HAL_GPIO_WritePin(type, F, SET);
    HAL_GPIO_WritePin(type, G, RESET);
  }
  else if(num == 3){
    HAL_GPIO_WritePin(type, A, RESET);
    HAL_GPIO_WritePin(type, B, RESET);
    HAL_GPIO_WritePin(type, C, RESET);
    HAL_GPIO_WritePin(type, D, RESET);
    HAL_GPIO_WritePin(type, E, SET);
    HAL_GPIO_WritePin(type, F, SET);
    HAL_GPIO_WritePin(type, G, RESET);
  }
  else if(num == 4){
    HAL_GPIO_WritePin(type, A, SET);
    HAL_GPIO_WritePin(type, B, RESET);
    HAL_GPIO_WritePin(type, C, RESET);
    HAL_GPIO_WritePin(type, D, SET);
    HAL_GPIO_WritePin(type, E, SET);
    HAL_GPIO_WritePin(type, F, RESET);
    HAL_GPIO_WritePin(type, G, RESET);
  }
  else if(num == 5){
    HAL_GPIO_WritePin(type, A, RESET);
    HAL_GPIO_WritePin(type, B, SET);
    HAL_GPIO_WritePin(type, C, RESET);
    HAL_GPIO_WritePin(type, D, RESET);
    HAL_GPIO_WritePin(type, E, SET);
    HAL_GPIO_WritePin(type, F, RESET);
    HAL_GPIO_WritePin(type, G, RESET);
  }
  else if(num == 6){
    HAL_GPIO_WritePin(type, A, RESET);
    HAL_GPIO_WritePin(type, B, SET);
```

```
59      HAL_GPIO_WritePin(type, C, RESET);
60      HAL_GPIO_WritePin(type, D, RESET);
61      HAL_GPIO_WritePin(type, E, RESET);
62      HAL_GPIO_WritePin(type, F, RESET);
63      HAL_GPIO_WritePin(type, G, RESET);
64    }
65    else if(num == 7){
66      HAL_GPIO_WritePin(type, A, RESET);
67      HAL_GPIO_WritePin(type, B, RESET);
68      HAL_GPIO_WritePin(type, C, RESET);
69      HAL_GPIO_WritePin(type, D, SET);
70      HAL_GPIO_WritePin(type, E, SET);
71      HAL_GPIO_WritePin(type, F, SET);
72      HAL_GPIO_WritePin(type, G, SET);
73    }
74    else if(num == 8){
75      HAL_GPIO_WritePin(type, A, RESET);
76      HAL_GPIO_WritePin(type, B, RESET);
77      HAL_GPIO_WritePin(type, C, RESET);
78      HAL_GPIO_WritePin(type, D, RESET);
79      HAL_GPIO_WritePin(type, E, RESET);
80      HAL_GPIO_WritePin(type, F, RESET);
81      HAL_GPIO_WritePin(type, G, RESET);
82    }
83    else if(num == 9){
84      HAL_GPIO_WritePin(type, A, RESET);
85      HAL_GPIO_WritePin(type, B, RESET);
86      HAL_GPIO_WritePin(type, C, RESET);
87      HAL_GPIO_WritePin(type, D, RESET);
88      HAL_GPIO_WritePin(type, E, SET);
89      HAL_GPIO_WritePin(type, F, RESET);
90      HAL_GPIO_WritePin(type, G, RESET);
91    }
92    else{
93      HAL_GPIO_WritePin(type, A, SET);
94      HAL_GPIO_WritePin(type, B, SET);
95      HAL_GPIO_WritePin(type, C, SET);
96      HAL_GPIO_WritePin(type, D, SET);
97      HAL_GPIO_WritePin(type, E, SET);
98      HAL_GPIO_WritePin(type, F, SET);
99      HAL_GPIO_WritePin(type, G, SET);
100   }
101 }
```

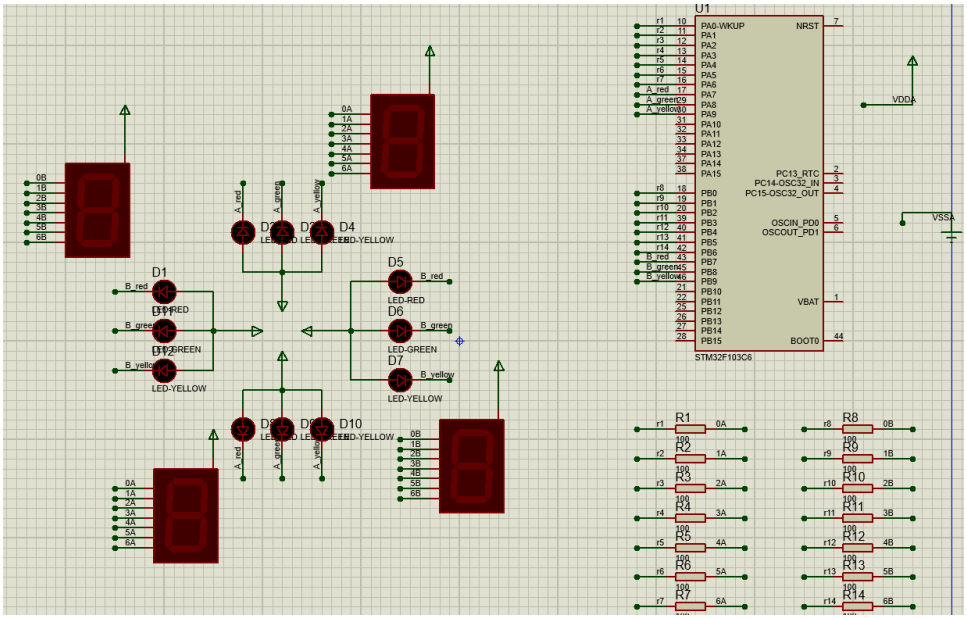
Program 1.5: Controlling 1 7SEG-LED

2.5 Exercise 5

Integrate the 7SEG-LED to the 4 way traffic light. In this case, the 7SEG-LED is used to display countdown value.

In this exercise, only source code is required to present. The function display7SEG in previous exercise can be re-used.

Solution: Base on exercise 4, we now can control the display of 7SEG-LED. Now let's implementate in the real traffic light



Hình 1.8: *Link GitHub*

The cycle of 1 traffic light is 20 sec, consisting of 10 sec for red light, 6 sec for green, and 4 sec for yellow, so, the variable "counter" show that the sec in traffic light cycle.

```
// pin PA0->6 use for 7SEG led A, PB0->6 for 7SEG B
// pin PA7->9 is red (10s), green (6s), yellow (4s) traffic
    light for led A
// pin PB7->9 is red (10s), green (6s), yellow (4s) traffic
    light for led B
////////////////////////////////////////////////////
int counter = 0;
while (1)
{
/* USER CODE END WHILE */
  if(counter >= 20) counter = 0;

  //traffic light A
  if(counter <= 9){ //red light
    display7SEG(9-counter, GPIOA, GPIO_PIN_0, GPIO_PIN_1,
   GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5,
   GPIO_PIN_6);
```

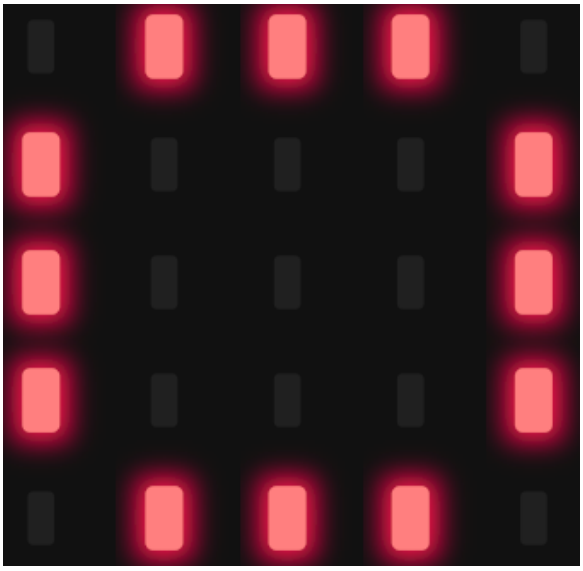
```
14      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, RESET);
15      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
16      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
17    }
18    else if(counter <= 15){ //green light
19      display7SEG(15-counter, GPIOA, GPIO_PIN_0, GPIO_PIN_1,
      GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5,
      GPIO_PIN_6);
20      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
21      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, RESET);
22      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
23    }
24    else{ //yellow light
25      display7SEG(19-counter, GPIOA, GPIO_PIN_0, GPIO_PIN_1,
      GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5,
      GPIO_PIN_6);
26      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
27      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
28      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, RESET);
29    }
30
31    //traffic light B
32    if(counter <= 5){ //green light
33      display7SEG(5-counter, GPIOB, GPIO_PIN_0, GPIO_PIN_1,
      GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5,
      GPIO_PIN_6);
34      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, SET);
35      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, RESET);
36      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, SET);
37    }
38    else if(counter <= 9){ //yellow light
39      display7SEG(9-counter, GPIOB, GPIO_PIN_0, GPIO_PIN_1,
      GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5,
      GPIO_PIN_6);
40      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, SET);
41      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, SET);
42      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, RESET);
43    }
44    else{ //red light
45      display7SEG(19-counter, GPIOB, GPIO_PIN_0, GPIO_PIN_1,
      GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5,
      GPIO_PIN_6);
46      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, RESET);
47      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, SET);
48      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, SET);
49    }
50
51    counter ++;
52    HAL_Delay(1000);
```

```
/* USER CODE BEGIN 3 */
}
```

Program 1.6: 4-way traffic light (full version)

2.6 Exercise 6

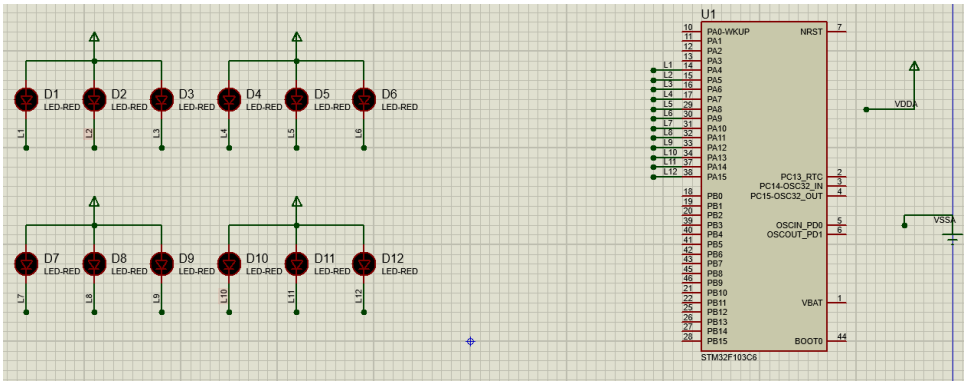
In this exercise, a new Proteus schematic is designed to simulate an analog clock, with 12 different number. The connections for 12 LEDs are supposed from PA4 to PA15 of the STM32. The arrangement of 12 LEDs is depicted as follows.



Hình 1.9: 12 LEDs for an analog clock

Report 1: Present the schematic.
Report 2: Implement a simple program to test the connection of every single LED. This testing program should turn every LED in a sequence.

Solution:



Hình 1.10: Link GitHub

```
// set begin case: (turn off all lights)
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, SET);
```

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);
/////////////////////////////////////////////////////////
while (1)
{
/* USER CODE END WHILE */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, RESET);
  HAL_Delay(500);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, SET);
```

```
52   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, RESET);
53   HAL_Delay(500);
54 /* USER CODE BEGIN 3 */
55 }
```

Program 1.7: Check connection

2.7 Exercise 7

Implement a function named **clearAllClock()** to turn off all 12 LEDs. Present the source code of this function.

```
1 void clearAllClock(){
2     //TODO
3 }
```

Program 1.8: Function Implementation

Solution:

```
1  void clearAllClock()
2  {
3    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, SET);
4    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, SET);
5    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
6    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
7    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
8    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
9    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, SET);
10   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, SET);
11   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, SET);
12   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, SET);
13   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, SET);
14   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);
15 }
```

Program 1.9: clearAllClock

2.8 Exercise 8

Implement a function named **setNumberOnClock(int num)**. The input for this function is from **0 to 11** and an appropriate LED is turn on. Present the source code of this function.

Solution:

```
1  void setNumberOnClock(int num)
2  {
3    if(num == 0) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, RESET);
4    else if(num == 1) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5,
     RESET);
5    else if(num == 2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6,
     RESET);
```

```
6   else if(num == 3) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7,
    RESET);
7   else if(num == 4) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,
    RESET);
8   else if(num == 5) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9,
    RESET);
9   else if(num == 6) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10,
    RESET);
10  else if(num == 7) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11,
    RESET);
11  else if(num == 8) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12,
    RESET);
12  else if(num == 9) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13,
    RESET);
13  else if(num == 10) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14,
    RESET);
14  else if(num == 11) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15,
    RESET);
15 }
```

Program 1.10: setNumberOnClock

2.9 Exercise 9

Implement a function named **clearNumberOnClock(int num)**. The input for this function is from **0 to 11** and an appropriate LED is turn off.

Solution:

```
1  void clearNumberOnClock(int num)
2  {
3    if(num == 0) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, SET);
4    else if(num == 1) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5,
     SET);
5    else if(num == 2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6,
     SET);
6    else if(num == 3) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7,
     SET);
7    else if(num == 4) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,
     SET);
8    else if(num == 5) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9,
     SET);
9    else if(num == 6) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10,
     SET);
10   else if(num == 7) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11,
     SET);
11   else if(num == 8) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12,
     SET);
12   else if(num == 9) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13,
     SET);
```

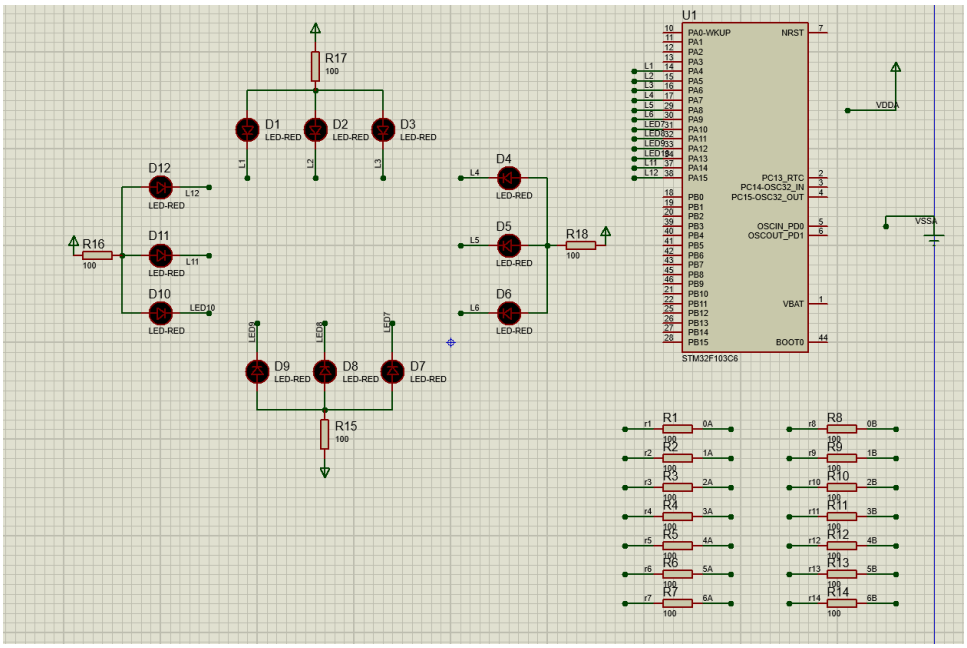
```
  else if(num == 10) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14,
   SET);
  else if(num == 11) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15,
   SET);
}
```

Program 1.11: clearNumberOnClock

2.10 Exercise 10

Integrate the whole system and use 12 LEDs to display a clock. At a given time, there are only 3 LEDs are turn on for hour, minute and second information.

Solution: Here, we are using 12 red leds to prepresent a simple clock, using from PA4 to PA15 to controll. And base on function we have built in exercise 8, and 9. In this clock project, we can also adjust the begin time.



Hình 1.11: *Link GitHub*

These codes below has been minimize the delay time for shorten the demo.

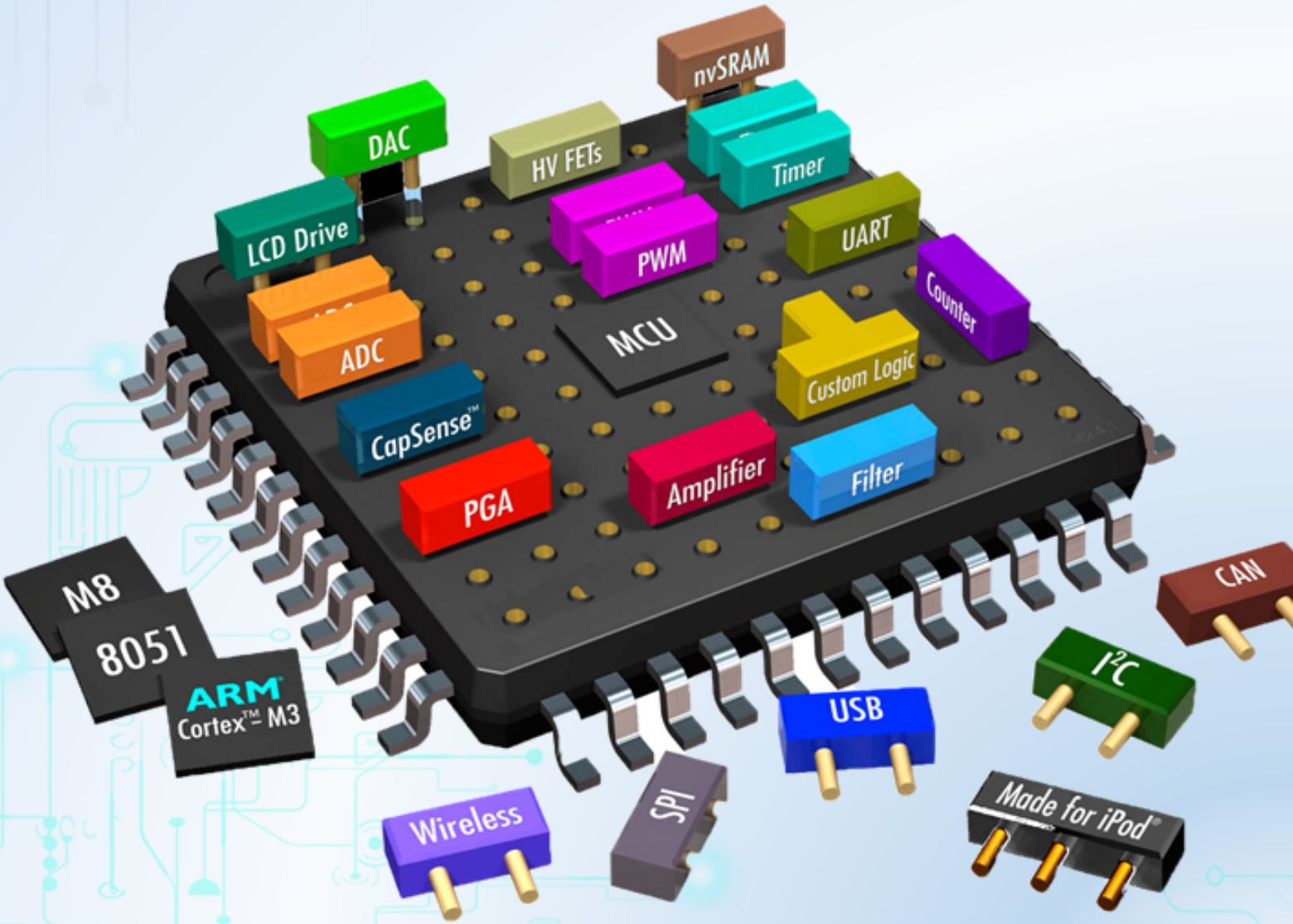
```
// set begin case:
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, SET);
```

```
13 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);
14 ////////////////////////////////////////////////////////
15 // set the begin time here
16 ////////////////////////////
17 int hour = 0;
18 int min = 0;
19 int sec = 0;
20 while (1)
21 {
22 /* USER CODE END WHILE */
23   setNumberOnClock(hour);
24   setNumberOnClock(min);
25   setNumberOnClock(sec);
26   HAL_Delay(400);
27   clearNumberOnClock(hour);
28   clearNumberOnClock(min);
29   clearNumberOnClock(sec);
30
31   sec++;
32
33   if(sec == 12) {
34     min++;
35     sec = 0;
36   }
37   if(min == 12){
38     hour++;
39     min = 0;
40   }
41   if(hour == 12){
42     hour = 0;
43   }
44 /* USER CODE BEGIN 3 */
45 }
```

Program 1.12: A clock

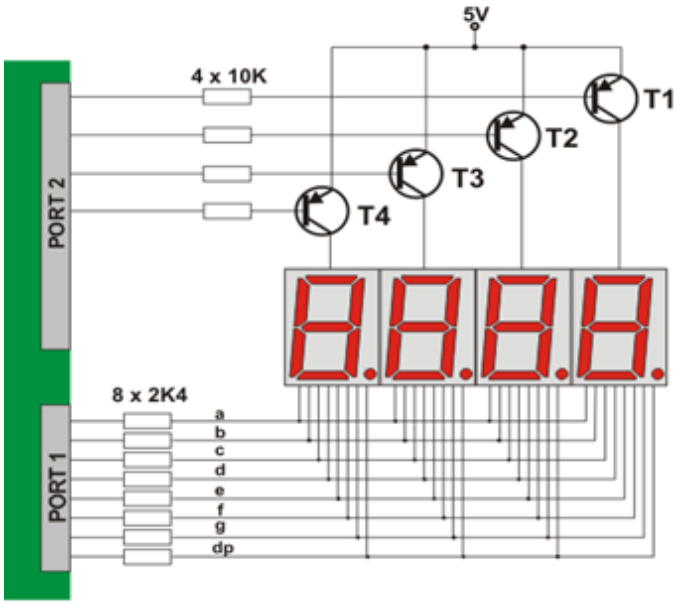
CHƯƠNG 2

Timer Interrupt and LED Scanning



1 Introduction

Timers are one of the most important features in modern micro-controllers. They allow us to measure how long something takes to execute, create non-blocking code, precisely control pin timing, and even run operating systems. In this manual, how to configure a timer using STM32CubeIDE is presented how to use them to flash an LED. Finally, students are proposed to finalize 10 exercises using timer interrupt for applications based LED Scanning.



Hình 2.1: *Four seven segment LED interface for a micro-controller*

Design an interface for with multiple LED (seven segment or matrix) displays which is to be controlled is depends on the number of input and output pins needed for controlling all the LEDs in the given matrix display, the amount of current that each pin can source and sink and the speed at which the micro-controller can send out control signals. With all these specifications, interfacing can be done for 4 seven segment LEDs with a micro-controller is proposed in the figure above.

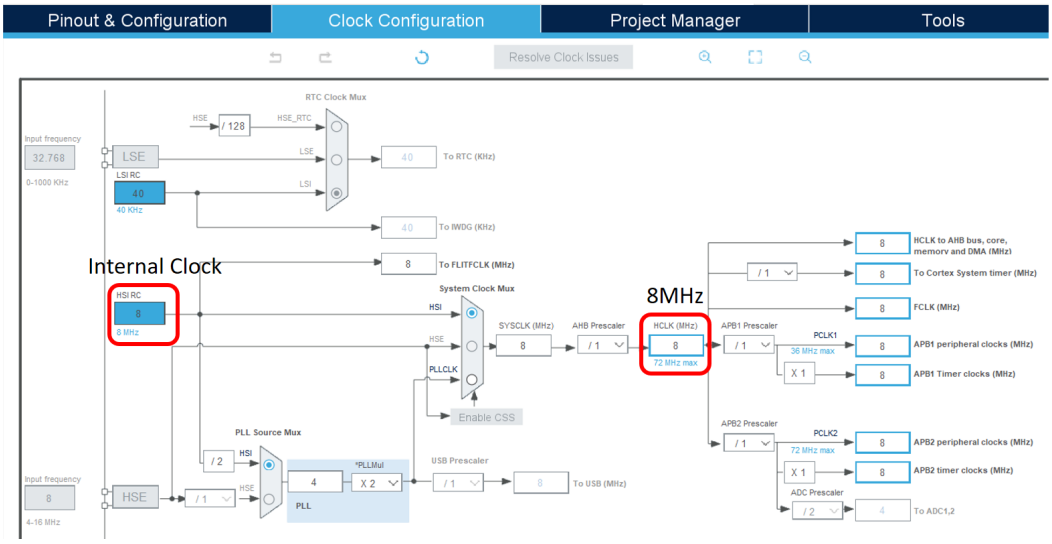
In the above diagram each seven segment display is having 8 internal LEDs, leading to the total number of LEDs is 32. However, not all the LEDs are required to turn ON, but one of them is needed. Therefore, only 12 lines are needed to control the whole 4 seven segment LEDs. By controlling with the micro-controller, we can turn ON an LED during a same interval T_S. Therfore, the period for controlling all 4 seven segment LEDs is $4T_S$. In other words, these LEDs are scanned at frequecy $f = 1/4T_S$. Finally, it is obviously that if the frequency is greater than 30Hz (e.g. f = 50Hz), it seems that all LEDs are turn ON at the same time.

In this manual, the timer interrupt is used to design the interval T_S for LED scanning. Unfortunately, the simulation on Proteus can not execute at high frequency, the frequency f is set to a low value (e.g. 1Hz). In a real implementation, this frequency should be 50Hz.

2 Timer Interrupt Setup

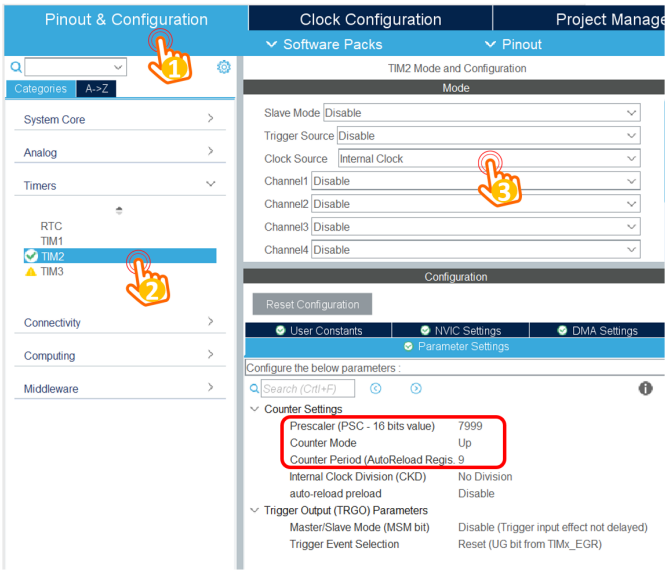
Step 1: Create a simple project, which LED connected to PA5. The manual can be found in the first lab.

Step 2: Check the clock source of the system on the tab **Clock Configuration** (from *.ioc file). In the default configuration, the internal clock source is used with 8MHz, as shown in the figure bellow.



Hình 2.2: Default clock source for the system

Step 3: Configure the timer on the **Parameter Settings**, as follows:



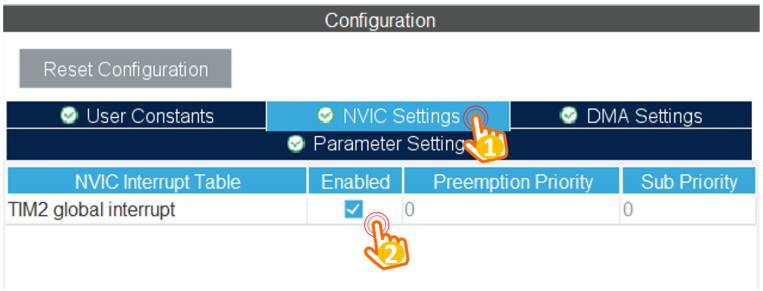
Hình 2.3: Configure for Timer 2

Select the clock source for timer 2 to the **Internal Clock**. Finally, set the prescaller and the counter to 7999 and 9, respectively. These values are explained as follows:

- The target is to set an interrupt timer to 10ms

- The clock source is 8MHz, by setting the prescaller to 7999, the input clock source to the timer is **8MHz/(7999+1) = 1000Hz**.
- The interrupt is raised when the timer counter is counted from 0 to 9, meaning that the frequency is divided by 10, which is 100Hz.
- The frequency of the timer interrupt is 100Hz, meaning that the period is **1/100Hz = 10ms**.

Step 4: Enable the timer interrupt by switching to **NIVC Settings** tab, as follows:



Hình 2.4: Enable timer interrupt

Finally, save the configuration file to generate the source code.

Step 5: On the **main()** function, call the timer init function, as follows:

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

  MX_GPIO_Init();
  MX_TIM2_Init();

  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */

  while (1){

  }
}
```

Program 2.1: Init the timer interrupt in main

Please put the init function in a right place to avoid conflicts when code generation is executed (e.g. ioc file is updated).

Step 6: Add the interrupt service routine function, this function is invoked every 10ms, as follows:

```
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
   {

}
/* USER CODE END 4 */
```

Program 2.2: Add an interrupt service routine

Step 7: To run a LED Blinky demo using interrupt, a short manual is presented as follows:

```
/* USER CODE BEGIN 4 */
int counter = 100;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
   {
  counter--;
  if(counter <= 0){
    counter = 100;
    HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
  }
}
/* USER CODE END 4 */
```

Program 2.3: LED Blinky using timer interrupt

The **HAL_TIM_PeriodElapsedCallback** function is an infinite loop, which is invoked every cycle of the timer 2, in this case, is 10ms.

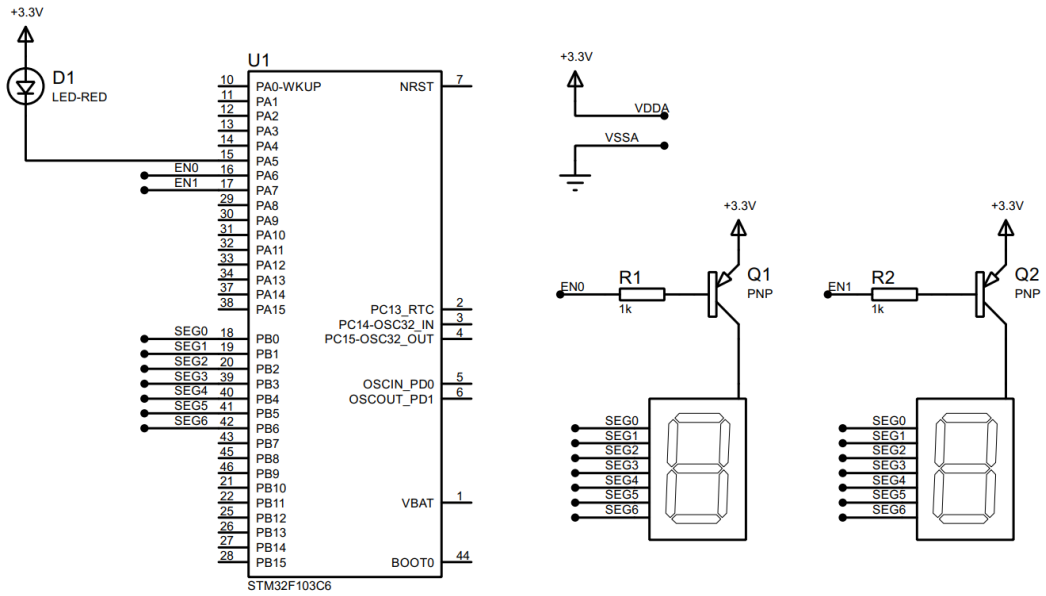
3 Exercise and Report

3.1 Exercise 1

The first exercise show how to interface for multiple seven segment LEDs to STM32F103C6 micro-controller (MCU). Seven segment displays are common anode type, meaning that the anode of all LEDs are tied together as a single terminal and cathodes are left alone as individual pins.

In order to save the resource of the MCU, individual cathode pins from all the seven segment LEDs are connected together, and connect to 7 pins of the MCU. These pins are popular known as the **signal pins**. Meanwhile, the anode pin of each seven segment LEDs are controlled under a power enabling circuit, for instance, an PNP transistor. At a given time, only one seven segment LED is turned on. However, if the delay is small enough, it seems that all LEDs are enabling.

Implement the circuit simulation in Proteus with two 7-SEGMENT LEDs as following:



Hình 2.5: *Simulation schematic in Proteus*

Components used in the schematic are listed bellow:

- 7SEG-COM-ANODE (connected from PB0 to PB6)
- LED-RED
- PNP
- RES
- STM32F103C6

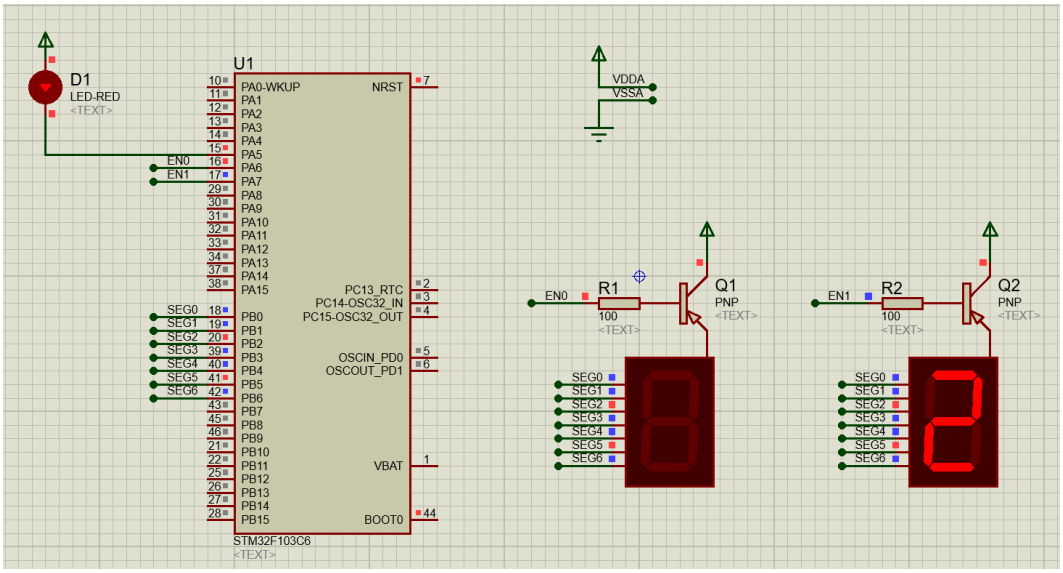
Students are proposed to use the function **display7SEG(int num)** in the Lab 1 in this exercise. Implement the source code in the interrupt callback function to display number **"1"** on the first seven segment and number **"2"** for second one. The

switching time between 2 LEDs is half of second.

Report 1: Capture your schematic from Proteus and show in the report.

Report 2: Present your source code in the **HAL_TIM_PeriodElapsedCallback** function.

Solution:



Hình 2.6: Simulation schematic in Proteus

```c
/* USER CODE BEGIN 4 */
// state = 0 -> print 1
// state = 1 -> print 2
int state = 0;
int counter = 0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim
   ){
  switch(state){
  case(0):
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
      display7SEG(1);
      if(counter >= 50){
        counter = 0;
        state = 1;
      }
      break;
  case(1):
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, RESET);
      display7SEG(2);
      if(counter >= 50){
        counter = 0;
```

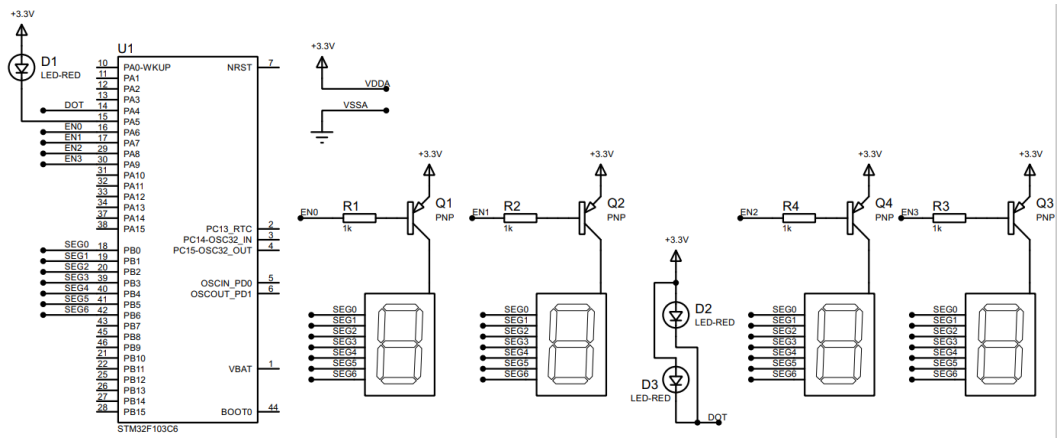
```
            state = 0;
        }
        break;
    }
    counter ++;
}
/* USER CODE END 4 */
```

Short question: What is the frequency of the scanning process?

⇒ ANS: the frequency of 2 led seg is 1Hz, which means that it's cycle take 1s long.

3.2 Exercise 2

Extend to 4 seven segment LEDs and two LEDs (connected to PA4, labeled as **DOT**) in the middle as following:



Hình 2.7: Simulation schematic in Proteus

Blink the two LEDs every second. Meanwhile, number 3 is displayed on the third seven segment and number 0 is displayed on the last one (to present 12 hour and a half). The switching time for each seven segment LED is also a half of second (500ms). **Implement your code in the timer interrupt function.**

Report 1: Capture your schematic from Proteus and show in the report.

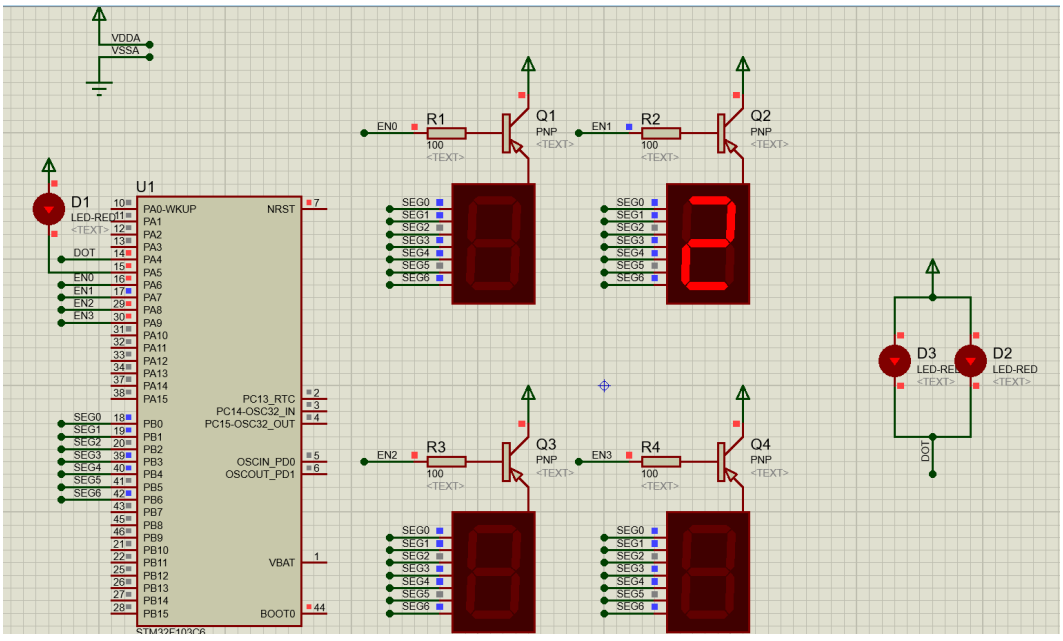
Report 2: Present your source code in the **HAL_TIM_PeriodElapsedCallback** function.

Solution:

```
/* USER CODE BEGIN 4 */
// 7SEG state:
//-------------------------
// state = 0 -> print 1
// state = 1 -> print 2
int state = 0;
int counter = 0;
```

```
8
9  // led state:   PA4 pin
10 //--------------------------
11 // state = 0 -> led on (2s)
12 // state = 1 -> led off (2s)
13 int led_state = 0;
14 int led_counter = 0;
15 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef  *htim
   ){
16   // two led
17   switch(led_state){
18   case(0):
19       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, RESET);
20       if(led_counter >= 100){
21         led_counter = 0;
22         led_state = 1;
23       }
24       break;
25   case(1):
26       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, SET);
27       if(led_counter >= 100){
28         led_counter = 0;
29         led_state = 0;
30       }
31       break;
32   }
33   led_counter ++;
34
35   // 7SEG led
36   switch(state){
37   case(0):
38       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, RESET);
39       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
40       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
41       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
42       display7SEG(1);
43       if(counter >= 50){
44         counter = 0;
45         state = 1;
46       }
47       break;
48   case(1):
49       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
50       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, RESET);
51       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
52       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
53       display7SEG(2);
54       if(counter >= 50){
55         counter = 0;
```

```
        state = 2;
      }
      break;
  case(2):
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
      display7SEG(3);
      if(counter >= 50){
        counter = 0;
        state = 3;
      }
      break;
  case(3):
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, RESET);
      display7SEG(0);
      if(counter >= 50){
        counter = 0;
        state = 0;
      }
      break;
  }
  counter ++;
}
/* USER CODE END 4 */
```



Hình 2.8: Simulation schematic in Proteus

Short question: What is the frequency of the scanning process?

⇒ ANS: the frequency of 4 led seg is 0.5Hz, which means that it's cycle take 2s long.

3.3 Exercise 3

Implement a function named **update7SEG(int index)**. An array of 4 integer numbers are declared in this case. The code skeleton in this exercise is presented as following:

```
const int MAX_LED = 4;
int index_led = 0;
int led_buffer[4] = {1, 2, 3, 4};
void update7SEG(int index){
    switch (index){
        case 0:
            //Display the first 7SEG with led_buffer[0]
            break;
        case 1:
            //Display the second 7SEG with led_buffer[1]
            break;
        case 2:
            //Display the third 7SEG with led_buffer[2]
            break;
        case 3:
            //Display the forth 7SEG with led_buffer[3]
            break;
        default:
            break;
    }
}
```

Program 2.4: An example for your source code

This function should be invoked in the timer interrupt, e.g update7SEG(index_led++). The variable **index_led** is updated to stay in a valid range, which is from 0 to 3.

Report 1: Present the source code of the update7SEG function.

Report 2: Present the source code in the HAL_TIM_PeriodElapsedCallback.

Students are proposed to change the values in the **led_buffer** array for unit test this function, which is used afterward.
Solution:

```
int led_buffer[4] = {1,2,3,0};
void update7SEG(int index){
  int value = led_buffer[index];
  display7SEG(value);

  switch(index){
```

```
  case(0):  // first 7SEG on
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
      break;
  case(1):  // second 7SEG on
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
      break;
  case(2):  // third 7SEG on
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
      break;
  case(3):  // last 7SEG on
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, RESET);
      break;
  default:
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
      break;
  }
}
```

Program 2.5: update7SEG() function

```
int single_led_counter = 0;
int counter = 0;
int index_led = 0;
const int MAX_LED = 4;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
   {
  if(counter >= 50){
    counter = 0;
    update7SEG(index_led++);

    if(index_led == MAX_LED) index_led = 0;
  }
  counter ++;

  if(single_led_counter >= 100){
```

```
    single_led_counter = 0;
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
  }
  single_led_counter++;
}
```

Program 2.6: HAL_TIM_PeriodElapsedCallback function

3.4 Exercise 4

Change the period of invoking update7SEG function in order to set the frequency of 4 seven segment LEDs to 1Hz. The DOT is still blinking every second.

Report 1: Present the source code in the **HAL_TIM_PeriodElapsedCallback**.

Solution:

```
int single_led_counter = 0;
int counter = 0;
int index_led = 0;
const int MAX_LED = 4;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
   {
  if(counter >= 25){  //0.25s
    counter = 0;
    update7SEG(index_led++);

    if(index_led == MAX_LED) index_led = 0;
  }
  counter ++;

  if(single_led_counter >= 100){
    single_led_counter = 0;
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
  }
  single_led_counter++;
}
```

Program 2.7: HAL_TIM_PeriodElapsedCallback function

3.5 Exercise 5

Implement a digital clock with **hour** and **minute** information displayed by 2 seven segment LEDs. The code skeleton in the **main** function is presented as follows:

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

while(1){
    second++;
    if (second >= 60){
        second = 0;
```

```
        minute++;
    }
    if(minute >= 60){
        minute = 0;
        hour++;
    }
    if(hour >=24){
        hour = 0;
    }
    updateClockBuffer();
    HAL_Delay(1000);
}
```

Program 2.8: An example for your source code

The function **updateClockBuffer** will generate values for the array **led_buffer** according to the values of hour and minute. In the case these values are 1 digit number, digit 0 is added.

Report 1: Present the source code in the **updateClockBuffer** function.
Solution:

```
// set begin time
int hour = 15, minute = 8, second = 50;
void updateClockBuffer(){
  led_buffer[0] = hour/10;
  led_buffer[1] = hour%10;

  led_buffer[2] = minute/10;
  led_buffer[3] = minute%10;
}
```

Program 2.9: UpdateClockBuffer() function

3.6 Exercise 6

The main target from this exercise to reduce the complexity (or reduce code processing) in the timer interrupt. The time consumed in the interrupt can lead to the nested interrupt issue, which can crash the whole system. A simple solution can disable the timer whenever the interrupt occurs, the enable it again. However, the real-time processing is not guaranteed anymore.

In this exercise, a software timer is created and its counter is count down every timer interrupt is raised (every 10ms). By using this timer, the **Hal_Delay(1000)** in the main function is removed. In a MCU system, non-blocking delay is better than blocking delay. The details to create a software timer are presented bellow. The source code is added to your current program, **do not delete the source code you have on Exercise 5.**

Step 1: Declare variables and functions for a software timer, as following:

```
/* USER CODE BEGIN 0 */
```

```
int timer0_counter = 0;
int timer0_flag = 0;
int TIMER_CYCLE = 10;
void set
Timer0(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 2.10: Software timer based timer interrupt

Please change the **TIMER_CYCLE** to your timer interrupt period. In the manual code above, it is **10ms**.

Step 2: The **timer_run()** is invoked in the timer interrupt as following:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
   {

  timer_run();

  //YOUR OTHER CODE
}
```

Program 2.11: Software timer based timer interrupt

Step 3: Use the timer in the main function by invoked setTimer0 function, then check for its flag (timer0_flag). An example to blink an LED connected to PA5 using software timer is shown as follows:

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

Program 2.12: Software timer is used in main fuction to blink the LED

Solution:
Report 1: if in line 1 of the code above is miss, what happens after that and why?

⇒ ANS: The program is still running. But, instead of waiting 1 second at the beginning, the new program skips waiting 1 second, and start to toggle pin right after the beginning. After that, every is the same.

Because the timer_run() funtion will turn on timer0_flag right at the beginning.

Report 2: if in line 1 of the code above is changed to setTimer0(1), what happens after that and why?

⇒ ANS: The program is still running. But, instead of waiting 1 second at the beginning, the new program just waits for 10 ms, and start to toggle pin right after the beginning. After that, every is the same.

Because the timer_run() funtion will count down from 1, which means $10ms \times 1 = 10ms$.

Report 3: if in line 1 of the code above is changed to setTimer0(10), what is changed compared to 2 first questions and why?

⇒ ANS: The program is still running. But, instead of waiting 1 second at the beginning, the new program just waits for 100 ms, and start to toggle pin right after the beginning. After that, every is the same.

Because the timer_run() funtion will count down from 10, which means $10ms \times 10 = 100ms$.

3.7 Exercise 7

Upgrade the source code in Exercise 5 (update values for hour, minute and second) by using the software timer and remove the HAL_Delay function at the end. Moreover, the DOT (connected to PA4) of the digital clock is also moved to main function.

Report 1: Present your source code in the while loop on main function.
Solution:

```
while (1)
{
    //second++; // do this in callback function

    if(second >= 30){ //60
        second = 0;
        minute++;
    }
    if(minute >= 30){ //60
        minute = 0;
        hour++;
    }
    if(hour >= 24){
        hour = 0;
    }

    // single led (1s)
```

```
18      if(toggle_flag == 1){
19          HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
20          toggle_flag = 0;
21      }
22
23      updateClockBuffer();
24      /* USER CODE END WHILE */
25  }
```

Program 2.13: while(1) function in main

3.8 Exercise 8

Move also the update7SEG() function from the interrupt timer to the main. Finally, the timer interrupt only used to handle software timers. All processing (or complex computations) is move to an infinite loop on the main function, optimizing the complexity of the interrupt handler function.

Report 1: Present your source code in the the main function. In the case more extra functions are used (e.g. the second software timer), present them in the report as well.
Solution:

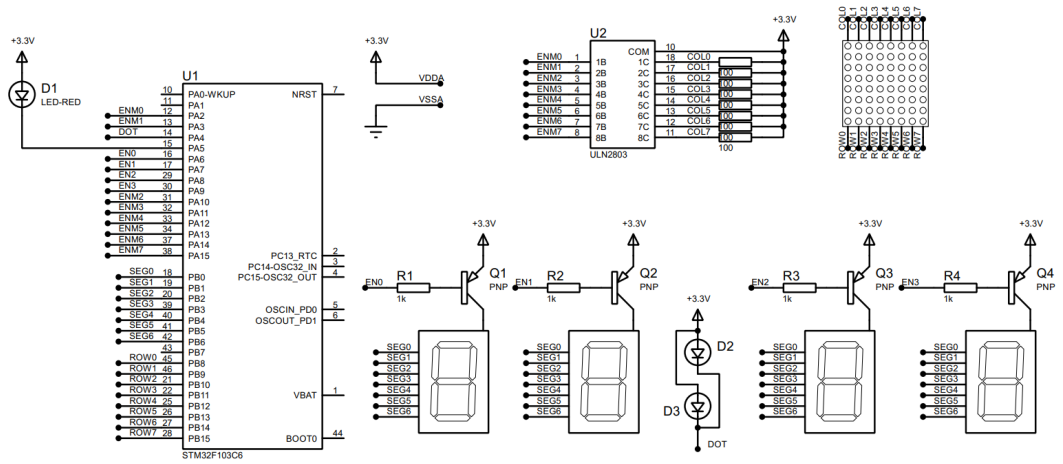
```
1   while (1)
2   {
3       //second++; // do this in callback function
4
5       if(second >= 30){ //60
6           second = 0;
7           minute++;
8       }
9       if(minute >= 30){ //60
10          minute = 0;
11          hour++;
12      }
13      if(hour >= 24){
14          hour = 0;
15      }
16      updateClockBuffer();
17
18      // single led (1s)
19      if(toggle_flag == 1){
20          HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
21          toggle_flag = 0;
22      }
23
24      // 7seg led (0.5s)
25      if(segled_flag == 1){
26          update7SEG(index_led++);
27          if(index_led == MAX_LED) index_led = 0;
```

```
28          segled_flag = 0;
29      }
30      /* USER CODE END WHILE */
31  }
```

Program 2.14: while(1) function in main

3.9 Exercise 9

This is an extra works for this lab. A LED Matrix is added to the system. A reference design is shown in figure bellow:



Hình 2.9: LED matrix is added to the simulation

In this schematic, two new components are added, including the **MATRIX-8X8-RED** and **ULN2803**, which is an NPN transistor array to enable the power supply for a column of the LED matrix. Students can change the enable signal (from ENM0 to ENM7) if needed. Finally, the data signal (from ROW0 to ROW7) is connected to PB8 to PB15.

Report 1: Present the schematic of your system by capturing the screen in Proteus.

Report 2: Implement the function, updateLEDMatrix(int index), which is similarly to 4 seven led segments.

```
1  const int MAX_LED_MATRIX = 8;
2  int index_led_matrix = 0;
3  uint8_t matrix_buffer[8] = {0x01, 0x02, 0x03, 0x04, 0x05, 0
      x06, 0x07, 0x08};
4  void updateLEDMatrix(int index){
5      switch (index){
6          case 0:
7              break;
8          case 1:
9              break;
10         case 2:
11             break;
```

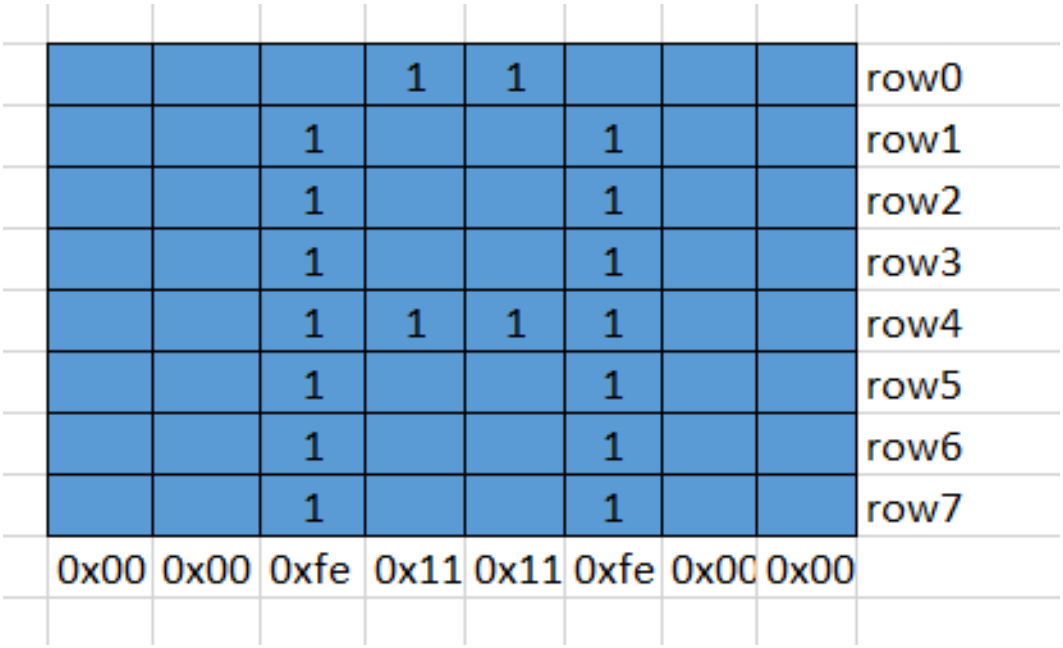
```
        case 3:
            break;
        case 4:
            break;
        case 5:
            break;
        case 6:
            break;
        case 7:
            break;
        default:
            break;
    }
}
```

Program 2.15: Function to display data on LED Matrix

Student are free to choose the invoking frequency of this function. However, this function is supposed to invoked in main function. Finally, please update the **matrix_buffer** to display character **"A"**.

Solution:

To design A character on led matrix, we can simply open excel to draw an "A" on it, or search google for "led matrix bit map". Here is an example for excel:



Hình 2.10: HOW TO SET "A" IN EXCEL

```
const int MAX_LED_MATRIX = 8;
int index_led_matrix = 0;
uint8_t matrix_led_buffer[8] = {0x00, 0x00, 0xfe, 0x11, 0
   x11, 0xfe, 0x00, 0x00};  // A character

void updateLEDMatrix(int index){
  uint8_t value = matrix_led_buffer[index];
```

```
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2,  !((value>>0)&1));
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3,  !((value>>1)&1));
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, !((value>>2)&1));
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, !((value>>3)&1));
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, !((value>>4)&1));
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, !((value>>5)&1));
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, !((value>>6)&1));
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, !((value>>7)&1));
}
```

Program 2.16: updateLEDMatrix() function

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2,  SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3,  SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);

HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);


int state = 0;
while (1)
{
    //second++;  // do this in callback function

    if(second >= 30){ //60
        second = 0;
        minute++;
    }
    if(minute >= 30){ //60
        minute = 0;
        hour++;
    }
    if(hour >= 24){
        hour = 0;
    }
    updateClockBuffer();

    // single led (1s)
```

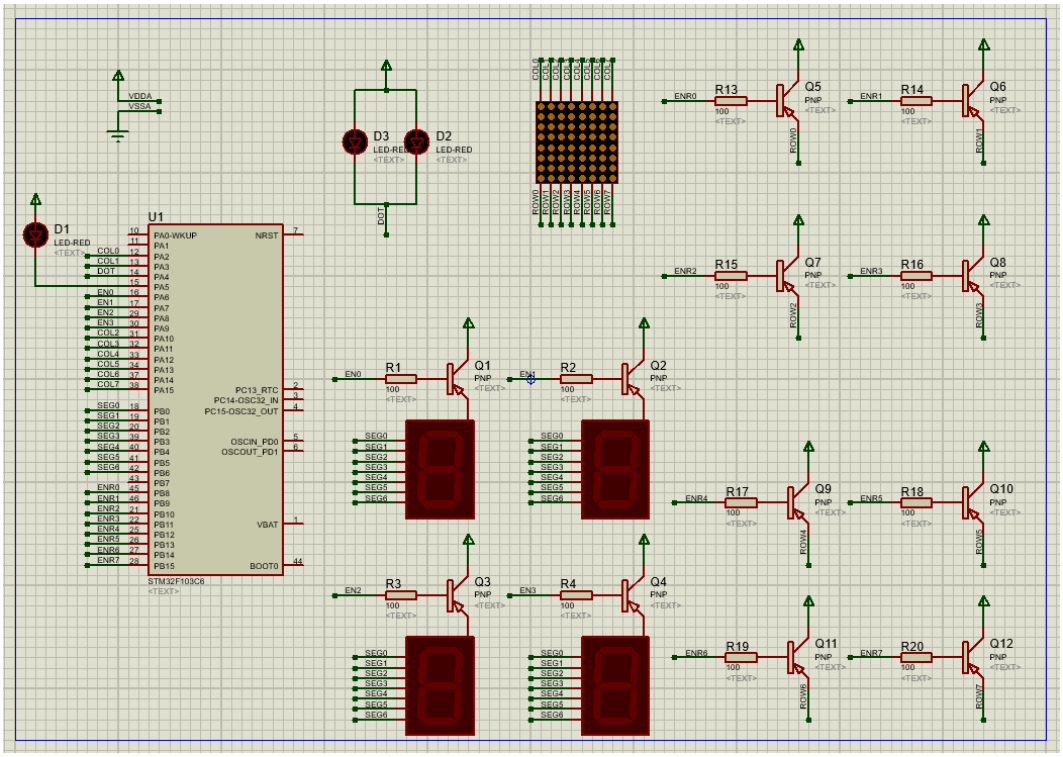
```
    if(toggle_flag == 1){
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
        toggle_flag = 0;
    }

    // 7seg led (0.5s)
    if(segled_flag == 1){
        update7SEG(index_led++);
        if(index_led == MAX_LED) index_led = 0;
        segled_flag = 0;
    }

    // matrix led ROW
    if(matrixled_flag == 1){
        matrixled_flag = 0;
        updateLEDMatrix(index_led_matrix++);
        if(index_led_matrix == MAX_LED_MATRIX)
index_led_matrix = 0;
        // scan led matrix
        switch(state){
        case(0):
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
      state = 1;
      break;
        case(1):
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
      state = 2;
      break;
        case(2):
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
```

```
87        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
88        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
89        state = 3;
90        break;
91          case(3):
92        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
93        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
94        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
95        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, RESET);
96        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
97        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
98        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
99        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
100       state = 4;
101       break;
102         case(4):
103       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
104       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
105       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
106       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
107       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, RESET);
108       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
109       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
110       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
111       state = 5;
112       break;
113         case(5):
114       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
115       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
116       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
117       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
118       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
119       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, RESET);
120       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
121       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
122       state = 6;
123       break;
124         case(6):
125       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
126       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
127       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
128       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
129       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
130       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
131       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, RESET);
132       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
133       state = 7;
134       break;
135         case(7):
```

```
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, RESET);
        state = 0;
        break;
          }
    }
    /* USER CODE END WHILE */
}
```

Program 2.17: while() function

Here is what we got under these code, the chematic



Hình 2.11: schematic

3.10 Exercise 10

Create an animation on LED matrix, for example, the character is shifted to the left.

Report 1: Briefly describe your solution and present your source code in the report.

Solution:

Base on what we got in exercise 9, to prepresent "A" from left to right, we can use a state machine to help build it.

```c
int led_matrix_state = 0;
void updateLEDMatrixBuffer(){
  switch(led_matrix_state){
  case(0):
      matrix_led_buffer[0] = 0xfe;
      matrix_led_buffer[1] = 0x00;
      matrix_led_buffer[2] = 0x00;
      matrix_led_buffer[3] = 0x00;
      matrix_led_buffer[4] = 0x00;
      matrix_led_buffer[5] = 0x00;
      matrix_led_buffer[6] = 0x00;
      matrix_led_buffer[7] = 0x00;
      led_matrix_state = 1;
      break;
  case(1):
      matrix_led_buffer[0] = 0x11;
      matrix_led_buffer[1] = 0xfe;
      matrix_led_buffer[2] = 0x00;
      matrix_led_buffer[3] = 0x00;
      matrix_led_buffer[4] = 0x00;
      matrix_led_buffer[5] = 0x00;
      matrix_led_buffer[6] = 0x00;
      matrix_led_buffer[7] = 0x00;
      led_matrix_state = 2;
      break;
  case(2):
      matrix_led_buffer[0] = 0x11;
      matrix_led_buffer[1] = 0x11;
      matrix_led_buffer[2] = 0xfe;
      matrix_led_buffer[3] = 0x00;
      matrix_led_buffer[4] = 0x00;
      matrix_led_buffer[5] = 0x00;
      matrix_led_buffer[6] = 0x00;
      matrix_led_buffer[7] = 0x00;
      led_matrix_state = 3;
      break;
  case(3):
      matrix_led_buffer[0] = 0xfe;
      matrix_led_buffer[1] = 0x11;
      matrix_led_buffer[2] = 0x11;
      matrix_led_buffer[3] = 0xfe;
      matrix_led_buffer[4] = 0x00;
      matrix_led_buffer[5] = 0x00;
      matrix_led_buffer[6] = 0x00;
      matrix_led_buffer[7] = 0x00;
      led_matrix_state = 4;
      break;
  case(4):
      matrix_led_buffer[0] = 0x00;
```

```
50        matrix_led_buffer[1] = 0xfe;
51        matrix_led_buffer[2] = 0x11;
52        matrix_led_buffer[3] = 0x11;
53        matrix_led_buffer[4] = 0xfe;
54        matrix_led_buffer[5] = 0x00;
55        matrix_led_buffer[6] = 0x00;
56        matrix_led_buffer[7] = 0x00;
57        led_matrix_state = 5;
58        break;
59    case(5):
60        matrix_led_buffer[0] = 0x00;
61        matrix_led_buffer[1] = 0x00;
62        matrix_led_buffer[2] = 0xfe;
63        matrix_led_buffer[3] = 0x11;
64        matrix_led_buffer[4] = 0x11;
65        matrix_led_buffer[5] = 0xfe;
66        matrix_led_buffer[6] = 0x00;
67        matrix_led_buffer[7] = 0x00;
68        led_matrix_state = 6;
69        break;
70    case(6):
71        matrix_led_buffer[0] = 0x00;
72        matrix_led_buffer[1] = 0x00;
73        matrix_led_buffer[2] = 0x00;
74        matrix_led_buffer[3] = 0xfe;
75        matrix_led_buffer[4] = 0x11;
76        matrix_led_buffer[5] = 0x11;
77        matrix_led_buffer[6] = 0xfe;
78        matrix_led_buffer[7] = 0x00;
79        led_matrix_state = 7;
80        break;
81    case(7):
82        matrix_led_buffer[0] = 0x00;
83        matrix_led_buffer[1] = 0x00;
84        matrix_led_buffer[2] = 0x00;
85        matrix_led_buffer[3] = 0x00;
86        matrix_led_buffer[4] = 0xfe;
87        matrix_led_buffer[5] = 0x11;
88        matrix_led_buffer[6] = 0x11;
89        matrix_led_buffer[7] = 0xfe;
90        led_matrix_state = 8;
91        break;
92    case(8):
93        matrix_led_buffer[0] = 0x00;
94        matrix_led_buffer[1] = 0x00;
95        matrix_led_buffer[2] = 0x00;
96        matrix_led_buffer[3] = 0x00;
97        matrix_led_buffer[4] = 0x00;
98        matrix_led_buffer[5] = 0xfe;
```

```
        matrix_led_buffer[6] = 0x11;
        matrix_led_buffer[7] = 0x11;
        led_matrix_state = 9;
        break;
    case(9):
        matrix_led_buffer[0] = 0x00;
        matrix_led_buffer[1] = 0x00;
        matrix_led_buffer[2] = 0x00;
        matrix_led_buffer[3] = 0x00;
        matrix_led_buffer[4] = 0x00;
        matrix_led_buffer[5] = 0x00;
        matrix_led_buffer[6] = 0xfe;
        matrix_led_buffer[7] = 0x11;
        led_matrix_state = 10;
        break;
    case(10):
        matrix_led_buffer[0] = 0x00;
        matrix_led_buffer[1] = 0x00;
        matrix_led_buffer[2] = 0x00;
        matrix_led_buffer[3] = 0x00;
        matrix_led_buffer[4] = 0x00;
        matrix_led_buffer[5] = 0x00;
        matrix_led_buffer[6] = 0x00;
        matrix_led_buffer[7] = 0xfe;
        led_matrix_state = 11;
        break;
    case(11):
        matrix_led_buffer[0] = 0x00;
        matrix_led_buffer[1] = 0x00;
        matrix_led_buffer[2] = 0x00;
        matrix_led_buffer[3] = 0x00;
        matrix_led_buffer[4] = 0x00;
        matrix_led_buffer[5] = 0x00;
        matrix_led_buffer[6] = 0x00;
        matrix_led_buffer[7] = 0x00;
        led_matrix_state = 0;
        break;
    }
}
```

Program 2.18: UpdateMatrixBuffer()

In main function, we have to change some code too!

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2,  SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3,  SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, SET);
```

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);

HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);

// set the beginning buffer of led matrix
updateLEDMatrixBuffer();

int state = 0;
while (1)
{
    //second++;  // do this in callback function

    if(second >= 30){ //60
        second = 0;
        minute++;
    }
    if(minute >= 30){ //60
        minute = 0;
        hour++;
    }
    if(hour >= 24){
        hour = 0;
    }
    updateClockBuffer();

    // single led (1s)
    if(toggle_flag == 1){
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
        toggle_flag = 0;
    }

    // 7seg led (0.5s)
    if(segled_flag == 1){
        update7SEG(index_led++);
        if(index_led == MAX_LED) index_led = 0;
        segled_flag = 0;
    }

    // matrix led ROW (50ms)
    if(matrixled_flag == 1){
        matrixled_flag = 0;
        updateLEDMatrix(index_led_matrix++); //display led
```

```
matrix
57        if(index_led_matrix == MAX_LED_MATRIX)
index_led_matrix = 0;
58
59        // scan led matrix
60        switch(state){ //bring up every single column
61        case(0):
62      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  RESET);
63      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
64      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
65      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
66      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
67      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
68      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
69      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
70      state = 1;
71      break;
72        case(1):
73      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
74      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  RESET);
75      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
76      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
77      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
78      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
79      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
80      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
81      state = 2;
82      break;
83        case(2):
84      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
85      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
86      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, RESET);
87      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
88      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
89      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
90      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
91      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
92      state = 3;
93      break;
94        case(3):
95      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
96      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
97      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
98      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, RESET);
99      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
100     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
101     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
102     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
103     state = 4;
```

```
104         break;
105           case(4):
106         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
107         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
108         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
109         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
110         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, RESET);
111         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
112         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
113         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
114         state = 5;
115         break;
116           case(5):
117         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
118         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
119         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
120         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
121         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
122         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, RESET);
123         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
124         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
125         state = 6;
126         break;
127           case(6):
128         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
129         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
130         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
131         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
132         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
133         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
134         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, RESET);
135         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, SET);
136         state = 7;
137         break;
138           case(7):
139         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8,  SET);
140         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9,  SET);
141         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, SET);
142         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, SET);
143         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, SET);
144         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, SET);
145         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, SET);
146         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, RESET);
147         state = 0;
148
149         // change the buffer before starting new cycle
150         updateLEDMatrixBuffer();
151         break;
152           }
```

```
153     }
154     /* USER CODE END WHILE */
155 }
```

Program 2.19: while() function

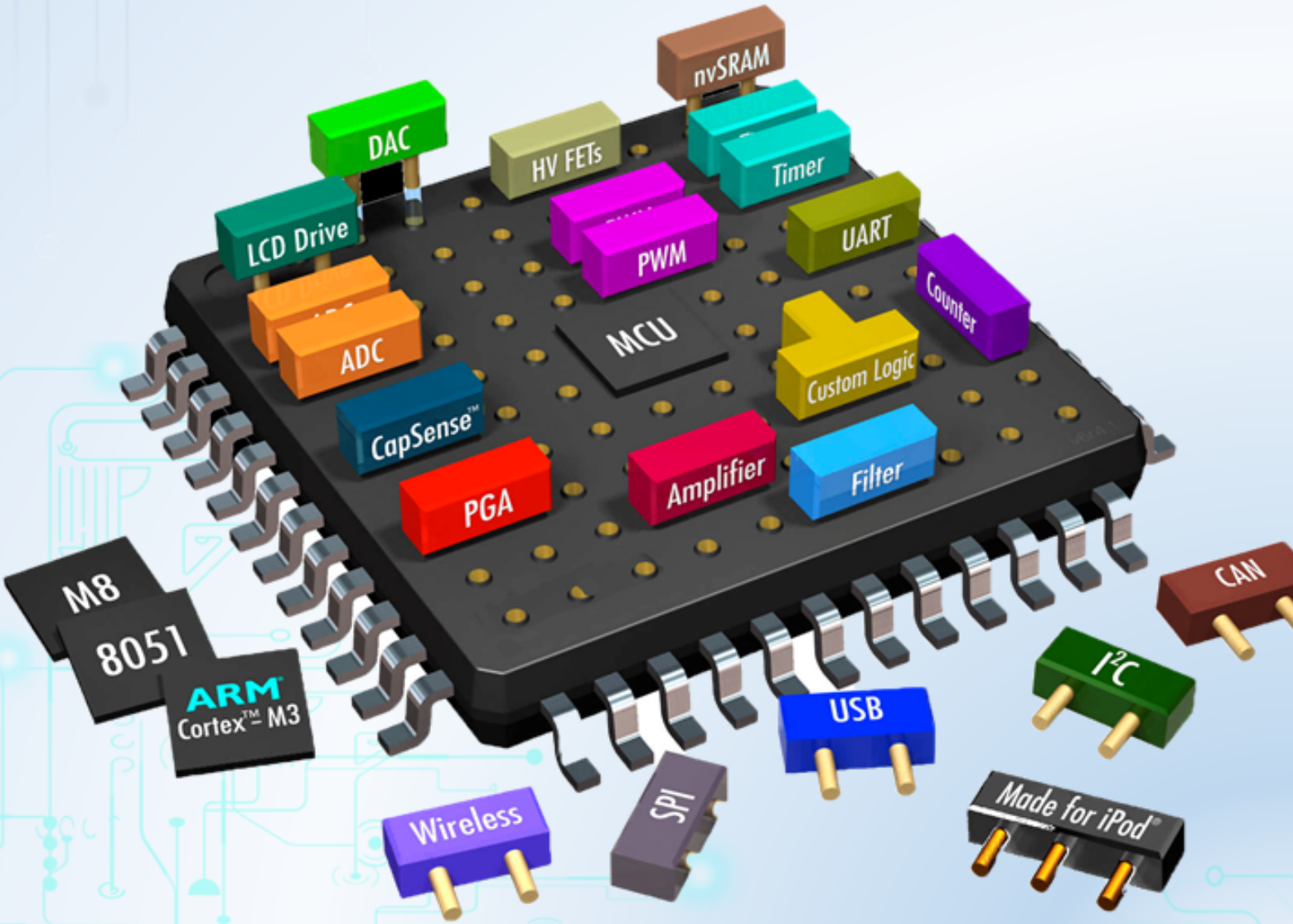
And here is my HAL_TIM_PeriodElapsedCallback() function:

```
1  int matrix_led_counter = 0;
2  int sec_counter = 0;
3  int counter = 0;
4  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef * htim
       ){
5    if(counter >= 50){
6      counter = 0;
7      segled_flag = 1;
8    }
9    counter ++;
10
11   if(sec_counter >= 100){
12     sec_counter = 0;
13     second ++;
14     toggle_flag = 1;
15   }
16   sec_counter ++;
17
18   if(matrix_led_counter >= 5){ //5
19     matrix_led_counter = 0;
20     matrixled_flag = 1;
21   }
22   matrix_led_counter ++;
23 }
```

Program 2.20: HAL_TIM_PeriodElapsedCallback() function

CHƯƠNG 3

Buttons/Switches



1 Objectives

In this lab, you will

- Learn how to add new C source files and C header files in an STM32 project,
- Learn how to read digital inputs and display values to LEDs using a timer interrupt of a microcontroller (MCU).
- Learn how to debounce when reading a button.
- Learn how to create an FSM and implement an FSM in an MCU.

2 Introduction

Embedded systems usually use buttons (or keys, or switches, or any form of mechanical contacts) as part of their user interface. This general rule applies from the most basic remote-control system for opening a garage door, right up to the most sophisticated aircraft autopilot system. Whatever the system you create, you need to be able to create a reliable button interface.
A button is generally hooked up to an MCU so as to generate a certain logic level when pushed or closed or “active" and the opposite logic level when unpushed or open or “inactive." The active logic level can be either ‘0’ or ‘1’, but for reasons both historical and electrical, an active level of ’0’ is more common.
We can use a button if we want to perform operations such as:

- Drive a motor while a switch is pressed.
- Switch on a light while a switch is pressed.
- Activate a pump while a switch is pressed.

These operations could be implemented using an electrical button without using an MCU; however, use of an MCU may well be appropriate if we require more complex behaviours. For example:

- Drive a motor while a switch is pressed.

  Condition: If the safety guard is not in place, don’t turn the motor. Instead sound a buzzer for 2 seconds.

- Switch on a light while a switch is pressed.

  Condition: To save power, ignore requests to turn on the light during daylight hours.

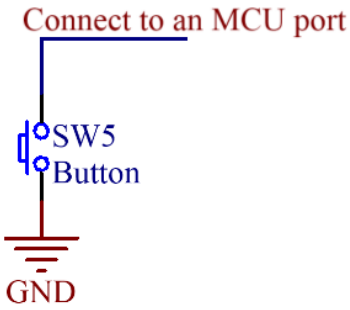
- Activate a pump while a switch is pressed

  Condition: If the main water reservoir is below 300 litres, do not start the main pump: instead, start the reserve pump and draw the water from the emergency tank.

In this lab, we consider how you read inputs from mechanical buttons in your embedded application using an MCU.

3 Basic techniques for reading from port pins

3.1 The need for pull-up resistors

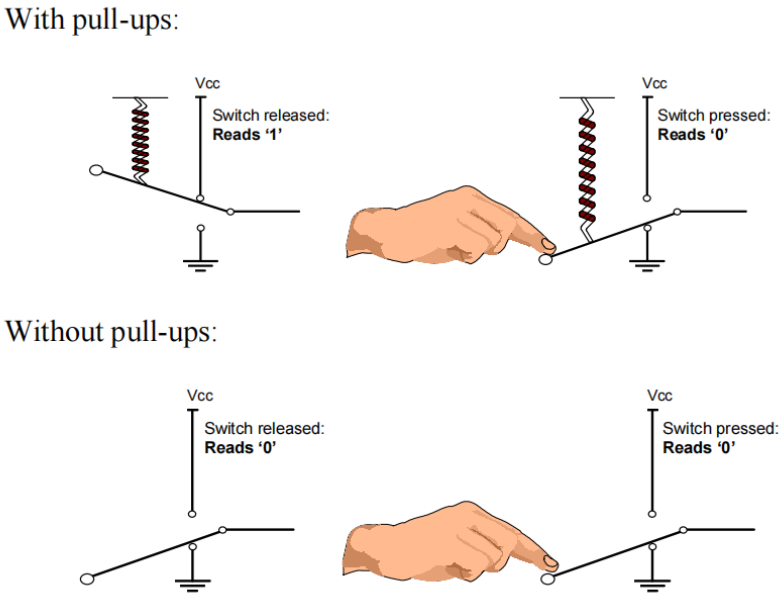


Hình 3.1: Connecting a button to an MCU

Figure 3.1 shows a way to connect a button to an MCU. This hardware operates as follows:

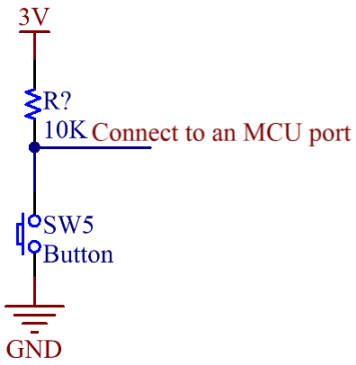
- When the switch is open, it has no impact on the port pin. An internal resistor on the port “pulls up" the pin to the supply voltage of the MCU (typically 3.3V for STM32F103). If we read the pin, we will see the value ‘1’.
- When the switch is closed (pressed), the pin voltage will be 0V. If we read the pin, we will see the value ‘0’.

However, if the MCU does not have a pull-up resistor inside, when the button is pressed, the read value will be ‘0’, but even we release the button, the read value is still ‘0’ as shown in Figure 3.2.



Hình 3.2: The need of pull up resistors

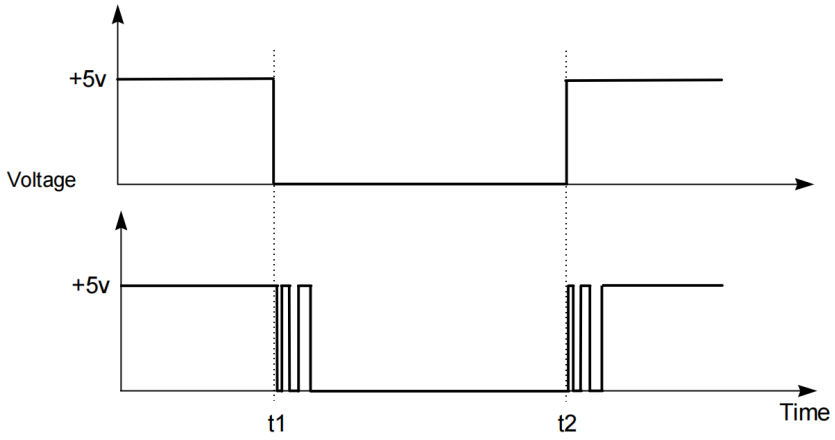
So a reliable way to connect a button/switch to an MCU is that we explicitly use an external pull-up resistor as shown in Figure 3.3.



Hình 3.3: A reliable way to connect a button to an MCU

3.2 Dealing with switch bounces

In practice, all mechanical switch contacts bounce (that is, turn on and off, repeatedly, for short period of time) after the switch is closed or opened as shown in Figure 3.4.



Hình 3.4: Switch bounces

Every system that uses any kind of mechanical switch must deal with the issue of debouncing. The key task is to make sure that one mechanical switch or button action is only read as one action by the MCU, even though the MCU will typically be fast enough to detect the unwanted switch bounces and treat them as separate events. Bouncing can be eliminated by special ICs or by RC circuitry, but in most cases debouncing is done in software because software is “free".
As far as the MCU concerns, each “bounce" is equivalent to one press and release of an “ideal” switch. Without appropriate software design, this can give several problems:

- Rather than reading ‘A’ from a keypad, we may read ‘AAAAA’

- Counting the number of times that a switch is pressed becomes extremely difficult
- If a switch is depressed once, and then released some time later, the 'bounce' may make it appear as if the switch has been pressed again (at the time of release).

The key to debouncing is to establish a minimum criterion for a valid button push, one that can be implemented in software. This criterion must involve differences in time - two button presses in 20ms must be treated as one button event, while two button presses in 2 seconds must be treated as two button events. So what are the relevant times we need to consider? They are these:

- Bounce time: most buttons seem to stop bouncing within 10ms
- Button press time: the shortest time a user can press and release a button seems to be between 50 and 100ms
- Response time: a user notices if the system response is 100ms after the button press, but not if it is 50ms after

Combining all of these times, we can set a few goals

- Ignore all bouncing within 10ms
- Provide a response within 50ms of detecting a button push (or release)
- Be able to detect a 50ms push and a 50ms release

The simplest debouncing method is to examine the keys (or buttons or switches) every N milliseconds, where N > 10ms (our specified button bounce upper limit) and N <= 50ms (our specified response time). We then have three possible outcomes every time we read a button:

- We read the button in the solid '0' state
- We read the button in the solid '1' state
- We read the button while it is bouncing (so we will get either a '0' or a '1')

Outcomes 1 and 2 pose no problems, as they are what we would always like to happen. Outcome 3 also poses no problem because during a bounce either state is acceptable. If we have just pressed an active-low button and we read a '1' as it bounces, the next time through we are guaranteed to read a '0' (remember, the next time through all bouncing will have ceased), so we will just detect the button push a bit later. Otherwise, if we read a '0' as the button bounces, it will still be '0' the next time after all bouncing has stopped, so we are just detecting the button push a bit earlier. The same applies to releasing a button. Reading a single bounce (with all bouncing over by the time of the next read) will never give us an invalid button state. It's only reading multiple bounces (multiple reads while bouncing is occurring) that can give invalid button states such as repeated push signals from one physical push.
So if we guarantee that all bouncing is done by the time we next read the button, we're good. Well, almost good, if we're lucky...

MCUs often live among high-energy beasts, and often control the beasts. High energy devices make electrical noise, sometimes great amounts of electrical noise. This noise can, at the worst possible moment, get into your delicate button-and-high-value-pullup circuit and act like a real button push. Oops, missile launched, sorry!
If the noise is too intense we cannot filter it out using only software, but will need hardware of some sort (or even a redesign). But if the noise is only occasional, we can filter it out in software without too much bother. The trick is that instead of regarding a single button 'make' or 'break' as valid, we insist on N contiguous makes or breaks to mark a valid button event. N will be a factor of your button scanning rate and the amount of filtering you want to add. Bigger N gives more filtering. The simplest filter (but still a big improvement over no filtering) is just an N of 2, which means compare the current button state with the last button state, and only if both are the same is the output valid.
Note that now we have not two but three button states: active (or pressed), inactive (or released), and indeterminate or invalid (in the middle of filtering, not yet filtered). In most cases we can treat the invalid state the same as the inactive state, since we care in most cases only about when we go active (from whatever state) and when we cease being active (to inactive or invalid). With that simplification we can look at simple **N = 2** filtering reading a button wired to STM32 MCU:

```
void button_reading(void){
    static unsigned char last_button;
    unsigned char raw_button;
    unsigned char filtered_button;
    last_button = raw_button;
    raw_button = HAL_GPIO_ReadPin(BUTTON_1_GPIO_Port,
   BUTTON_1_Pin);
    if(last_button == raw_button){
        filtered_button = raw_button;
    }
}
```

Program 3.1: Read port pin and deboucing

The function button_reading() must be called no more often than our debounce time (10ms).
To expand to greater filtering (larger N), keep in mind that the filtering technique essentially involves reading the current button state and then either counting or reseting the counter. We count if the current button state is the same as the last button state, and if our count reaches N we then report a valid new button state. We reset the counter if the current button state is different than the last button state, and we then save the current button state as the new button state to compare against the next time. Also note that the larger our value of N the more often our filtering routine must be called, so that we get a filtered response within our specified 50ms deadline. So for example with an N of 8 we should be calling our filtering routine every 2 - 5ms, giving a response time of 16 - 40ms (>10ms and <50ms).

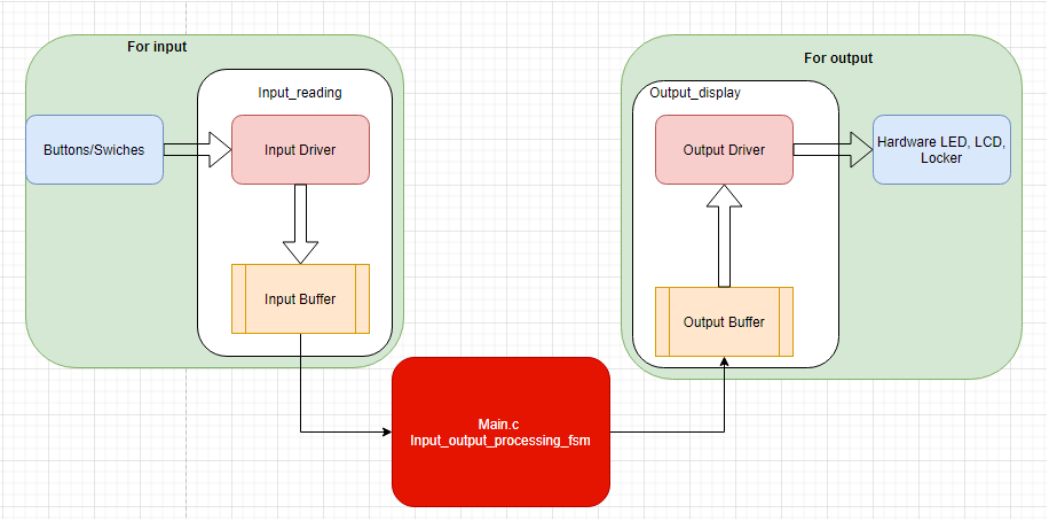
4 Reading switch input (basic code) using STM32

To demonstrate the use of buttons/switches in STM32, we use an example which requires to write a program that

- Has a timer which has an interrupt in every 10 milliseconds.
- Reads values of button PB0 every 10 milliseconds.
- Increases the value of LEDs connected to PORTA by one unit when the button PB0 is pressed.
- Increases the value of PORTA automatically in every 0.5 second, if the button PB0 is pressed in more than 1 second.

4.1 Input Output Processing Patterns

For both input and output processing, we have a similar pattern to work with. Normally, we have a module named driver which works directly to the hardware. We also have a buffer to store temporarily values. In the case of input processing, the driver will store the value of the hardware status to the buffer for further processing. In the case of output processing, the driver uses the buffer data to output to the hardware.



Hình 3.5: Input Output Processing Patterns

Figure 3.5 shows that we should have an *input_reading* module to processing the buttons, then store the processed data to the buffer. Then a module of *input_output_processin* will process the input data, and update the output buffer. The output driver gets the value from the output buffer to transfer to the hardware.

4.2 Setting up

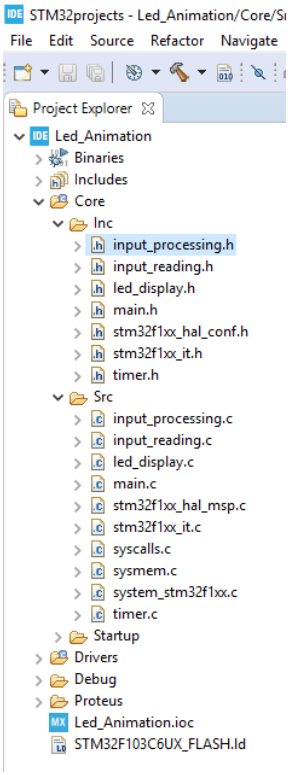
4.2.1 Create a project

Please follow the instruction in Labs 1 and 2 to create a project that includes:

- PB0 as an input port pin,
- PA0-PA7 as output port pins, and
- Timer 2 10ms interrupt

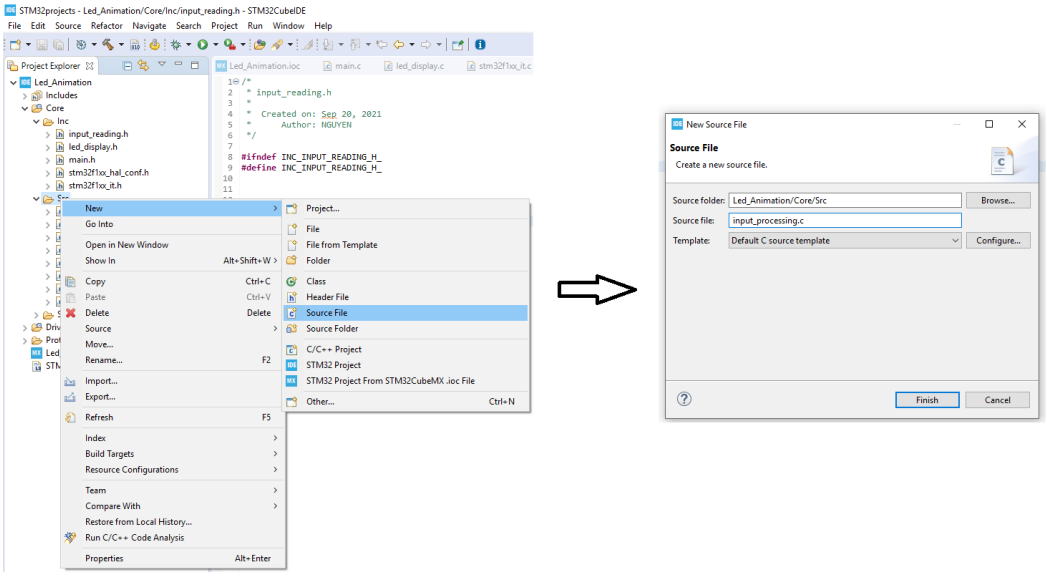
4.2.2 Create a file C source file and header file for input reading

We are expected to have files for button processing and led display as shown in Figure 3.6.

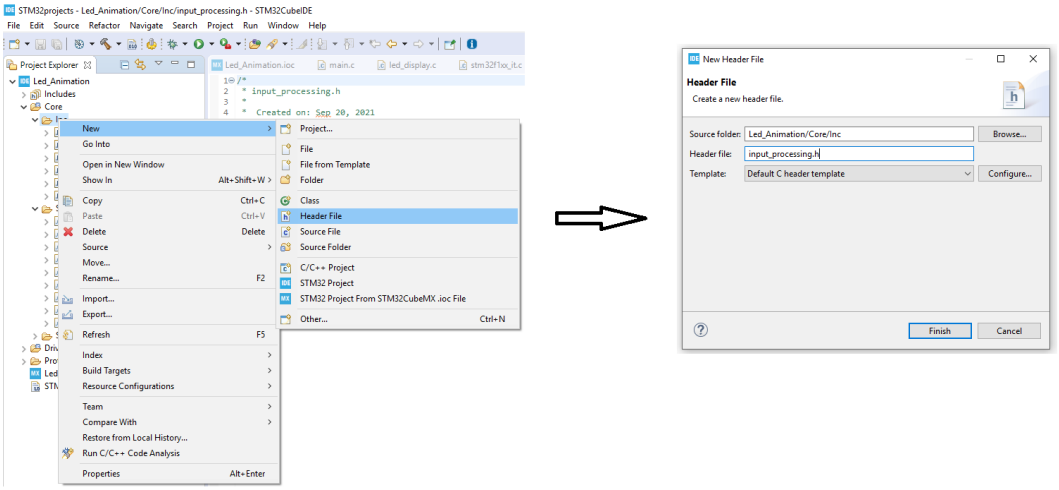


Hình 3.6: *File Organization*

Steps 1 (Figure 3.7): Right click to the folder **Src**, select **New**, then select **Source File**. There will be a pop-up. Please type the file name, then click **Finish**.
Step 2 (Figure 3.8): Do the same for the C header file in the folder **Inc**.



Hình 3.7: Step 1: Create a C source file for input reading



Hình 3.8: Step 2: Create a C header file for input processing

4.3 Code For Read Port Pin and Debouncing

4.3.1 The code in the input_reading.c file

```
#include "main.h"
//we aim to work with more than one buttons
#define NO_OF_BUTTONS                 1
//timer interrupt duration is 10ms, so to pass 1 second,
//we need to jump to the interrupt service routine 100 time
#define DURATION_FOR_AUTO_INCREASING     100
#define BUTTON_IS_PRESSED                  GPIO_PIN_RESET
#define BUTTON_IS_RELEASED                 GPIO_PIN_SET
//the buffer that the final result is stored after
//debouncing
static GPIO_PinState buttonBuffer[NO_OF_BUTTONS];
//we define two buffers for debouncing
static GPIO_PinState debounceButtonBuffer1[NO_OF_BUTTONS];
static GPIO_PinState debounceButtonBuffer2[NO_OF_BUTTONS];
//we define a flag for a button pressed more than 1 second.
static uint8_t flagForButtonPress1s[NO_OF_BUTTONS];
//we define counter for automatically increasing the value
//after the button is pressed more than 1 second.
static uint16_t counterForButtonPress1s[NO_OF_BUTTONS];
void button_reading(void){
  for(char i = 0; i < NO_OF_BUTTONS; i ++){
    debounceButtonBuffer2[i] =debounceButtonBuffer1[i];
    debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(
   BUTTON_1_GPIO_Port, BUTTON_1_Pin);
    if(debounceButtonBuffer1[i] == debounceButtonBuffer2[i
   ])
       buttonBuffer[i] = debounceButtonBuffer1[i];
       if(buttonBuffer[i] == BUTTON_IS_PRESSED){
       //if a button is pressed, we start counting
         if(counterForButtonPress1s[i] <
   DURATION_FOR_AUTO_INCREASING){
           counterForButtonPress1s[i]++;
         } else {
         //the flag is turned on when 1 second has passed
         //since the button is pressed.
           flagForButtonPress1s[i] = 1;
           //todo
         }
       } else {
         counterForButtonPress1s[i] = 0;
         flagForButtonPress1s[i] = 0;
       }
  }
}
```

Program 3.2: Define constants buffers and button_reading function

```c
unsigned char is_button_pressed(uint8_t index){
  if(index >= NO_OF_BUTTONS) return 0;
  return (buttonBuffer[index] == BUTTON_IS_PRESSED);
}
```

Program 3.3: Checking a button is pressed or not

```c
unsigned char is_button_pressed_1s(unsigned char index){
  if(index >= NO_OF_BUTTONS) return 0xff;
  return (flagForButtonPress1s[index] == 1);
}
```

Program 3.4: Checking a button is pressed more than a second or not

4.3.2 The code in the input_reading.h file

```c
#ifndef INC_INPUT_READING_H_
#define INC_INPUT_READING_H_
void button_reading(void);
unsigned char is_button_pressed(unsigned char index);
unsigned char is_button_pressed_1s(unsigned char index);
#endif /* INC_INPUT_READING_H_ */
```

Program 3.5: Prototype in input_reading.h file

4.3.3 The code in the timer.c file

```c
#include "main.h"
#include "input_reading.h"

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if(htim->Instance == TIM2){
    button_reading();
  }
}
```

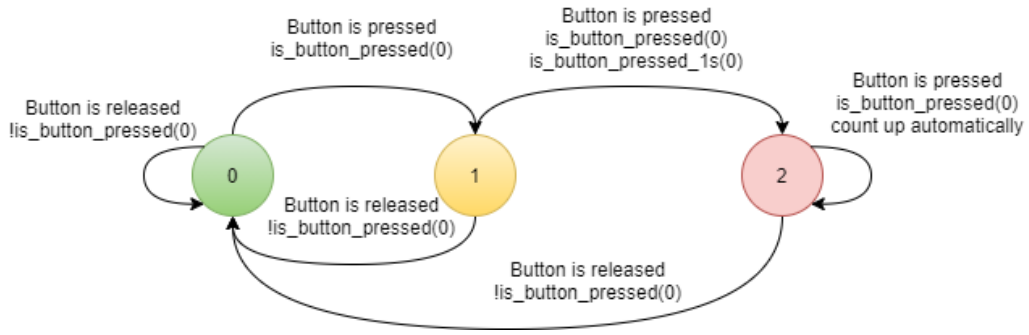
Program 3.6: Timer interrupt callback function

4.4 Button State Processing

4.4.1 Finite State Machine

To solve the example problem, we define 3 states as follows:

- State 0: The button is released or the button is in the initial state.
- State 1: When the button is pressed, the FSM will change to State 1 that is increasing the values of PORTA by one value. If the button is released, the FSM goes back to State 0.
- State 2: while the FSM is in State 1, the button is kept pressing more than 1 second, the state of FSM will change from 1 to 2. In this state, if the button is kept pressing, the value of PORTA will be increased automatically in every 500ms. If the button is released, the FSM goes back to State 0.



Hình 3.9: An FSM for processing a button

4.4.2 The code for the FSM in the input_processing.c file

Please note that *fsm_for_input_processing* function should be called inside the super loop of the main functin.

```
#include "main.h"
#include "input_reading.h"

enum ButtonState{BUTTON_RELEASED, BUTTON_PRESSED,
   BUTTON_PRESSED_MORE_THAN_1_SECOND} ;
enum ButtonState buttonState = BUTTON_RELEASED;
void fsm_for_input_processing(void){
  switch(buttonState){
  case BUTTON_RELEASED:
    if(is_button_pressed(0)){
      buttonState = BUTTON_PRESSED;
      //INCREASE VALUE OF PORT A BY ONE UNIT
    }
    break;
  case BUTTON_PRESSED:
    if(!is_button_pressed(0)){
      buttonState = BUTTON_RELEASED;
    } else {
      if(is_button_pressed_1s(0)){
        buttonState = BUTTON_PRESSED_MORE_THAN_1_SECOND;
      }
    }
    break;
  case BUTTON_PRESSED_MORE_THAN_1_SECOND:
    if(!is_button_pressed(0)){
      buttonState = BUTTON_RELEASED;
    }
    //todo
    break;
  }
}
```

Program 3.7: The code in the input_processing.c file

4.4.3 The code in the input_processing.h

```
#ifndef INC_INPUT_PROCESSING_H_
#define INC_INPUT_PROCESSING_H_

void fsm_for_input_processing(void);

#endif /* INC_INPUT_PROCESSING_H_ */
```

Program 3.8: Code in the input_processing.h file

4.4.4 The code in the main.c file

```
#include "main.h"
#include "input_processing.h"
//don't modify this part
int main(void){
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_TIM2_Init();
    while (1)
    {
        //you only need to add the fsm function here
        fsm_for_input_processing();
    }
}
```

Program 3.9: The code in the main.c file

5 Exercises and Report

5.1 Specifications

You are required to build an application of a traffic light in a cross road which includes some features as described below:

- The application has 12 LEDs including 4 red LEDs, 4 amber LEDs, 4 green LEDs.
- The application has 4 seven segment LEDs to display time with 2 for each road. The 2 seven segment LEDs will show time for each color LED corresponding to each road.
- The application has three buttons which are used

  - to select modes,

  - to modify the time for each color led on the fly, and

  - to set the chosen value.

- The application has at least 4 modes which is controlled by the first button. Mode 1 is a normal mode, while modes 2 3 4 are modification modes. You can press the first button to change the mode. Modes will change from 1 to 4 and back to 1 again.

  Mode 1 - Normal mode:

  - The traffic light application is running normally.

  Mode 2 - Modify time duration for the red LEDs: This mode allows you to change the time duration of the red LED in the main road. The expected behaviours of this mode include:

  - All single red LEDs are blinking in 2 Hz.

  - Use two seven-segment LEDs to display the value.

  - Use the other two seven-segment LEDs to display the mode.

  - The second button is used to increase the time duration value for the red LEDs.

  - The value of time duration is in a range of 1 - 99.

  - The third button is used to set the value.

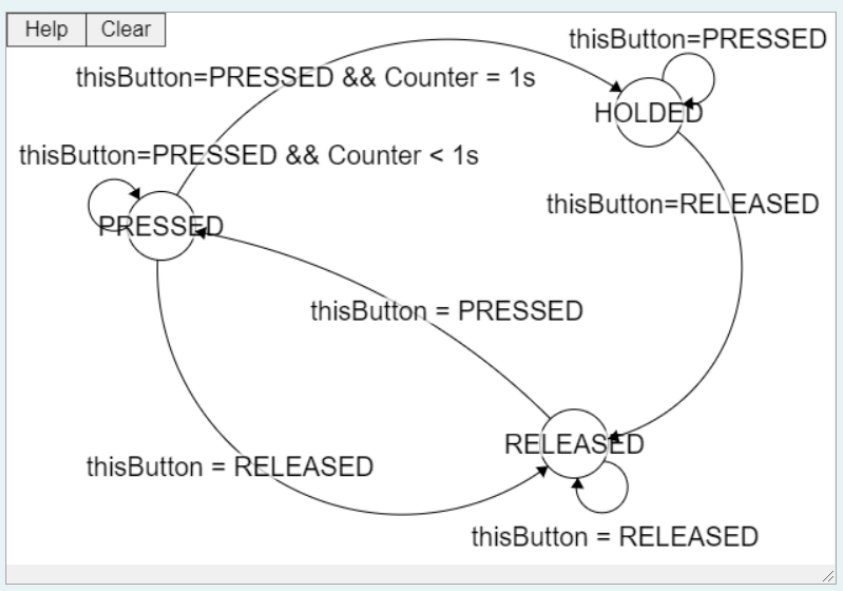
  Mode 3 - Modify time duration for the amber LEDs: Similar for the red LEDs described above with the amber LEDs.

  Mode 4 - Modify time duration for the green LEDs: Similar for the red LEDs described above with the green LEDs.

5.2 Exercise 1: Sketch an FSM

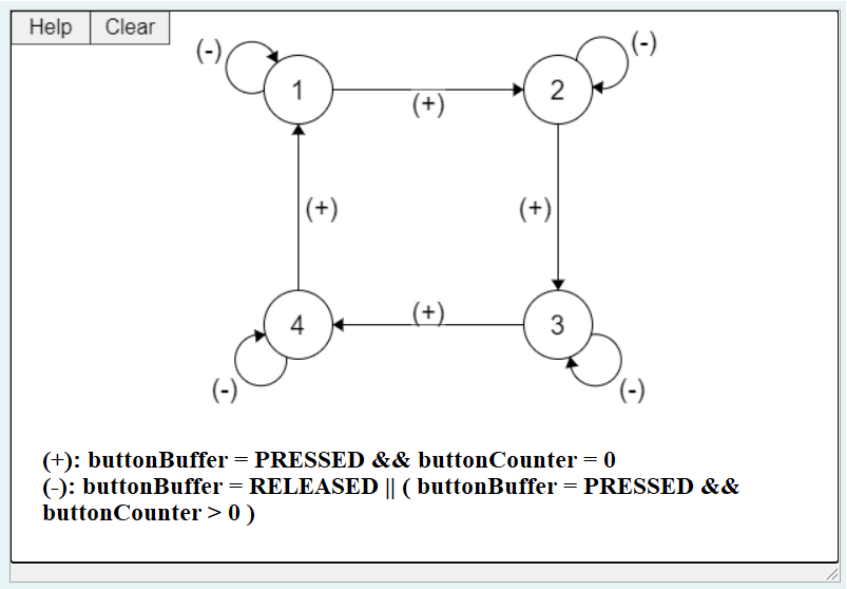
Your task in this exercise is to sketch an FSM that describes your idea of how to solve the problem.

We have 2 fsm, one of them is used for reading a button (it's fsm for reading button).



Hình 3.10: FSM for button reading

This fsm is to process 4 modes in lab requirements. Because there are only 4 mode, so holding button to change between 4 mode is unnecessary. To avoid holding the button, we just change state at the beginning when we press the button. (when counter == 0).



Hình 3.11: FSM for mode changing

5.3 Exercise 2: Proteus Schematic

Your task in this exercise is to draw a Proteus schematic for the problem above.

This schematic below shows that the lowest 7seg led display the mode of the module, the middle two 7seg leds represent the time for the horizontal road, and the others 2 7seg led are for the vertical road. However, in mode 2, 3, and 4, they are a litle bit different. The 2 middle led display the current value (not the time duration), and the top 2 seg led display the time duration of red led in mode 2, green led in mode 3, and yellow led in mode 4.

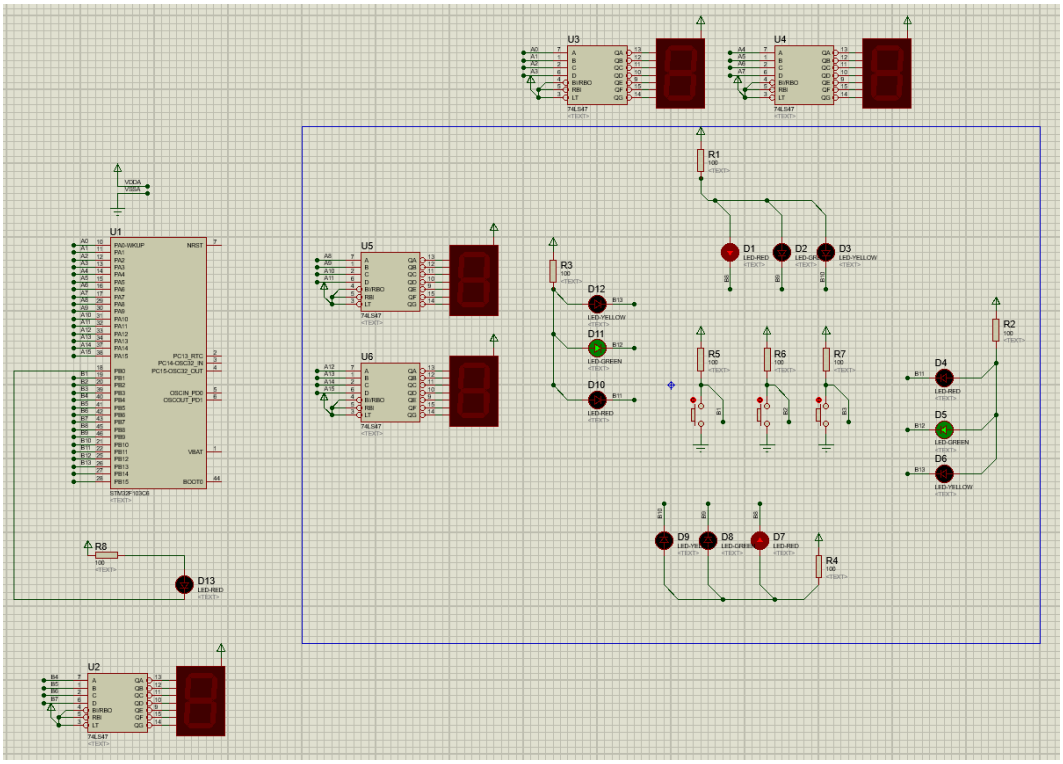
5.4 Exercise 3: Create STM32 Project

Your task in this exercise is to create a project that has pin corresponding to the Proteus schematic that you draw in previous section. You need to set up your timer interrupt is about 10ms.

Here is some parameters in **MX_TIM2_Init()** for setting **10ms** timer-interupt.

```
htim2.Instance = TIM2;
htim2.Init.Prescaler = 7999;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 9;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload =
 TIM_AUTORELOAD_PRELOAD_DISABLE;
```

Program 3.10: The code in the MX_TIM2_Init() in main.c



Hình 3.12: Skematic

5.5 Exercise 4: Modify Timer Parameters

Your task in this exercise is to modify the timer settings so that when we want to change the time duration of the timer interrupt, we change it the least and it will not affect the overall system. For example, the current system we have implemented is that it can blink an LED in 2 Hz, with the timer interrupt duration is 10ms. However, when we want to change the timer interrupt duration to 1ms or 100ms, it will not affect the 2Hz blinking LED.

When changing the timer interupt later, we can avoid opening the code and changing it by setting some parameter (it'll increase user's experience).

```
#ifndef INC_TIMER_H_
#define INC_TIMER_H_

#include "main.h"

#define TIMER_INTERUPT    10    //milisecond

// used in buttonCounter
////////////
#define TIME_1S        1000  //milisecond
static int COUNT_1S = TIME_1S / TIMER_INTERUPT;

#define TIME_50MS      50
static int COUNT_50MS = TIME_50MS / TIMER_INTERUPT;

```

```
#define TIME_500MS       500    // for blinking
static int COUNT_500MS = TIME_500MS / TIMER_INTERUPT;

#endif /* INC_TIMER_H_ */
```

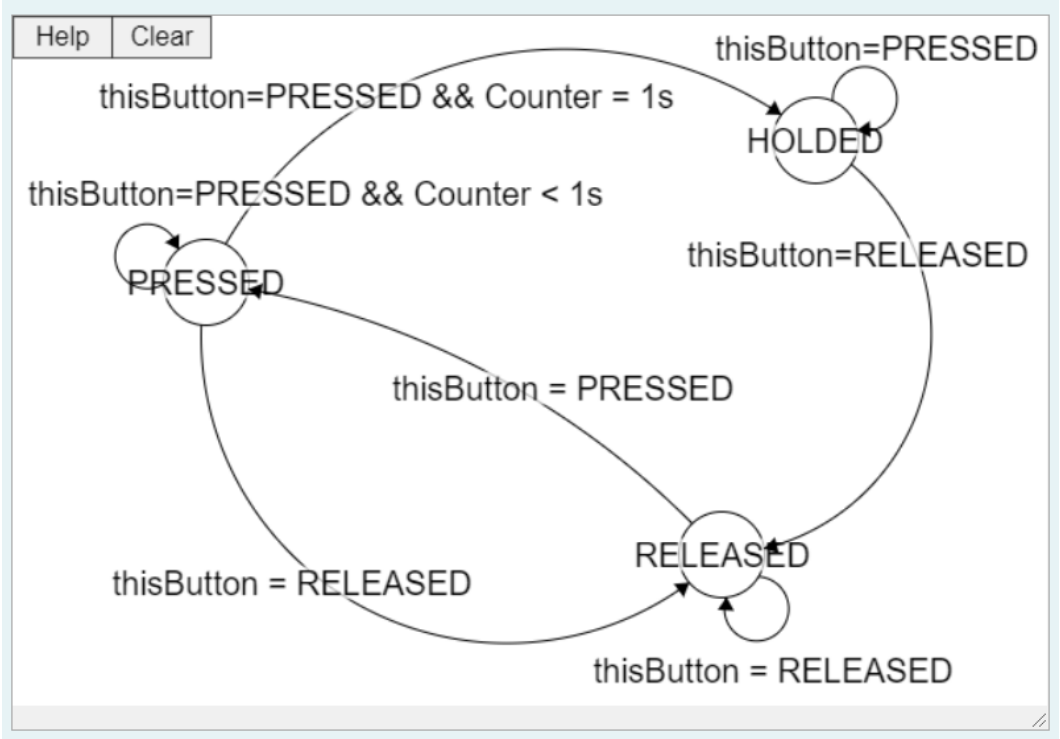
Program 3.11: The code in the timer.h

5.6 Exercise 5: Adding code for button debouncing

Following the example of button reading and debouncing in the previous section, your tasks in this exercise are:

- To add new files for input reading and output display,
- To add code for button debouncing,
- To add code for increasing mode when the first button is pressed.

After checking debouncing, we follow a Finte State Machine to upadte the state of buttonBuffer:



Hình 3.13: FSM for button reading

The increasement of mode or time duration when pressing button 1 and 2 is also done in the readButton() function. Moreover, ReadButton will raise up a flag when user press button 3.

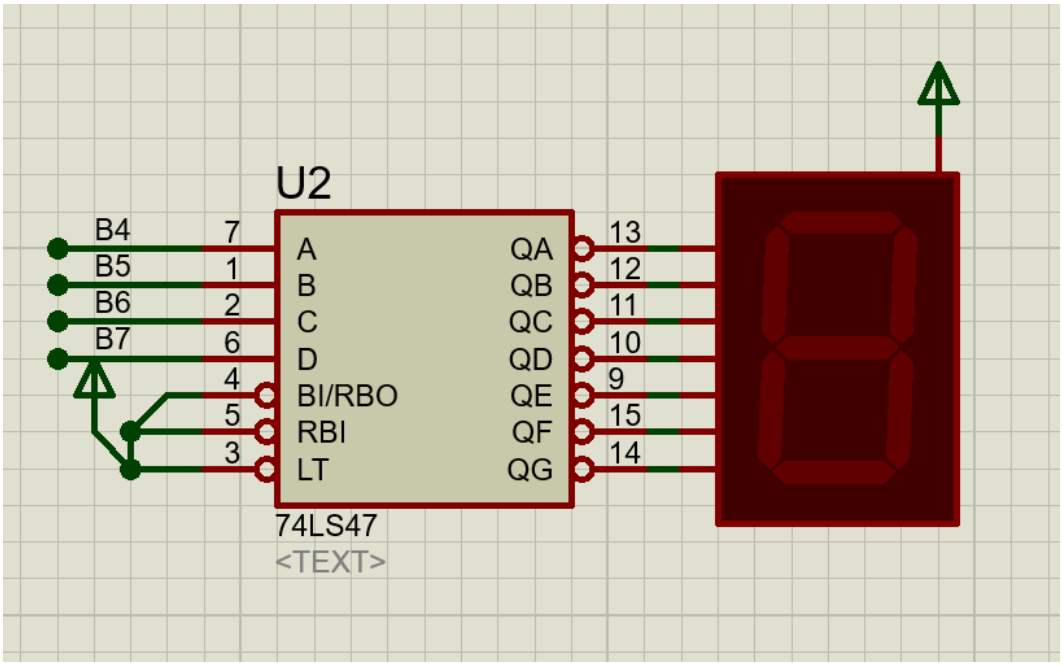
```
void readButton(void){
    for (int i = 0; i < NUM_OF_BUTTONS; i++){
        lastButton[i] = thisButton[i];
```

```
4         thisButton[i] = HAL_GPIO_ReadPin(portBuffer[i],
  pinBuffer[i]);
5
6         if(thisButton[i] == lastButton[i]){     // it is
  not a debounce
7             switch (buttonBuffer[i]){
8             case RELEASED:
9                 buttonBuffer[i] = thisButton[i];
10                break;
11
12            case PRESSED:
13                if(thisButton[i] == PRESSED){
14                  // Todo when button is pressed
15                  if(i==0 && buttonCounter[0] == 0){
16                    flag_reset = 1;
17                    value = 1;
18                    mode ++;       // first button
19                  }
20                  else if(i == 1 && buttonCounter[1] == 0)
  value ++;  // second counter
21                  else if(i == 2 && buttonCounter[2] == 0)
  flag_set = 1;
22
23                  // Button reading
24                    if(buttonCounter[i] >= COUNT_1S){
25                        buttonCounter[i] = 0;
26                        buttonBuffer[i] = HOLDED;
27                        break;
28                    }
29                    buttonCounter[i]++;
30                }
31                else if(thisButton[i] == RELEASED) {
32                  buttonCounter[i] = 0;
33                  buttonBuffer[i] = RELEASED;
34                }
35                break;
36
37            case HOLDED:
38              // Todo when button is holded
39              if(i == 1){   // for button 2
40                  if(buttonCounter_500ms >= COUNT_500MS){
41                    buttonCounter_500ms = 0;
42                    value ++;
43                  }
44                  buttonCounter_500ms ++;
45
46                  if(thisButton[i] == RELEASED) {
47                    buttonCounter_500ms = 0;
48                    buttonBuffer[i] = RELEASED;
```

```
49                      }
50                      break;
51                  }
52
53                  // other button
54                    if(thisButton[i] == RELEASED) buttonBuffer[
    i] = RELEASED;
55                    break;
56
57              default:
58                  break;
59              }
60          }
61      }
62  }
```

Program 3.12: The code in the readButton() in input_reading.c

For **display7SEG()**, we are now using a decoder **74LS47** due to lack of pins in MCU (instead of using 7 pins to control one 7seg led, now we just need 4 to do it). In this 74LS47 decoder, the A port is the LSB, and D port is fot MSB), so if you want to represent a 5 on 7seg led, just assign 0101 to DCBA pins in order, and when you want to turn off the led, just assign 1 to all the pins.



Hình 3.14: FSM for button reading

```
void display7SEG (int num, GPIO_TypeDef* type, uint16_t A,
   uint16_t B, uint16_t C, uint16_t D){
  switch(num){
  case -1: // turn off seg led
    HAL_GPIO_WritePin(type, D|C|B|A, SET);
    break;
  case 0:
    HAL_GPIO_WritePin(type, D|C|B|A, RESET);
    break;
  case 1:
    HAL_GPIO_WritePin(type, D|C|B, RESET);
    HAL_GPIO_WritePin(type, A, SET);
    break;
  case 2:
    HAL_GPIO_WritePin(type, D|C|A, RESET);
    HAL_GPIO_WritePin(type, B, SET);
    break;
  case 3:
    HAL_GPIO_WritePin(type, D|C, RESET);
    HAL_GPIO_WritePin(type, B|A, SET);
    break;
  case 4:
    HAL_GPIO_WritePin(type, D|B|A, RESET);
    HAL_GPIO_WritePin(type, C, SET);
    break;
  case 5:
    HAL_GPIO_WritePin(type, D|B, RESET);
    HAL_GPIO_WritePin(type, C|A, SET);
```

```
    break;
  case 6:
    HAL_GPIO_WritePin(type, D|A, RESET);
    HAL_GPIO_WritePin(type, C|B, SET);
    break;
  case 7:
    HAL_GPIO_WritePin(type, D, RESET);
    HAL_GPIO_WritePin(type, C|B|A, SET);
    break;
  case 8:
    HAL_GPIO_WritePin(type, C|B|A, RESET);
    HAL_GPIO_WritePin(type, D, SET);
    break;
  case 9:
    HAL_GPIO_WritePin(type, C|B, RESET);
    HAL_GPIO_WritePin(type, D|A, SET);
    break;
  default:
    HAL_GPIO_WritePin(type, D|C|B|A, SET);
    break;
  }
}
```

Program 3.13: The code to display on 7seg led in input_processing.c

5.7 Exercise 6: Adding code for displaying modes

Your tasks in this exercise are:

- To add code for display mode on seven-segment LEDs, and
- To add code for blinking LEDs depending on the mode that is selected.

In mode 1, to create a 4-way traffic light, using basic finite state machine to increase the performance of the machine. Beside, blink led depends on mode 2, 3, and 4, so it is done inside these mode.

```
void display7SEG (int num, GPIO_TypeDef* type, uint16_t A,
   uint16_t B, uint16_t C, uint16_t D){
void traffic_func(void){
  vertical_counter--;
  int vertical_counter_high = vertical_counter / 10;
  int vertical_counter_low  = vertical_counter % 10;
  switch(vertical_state){
  case RED_LIGHT:
    display7SEG(vertical_counter_high, GPIOA, GPIO_PIN_0,
   GPIO_PIN_1,
        GPIO_PIN_2, GPIO_PIN_3);
    display7SEG(vertical_counter_low, GPIOA, GPIO_PIN_4,
   GPIO_PIN_5,
        GPIO_PIN_6, GPIO_PIN_7);
```

```
12
13      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, RESET);
14      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9|GPIO_PIN_10, SET);
15
16      if(vertical_counter <= 0) {
17        vertical_counter = green_time;
18        vertical_state = GREEN_LIGHT;
19      }
20      break;
21
22    case GREEN_LIGHT:
23      display7SEG(vertical_counter_high, GPIOA, GPIO_PIN_0,
     GPIO_PIN_1,
24            GPIO_PIN_2, GPIO_PIN_3);
25      display7SEG(vertical_counter_low, GPIOA, GPIO_PIN_4,
     GPIO_PIN_5,
26            GPIO_PIN_6, GPIO_PIN_7);
27
28      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, RESET);
29      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8|GPIO_PIN_10, SET);
30
31      if(vertical_counter <= 0) {
32        vertical_counter = yellow_time;
33        vertical_state = YELLOW_LIGHT;
34      }
35      break;
36
37    case YELLOW_LIGHT:
38      display7SEG(vertical_counter_high, GPIOA, GPIO_PIN_0,
     GPIO_PIN_1,
39            GPIO_PIN_2, GPIO_PIN_3);
40      display7SEG(vertical_counter_low, GPIOA, GPIO_PIN_4,
     GPIO_PIN_5,
41            GPIO_PIN_6, GPIO_PIN_7);
42
43      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, RESET);
44      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8|GPIO_PIN_9, SET);
45
46      if(vertical_counter <= 0) {
47        vertical_counter = red_time;
48        vertical_state = RED_LIGHT;
49      }
50      break;
51
52    default:
53      break;
54    }
55
56
```

```
57    horizontal_counter--;
58    int horizontal_counter_high = horizontal_counter / 10;
59    int horizontal_counter_low  = horizontal_counter % 10;
60    switch(horizontal_state){
61      case RED_LIGHT:
62        display7SEG(horizontal_counter_high, GPIOA,
      GPIO_PIN_8,
63            GPIO_PIN_9, GPIO_PIN_10, GPIO_PIN_11);
64        display7SEG(horizontal_counter_low, GPIOA,
      GPIO_PIN_12,
65            GPIO_PIN_13, GPIO_PIN_14, GPIO_PIN_15);
66
67        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, RESET);
68        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12|GPIO_PIN_13, SET
      );
69
70        if(horizontal_counter <= 0) {
71          horizontal_counter = green_time;
72          horizontal_state = GREEN_LIGHT;
73        }
74        break;
75
76      case GREEN_LIGHT:
77        display7SEG(horizontal_counter_high, GPIOA,
      GPIO_PIN_8,
78            GPIO_PIN_9, GPIO_PIN_10, GPIO_PIN_11);
79        display7SEG(horizontal_counter_low, GPIOA,
      GPIO_PIN_12,
80            GPIO_PIN_13, GPIO_PIN_14, GPIO_PIN_15);
81
82        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, RESET);
83        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11|GPIO_PIN_13, SET
      );
84
85        if(horizontal_counter <= 0) {
86          horizontal_counter = yellow_time;
87          horizontal_state = YELLOW_LIGHT;
88        }
89        break;
90
91      case YELLOW_LIGHT:
92        display7SEG(horizontal_counter_high, GPIOA,
      GPIO_PIN_8,
93            GPIO_PIN_9, GPIO_PIN_10, GPIO_PIN_11);
94        display7SEG(horizontal_counter_low, GPIOA,
      GPIO_PIN_12,
95            GPIO_PIN_13, GPIO_PIN_14, GPIO_PIN_15);
96
97        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, RESET);
```

```
98        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11|GPIO_PIN_12, SET
    );
99
100       if(horizontal_counter <= 0) {
101         horizontal_counter = red_time;
102         horizontal_state = RED_LIGHT;
103       }
104       break;
105
106     default:
107       break;
108     }
109 }
```

Program 3.14: The code to display on 7seg led in input_processing.c

5.8 Exercise 7: Adding code for increasing time duration value for the red LEDs

Your tasks in this exercise are:

- to use the second button to increase the time duration value of the red LEDs
- to use the third button to set the value for the red LEDs.

In this exercise, button 2 is used to adjust a variable (which named "value"), if we don't push button 3 (set), the time duration of red led won't change. It makes user easier to increase or not when using. Moreover, because the time duration maximum is up to 99, so what if user want to set 99 at time duration? Do they need to push the button 99 times? No sence, so we use add a "HOLED" state to a button. If a button is hold, every 0.5s, the value will increase 1.

```
1  void fsm_input_processing(void){
2    // show mode on 7SEG
3    display7SEG(mode, GPIOB, GPIO_PIN_4, GPIO_PIN_5,
      GPIO_PIN_6, GPIO_PIN_7);
4
5    switch(mode){
6    case 1:   // traffic light
7      if(flag_1s) { //call every 1s
8        traffic_func();
9        flag_1s = 0;
10     }
11     break;
12   case 2:   // modify red time
13     // show value on 7SEG
14     value_high = value/10;
15     value_low = value%10;
16     display7SEG(value_high, GPIOA, GPIO_PIN_8,
17         GPIO_PIN_9, GPIO_PIN_10, GPIO_PIN_11);
18     display7SEG(value_low, GPIOA, GPIO_PIN_12,
```

```
19          GPIO_PIN_13, GPIO_PIN_14, GPIO_PIN_15);
20
21      // show red time on 7SEG
22      int red_time_high = red_time/10;
23      int red_time_low  = red_time%10;
24      display7SEG(red_time_high, GPIOA, GPIO_PIN_0,
    GPIO_PIN_1,
25            GPIO_PIN_2, GPIO_PIN_3);
26      display7SEG(red_time_low, GPIOA, GPIO_PIN_4, GPIO_PIN_5
    ,
27            GPIO_PIN_6, GPIO_PIN_7);
28
29      // blink red led
30      if(flag_500ms){
31        flag_500ms = 0;
32        HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_8);
33        HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_11);
34      }
35
36      // enter set button
37      if(flag_set){
38        flag_set = 0;
39        red_time = value;
40      }
41      break;
42      ...
43    }
44  }
```

Program 3.15: The code in mode 2 in fsm_input_processing() in input_processing.c

5.9 Exercise 8: Adding code for increasing time duration value for the amber LEDs

Your tasks in this exercise are:

- to use the second button to increase the time duration value of the amber LEDs
- to use the third button to set the value for the amber LEDs.

This is similar to exercise 7.

5.10 Exercise 9: Adding code for increasing time duration value for the green LEDs

Your tasks in this exercise are:

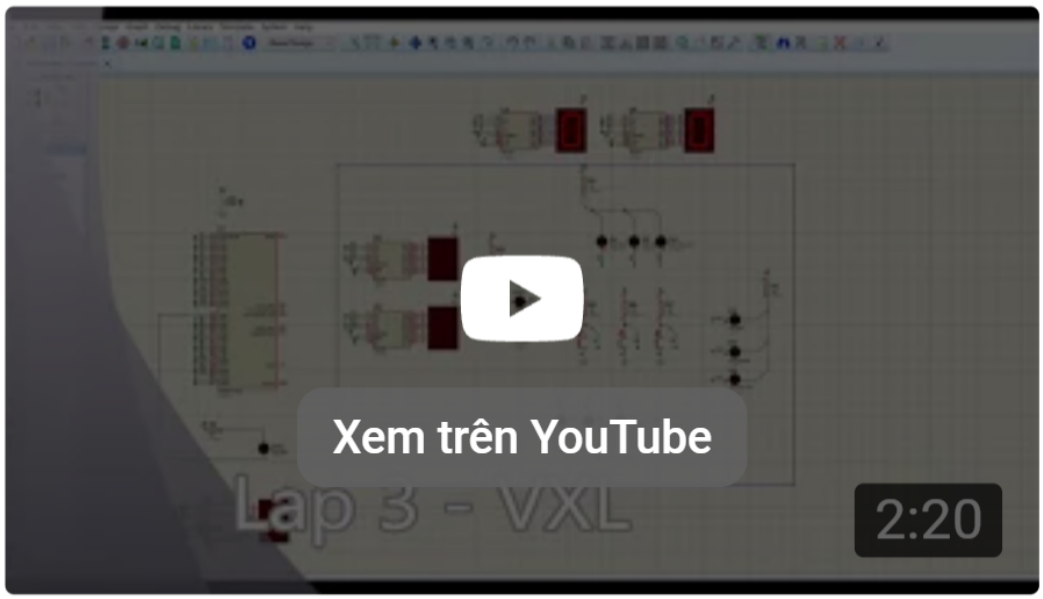
- to use the second button to increase the time duration value of the green LEDs

- to use the third button to set the value for the green LEDs.

This is similar to exercise 7.

5.11 Exercise 10: To finish the project

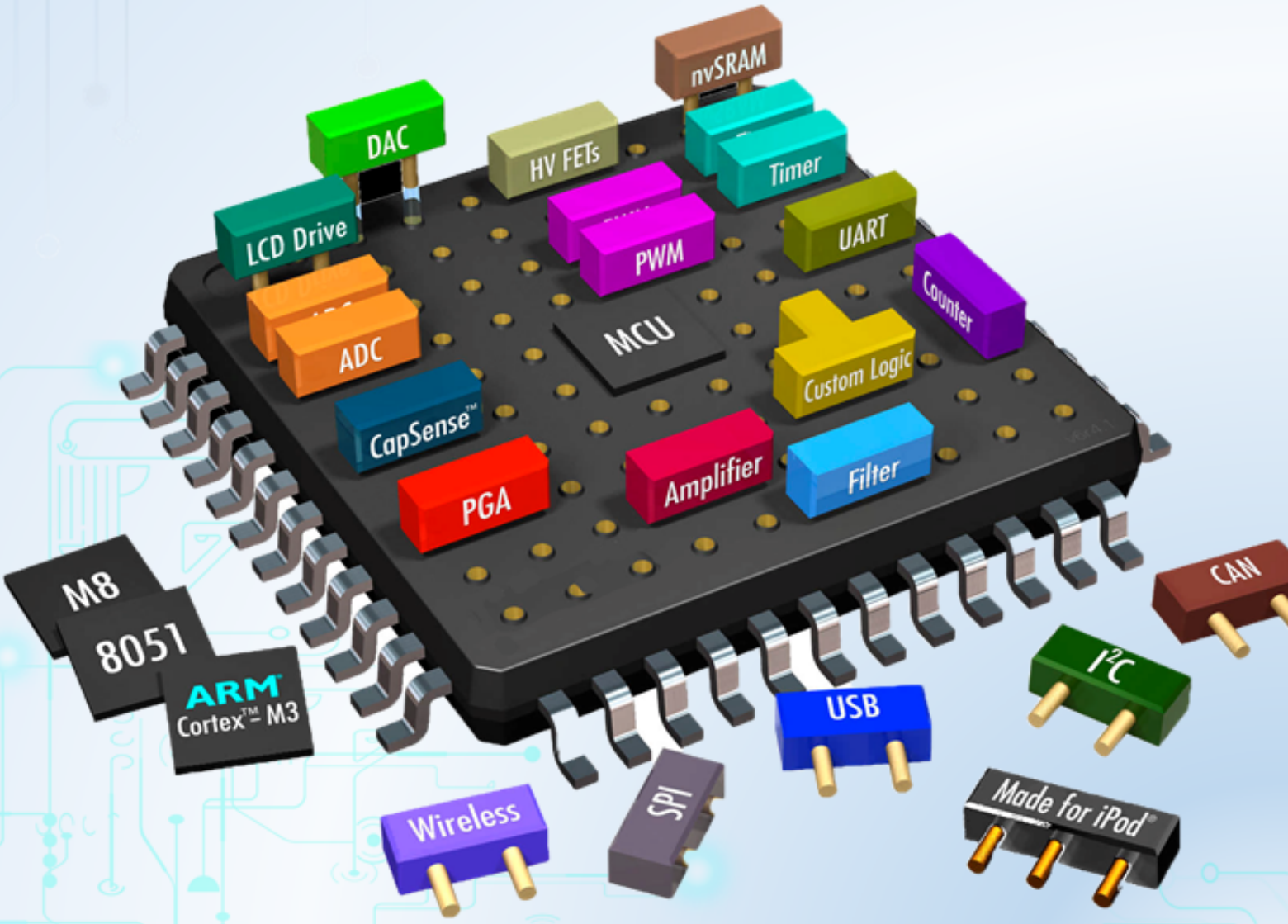
To know more about the states and what steps in that state, take a look at this Flowchart for more details. Or, watching this below demo may help.



Hình 3.15: Touch the image or this link

CHƯƠNG 4

Report A cooperative scheduler



1 Solution

A scheduler has the following key components:

- The scheduler data structure.
- An initialization function.
- A single interrupt service routine (ISR), used to update the scheduler at regular time intervals.
- A function for adding tasks to the scheduler.
- A dispatcher function that causes tasks to be executed when they are due to run.
- A function for removing tasks from the scheduler (not required in all applications).

1.1 The scheduler data structure and task array

At the heart of the scheduler is the scheduler data structure: this is a user-defined data type which collects together the information required about each task.

```
typedef struct{
  void (*pTask)(void);
  uint32_t Delay;
  uint32_t Period;
  uint8_t RunMe;
  uint32_t TaskID;
} sTask;
// MUST BE ADJUSTED FOR EACH NEW PROJECT
#define SCH_MAX_TASKS 10
sTask SCH_Task[SCH_MAX_TASKS];
```

Program 4.1: A struct of a task

The size of the task array
You must ensure that the task array is sufficiently large to store the tasks required in your application, by adjusting the value of SCH_MAX_TASKS.
In this lab4, now we have five tasks:

```
//Function of tasks
void Function1(){   //for pinA1
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_1);
}
void Function2(){   //for pinA2
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_2);
}
void Function3(){   //for pinA3
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_3);
}
void Function4(){   //for pinA4
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
```

```
}
void Function5(){   //for pinA5
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
}
// Add tasks to the task array
SCH_Add_Task(Function1, 0, 50);
SCH_Add_Task(Function2, 50, 100);
SCH_Add_Task(Function3, 100, 150);
SCH_Add_Task(Function4, 150, 200);
SCH_Add_Task(Function5, 200, 250);
```

1.2 The initialization function

Like most of the tasks we wish to schedule, the scheduler itself requires an initialization function. While this performs various important operations – such as preparing the scheduler array (discussed earlier) and the error code variable (discussed later) – the main purpose of this function is to set up a timer that will be used to generate the regular 'ticks' that will drive the scheduler.

```
void SCH_Init(void){
  unsigned char i;
  for(i=0; i<SCH_MAX_TASKS; i++){
    SCH_Delete_Task(i);
  }
}
```

1.3 The 'Update' function

The 'Update' function is involved in the ISR. It is invoked when the timer is overflow.

When it determines that a task is due to run, the update function increments the RunMe field for this task: the task will then be executed by the dispatcher, as we discuss later.

```
void SCH_Update(void){
  unsigned char index;

  for(index=0; index<SCH_MAX_TASKS; index++){
    if(SCH_Task[index].pTask){

      if(SCH_Task[index].Delay == 0){
        SCH_Task[index].RunMe+= 1;
        if(SCH_Task[index].Period)
            SCH_Task[index].Delay = SCH_Task[index].Period;
      }

      else {
        SCH_Task[index].Delay -= 1;
```

```
      }
    }
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  SCH_Update();
}
```

1.4 The 'Add Task' function

As the name suggests, the 'Add Task' function is used to add tasks to the task array, to ensure that they are called at the required time(s). Here is the example of add task function: **unsigned char SCH_Add_Task (Task_Name , Initial_Delay, Period)**
The parameters for the 'Add Task' function are described as follows:

- **Task_Name**: the name of the function (task) that you wish to schedule
- **Initial_Delay**: the delay (in ticks) before task is first executed. If set to 0, the task is executed immediately.
- **Period**: the interval (in ticks) between repeated executions of the task. If set to 0, the task is executed only once

Here are some examples.

Our timer interpt is 10ms. And when we have **SCH_Add_Task(Function1,0,50);**. Function1 will be executed regularly every 50 scheduler ticks, and the task will first at T=0.

SCH_Add_Task(Function2,50,100); is working the same to the above but having a few differences. Function2 will run first at T=50, and will be executed every 100 ticks, which means T=150, T=25-, T=350, ...

Here my code for handling it:

```
unsigned char SCH_Add_Task(void(*pFunction)(), uint32_t delay,
uint32_t period){
  unsigned char index=0;
  while(SCH_Task[index].pTask!=0 && index<SCH_MAX_TASKS){
    index++;
  }

  if(index == SCH_MAX_TASKS) return index;

  SCH_Task[index].pTask = pFunction;
  SCH_Task[index].Delay = delay;
  SCH_Task[index].Period = period;
  SCH_Task[index].RunMe = 0;
```

```
14   return index;
15 }
```

Program 4.2: An implementation of the scheduler 'add task' function

1.5 The 'Dispatcher'

As we have seen, the 'Update' function does not execute any tasks: the tasks that are due to run are invoked through the 'Dispatcher' function.

```
1  void SCH_Dispatcher_Task(void){
2    unsigned char index;
3
4    for(index=0; index<SCH_MAX_TASKS; index++){
5      if(SCH_Task[index].RunMe > 0){
6        (*SCH_Task[index].pTask)();
7        SCH_Task[index].RunMe -= 1;
8        if(SCH_Task[index].Period == 0) SCH_Delete_Task(index);
9      }
10   }
11 }
```

Program 4.3: An implementation of the scheduler 'dispatch task' function

The dispatcher is the only component in the Super Loop:

```
1      while(1){
2          SCH_Dispatch_Task();
3      }
```

Program 4.4: The dispatcher in the super loopin main.c

1.6 The 'Delete Task' function

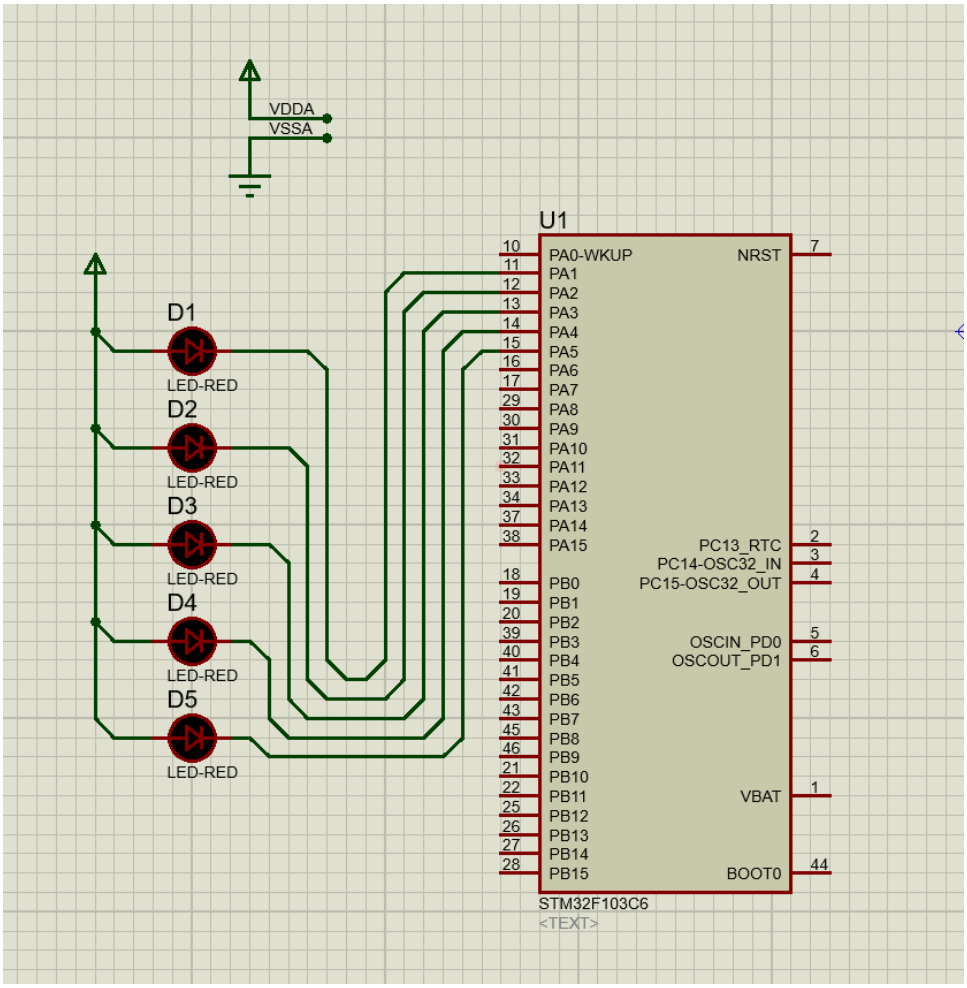
Sometimes it can be necessary to delete tasks from the array. To do so, SCH_Delete_Task() can be used as follows: SCH_Delete_Task(Task_ID)

```
1  void SCH_Delete_Task(unsigned char index){
2    SCH_Task[index].pTask = 0x0000;
3    SCH_Task[index].Delay = 0;
4    SCH_Task[index].Period = 0;
5    SCH_Task[index].RunMe = 0;
6  }
```

Program 4.5: An implementation of the scheduler 'delete task' function

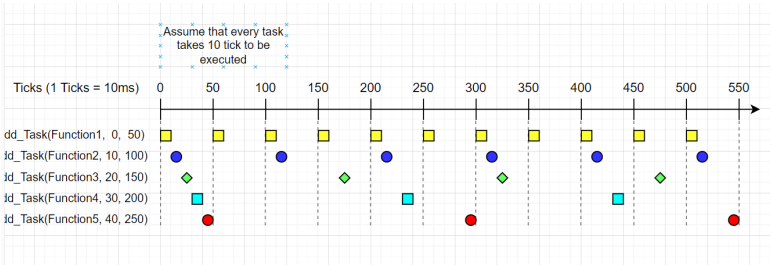
2 Schematic and Chart Diagram

The schematic from Proteus simulation



Hình 4.1: for further information, take a look at [my github]

The Chart Diagram :



*Hình 4.2: for further information, download **chart.drawio** at [my github] , open it with [draw.io]*

3 Implementation

Now we put every thing together, Firstly, in timer interupt, we set the timer cycle is 10ms, and call the **SCH_Update()**:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  SCH_Update();
}
```

Program 4.6: In timer interupt

Now we define the **sTask** struct and some definitions:

```
typedef struct{
  void (*pTask)(void);
  uint32_t Delay;
  uint32_t Period;
  uint8_t RunMe;
  uint32_t TaskID;
} sTask;
// MUST BE ADJUSTED FOR EACH NEW PROJECT
#define SCH_MAX_TASKS 10
sTask SCH_Task[SCH_MAX_TASKS];
```

Program 4.7: In timer interupt

After that, we create function in the same file main.c:

```
// ========= SCH func ========= //
////////////////////////////////
void SCH_Init(void){
  unsigned char i;
  for(i=0; i<SCH_MAX_TASKS; i++){
    SCH_Delete_Task(i);
  }
}

void SCH_Update(void){
  unsigned char index;

  for(index=0; index<SCH_MAX_TASKS; index++){
    if(SCH_Task[index].pTask){

      if(SCH_Task[index].Delay == 0){
        SCH_Task[index].RunMe+= 1;
        if(SCH_Task[index].Period)
            SCH_Task[index].Delay = SCH_Task[index].Period;
      }

      else {
        SCH_Task[index].Delay -= 1;
      }
    }
  }
}

```

```
unsigned char SCH_Add_Task(void(*pFunction)(), uint32_t delay,
uint32_t period){
  unsigned char index=0;
  while(SCH_Task[index].pTask!=0 && index<SCH_MAX_TASKS){
    index++;
  }

  if(index == SCH_MAX_TASKS) return index;

  SCH_Task[index].pTask = pFunction;
  SCH_Task[index].Delay = delay;
  SCH_Task[index].Period = period;
  SCH_Task[index].RunMe = 0;
  return index;
}

void SCH_Dispatcher_Task(void){
  unsigned char index;

  for(index=0; index<SCH_MAX_TASKS; index++){
    if(SCH_Task[index].RunMe > 0){
      (*SCH_Task[index].pTask)();
      SCH_Task[index].RunMe -= 1;
      if(SCH_Task[index].Period == 0) SCH_Delete_Task(index);
    }
  }
}

// ========= five func ========== //
////////////////////////////////////
void Function1(){
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_1);
}
void Function2(){
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_2);
}
void Function3(){
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_3);
}
void Function4(){
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
}
void Function5(){
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
}
```

Program 4.8: Some function here

Let's move on, and go to main dishes. Now we have to call **SCH_Init()** first, and then add our 5 task with the lab requirement. And finally, in the main's superloop, we call the **SCH_Dispatcher_Task()** function.

NOTE: There are 5 tasks with their own period of time: 0.5s, 1s, 1.5s, 2s, 2.5s. So

some of they may be executed in the same time , for example:

- **[T=0 (0s)]** 5 tasks're executed concurrently
- **[T=100 (1s)]** task1 and task2 are executed concurrently
- **[...]**

And when they run concurrently, these function may not be called at correct time. So we apply a small delay to prevent. (**ASSUME THAT EVERY FUNCTION CAN BE FULLY EXECUTED IN <= 10 TICKS**)

```
int main(void){

    MX_GPIO_Init();
    MX_TIM2_Init();
    HAL_TIM_Base_Start_IT(&htim2);

    SCH_Init();
    SCH_Add_Task(Function1,  0,  50);
    SCH_Add_Task(Function2, 10, 100);
    SCH_Add_Task(Function3, 20, 150);
    SCH_Add_Task(Function4, 30, 200);
    SCH_Add_Task(Function5, 40, 250);

    while(1){
      SCH_Dispatcher_Task();
    }

}
```

Program 4.9: In main.c