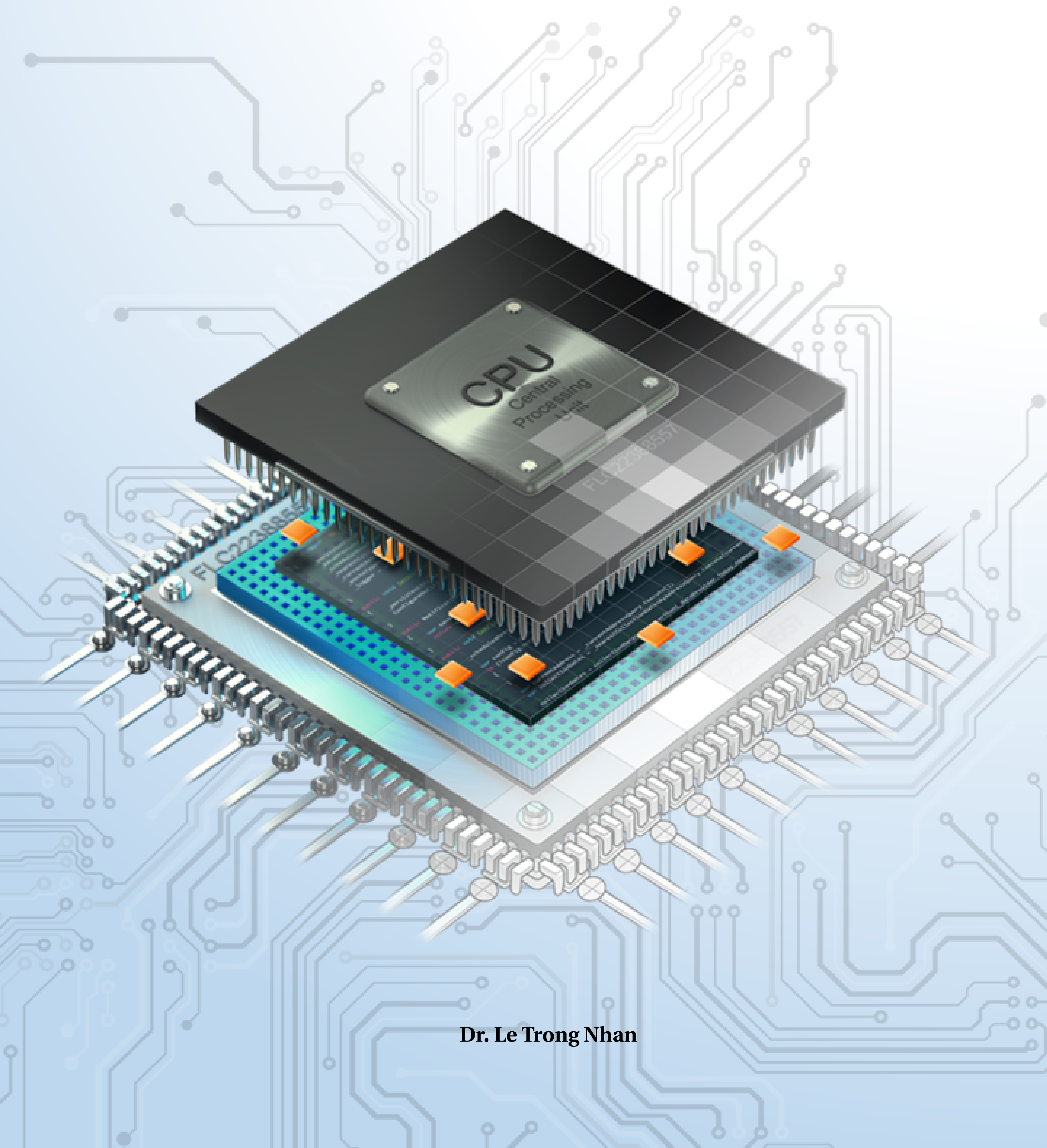
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

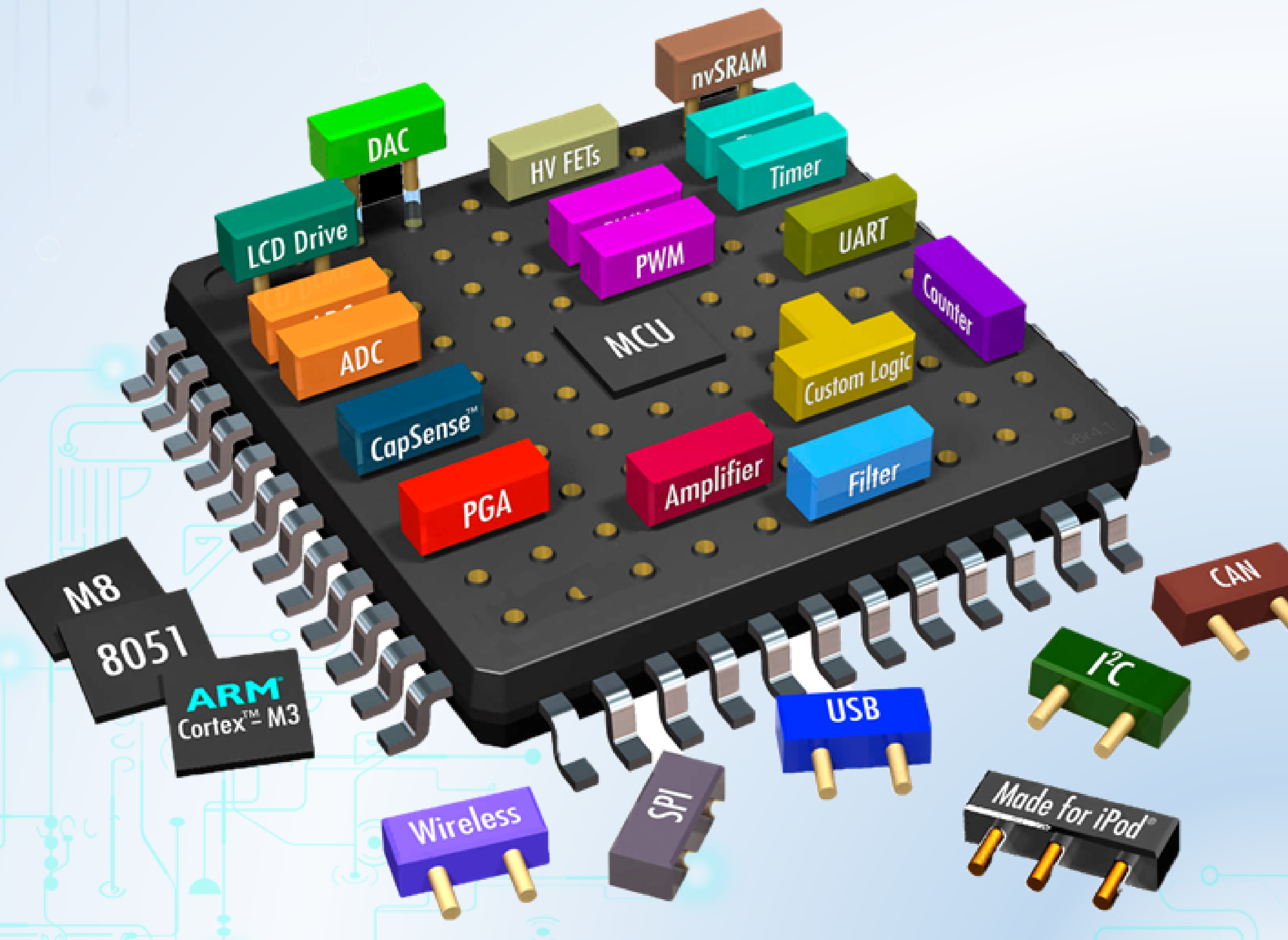


Dr. Le Trong Nhan

Mục lục

CHƯƠNG 1

LED Animations



1 Introduction

2 First project on STM32Cube

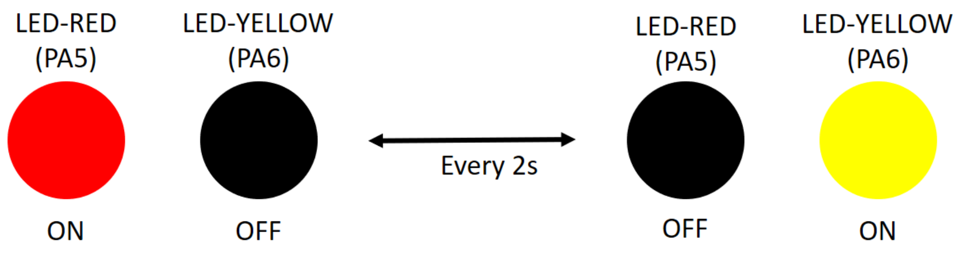
3 Simulation on Proteus

4 Exercise and Report

4.1 Exercise 1

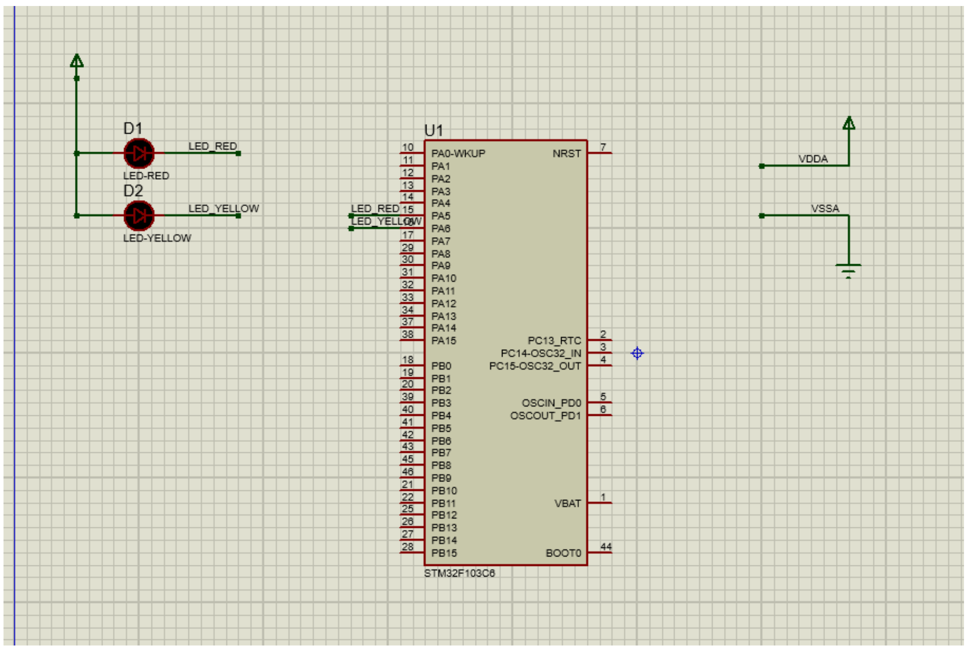
From the simulation on Proteus, one more LED is connected to pin **PA6** of the STM32 (negative pin of the LED is connected to PA6). The component suggested in this exercise is **LED-YELLOW**, which can be found from the device list.

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: Schematic:



*Hình 1.2: Schematic for Exercise 1 - **Lab1_ex1.pdsprj***

Report 2: Source code:

This code will toggle two led every 2 seconds. It use variable i to control what Led to turn on or off.

```c
int i = 0;
while (1)
{
    if (i == 1) {
    HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_SET);
    } else {
    HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET);
    }
    HAL_Delay(2000);
    i = (i + 1) % 2;
}
```

Program 1.1: Source code for lab 1

4.2 Exercise 2

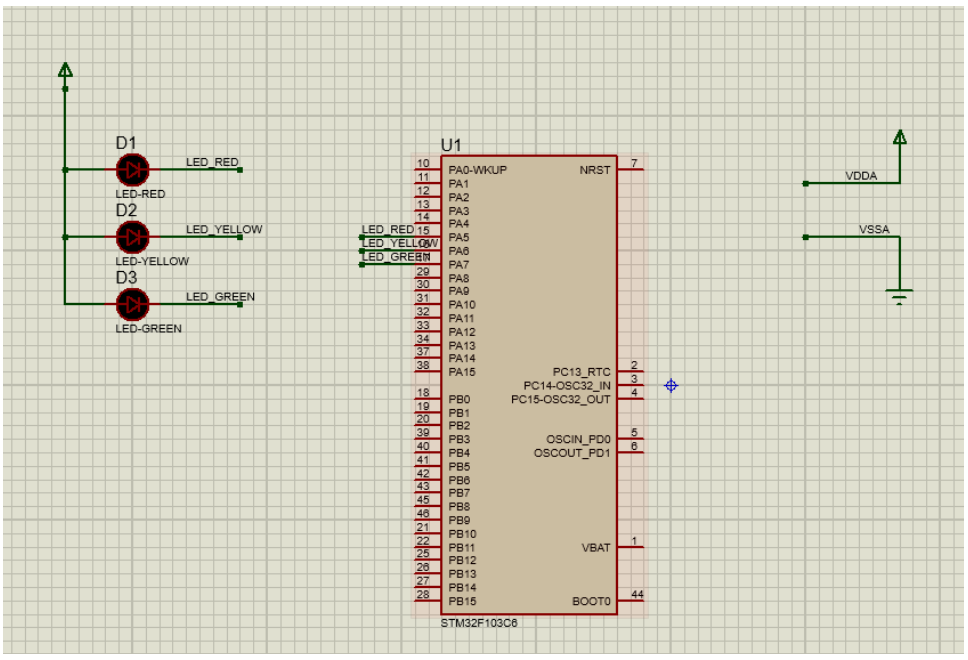
Extend the first exercise to simulate the behavior of a traffic light. A third LED, named **LED-GREEN** is added to the system, which is connected to **PA7**. A cycle in this traffic light is 5 seconds for the RED, 2 seconds for the YELLOW and 3 seconds for the GREEN. The LED-GREEN is also controlled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

Report 1: Schematic:

Report 2: Source code:

Variable i is decrease every 1 second, start from 10 to 0 and return back to 10. If i >= 5, the red light will turn on. If i >= 3, yellow light turn on or else the green light turn on.

```c
int i = 10;
while (1)
{
    i = i - 1;
    if (i >= 5)
    {
    HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);
    } else if (i >= 3)
```



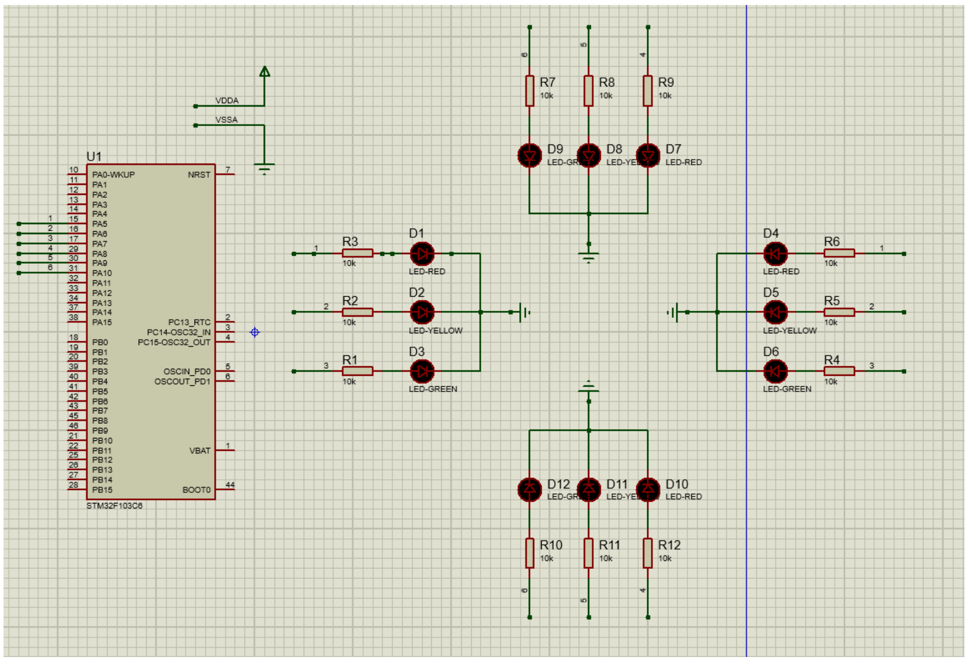
*Hình 1.3: Schematic for Exercise 2 - **Lab1_ex2.pdsprj***

```
    {
    HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);
    } else
    {
    HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_RESET);
    }
    HAL_Delay(1000);
    if (i == 0) {
    i = 10;
    }
}
```
Program 1.2: Source code for lab 2

4.3 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.

Report 1: Schematic:



*Hình 1.4: Schematic for Exercise 3 - **Lab1_ex3.pdsprj***

Report 2: Source code:

This code control 4 way traffic light.

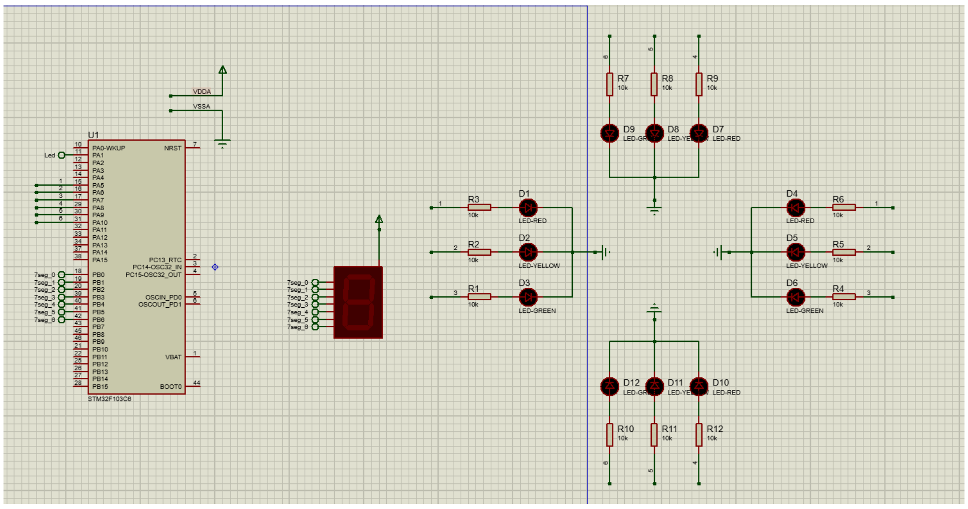
```
int i = 10;
while (1)
  {
  i = i - 1;
  switch(i) {
  case 9: {
          // led 2 Xanh
  HAL_GPIO_WritePin(LED_RED_1_GPIO_Port, LED_RED_1_Pin,
   GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED_YELLOW_1_GPIO_Port,
   LED_YELLOW_1_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED_GREEN_1_GPIO_Port, LED_GREEN_1_Pin,
    GPIO_PIN_SET);
          // led 1 do
  HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin,
   GPIO_PIN_SET);
  HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin,
   GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin,
   GPIO_PIN_RESET);
  break;
  }
  case 6: {
          // led 2 vang
  HAL_GPIO_WritePin(LED_RED_1_GPIO_Port, LED_RED_1_Pin,
   GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED_YELLOW_1_GPIO_Port,
 LED_YELLOW_1_Pin, GPIO_PIN_SET);
HAL_GPIO_WritePin(LED_GREEN_1_GPIO_Port, LED_GREEN_1_Pin,
  GPIO_PIN_RESET);
break;
  }
case 4: {
        //led 2 do
HAL_GPIO_WritePin(LED_RED_1_GPIO_Port, LED_RED_1_Pin,
 GPIO_PIN_SET);
HAL_GPIO_WritePin(LED_YELLOW_1_GPIO_Port,
 LED_YELLOW_1_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LED_GREEN_1_GPIO_Port, LED_GREEN_1_Pin,
  GPIO_PIN_RESET);
        // led 1 xanh

HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin,
 GPIO_PIN_RESET);
HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin,
 GPIO_PIN_RESET);
HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin,
 GPIO_PIN_SET);
break;
}
case 1:
{
      //led 1 vang
HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin,
 GPIO_PIN_RESET);
HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin,
 GPIO_PIN_SET);
HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin,
 GPIO_PIN_RESET);
 break;
 }
  }
HAL_Delay(1000);
if (i == 0) {
  i = 10;
}
}
```

Program 1.3: Source code for lab 3

4.4 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).

Report 1: Schematic:



*Hình 1.5: Schematic for Exercise 4- **Lab1_ex4.pdsprj***

Report 2: Source code:

We write a specific function to display each number on the led, and then combine it in the display7SEG function.

```
void Led_7Seg_0() {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
}

void Led_7Seg_1() {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
}

void Led_7Seg_2() {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
```

```c
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
}

void Led_7Seg_3() {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
}

void Led_7Seg_4() {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
}

void Led_7Seg_5() {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
}

void Led_7Seg_6() {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
}

void Led_7Seg_7() {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
}

void Led_7Seg_8() {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
}

void Led_7Seg_9() {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
}

void display7SEG(int value) {
  switch(value) {
  case 0:{
      Led_7Seg_0();
      break;
  }
  case 1:{
      Led_7Seg_1();
      break;
    }
  case 2:{
      Led_7Seg_2();
      break;
    }
  case 3:{
      Led_7Seg_3();
      break;
    }
  case 4:{
      Led_7Seg_4();
      break;
    }
  case 5:{
      Led_7Seg_5();
```

```c
            break;
        }
    case 6:{
            Led_7Seg_6();
            break;
        }
    case 7:{
            Led_7Seg_7();
            break;
        }
    case 8:{
            Led_7Seg_8();
            break;
        }
    case 9:{
            Led_7Seg_9();
            break;
        }
    }
}
```

Program 1.4: Source code for lab 4

4.5 Exercise 5

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

Report 1: Schematic:

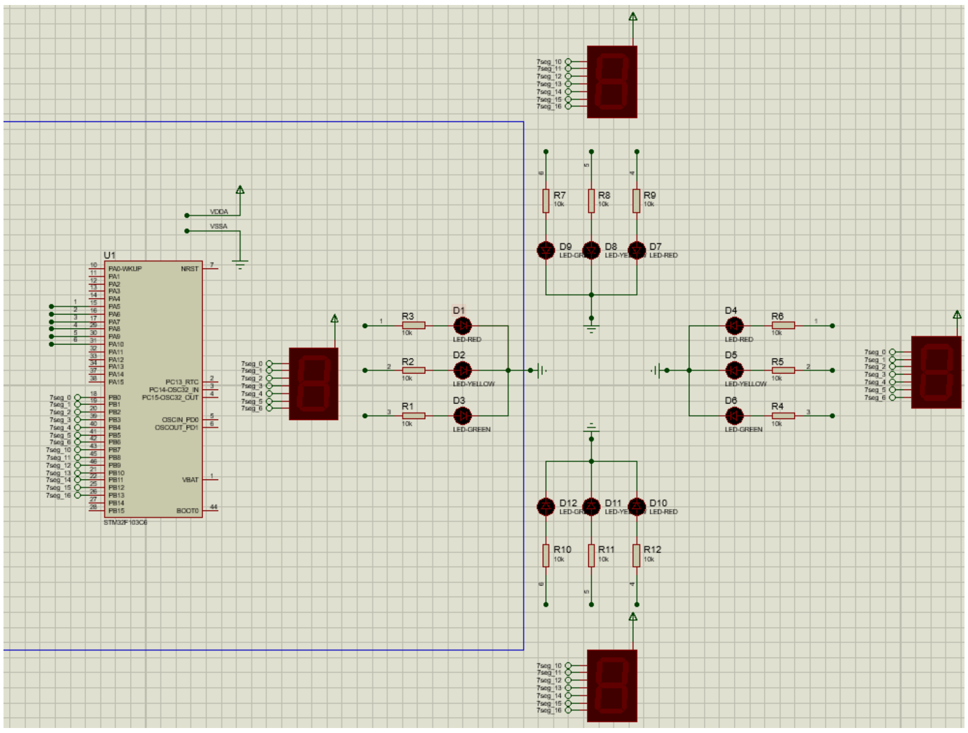
Report 2: Source code:

This code control 4 way traffic light and 4 7-segment LED. Since 2 7-segment LEDs which oppose each other count in the same way, we only need to control 2 7-segment LEDs.

Variable flag1,flag2 are used to control the state of the traffic light and the LED counter (road1,road2). When the counter for each road1,road2 is less than 0, we look at the state it's currently in to decide the next state and starting number to countdown.

Since we use HAL_Delay(10), we need to count 100 times and then decrease the time in road1,road2 so that the time will change every 1 second.

```c
enum color{RED, YELLOW, GREEN};
int count = 100 - 1;

int red = 5 - 1;
int yellow = 2 - 1;
int green = 3 - 1;

int flag1 = RED;
```



*Hình 1.6: Schematic for Exercise 5- **Lab1_ex5.pdsprj***

```
int road1 = red;

int flag2 = GREEN;
int road2 = green;
while (1)
{
  display7SEG(road1);
  display7SEG_1(road2);

  switch(flag1) {
  case RED:
  {
    HAL_GPIO_WritePin(GPIOA, LED_RED_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_YELLOW_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, LED_GREEN_Pin, GPIO_PIN_RESET);
    break;
  }
  case YELLOW:
  {
    HAL_GPIO_WritePin(GPIOA, LED_RED_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, LED_YELLOW_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_GREEN_Pin, GPIO_PIN_RESET);
    break;
  }
  case GREEN:
  {
    HAL_GPIO_WritePin(GPIOA, LED_RED_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, LED_YELLOW_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, LED_GREEN_Pin, GPIO_PIN_SET);
    break;
  }
  }

  switch(flag2) {
  case RED:
  {
    HAL_GPIO_WritePin(GPIOA, LED_RED_1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_YELLOW_1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, LED_GREEN_1_Pin, GPIO_PIN_RESET);
    break;
  }
  case YELLOW:
  {
    HAL_GPIO_WritePin(GPIOA, LED_RED_1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, LED_YELLOW_1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_GREEN_1_Pin, GPIO_PIN_RESET);
    break;
  }
  case GREEN:
  {
    HAL_GPIO_WritePin(GPIOA, LED_RED_1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, LED_YELLOW_1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, LED_GREEN_1_Pin, GPIO_PIN_SET);
    break;
  }
  }
  count--;
  if (count < 0) {
    count = 100 - 1;
```

```
      road1--;
      road2--;
      if (road1 < 0) {
        if (flag1 == RED) {
          road1 = green;
          flag1 = GREEN;
        } else if (flag1 == GREEN) {
          road1 = yellow;
          flag1 = YELLOW;
        } else { //if yellow -> turn red
          road1 = red;
          flag1 = RED;
        }
      }
      if (road2 < 0) {
        if (flag2 == RED) {
          road2 = green;
          flag2 = GREEN;
        } else if (flag2 == GREEN) {
          road2 = yellow;
          flag2 = YELLOW;
        } else { //if yellow -> turn red
          road2 = red;
          flag2 = RED;
        }
      }
    }
    HAL_Delay(10);
  }
}
```

Program 1.5: Source code for lab 5

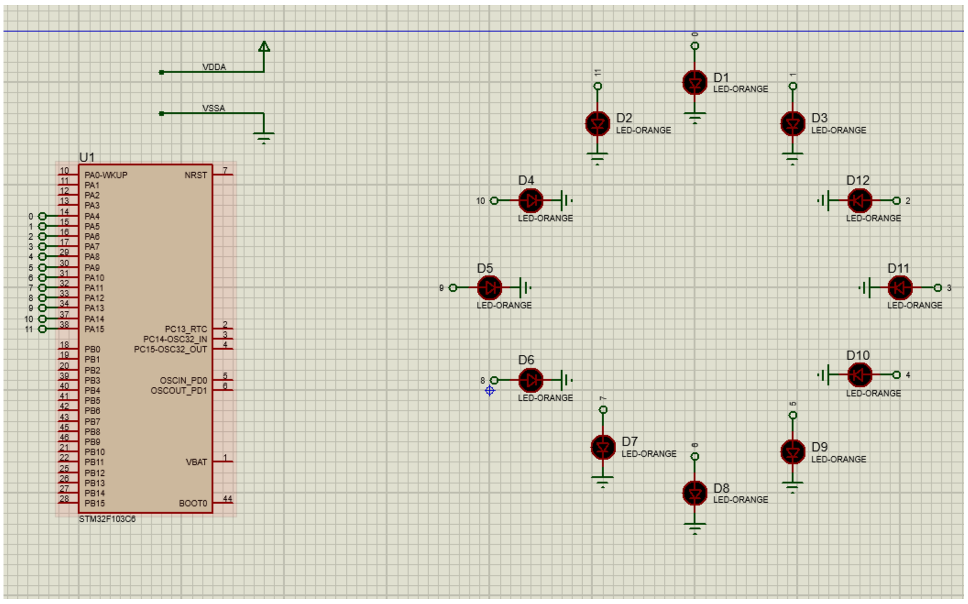
4.6 Exercise 6

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.

Report 1: Schematic:

Report 2: Source code:

```
int counter = 0;
while (1)
{
  switch(counter) {
  case 0:
  {
    HAL_GPIO_TogglePin(GPIOA, LED1_Pin);
    break;
```



Hình 1.7: Schematic for Exercise 6- ***Lab1_ex6.pdsprj***

```
    }
    case 1:
    {
      HAL_GPIO_TogglePin(GPIOA, LED2_Pin);
      break;
    }
    case 2:
    {
      HAL_GPIO_TogglePin(GPIOA, LED3_Pin);
      break;
    }
    case 3:
    {
      HAL_GPIO_TogglePin(GPIOA, LED4_Pin);
      break;
    }
    case 4:
    {
      HAL_GPIO_TogglePin(GPIOA, LED5_Pin);
      break;
    }
    case 5:
    {
      HAL_GPIO_TogglePin(GPIOA, LED6_Pin);
      break;
    }
    case 6:
    {
      HAL_GPIO_TogglePin(GPIOA, LED7_Pin);
      break;
```

```c
  }
  case 7:
  {
    HAL_GPIO_TogglePin(GPIOA, LED8_Pin);
    break;
  }
  case 8:
  {
    HAL_GPIO_TogglePin(GPIOA, LED9_Pin);
    break;
  }
  case 9:
  {
    HAL_GPIO_TogglePin(GPIOA, LED10_Pin);
    break;
  }
  case 10:
  {
    HAL_GPIO_TogglePin(GPIOA, LED11_Pin);
    break;
  }
  case 11:
  {
    HAL_GPIO_TogglePin(GPIOA, LED12_Pin);
    break;
  }

  }
  counter = (counter + 1) % 12;
  HAL_Delay(100);
}
```

Program 1.6: Source code for lab 6

4.7 Exercise 7

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

Report Source code:

To turn off all the led, we simply set all the pin to zero (GPIO_PIN_RESET)

```c
void clearAllClock(void)
{
  HAL_GPIO_WritePin(GPIOA, LED1_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED2_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED3_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED4_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED5_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED6_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED7_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED8_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED9_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED10_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED11_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, LED12_Pin, GPIO_PIN_RESET);
}
```

Program 1.7: Source code for lab 7

4.8 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.

Report Source code:

To turn on an invidual Led, we only need to set its Pin to high. Using switch case, turn on a single led won't affect other leds.

```
void setNumberOnClock(int num) {
  switch(num)
  {
      case 0:
      {
        HAL_GPIO_WritePin(GPIOA, LED1_Pin, GPIO_PIN_SET);
        break;
      }
      case 1:
      {
      HAL_GPIO_WritePin(GPIOA, LED2_Pin, GPIO_PIN_SET);
        break;
      }
      case 2:
      {
      HAL_GPIO_WritePin(GPIOA, LED3_Pin, GPIO_PIN_SET);
        break;
      }
      case 3:
      {
      HAL_GPIO_WritePin(GPIOA, LED4_Pin, GPIO_PIN_SET);
        break;
      }
      case 4:
      {
      HAL_GPIO_WritePin(GPIOA, LED5_Pin, GPIO_PIN_SET);
        break;
      }
      case 5:
      {
      HAL_GPIO_WritePin(GPIOA, LED6_Pin, GPIO_PIN_SET);
```

```
            break;
        }
        case 6:
        {
        HAL_GPIO_WritePin(GPIOA, LED7_Pin, GPIO_PIN_SET);
            break;
        }
        case 7:
        {
        HAL_GPIO_WritePin(GPIOA, LED8_Pin, GPIO_PIN_SET);
            break;
        }
        case 8:
        {
        HAL_GPIO_WritePin(GPIOA, LED9_Pin, GPIO_PIN_SET);
            break;
        }
        case 9:
        {
        HAL_GPIO_WritePin(GPIOA, LED10_Pin, GPIO_PIN_SET);
            break;
        }
        case 10:
        {
        HAL_GPIO_WritePin(GPIOA, LED11_Pin, GPIO_PIN_SET);
            break;
        }
        case 11:
        {
        HAL_GPIO_WritePin(GPIOA, LED12_Pin, GPIO_PIN_SET);
            break;
        }
    }
}
```

Program 1.8: Source code for lab 8

4.9 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.

Report: Source code:

This code is the same as function setNumberOnClock, but now the pin will be set to low.

```
void clearNumberOnClock(int num) {
    switch(num)
    {
    case 0:
```

```
    {
      HAL_GPIO_WritePin(GPIOA, LED1_Pin, GPIO_PIN_RESET);
      break;
    }
    case 1:
    {
    HAL_GPIO_WritePin(GPIOA, LED2_Pin, GPIO_PIN_RESET);
      break;
    }
    case 2:
    {
    HAL_GPIO_WritePin(GPIOA, LED3_Pin, GPIO_PIN_RESET);
      break;
    }
    case 3:
    {
    HAL_GPIO_WritePin(GPIOA, LED4_Pin, GPIO_PIN_RESET);
      break;
    }
    case 4:
    {
    HAL_GPIO_WritePin(GPIOA, LED5_Pin, GPIO_PIN_RESET);
      break;
    }
    case 5:
    {
    HAL_GPIO_WritePin(GPIOA, LED6_Pin, GPIO_PIN_RESET);
      break;
    }
    case 6:
    {
    HAL_GPIO_WritePin(GPIOA, LED7_Pin, GPIO_PIN_RESET);
      break;
    }
    case 7:
    {
    HAL_GPIO_WritePin(GPIOA, LED8_Pin, GPIO_PIN_RESET);
      break;
    }
    case 8:
    {
    HAL_GPIO_WritePin(GPIOA, LED9_Pin, GPIO_PIN_RESET);
      break;
    }
    case 9:
    {
    HAL_GPIO_WritePin(GPIOA, LED10_Pin, GPIO_PIN_RESET);
      break;
    }
    case 10:
    {
    HAL_GPIO_WritePin(GPIOA, LED11_Pin, GPIO_PIN_RESET);
      break;
    }
    case 11:
    {
    HAL_GPIO_WritePin(GPIOA, LED12_Pin, GPIO_PIN_RESET);
      break;
    }
    }
}
```

Program 1.9: Source code for lab 9

4.10 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.

Report: Source code:

We use variable hour, minute and second; second will count from 0 to 59, when it change to 0, minute incease by 1. This is the same with minute to hour. Every loop, we clear the clock, and then set on it second, minute and hour consecutively.

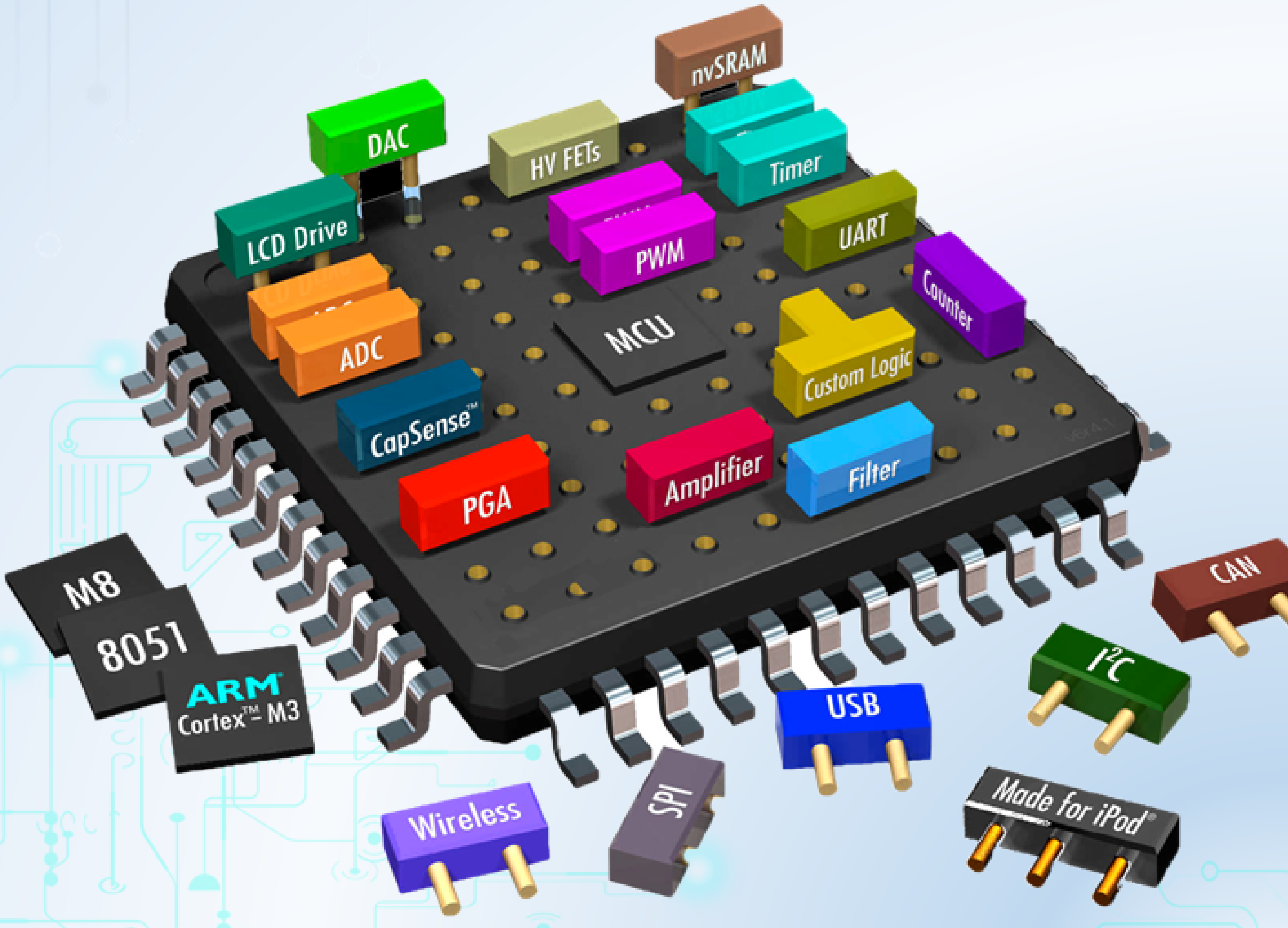
```
int counter = 100 - 1;
uint32_t second = 0;
uint32_t minute = 0;
uint32_t hour = 0;
void showNumber60_unit(int value) {
  setNumberOnClock(value / 5);
}

void showNumber12_unit(int value) {
  setNumberOnClock(value);
}
while (1)
{
  clearAllClock();
  showNumber60_unit(second);
  showNumber60_unit(minute);
  showNumber12_unit(hour);
    counter--;
    if (counter < 0) {
      counter = 100 - 1;
      second = (second + 1) % 60;
      if (second == 0) {
        minute = (minute + 1) % 60;
        if (minute == 0) {
          hour = (hour + 1) % 12;
```

```
            }
        }
    }
    HAL_Delay(10);
}
```

Program 1.10: Source code for lab 10

CHƯƠNG 2

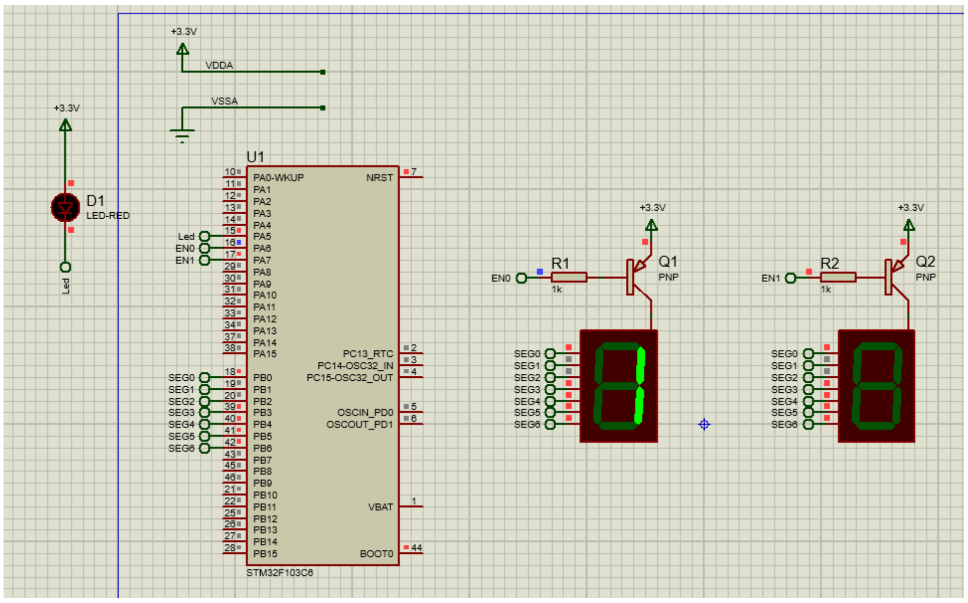
Timer Interrupt and LED Scanning



1 Exercise and Report

1.1 Exercise 1

Report 1: Schematic:



*Hình 2.1: Schematic for Exercise 1 - **Lab2_ex1.pdsprj***

Report 2: Source code:

Because the timer is set to interrupt every 10ms, so we use variable counter, count 100 times equal 1 second.

In this code, to make sure that each 7-seg LED is turned on half a second, we use variable counter, count from 99 to 0. When counter is >= 50 (from 99 to 50, count 50 times), state is LED1 and the first 7-seg LED is turned on. Otherwise, state = LED2 and the second 7-seg LED is turned on.

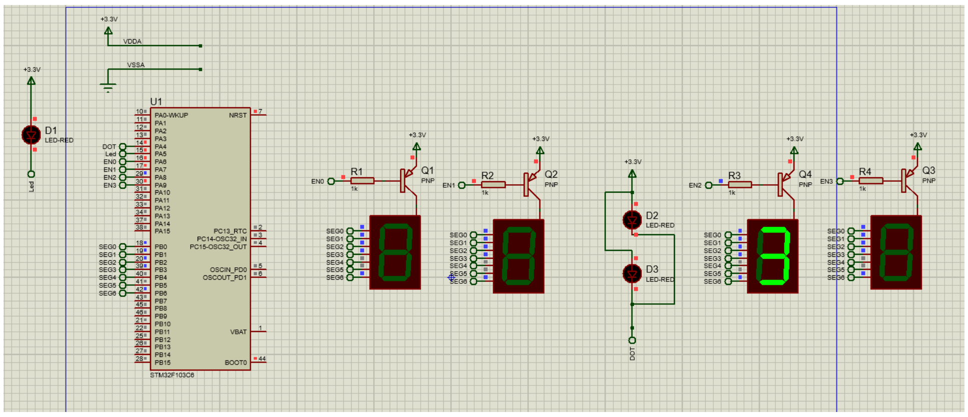
```c
int counter = 100 - 1;     //count from 99 to 0
enum scanLed{LED1, LED2};
int state = LED1;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
    {
  if (counter >= 50) {
    state = LED1;
  } else {
    state = LED2;
  }
  switch(state) {
  case LED1:
  {
    HAL_GPIO_WritePin(GPIOA, LED_SEG_0_Pin, GPIO_PIN_RESET)
   ;
    HAL_GPIO_WritePin(GPIOA, LED_SEG_1_Pin, GPIO_PIN_SET);
    display7SEG(1);
    break;
  }
  case LED2:
  {
    HAL_GPIO_WritePin(GPIOA, LED_SEG_0_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_SEG_1_Pin, GPIO_PIN_RESET)
   ;
    display7SEG(2);
    break;
  }
  }
  counter--;
  if (counter < 0) {
    counter = 100 - 1;
    HAL_GPIO_TogglePin(Led_GPIO_Port, Led_Pin);
  }
}
```

Program 2.1: Source code for ex1

Short question: Because 2 7-seg LED is scanned every 1 second, so the frequency is 1/1 = 1Hz.

1.2 Exercise 2

Report 1: Schematic:



Hình 2.2: Schematic for Exercise x- ***Lab2_ex2.pdsprj***

Report 2: Source code:

We extend the above code from Exercise 1: add more state in scanLed and turn on each 7-seg LED with at each state. For the two dot, we can wait till the counter reset its value (1 second) and toggle the dot.

```
int counter = 100 - 1;    //count from 99 to 0
enum scanLed{LED1, LED2, LED3, LED4};
int state = LED4;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
    {
```

```
  //state control
  if (counter % 50 == 49) {
    switch(state) {
    case LED1:
    {
      state = LED2;
      break;
    }
    case LED2:
    {
      state = LED3;
      break;
    }
    case LED3:
    {
      state = LED4;
      break;

    }
    case LED4:
    {
      state = LED1;
      break;
    }
    }
  }
  //what do we have at each state?
  //turn on the LED SEG by setting its pin to LOW
  switch(state) {
  case LED1:
  {
    HAL_GPIO_WritePin(GPIOA, LED_SEG_0_Pin, GPIO_PIN_RESET)
   ;
    HAL_GPIO_WritePin(GPIOA, LED_SEG_1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_SEG_2_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_SEG_3_Pin, GPIO_PIN_SET);
    display7SEG(1);
    break;
  }
  case LED2:
  {
    HAL_GPIO_WritePin(GPIOA, LED_SEG_0_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_SEG_1_Pin, GPIO_PIN_RESET)
   ;
    HAL_GPIO_WritePin(GPIOA, LED_SEG_2_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_SEG_3_Pin, GPIO_PIN_SET);
    display7SEG(2);
    break;
  }
  case LED3:
  {
    HAL_GPIO_WritePin(GPIOA, LED_SEG_0_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_SEG_1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_SEG_2_Pin, GPIO_PIN_RESET)
 ;
    HAL_GPIO_WritePin(GPIOA, LED_SEG_3_Pin, GPIO_PIN_SET);
    display7SEG(3);
    break;
  }
  case LED4:
  {
    HAL_GPIO_WritePin(GPIOA, LED_SEG_0_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_SEG_1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_SEG_2_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LED_SEG_3_Pin, GPIO_PIN_RESET)
 ;
    display7SEG(0);
    break;
  }

  }
  counter--;
  if (counter < 0) {
    counter = 100 - 1;
    HAL_GPIO_TogglePin(Led_GPIO_Port, Led_Pin);
    HAL_GPIO_TogglePin(DOT_GPIO_Port, DOT_Pin);
  }
}
```

Program 2.2: Source code for ex 2

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

In order to scan 4 7-seg LED, we need 2 second (half a second for each 7-seg LED). So, the frequency is 1/2 = 0.5Hz

1.3 Exercise 3

We can write update7SEG function base on the above Exercise. In this Exercise, I noticed there was a delay between each time we turn on the LED, the value of the previous 7-seg LED is carried to the next 7-seg LED for a little time before updated to its value. So, I decided when change to a new 7-seg LED, we have to turn off the previous 7seg LED first, then update the right value for the new 7seg LED, and turn the new LED on. Doing that will make sure that the new 7seg LED won't carry old value, or the previous 7-seg LED change to the new value right before we turn it off.

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

```
void update7SEG(int index) {
```

```
switch(index) {
case 0:
{
  //truoc khi bat den moi, ta phai
  // + tat den cu~ di (neu ko tat den cu~ truoc, den cu~
 se
  // hien gia tri moi, gay ra delay.)
  // + cap nhat lai gia tri cho den moi
  // + bat den moi len.
  HAL_GPIO_WritePin(GPIOA, LED_SEG_3_Pin, GPIO_PIN_SET);
  display7SEG(led_buffer[0]);
  HAL_GPIO_WritePin(GPIOA, LED_SEG_0_Pin, GPIO_PIN_RESET)
 ;
  HAL_GPIO_WritePin(GPIOA, LED_SEG_1_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOA, LED_SEG_2_Pin, GPIO_PIN_SET);

  break;
}
case 1:
{
  HAL_GPIO_WritePin(GPIOA, LED_SEG_0_Pin, GPIO_PIN_SET);
  display7SEG(led_buffer[1]);

  HAL_GPIO_WritePin(GPIOA, LED_SEG_1_Pin, GPIO_PIN_RESET)
 ;
  HAL_GPIO_WritePin(GPIOA, LED_SEG_2_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOA, LED_SEG_3_Pin, GPIO_PIN_SET);
  break;
}

case 2:
{
  HAL_GPIO_WritePin(GPIOA, LED_SEG_1_Pin, GPIO_PIN_SET);
  display7SEG(led_buffer[2]);
  HAL_GPIO_WritePin(GPIOA, LED_SEG_0_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(GPIOA, LED_SEG_2_Pin, GPIO_PIN_RESET)
 ;
  HAL_GPIO_WritePin(GPIOA, LED_SEG_3_Pin, GPIO_PIN_SET);
  break;
}

case 3:
{
  HAL_GPIO_WritePin(GPIOA, LED_SEG_2_Pin, GPIO_PIN_SET);
  display7SEG(led_buffer[3]);
  HAL_GPIO_WritePin(GPIOA, LED_SEG_0_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOA, LED_SEG_1_Pin, GPIO_PIN_SET);

```

```
    HAL_GPIO_WritePin(GPIOA, LED_SEG_3_Pin, GPIO_PIN_RESET)
  ;
    break;
  }
  default:
  {
    break;
  }

  }
}
```

Program 2.3: Source code for ex 3

Report 2: Present the source code in the

We put function update7SEG inside the if (counter % 50 == 49) statement, to make sure that update7SEG is called every time half a second has passed.

```
int counter = 100 - 1;    //count from 99 to 0
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
  {
  //state control
  if (counter % 50 == 49) {
    update7SEG(index_led);
    index_led = (index_led + 1) % MAX_LED;
  }
  counter--;
  if (counter < 0) {
    counter = 100 - 1;
    HAL_GPIO_TogglePin(Led_GPIO_Port, Led_Pin);
    HAL_GPIO_TogglePin(DOT_GPIO_Port, DOT_Pin);
  }
}
```

Program 2.4: Source code for ex 3

1.4 Exercise 4

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

In order increase the frequency to 1Hz, we need to scan all the 7seg LED every 1 second, then each 7-seg LED is turn on for one fourth of a second. We simply change the statement if to (counter % 25 == 24). Then, every time the counter has counted 25 times, it trigger function update7SEG and also, increase the value index_led in range 0 to 3.

```
int counter = 100 - 1;    //count from 99 to 0
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
  {
  //state control
  if (counter % 25 == 24) {
    update7SEG(index_led);
    index_led = (index_led + 1) % MAX_LED;
  }
  counter--;
  if (counter < 0) {
    counter = 100 - 1;
    HAL_GPIO_TogglePin(Led_GPIO_Port, Led_Pin);
    HAL_GPIO_TogglePin(DOT_GPIO_Port, DOT_Pin);
  }
}
```

Program 2.5: Source code for ex 4

1.5 Exercise 5

We can write updateClockBuffer() function as follow. Variable hour, minute, second and array led_buffer is declared as global variable, so we don't need to pass it to the function. led_buffer[0] and led_buffer[2] is created by divide hour and minute by 10. led_buffer[1] and led_buffer[3] is created by mod hour and minute by 10.

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

Program 2.6: Source code for ex 5

1.6 Exercise 6

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

if setTimer0(1000); is missing, then timer_counter = 0, because of the condition in timer_run(), since timer_counter = 0, timer_run() will do nothing. That mean that timer0_flag will always equal 0, so the Led won't toggle.

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

if setTimer0(1000) is changed to setTimer0(1), timer_counter = 0. It act the same as report 1, Led won't toggle.

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?

if setTimer0(1000) is changed to setTimer0(10), timer_counter = 1; this time, timer_run() can do it work and set the timer0_flag to 1 after 10ms. After that, LED will toggle every 2 seconds as written in while().

1.7 Exercise 7

Report 1: Source code

In this code, we only update values for hour, minute and second when timer0_flag is set. We also change Hal_Delay with setTimer0(1000), and bring the control code for the DOT into while(1). Before while(1), we use setTimer0(1000) to make sure that timer work (the flag is set), and the clock will update its value after every 1 second.

```
setTimer0(1000);    //1000ms = 1s
while (1)
{
  if (timer0_flag == 1) {
    second++;
    if (second >= 60) {
      second = 0;
      minute++;
    }
    if (minute >= 60) {
      minute = 0;
      hour++;
    }
    if (hour >= 24) {
      hour = 0;
    }
    updateClockBuffer();
    HAL_GPIO_TogglePin(Led_GPIO_Port, Led_Pin);
    HAL_GPIO_TogglePin(DOT_GPIO_Port, DOT_Pin);
    setTimer0(1000);
  }
}
```

Program 2.7: Source code for ex 7

1.8 Exercise 8

Report 1: Source code:

In this code, we change the timer to interrupt every 250ms. Because of that, variable counter is used, which decrease after 250ms - whenever the timer0_flag is set. DOT, hour, minute and second will only update its value if counter = 3. When counter decrease from 3 to 0 and reset to 3 again, 1 second has passed.

Beside, in order to scan 4 7-seg LED, index_led change every 250ms (timer0_flag is set) and we use it to turn on each 7-seg LED using update7SEG function.

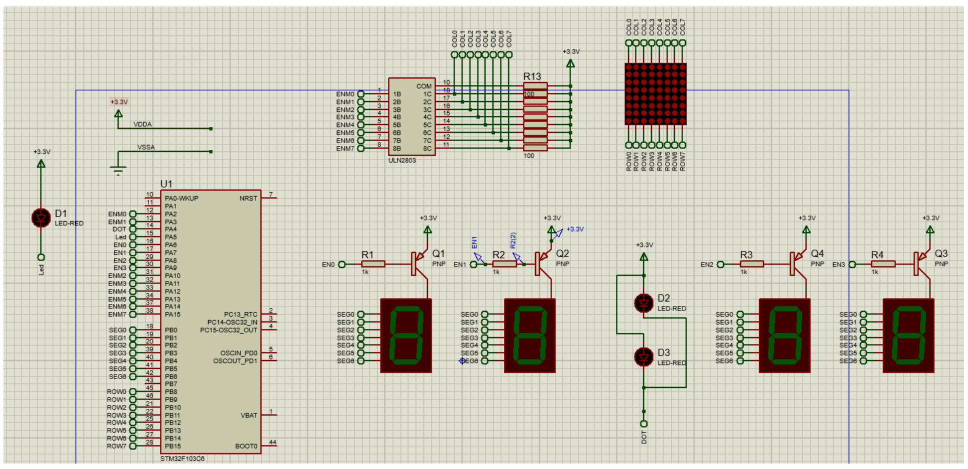
```
setTimer0(1000);
int counter = 4 - 1;
while (1)
{
  if (timer0_flag == 1) {
    //cap nhat TRUOC khi in
    if (counter == 4 - 1) {
      second++;
      if (second >= 60) {
```

```c
        second = 0;
        minute++;
      }
      if (minute >= 60) {
        minute = 0;
        hour++;
      }
      if (hour >= 24) {
        hour = 0;
      }
      HAL_GPIO_TogglePin(Led_GPIO_Port, Led_Pin);
      HAL_GPIO_TogglePin(DOT_GPIO_Port, DOT_Pin);
    }
    updateClockBuffer();
    update7SEG(index_led);
    index_led = (index_led + 1) % 4;
    counter--;
    if (counter < 0) {
      counter = 4 - 1;
    }
    setTimer0(250);
  }
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  timer0_run();
}
```

Program 2.8: Source code for ex 8

1.9 Exercise 9

Report 1: Schematic:



*Hình 2.3: Schematic for Exercise 9- **Lab2_ex9.pdsprj***

Report 2: Source code:

In this code, we do the following:

- Since we can't control each individual LED on a led-matrix, we'll deal with it by quickly scan through (turn on) one column of the led-matrix from column 0 to column 7. At a specific time, only one column is turned on, and we will control which LED row should be turned on for that column (by setting the row pin to low).
- We see each data in matrix_buffer correspond to which led row should be turned on when we turn on a specific column. For example, matrix_buffer[0] = 0x02 (0000_0010), it mean that when we turn on column 0 of the 8x8 led matrix, led at row 1 will be turn on, other leds is turned off.
- To be able to use data in matrix_buffer, we need a way to check the data and decide which led row to turn on. Function **displayMatrixCol** is designed to do that. It extract the binary representation of the data by continuously get the remainder of data when divided by 2, and replace the data with the result of that data divided by 2. Do that 8 times, using the remainder, we could know and control the state of each Led row.
- Now return back to the **updateLEDMatrix** function. To simplify code, each time the function is called, it first turn off all the column. And then, base on the index passed to it, the function decide which column to turned on, and also which led rows should be turned on in that column. For example, when index = 0, **displayMatrixCol** will read the data at matrix_buffer[0] and turn on the right row (by setting the row pin to low) and then turn on column 0 (by setting the column pin to high)
- To control led-matrix, we also create a new timer for it, and let it scan each column every 10ms. It mean that, the frequency of led-matrix is 1/0.08 = 12.5Hz

```
//CONTROL LED MATRIX 8X8
void displayMatrixCol(uint8_t value) {
  int i = 0;
  //ROW 0
  i = value % 2;
  if (i == 1) {
    //bat den
    HAL_GPIO_WritePin(ROW0_GPIO_Port, ROW0_Pin,
   GPIO_PIN_RESET);
  } else {
    //tat den
    HAL_GPIO_WritePin(ROW0_GPIO_Port, ROW0_Pin,
   GPIO_PIN_SET);
  }
  value = value >> 1;
  //ROW 1
  i = value % 2;
  if (i == 1) {
    //bat den
    HAL_GPIO_WritePin(ROW1_GPIO_Port, ROW1_Pin,
 GPIO_PIN_RESET);
  } else {
    //tat den
    HAL_GPIO_WritePin(ROW1_GPIO_Port, ROW1_Pin, GPIO_PIN_SET);
  }
  value = value >> 1;
  //ROW 2
  i = value % 2;
  if (i == 1) {
    //bat den
    HAL_GPIO_WritePin(ROW2_GPIO_Port, ROW2_Pin, GPIO_PIN_RESET);
  } else {
    //tat den
    HAL_GPIO_WritePin(ROW2_GPIO_Port, ROW2_Pin, GPIO_PIN_SET);
  }
  value = value >> 1;
  //ROW 3
  i = value % 2;
  if (i == 1) {
    //bat den
    HAL_GPIO_WritePin(ROW3_GPIO_Port, ROW3_Pin, GPIO_PIN_RESET);
  } else {
    //tat den
    HAL_GPIO_WritePin(ROW3_GPIO_Port, ROW3_Pin, GPIO_PIN_SET);
  }
  value = value >> 1;
  //ROW 4
  i = value % 2;
  if (i == 1) {
    //bat den
    HAL_GPIO_WritePin(ROW4_GPIO_Port, ROW4_Pin, GPIO_PIN_RESET);
  } else {
    //tat den
    HAL_GPIO_WritePin(ROW4_GPIO_Port, ROW4_Pin, GPIO_PIN_SET);
  }
  value = value >> 1;
  //ROW 5
  i = value % 2;
  if (i == 1) {
    //bat den
    HAL_GPIO_WritePin(ROW5_GPIO_Port, ROW5_Pin, GPIO_PIN_RESET);
```

```c
  } else {
    //tat den
    HAL_GPIO_WritePin(ROW5_GPIO_Port, ROW5_Pin, GPIO_PIN_SET);
  }
  value = value >> 1;
  //ROW 6
  i = value % 2;
  if (i == 1) {
    //bat den
    HAL_GPIO_WritePin(ROW6_GPIO_Port, ROW6_Pin, GPIO_PIN_RESET);
  } else {
    //tat den
    HAL_GPIO_WritePin(ROW6_GPIO_Port, ROW6_Pin, GPIO_PIN_SET);
  }
  value = value >> 1;
  //ROW 7
  i = value % 2;
  if (i == 1) {
    //bat den
    HAL_GPIO_WritePin(ROW7_GPIO_Port, ROW7_Pin, GPIO_PIN_RESET);
  } else {
    //tat den
    HAL_GPIO_WritePin(ROW7_GPIO_Port, ROW7_Pin, GPIO_PIN_SET);
  }
}
const int MAX_LED_MATRIX = 8;
int index_led_matrix = 0;
uint8_t matrix_buffer[8] = {0x00, 0xfc, 0xfe, 0x33, 0x33, 0xfe, 0xfc, 0x00};
void updateLEDMatrix(int index) {
  HAL_GPIO_WritePin(GPIOA, ENM0_Pin|ENM1_Pin|ENM2_Pin|ENM3_Pin|ENM4_Pin|ENM5_Pin|ENM6_Pin|ENM7_Pin, GPIO_PIN_SET);
  switch(index) {
  case 0:
  {
    displayMatrixCol(matrix_buffer[0]);
    HAL_GPIO_WritePin(ENM0_GPIO_Port, ENM0_Pin, GPIO_PIN_RESET);
    break;
  }
  case 1:
  {
    displayMatrixCol(matrix_buffer[1]);
```

```
      HAL_GPIO_WritePin(ENM1_GPIO_Port, ENM1_Pin,
     GPIO_PIN_RESET);
      break;
    }
    case 2:
    {
      displayMatrixCol(matrix_buffer[2]);
      HAL_GPIO_WritePin(ENM2_GPIO_Port, ENM2_Pin,
     GPIO_PIN_RESET);
      break;
    }
    case 3:
    {
      displayMatrixCol(matrix_buffer[3]);
      HAL_GPIO_WritePin(ENM3_GPIO_Port, ENM3_Pin,
     GPIO_PIN_RESET);
      break;
    }
    case 4:
    {
      displayMatrixCol(matrix_buffer[4]);
      HAL_GPIO_WritePin(ENM4_GPIO_Port, ENM4_Pin,
     GPIO_PIN_RESET);
      break;
    }
    case 5:
    {
      displayMatrixCol(matrix_buffer[5]);
      HAL_GPIO_WritePin(ENM5_GPIO_Port, ENM5_Pin,
     GPIO_PIN_RESET);
      break;
    }
    case 6:
    {
      displayMatrixCol(matrix_buffer[6]);
      HAL_GPIO_WritePin(ENM6_GPIO_Port, ENM6_Pin,
     GPIO_PIN_RESET);
      break;
    }
    case 7:
    {
      displayMatrixCol(matrix_buffer[7]);
      HAL_GPIO_WritePin(ENM7_GPIO_Port, ENM7_Pin,
     GPIO_PIN_RESET);
      break;
    }
    default:
    {
      break;
```

```
  }
  }
}
```

Program 2.9: Source code for ex 9

1.10 Exercise 10

Report 1: Source code:

We decided to make character is shifted to the right and turn back. We create a function that shift the data in matrix_buffer : **shiftMatrixBuffer**.

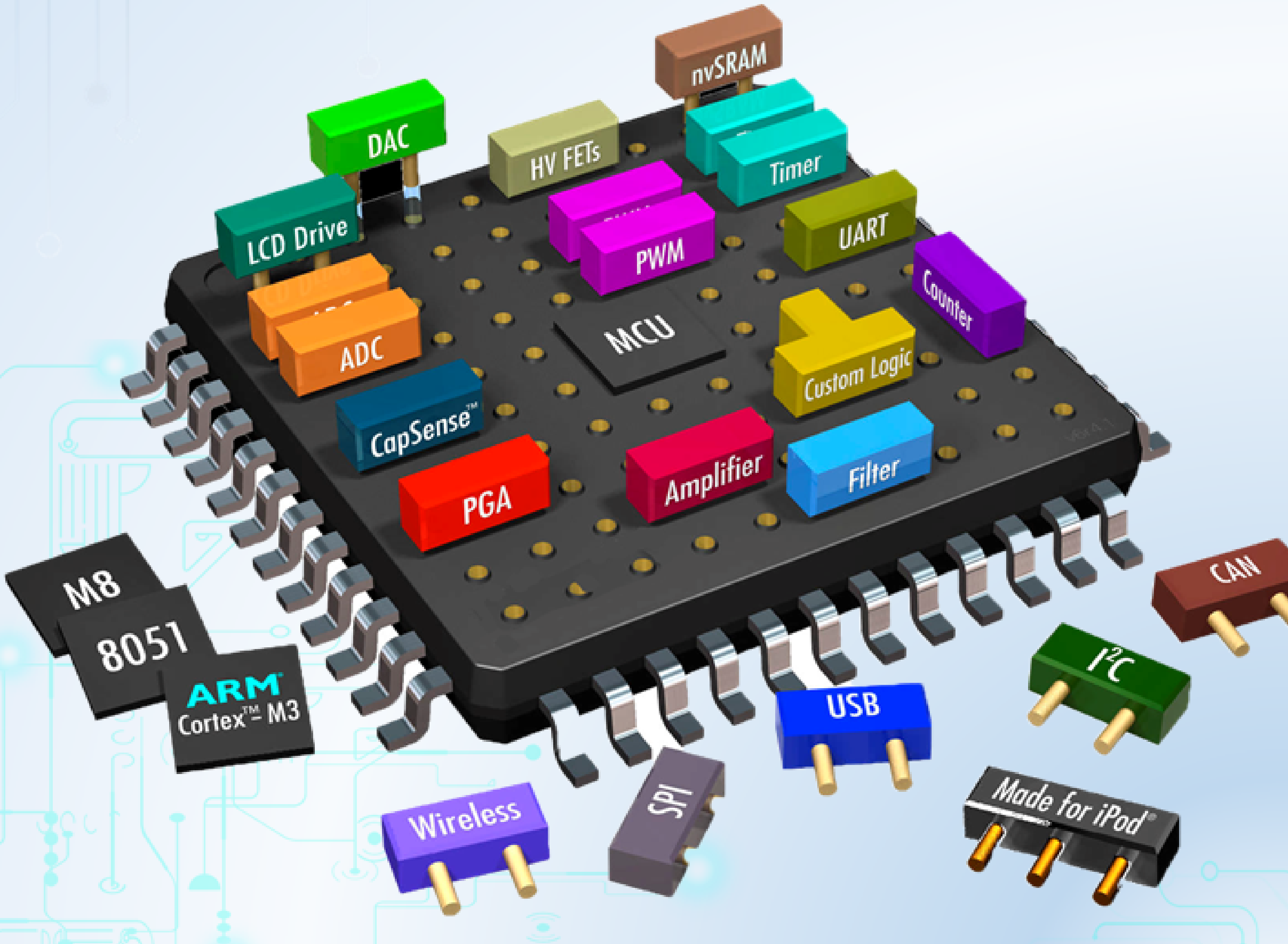
We also introduce a new variable *shiftCounter* to make the shift happen (call shiftMatrixBuffer) every 2 seconds. We use the previous function from Exercise 9 to print led-matrix.

```
void shiftMatrixBuffer() {
  uint8_t temp = matrix_buffer[7];
  matrix_buffer[7] = matrix_buffer[6];
  matrix_buffer[6] = matrix_buffer[5];
  matrix_buffer[5] = matrix_buffer[4];
  matrix_buffer[4] = matrix_buffer[3];
  matrix_buffer[3] = matrix_buffer[2];
  matrix_buffer[2] = matrix_buffer[1];
  matrix_buffer[1] = matrix_buffer[0];
  matrix_buffer[0] = temp;
}

setTimer0(10);
setTimer1(10);
int first = 1;
int counter = 4 - 1;
int shiftCounter = 2 - 1;
while (1)
{
  if (timer0_flag == 1) {
    //cap nhat TRUOC khi in
    if (counter == 4 - 1) {
      if (first != 1) {
        second++;
        if (second >= 60) {
          second = 0;
          minute++;
        }
        if (minute >= 60) {
          minute = 0;
          hour++;
        }
        if (hour >= 24) {
          hour = 0;
        }
      } else {
        first = 0;
      }
      HAL_GPIO_TogglePin(Led_GPIO_Port, Led_Pin);
      HAL_GPIO_TogglePin(DOT_GPIO_Port, DOT_Pin);

      //xu li dich phai.
      shiftCounter--;
      if (shiftCounter < 0) {
        shiftCounter = 2 - 1;
        shiftMatrixBuffer();
      }
    }
    updateClockBuffer();
    update7SEG(index_led);
    index_led = (index_led + 1) % 4;
    counter--;
    if (counter < 0) {
      counter = 4 - 1;
    }
    setTimer0(250);
  }
  if (timer1_flag == 1) {
    updateLEDMatrix(index_led_matrix);
    index_led_matrix = (index_led_matrix + 1) %
 MAX_LED_MATRIX;
    setTimer1(10);
  }
}
```

Program 2.10: Source code for ex 10

CHƯƠNG 3

Buttons/Switches



1 Exercises and Report

1.1 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.

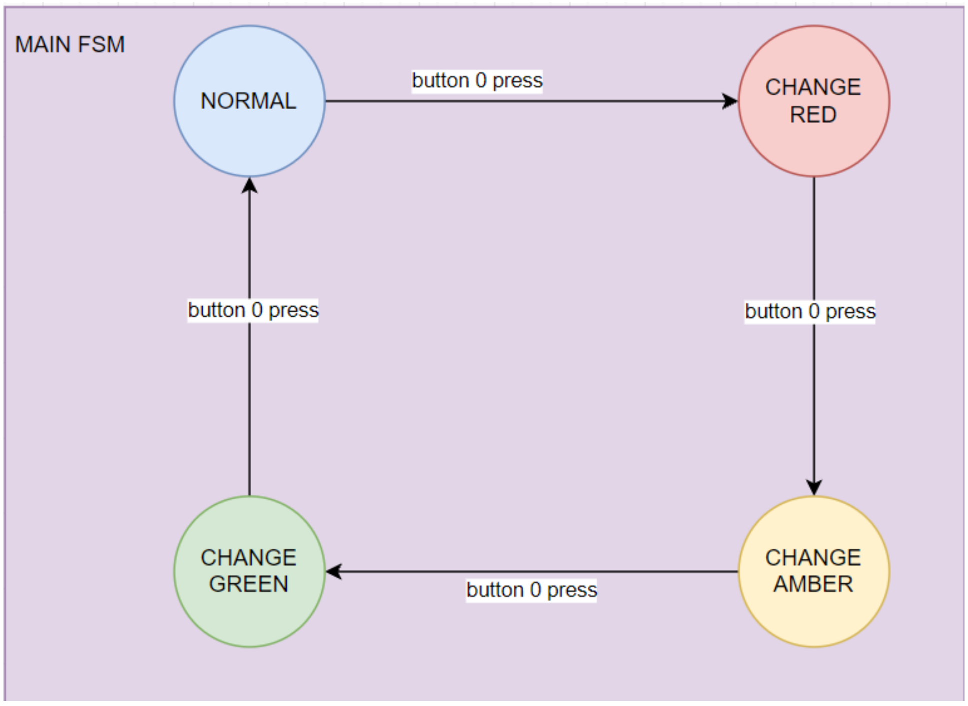
  - The third button is used to set the value.

  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.

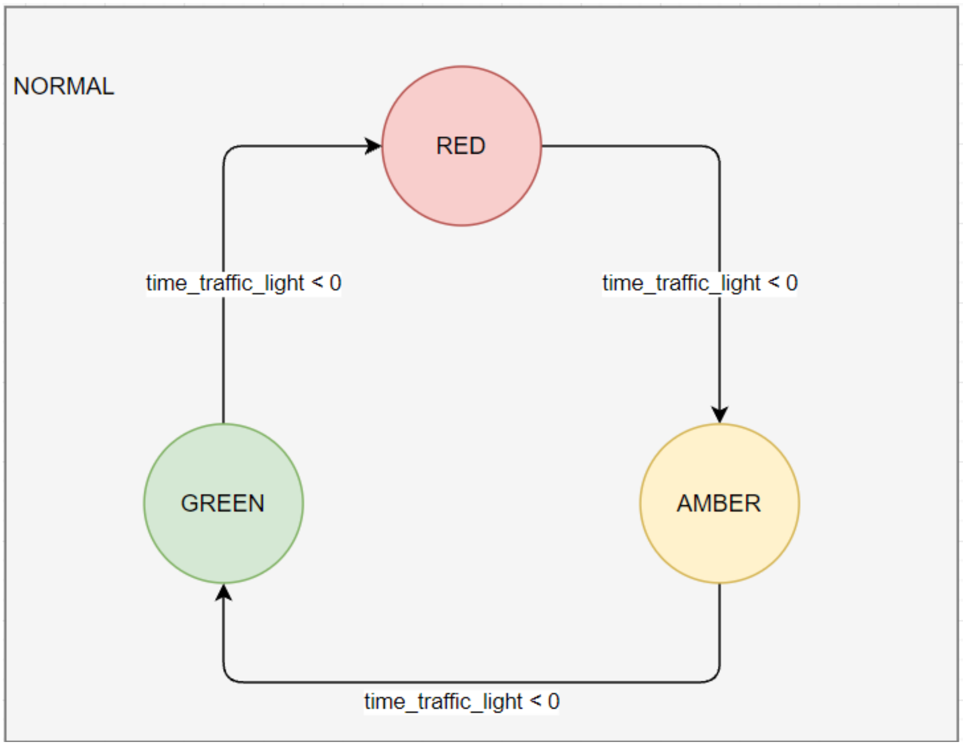
1.2 Exercise 1: Sketch an FSM

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



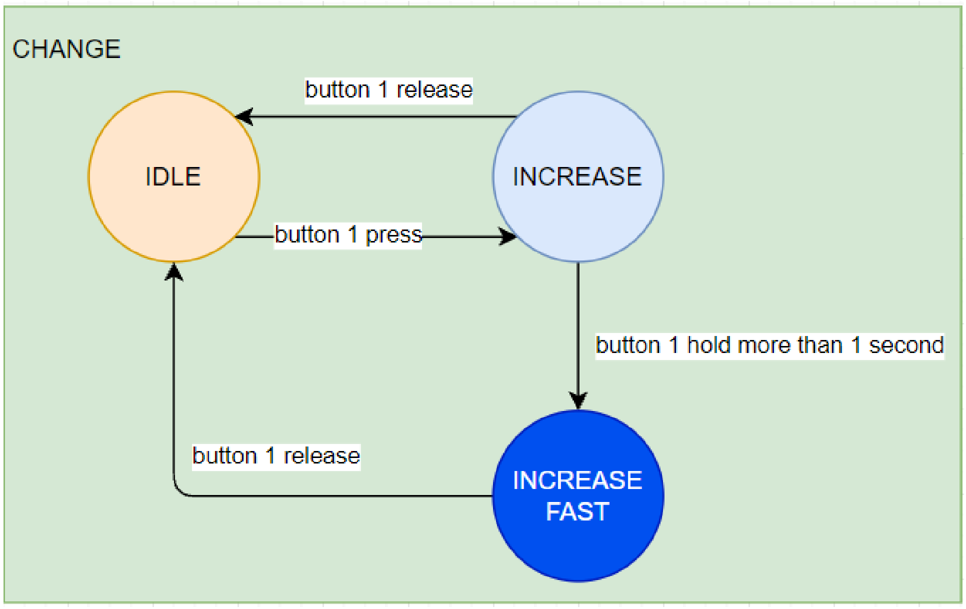
Hình 3.1: Main FSM

This is our main FSM, which have 4 states for 4 Modes. In Normal mode (mode when traffic light work normally) and Change mode (change red, change amber or change green) each of them have one more FSM to handle functioning



Hình 3.2: FSM for normal mode

In this normal mode, we have a time_traffic_light act as a counter and time. For each road, base on the state the traffic light is currently in, when the time_traffic_light is less than 0, it change to the next state.

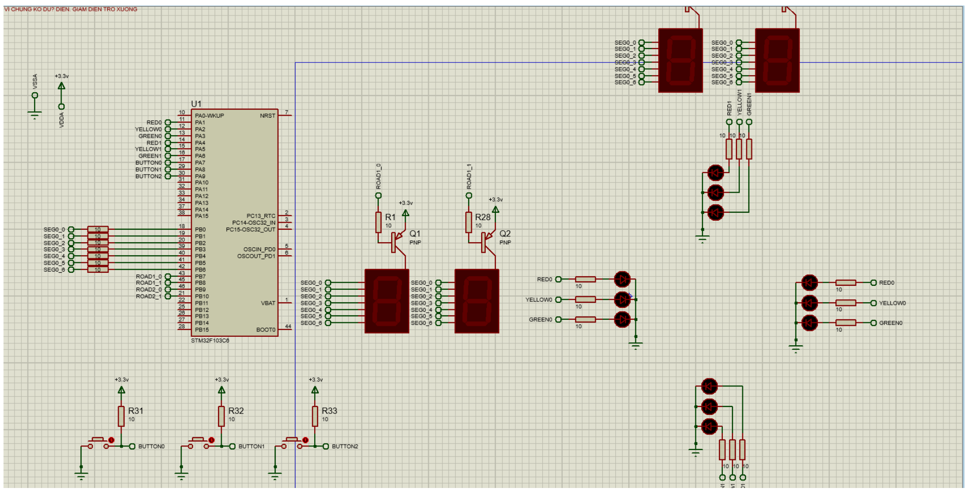


Hình 3.3: FSM for change mode

This FSM is used for 3 mode: CHANGE_RED, CHANGE_AMBER and CHANGE_GREEN. It has 3 state. When user press button 1, it increase the value of the time_duration in the mode it's currently in. If button is press longer than 1 second, the value increase quickly (5 time per sec).

1.3 Exercise 2: Proteus Schematic

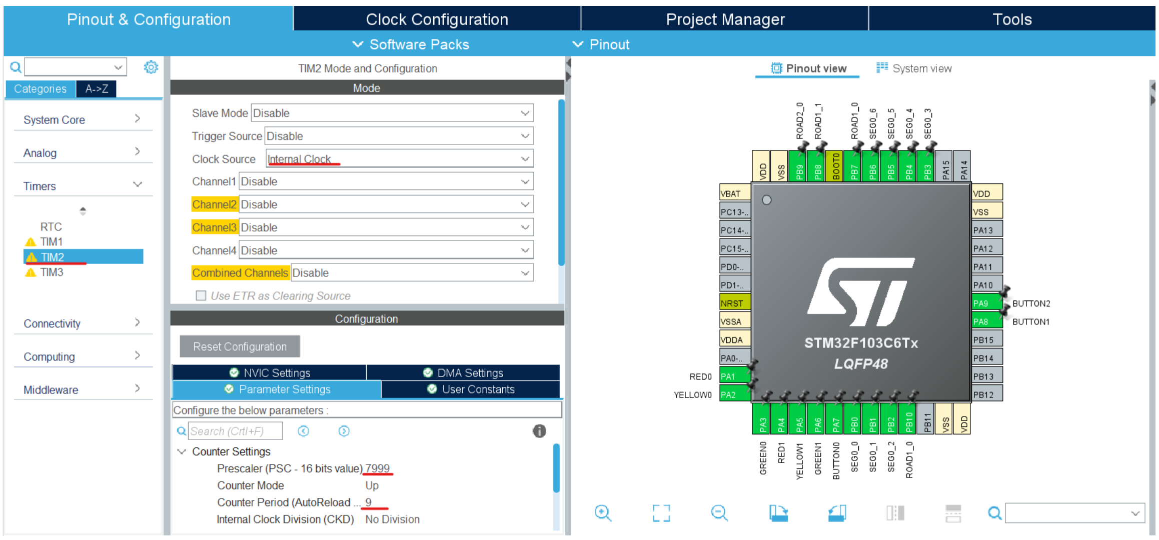
Your task in this exercise is to draw a Proteus schematic for the problem above.



Hình 3.4: Schematic for traffic light - ***light_traffic.pdsprj***

1.4 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.



TIM2 Mode and Configuration

Mode

Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Disable
Channel2 Disable
Channel3 Disable
Channel4 Disable
Combined Channels Disable
Use ETR as Clearing Source

Configuration

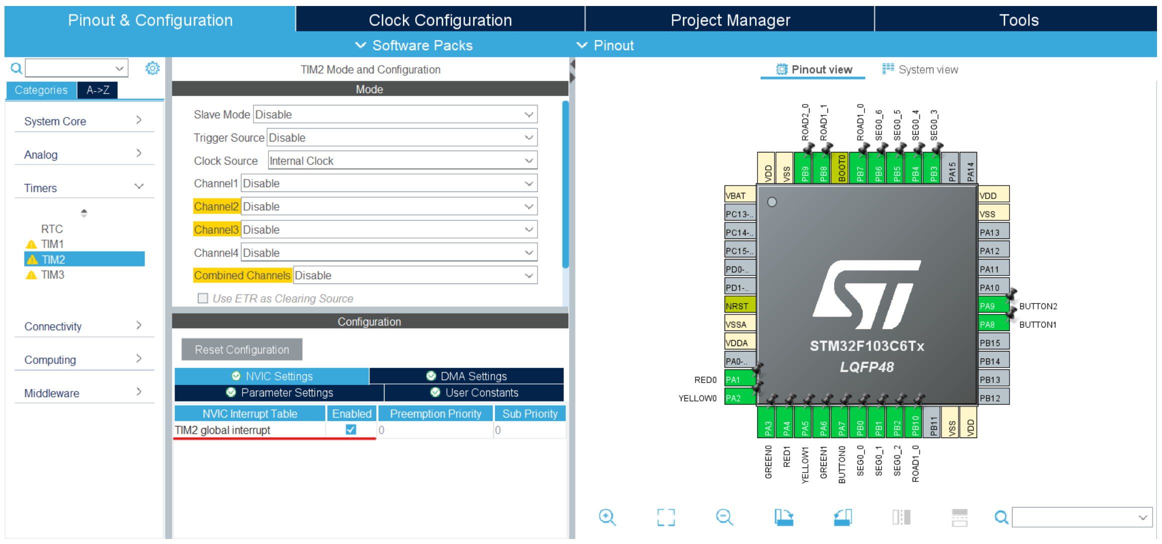
Reset Configuration

NVIC Settings | DMA Settings | Parameter Settings | User Constants

Configure the below parameters :

Counter Settings
Prescaler (PSC - 16 bits value) 7999
Counter Mode Up
Counter Period (AutoReload ... 9
Internal Clock Division (CKD) No Division

Hình 3.5: Create project, set up timer and pins



TIM2 Mode and Configuration

Mode

Slave Mode	Disable
Trigger Source	Disable
Clock Source	Internal Clock
Channel1	Disable
Channel2	Disable
Channel3	Disable
Channel4	Disable
Combined Channels	Disable

☐ Use ETR as Clearing Source

Configuration

Reset Configuration

NVIC Settings | DMA Settings | Parameter Settings | User Constants

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM2 global interrupt	✓	0	0

Hình 3.6: Create project, set up timer and pins

1.5 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.

To do this, I've write 2 file timer.h and timer.c as follow:

```c
/*
 * timer.h
 *
 *  Created on: Sep 28, 2021
 *      Author: ngocc
 */

#ifndef INC_TIMER_H_
#define INC_TIMER_H_

#define NO_OF_TIMER 3
//change this value as interrupt time changed.
#define TIMER_CYCLE 10

void setTimer(int num_of_timer, int duration);

void initTimer(int num_of_timer);

void timer_run(int num_of_timer);

int getTimerFlag(int num_of_timer);

```

```
#endif /* INC_TIMER_H_ */
```

Program 3.1: timer.h

```
/*
 * timer.c
 *
 *  Created on: Sep 28, 2021
 *      Author: ngocc
 */
#include "timer.h"

static int timer_flag[NO_OF_TIMER];
static int timer_counter[NO_OF_TIMER];

void setTimer(int num_of_timer, int duration) {
  if (num_of_timer >= NO_OF_TIMER) return;
  timer_counter[num_of_timer] = duration / TIMER_CYCLE;
  timer_flag[num_of_timer] = 0;
}

void initTimer(int num_of_timer) {
  if (num_of_timer >= NO_OF_TIMER) return;
  timer_flag[num_of_timer] = 1;
}

void timer_run(int num_of_timer) {
  if (num_of_timer >= NO_OF_TIMER) return;
  if(timer_counter[num_of_timer] > 0) {
    timer_counter[num_of_timer]--;
    if (timer_counter[num_of_timer] == 0) {
      timer_flag[num_of_timer] = 1;
    }
  }
}

int getTimerFlag(int num_of_timer) {
  if (num_of_timer >= NO_OF_TIMER) return -1;
  return timer_flag[num_of_timer];
}
```

Program 3.2: timer.c

When every we need to change timer interrupt duration, simply change the value of TIMER_CYCLE in timer.h file. Lowest value for this TIMER_CYCLE is 1, which is 1 milisecond for timer interrupt duration. We can easily change number of timer needed to use by changing NO_OF_TIMER. When changing TIMER_CYCLE, as long as duration passed to setTimer() function is divisible by TIMER_CYCLE, then the program will work the same as before changing.

1.6 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.

Below is the code section for input reading:

```c
#ifndef INC_INPUT_READING_H_
#define INC_INPUT_READING_H_

//if change NO_OF_BUTTON , go to input_reading and change
   pinValue + buttonBuffer
#define NO_OF_BUTTON    3
//wait time until debouncing
#define WAIT_TIME       2
#define DURATION_1_SECOND   100

#define BUTTON_PORT   GPIOA
#define PRESS      GPIO_PIN_RESET
#define RELEASE     GPIO_PIN_SET

void read_button();

int getButtonValue(int num_of_button);
int getFlagButtonPress1s(int num_of_button);

//this flag can only be pull down if button is released.
void setFlagButtonDelay(int num_of_button);
int getFlagButtonDelay(int num_of_button);
#endif /* INC_INPUT_READING_H_ */
```

Program 3.3: input_reading.h

In input_reading.h, We have

- **NO_OF_BUTTON**: number of button needed.
- **WAIT_TIME**: amount of time need to wait to debouncing button
- **BUTTON_PORT**: since I usually connect all the button in one port, we specify this port
- **PRESS, RELEASE**: state of the input pin when a button when pressed or released.

```c
#include "main.h"
#include "input_reading.h"
//button pin value
static int pinValue[NO_OF_BUTTON] = {0x0080, 0x0100, 0x0200
   };
```

```c

//buffer for button and a buffer before debouncing
static int buttonBufferBeforeDebouncing[NO_OF_BUTTON]= {
    RELEASE, RELEASE, RELEASE};
static int buttonBuffer[NO_OF_BUTTON] = {RELEASE, RELEASE,
    RELEASE};

//counter for button press and debouncing wait time
static int counterForButtonPress[NO_OF_BUTTON];
static int waitTimeTillDebouncing[NO_OF_BUTTON];

//some flags
static int flagButtonPress1s[NO_OF_BUTTON];
static int flagButtonDelay[NO_OF_BUTTON];

void read_button() {
  for(int i = 0; i < NO_OF_BUTTON; i++) {
    if (waitTimeTillDebouncing[i] > 1) {
      waitTimeTillDebouncing[i]--;
    } else {
      if (buttonBufferBeforeDebouncing[i] ==
    HAL_GPIO_ReadPin(BUTTON_PORT, pinValue[i])) {
        buttonBuffer[i] = buttonBufferBeforeDebouncing[i];
      } else {
        buttonBufferBeforeDebouncing[i] = HAL_GPIO_ReadPin(
    BUTTON_PORT, pinValue[i]);
        waitTimeTillDebouncing[i] = WAIT_TIME;
      }
      if (buttonBuffer[i] == PRESS) {
        if (counterForButtonPress[i] < DURATION_1_SECOND) {
          counterForButtonPress[i]++;
        } else {
          flagButtonPress1s[i] = 1;
        }
      } else {
        counterForButtonPress[i] = 0;
        flagButtonPress1s[i] = 0;
        flagButtonDelay[i] = 0;
      }
    }
  }
}

int getButtonValue(int num_of_button) {
  if (num_of_button >= NO_OF_BUTTON) return -1;
  return buttonBuffer[num_of_button];
}

int getFlagButtonPress1s(int num_of_button) {
  if (num_of_button >= NO_OF_BUTTON) return -1;
  return flagButtonPress1s[num_of_button];
}

//this flag can only be pull down if button is released.
void setFlagButtonDelay(int num_of_button) {
  if (num_of_button >= NO_OF_BUTTON) return;
  flagButtonDelay[num_of_button] = 1;
}

int getFlagButtonDelay(int num_of_button) {
  if(num_of_button >= NO_OF_BUTTON) return -1;
  return flagButtonDelay[num_of_button];
}
```

Program 3.4: input_reading.c

- **pinValue**: this array specify the address of the pin used for array.
- Base on NO_OF_BUTTON, we have a **buffer** and **bufferBeforeDebouncing** to store value of a button. bufferBeforeDebouncing is needed for debouncing, as buttonBuffer of a button will only be updated if **buttonBufferBeforeDebouncing** have value equal to the state of the button read currently from pin. If not, we'll assign the currently state of the button to the buttonBufferBeforeDebouncing, and wait for a little time before checking again.
- **counterForButtonPress**: each button has a counter, used to check duration of a button press.
- **waitTimeTillDebouncing**: each button has a wait time like this, when ever the button change state for the first time, this waitTime will be set to a value and decrease overtime. We only continue reading and processing value of button if this waitTime is equal to 1
- **flagButtonPress1s**: a flag that will be set if a button is press longer than 1 second
- **flagButtonDelay**: a special flag that only be set to low if button is released.
- **read_button()**: a function that handle reading button input value and debouncing.
- **getButtonValue()**: get value of a button from buffer
- **getFlagButtonPress1s()**: get the flag if button is press longer than 1 second
- **setFlagButtonDelay()**: a function to set the flagButtonDelay to high
- **getFlagButtonDelay()**: get value of flagButtonDelay

For displaying on 4 7-segment LED, we have the following code:

```
/*
 * 7_seg_display.h
 *
 *  Created on: Sep 22, 2021
```

```c
 *      Author: ngocc
 */

#ifndef INC_7_SEG_DISPLAY_H_
#define INC_7_SEG_DISPLAY_H_

#define NO_OF_LED7       4
#define PORT_USED_DISPLAY GPIOB
#define PORT_ENABLE      GPIOB
#define TURN_ON_SEGMENT   GPIO_PIN_RESET
#define TURN_OFF_SEGMENT  GPIO_PIN_SET
#define ENABLE_LED       GPIO_PIN_RESET
#define DISABLE_LED      GPIO_PIN_SET
//vi minh thuong noi cac pin cua led 7 doan chung voi nhau
//nen minh chi ro gia tri pin bat dau la bao nhieu, roi
//dung vong for shift qua gan gia tri cho nhanh
#define START_PIN      0x0001
//chan enable nhieu led minh cung thuong gan chung voi nhau
   , nen
//tuong tu, co start_enable_pin
#define START_ENABLE_PIN  0x0080

void display7SEG(int value);
//call this function to switch to the next LED and it's
   value
void scanLED();
//call this function to update buffer
void updateBuffer(int value1, int value2);

#endif /* INC_7_SEG_DISPLAY_H_ */
```

Program 3.5: 7_seg_display.h

- **NO_OF_LED7**: number of 7-seg LED used
- **PORT_USED_DISPLAY**: Port used to control the display of LED - port that connect to the segment of 7-seg LED
- **PORT_ENABLE**: Port used to turn on or off the 7-seg LED.
- **TURN_ON_SEGMENT, TURN_OFF_SEGMENT**: state of GPIO output pin needed to turn on or off a segment
- **ENABLE_LED, DISABLE_LED**: state of GPIO output pin needed to turn on or off the entire 7-seg LED.
- **START_PIN**: since we usally connect 7 pin of the chip to 7 pin of 7-seg LED continuously, we define a Starting Pin number
- **START_ENABLE_PIN**: same as START_PIN, but these pins is need to enable the 7-seg LED.
- **display7SEG**: this function is used to display value to the 7-seg LED

- **scanLED**: When this function is called, it will switch to the next led and it's value base on the buffer.

- **updateBuffer**: update buffer for 4 7-seg LED we're using.

```c
/*
 * 7_seg_display.c
 *
 *  Created on: Sep 22, 2021
 *      Author: ngocc
 */
#include "main.h"
#include "7_seg_display.h"



static int Led7Buffer[NO_OF_LED7] = {0,0,0,0};
static int currentLED = 0;
static uint8_t displayData[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D
                ,0x7D, 0X07, 0X7F, 0X6F};

//this function display value on a 7-seg LED
void display7SEG(int value) {
  uint8_t temp = displayData[value];
  int turnOnGroup = 0, turnOffGroup = 0;
  for(int i = 0; i < 7; i++) {
    if (temp % 2 == 1) {
      turnOnGroup = turnOnGroup | START_PIN << i;
    } else {
      turnOffGroup = turnOffGroup | START_PIN << i;
    }
    temp = temp >> 1;
  }
  HAL_GPIO_WritePin(PORT_USED_DISPLAY, turnOnGroup, TURN_ON_SEGMENT);
  HAL_GPIO_WritePin(PORT_USED_DISPLAY, turnOffGroup, TURN_OFF_SEGMENT);
}

//this function choose a 7-seg LED among multiple LED to turn on.
void scanLED() {
  //turn off all LED
  int group = START_ENABLE_PIN;
  for(int i = 0; i < NO_OF_LED7; i++) {
    group = group | (START_ENABLE_PIN << i);
  }
  HAL_GPIO_WritePin(PORT_ENABLE, group, DISABLE_LED);
  //update LED value
  display7SEG(Led7Buffer[currentLED]);
  //turn that LED on
  HAL_GPIO_WritePin(PORT_ENABLE, START_ENABLE_PIN <<
   currentLED, ENABLE_LED);

  currentLED = (currentLED + 1) % NO_OF_LED7;
}

//this function update Buffer. this buffer will be showed
   on all LEDs
void updateBuffer(int value1, int value2) {
  Led7Buffer[0] = value1/10;
  Led7Buffer[1] = value1%10;
  Led7Buffer[2] = value2/10;
  Led7Buffer[3] = value2%10;

}
```

Program 3.6: 7_seg_display.c

- **displayData** is a array contain information used by display7SEG. displayData[0] will help the function to display value 0 to the 7-seg LED, displayData[1] to display value 1, and so on.
- **display7SEG**: when receiving value, it get the displayData[value]. Using the data from displayData, it will extract the binary representation of the displayData, and if the bit is 1, turn on the segment, else turn it off. To prevent calling function to WritePin multiple time, we have turnOnGroup that group (using or operator) all the 1 bit that specify location to turn on segment on 7-seg LED and turnOffGroup for location to turn off segement, and put this into function WritePin once for each.
- **scanLED**: everytime this function is called, it will display the LED and its value base on currentLED variable. First, it turn off all the 7-seg LED. Then, it write the display Value of the currentLED buffer from Led7Buffer[currentLED], then, base on the currentLED, to turn on the right 7-seg LED. Since we connect all the enable pin in one port continuously, we can shift START_ENABLE_PIN to the left to get the next pin value. Then, currentLED is increase by 1, % NO_OF_LED7.
- **updateBuffer**: this function update the buffer of 7-seg LED based on the value passed.

We add more files to control the traffic light:

```c
#ifndef INC_TRAFFIC_LIGHT_H_
#define INC_TRAFFIC_LIGHT_H_

#include "7_seg_display.h"
#include "main.h"
//this traffic light has it own timer for 1 second count.
#include "timer.h"

```

```c
#define NO_OF_ROAD 2
enum State_Traffic_Light{RED, AMBER, GREEN};

//initialization is inside traffic_light.c
extern int time_traffic_light[NO_OF_ROAD];
extern enum State_Traffic_Light state_traffic_light[
    NO_OF_ROAD];


extern int red_duration;
extern int amber_duration;
extern int green_duration;

void init_traffic_light();
//this function will be called every 1 second.
void run_traffic_light();
void run_traffic_light_timer();
void update_time_traffic_light();
void display_traffic_light();

#endif /* INC_TRAFFIC_LIGHT_H_ */
```

Program 3.7: traffic_light.h

```c
#include "traffic_light.h"
#define NO_OF_ROAD 2

//khong the su dung ~TRAFFIC_LIGHT_ENABLE duoc. no =/=
    GPIO_PIN_RESET
#define TRAFFIC_LIGHT_ENABLE  GPIO_PIN_SET
#define TRAFFIC_LIGHT_DISABLE GPIO_PIN_RESET
#define PORT_TRAFFIC_LIGHT    GPIOA
int red_duration = 5;
int amber_duration = 2;
int green_duration = 3;

int time_traffic_light[NO_OF_ROAD];
enum State_Traffic_Light state_traffic_light[NO_OF_ROAD];
static int trafficLightPin[NO_OF_ROAD][3] = {{0x0002, 0
    x0004, 0x0008},{0x0010, 0x0020, 0x0040}};


void init_traffic_light() {
  time_traffic_light[0] = red_duration;
  time_traffic_light[1] = green_duration;
  state_traffic_light[0] = RED;
  state_traffic_light[1] = GREEN;
  setTimer(1,1000);
}

```

```
//this function has it own timer. after 1 second, it update
   the traffic time.
void run_traffic_light() {
  if (getTimerFlag(1) == 1) {
    for(int i = 0; i < NO_OF_ROAD; i++) {
      time_traffic_light[i]--;
      if (time_traffic_light[i] < 0) {
        switch(state_traffic_light[i]) {
        case RED:
          state_traffic_light[i] = AMBER;
          time_traffic_light[i] = amber_duration;
          break;
        case AMBER:
          state_traffic_light[i] = GREEN;
          time_traffic_light[i] = green_duration;
          break;
        case GREEN:
          state_traffic_light[i] = RED;
          time_traffic_light[i] = red_duration;
          break;
        }
      }
    }
    setTimer(1,1000);
  }
}

void run_traffic_light_timer() {
  timer_run(1);
}

void display_traffic_light() {
  for(int i = 0; i < NO_OF_ROAD; i++) {
    //tai sao ta khong tat het (dung vong for) roi bat lai?
    //khi dung vong for tat het, proteus xu ly khong noi.
   co le vong for xu ly viec tat het
    //da khien code nang hon.
    //sau do bat den len theo trang thai:
    switch(state_traffic_light[i]) {
    case RED:
      HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
   [i][0], TRAFFIC_LIGHT_ENABLE);
      HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
   [i][1] | trafficLightPin[i][2]
                                    , TRAFFIC_LIGHT_DISABLE
   );
      break;
    case AMBER:
      HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
[i][1], TRAFFIC_LIGHT_ENABLE);
        HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
[i][0] | trafficLightPin[i][2]
                                            , TRAFFIC_LIGHT_DISABLE
);
        break;
      case GREEN:
        HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
[i][0] | trafficLightPin[i][1]
                                            , TRAFFIC_LIGHT_DISABLE
);
        HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
[i][2], TRAFFIC_LIGHT_ENABLE);
        break;
      }
  }
}

void update_time_traffic_light() {
  updateBuffer(time_traffic_light[0], time_traffic_light
[1]);
}
```

Program 3.8: traffic_light.c

- this traffic light file has its own timer: timer1. It also contain red_duration, amber_duration and green_duration - which are extern variable and can be used outside of the file.
- Each road has a time_traffic_light and state_traffic_light. time_traffic_light is the time of traffic light that will be display on 7-seg LED. state_traffic_light is needed for the small fsm inside run_traffic_function that help the traffic light changing light and time for counting down.
- trafficLightPin is the pin value that we use to control traffic light. We need to specify these value.
- init_traffic_light(): a function that initialize time and state for each traffic light on a road. It also set the timer for 1 second.
- run_traffic_light(): main function to run the traffic light. Only if the timer1 flag is set (after 1 second), time for both road will be decrease. If the time of a road is less than 0, It will look at the state it's currently in and change to another state and set new time value (duration) to the time_traffic_light.
- run_traffic_light_timer(): this function simply call timer_run(1) to run timer1. Need to be placed inside interrupt function
- display_traffic_light(): a function that use the state of each road to display traffic light.
- update_time_traffic_light(): a function that get the time of traffic light and pass to update function of 7-seg LED to display them.

And we also implement the main fsm, which is not complete as follow:

```c
#include "fsm_traffic_light.h"

enum Mode{NORMAL, CHANGE_RED, CHANGE_AMBER, CHANGE_GREEN};

enum Mode mode = NORMAL;
static int blinkingCounter = 25;

void fsm_traffic_light() {
  switch(mode) {
  case NORMAL:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
    == 0) {
      mode = CHANGE_RED;
      setFlagButtonDelay(0);
    }
    run_traffic_light();
    update_time_traffic_light();
    display_traffic_light();
    break;
  case CHANGE_RED:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
    == 0) {
      mode = CHANGE_AMBER;
      setFlagButtonDelay(0);
    }
    updateBuffer(0, 02);
    break;
  case CHANGE_AMBER:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
    == 0) {
      mode = CHANGE_GREEN;
      setFlagButtonDelay(0);
    }
    updateBuffer(0, 03);
    break;
  case CHANGE_GREEN:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
    == 0) {
      mode = NORMAL;
      setFlagButtonDelay(0);
      init_traffic_light();
    }
    updateBuffer(0, 04);
    break;
  }

}
```

Program 3.9: fsm_traffic_light.c

As for now, we can press button 0 to change between mode. To prevent mode changing continuously, only change the mode if button is pressed and flagButtonDelay is not set. Right before switching to the next mode, we set the flagButtonDelay high, and this flag will only set to low again if the button is release. With this, if we press button 0 and even hold it, we can only move to the next mode only once, and when we need to go to the next mode, we have to release the button and press it again.

1.7 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 traffic_light.c, we add 3 more function to help with toggling the LEDs. These function will toggle the LED whenever it's called.

```c
void toggleRedLight() {
  for(int i = 0; i < NO_OF_ROAD; i++) {
    if (toggleCounter == 1){
      HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
 [i][0], TRAFFIC_LIGHT_ENABLE);
    } else {
      HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
 [i][0], TRAFFIC_LIGHT_DISABLE);
    }
    HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin[i
][1] | trafficLightPin[i][2]
                                 , TRAFFIC_LIGHT_DISABLE);
  }
  toggleCounter--;
  if (toggleCounter < 0) toggleCounter = 1;
}

void toggleAmberLight() {
  for(int i = 0; i < NO_OF_ROAD; i++) {
    if (toggleCounter == 1){
      HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
 [i][1], TRAFFIC_LIGHT_ENABLE);
    } else {
      HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
 [i][1], TRAFFIC_LIGHT_DISABLE);
    }
    HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin[i
][0] | trafficLightPin[i][2]
                                 , TRAFFIC_LIGHT_DISABLE);
  }
  toggleCounter--;
  if (toggleCounter < 0) toggleCounter = 1;
}

void toggleGreenLight() {
  for(int i = 0; i < NO_OF_ROAD; i++) {
    if (toggleCounter == 1){
      HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
   [i][2], TRAFFIC_LIGHT_ENABLE);
    } else {
      HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin
   [i][2], TRAFFIC_LIGHT_DISABLE);
    }
    HAL_GPIO_WritePin(PORT_TRAFFIC_LIGHT, trafficLightPin[i
   ][1] | trafficLightPin[i][0]
                                      , TRAFFIC_LIGHT_DISABLE);
  }
  toggleCounter--;
  if (toggleCounter < 0) toggleCounter = 1;
}
```

Program 3.10: traffic_light.c +

We update fsm_traffic_light.c with blinking function. Add a blinkingCounter that will toggle the LED on each mode when blinkingCounter = 25.

```
#include "fsm_traffic_light.h"

enum Mode{NORMAL, CHANGE_RED, CHANGE_AMBER, CHANGE_GREEN};

enum Mode mode = NORMAL;
static int blinkingCounter = 25;

void fsm_traffic_light() {
  switch(mode) {
  case NORMAL:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
    == 0) {
      mode = CHANGE_RED;
      setFlagButtonDelay(0);
    }
    run_traffic_light();
    update_time_traffic_light();
    display_traffic_light();
    break;
  case CHANGE_RED:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
    == 0) {
      mode = CHANGE_AMBER;
      setFlagButtonDelay(0);
```

```c
      blinkingCounter = 25;
    }
    updateBuffer(red_duration, 02);
    if (blinkingCounter == 25) {
      toggleRedLight();
    }
    blinkingCounter--;
    if (blinkingCounter <= 0) {
      blinkingCounter = 25;
    }
    break;
  case CHANGE_AMBER:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
   == 0) {
      mode = CHANGE_GREEN;
      setFlagButtonDelay(0);
      blinkingCounter = 25;
    }
    updateBuffer(amber_duration, 03);
    if (blinkingCounter == 25) {
      toggleAmberLight();
    }
    blinkingCounter--;
    if (blinkingCounter <= 0) {
      blinkingCounter = 25;
    }
    break;
  case CHANGE_GREEN:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
   == 0) {
      mode = NORMAL;
      setFlagButtonDelay(0);
      init_traffic_light();
    }
    updateBuffer(green_duration, 04);
    if (blinkingCounter == 25) {
      toggleGreenLight();
    }
    blinkingCounter--;
    if (blinkingCounter <= 0) {
      blinkingCounter = 25;
    }
    break;
  }

}
```

Program 3.11: fsm_traffic_light.c

1.8 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.

We add a small fsm **fsm_increase_mode** to help with increasing function. We can pass a variable to the fsm and it will change the value passed. If button is pressed, value will be increase by 1. If hold longer than 1 second, value will increase quickly. We only need to call this fsm_increase_mode function, and it will handle the increasing feature.

To prevent changing the duration of traffic_light, we only pass tempDuration to it fsm_increase_mode. Only if button 2 is pressed, tempDuration is assigned to the red_duration to update the value. When pressing button 2, 2 7-seg LED used to display mode will display "88" to indicate that the red_duration has been set.

```c
#include "fsm_traffic_light.h"

enum Mode{NORMAL, CHANGE_RED, CHANGE_AMBER, CHANGE_GREEN};
enum IncreaseMode{IDLE, INCREASE, INCREASE_FAST};


enum Mode mode = NORMAL;
enum IncreaseMode increaseMode = IDLE;
static int blinkingCounter = 25;
static int incCounter = 20;
static int tempDuration = 0;

void fsm_increase_mode(int *duration) {
  switch(increaseMode) {
  case IDLE:
    if (getButtonValue(1) == PRESS) {
      increaseMode = INCREASE;
      *duration = (*duration + 1) % 100;
    }
    break;
  case INCREASE:
    if (getButtonValue(1) == RELEASE) {
      increaseMode = IDLE;
    } else if (getFlagButtonPress1s(1) == 1) {
      increaseMode = INCREASE_FAST;
      incCounter = 20;
    }
    break;
  case INCREASE_FAST:
    if (getButtonValue(1) == RELEASE) {
      increaseMode = IDLE;
    }
    if (incCounter == 20) {
      *duration = (*duration + 1) % 100;
    }
    incCounter--;
    if (incCounter <= 0) incCounter = 20;
    break;
  }
}

void fsm_traffic_light() {
  switch(mode) {
  case NORMAL:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
    == 0) {
      mode = CHANGE_RED;
      increaseMode = IDLE;
      tempDuration = red_duration;
      setFlagButtonDelay(0);
      blinkingCounter = 25;
    }
    run_traffic_light();
    update_time_traffic_light();
    display_traffic_light();
    break;
  case CHANGE_RED:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
    == 0) {
      mode = CHANGE_AMBER;
      increaseMode = IDLE;
      tempDuration = amber_duration;
      setFlagButtonDelay(0);
      blinkingCounter = 25;
    }
    fsm_increase_mode(&tempDuration);
    updateBuffer(tempDuration, 02);
    if (getButtonValue(1) == RELEASE && getButtonValue(2)
   == PRESS) {
      red_duration = tempDuration;
      updateBuffer(tempDuration, 88);
    }
    //these are used to blink the led
    if (blinkingCounter == 25) {
      toggleRedLight();
    }
    blinkingCounter--;
    if (blinkingCounter <= 0) {
      blinkingCounter = 25;
    }
    break;
  case CHANGE_AMBER:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
   == 0) {
      mode = CHANGE_GREEN;
      increaseMode = IDLE;
      setFlagButtonDelay(0);
      tempDuration = green_duration;
      blinkingCounter = 25;
    }
    fsm_increase_mode(&tempDuration);
    updateBuffer(tempDuration, 03);
    //these are used to blink the led
    if (blinkingCounter == 25) {
      toggleAmberLight();
    }
    blinkingCounter--;
    if (blinkingCounter <= 0) {
      blinkingCounter = 25;
    }
    break;
  case CHANGE_GREEN:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
   == 0) {
      mode = NORMAL;
      setFlagButtonDelay(0);
      init_traffic_light();
    }
    fsm_increase_mode(&tempDuration);
    updateBuffer(tempDuration, 04);
    //these are used to blink the led
    if (blinkingCounter == 25) {
      toggleGreenLight();
    }
    blinkingCounter--;
    if (blinkingCounter <= 0) {
      blinkingCounter = 25;
    }
    break;
  }
}
```

Program 3.12: fsm_traffic_light.c +

1.9 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.

Inside fsm_traffic_light function, we only need to add these code inside CHANGE_AMBER state:

```c
case CHANGE_AMBER:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
    == 0) {
      mode = CHANGE_GREEN;
      increaseMode = IDLE;
      setFlagButtonDelay(0);
      tempDuration = green_duration;
      blinkingCounter = 25;
    }
    fsm_increase_mode(&tempDuration);
    updateBuffer(tempDuration, 03);
    if (getButtonValue(1) == RELEASE && getButtonValue(2)
   == PRESS) {
      amber_duration = tempDuration;
      updateBuffer(tempDuration, 88);
    }
    //these are used to blink the led
    if (blinkingCounter == 25) {
      toggleAmberLight();
    }
    blinkingCounter--;
    if (blinkingCounter <= 0) {
      blinkingCounter = 25;
    }
    break;
```

Program 3.13: fsm_traffic_light.c +

1.10 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.

Inside fsm_traffic_light function, we only need to add these code inside CHANGE_GREEN state:

```c
case CHANGE_GREEN:
    if (getButtonValue(0) == PRESS && getFlagButtonDelay(0)
    == 0) {
      mode = NORMAL;
      setFlagButtonDelay(0);
      init_traffic_light();
    }
    fsm_increase_mode(&tempDuration);
    updateBuffer(tempDuration, 04);
    if (getButtonValue(1) == RELEASE && getButtonValue(2)
  == PRESS) {
      green_duration = tempDuration;
      updateBuffer(tempDuration, 88);
    }
    //these are used to blink the led
    if (blinkingCounter == 25) {
      toggleGreenLight();
    }
    blinkingCounter--;
    if (blinkingCounter <= 0) {
      blinkingCounter = 25;
    }
    break;
```

Program 3.14: fsm_traffic_light.c +

1.11 Exercise 10: To finish the project

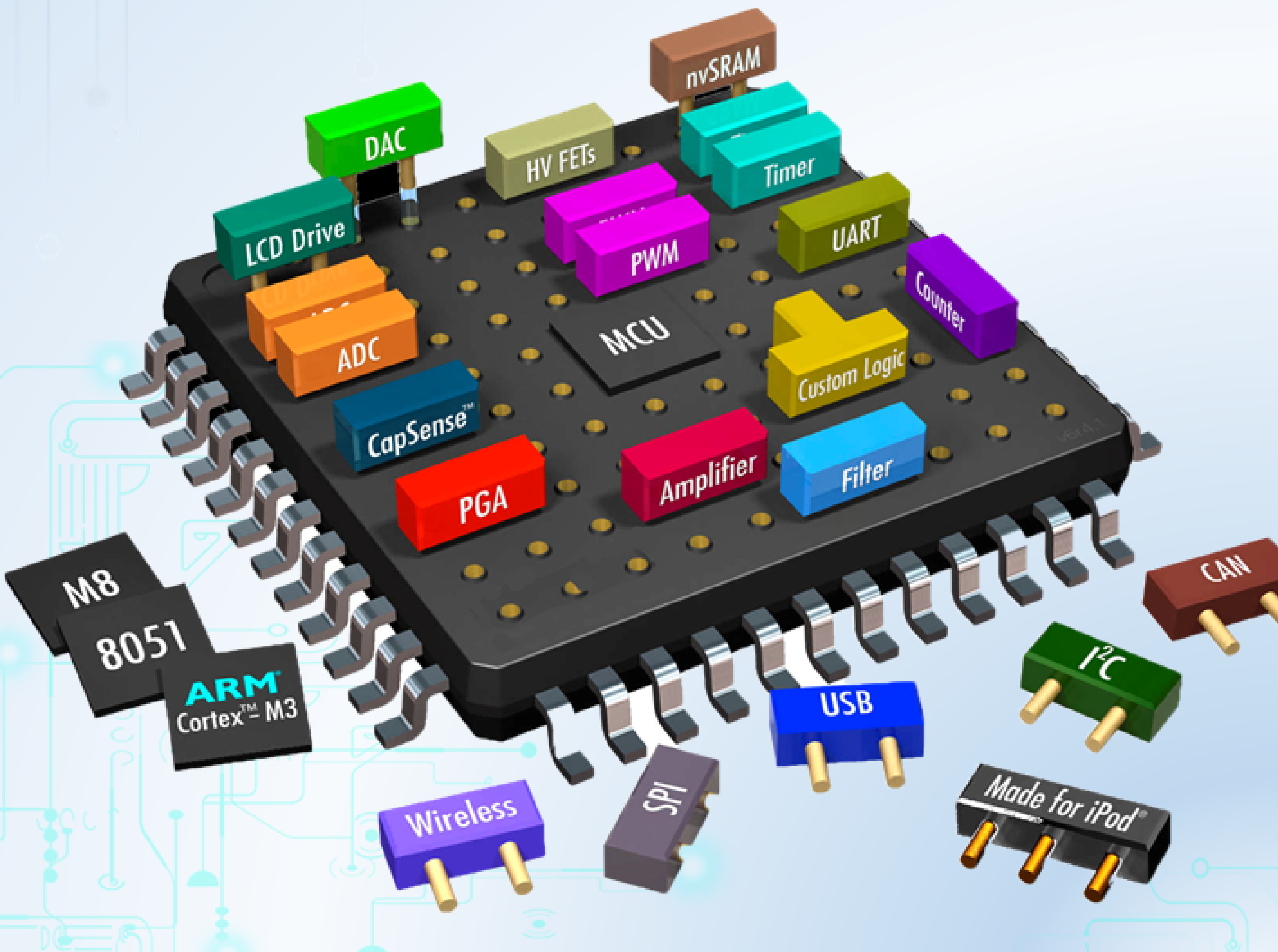
Your tasks in this exercise are:

- To integrate all the previous tasks to one final project
- To create a video to show all features in the specification
- To add a report to describe your solution for each exercise.
- To submit your report and code on the BKeL

Link to my video: traffic_light.mp4

CHƯƠNG 4

A Cooperative Scheduler



1 Objective

The aim of this lab is to design and implement a cooperate scheduler to accurately provide timeouts and trigger activities. You should add a file for the scheduler implementation and modify the main system call loop to handle timer interrupts.

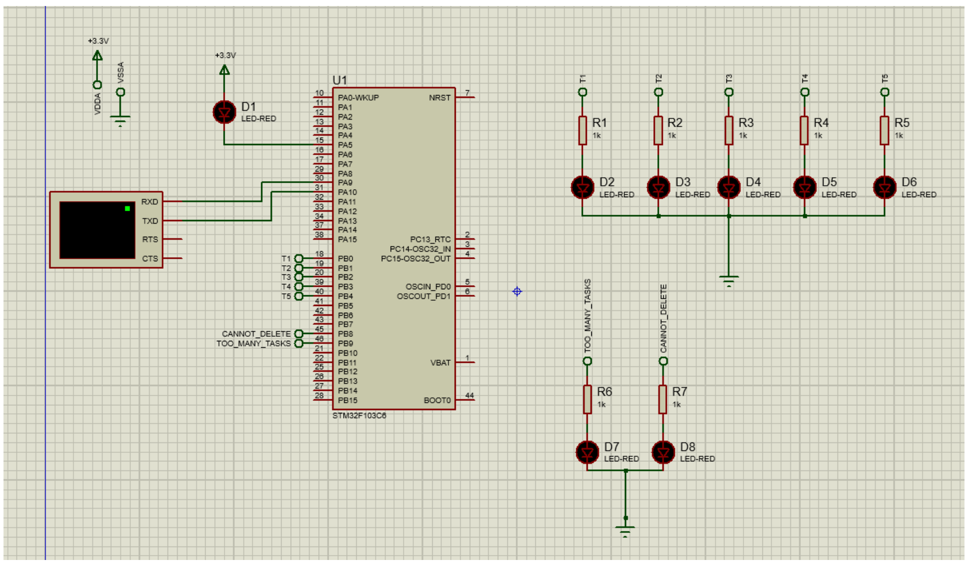
2 Problem

- Your system should have at least four functions:

- **void SCH_Update(void)**:This function will be updated the remaining time of each tasks that are added to a queue. It will be called in the interrupt timer, for example 10 ms.
- **void SCH_Dispatch_Tasks(void)**: This functionwill get the task in the queue to run.
- **uint32_tSCH_Add_Task(void (* pFunction) (), uint32_tDELAY, uint32_tPERIOD)**: This function is used to add a task to the queue. It should return an ID that is corresponding with the added task.
- **uint8_t SCH_Delete_Task(uint32_t taskID)**: This function is used to delete the task based on its ID.

You should add more functions if you think it will help you to solve this problem. Your main program must have 5 tasks running periodically in 0.5 second, 1 second, 1.5 seconds, 2 seconds, 2.5 seconds.

3 Implementation



Hình 4.1: Schematic for Scheduler test- ***Scheduler.pdsprj***

With the above problem, I've write a scheduler base on instruction of Lab4. My Scheduler is implement in Scheduler.h and Scheduler.c.

3.1 Scheduler.h

```c
/*
 * Scheduler.h
 *
 *  Created on: Oct 12, 2021
 *      Author: ngocc
 */

#ifndef INC_SCHEDULER_H_
#define INC_SCHEDULER_H_

#include "main.h"
#include <stdio.h>

//SCHEDULER SECTION
   ----------------------------------------------------------------

struct SCH_Task {
  //Pointer to the task (must be a 'void (void)' function)
  void (*pFunc)(void);
  //Delay (ticks) until the function will (next) be run
  int delay;
  //Interval (ticks) between subsequent runs
  int period;
  //Incremented (by scheduler) when task is due to execute
  int ready;
  //This is a hint to solve the question below
  uint32_t TaskID;
};

#define MAX_TASK  5
#define NO_TASK_ID  0
#define TICK_TIME   10

#define RETURN_ERROR  0
#define RETURN_NORMAL 1

void SCH_Init();
uint32_t SCH_Add_Task(void(*pFunction)(), uint32_t DELAY,
   uint32_t PERIOD);
uint8_t SCH_Delete_Task(uint32_t taskID);
void SCH_Update(void);
void SCH_Dispactch_Tasks(void);
void SCH_Check_Ready_Task();

//
   ----------------------------------------------------------------
```

```
//ERROR SECTION
   -------------------------------------------------------------------

#define ERROR_SCH_CANNOT_DELETE_TASK 0x01
#define ERROR_SCH_TOO_MANY_TASKS   0x02

extern unsigned char Error_code_G;

void SCH_Report_Status(void);

//
   ----------------------------------------------------------------------

#endif /* INC_SCHEDULER_H_ */
```

For easy testing, I only define MAX_TASK is 5, so we can easily fill up the task queue. Also, I define TICK_TIME as 10, TICK_TIME is used to increase the tick_time value of the scheduler which keep track of the time to help the scheduler scheduling the task. Since our SCH_Update function is placed inside 10ms timer interrupt, so everytime SCH_Update is called, we increase tick_time value by TICK_TIME (10ms). I also define some ERROR and to report the status of scheduler. We'll discuss more detail about all function in this header file in Scheduler.c file.

3.2 Scheduler.c

This file contain our implementation of the Scheduler. With this scheduler, we have some special variable:

```
static struct SCH_Task sch_Task[MAX_TASK];
static uint32_t tick_time = 0; //TICK_TIME
static uint32_t min_delay = 0xffffffff;
UART_HandleTypeDef huart1;    //NEEDED to use UART
IWDG_HandleTypeDef hiwdg;   //NEEDED for watchdog Timer

unsigned char Error_code_G;
unsigned char Last_error_code_G;
static uint32_t Error_tick_count_G;
```
Program 4.1: Variable used

- sch_Task is an array which keep the task information such as function poiner to the task, delay, period and ready state that indicate if the process is scheduled to run or not
- tick_time a variable that we use to keep track of the time, this variable is compared again min_delay, decide which tasks will be scheduled to run
- min_delay: a variable that let us know the minimum delay of all task in the array of task in sch_Task

My ideal to implement this is: We always have a min_delay, which is the smallest delay value of all task currently in sch_Task. tick_time is our timer, which will increase overtime. When tick_time is equal to or greater than min_delay, that mean one, or more task can be schedule to run, so we search through every task inside sch_Task and schedule them to run.

3.2.1 SCH_Init()

```
void SCH_Init(void) {
  for(int i = 0; i < MAX_TASK; i++) {
    sch_Task[i].delay = 0;
    sch_Task[i].pFunc = NULL;
    sch_Task[i].period = 0;
    sch_Task[i].ready = 0;
  }
  //init Error code
  Error_code_G = 0;
  HAL_IWDG_Init(&hiwdg);
}
```

Program 4.2: SCH_Init

First, we have an Init function to initialize our task array and start the watchdog timer.

3.2.2 SCH_Update()

Let first start with our update function, SCH_Update

```
void SCH_Update(void) {
  tick_time += TICK_TIME;
  HAL_IWDG_Refresh(&hiwdg);
}
```

Program 4.3: SCH_Update

This Update function is now a function of O(1) complexity.

SCH_Update is put inside hardware timer interrupt TIM2, which will fire an interrupt every 10ms. When this function is call, tick_time will increase by TICK_TIME (10). Also, since we're using watchdog timer for our scheduler, we need to refresh watchdog timer. If our system run normally, this watchdog timer will always be refresh after 10ms, prevent watchdog timer to reset our system. But, if in case our system is stuck, this function can't be call, watchdog timer can't be refresh, then after a timeout, our microprocessor will be reset and start running from the beginning.

As you can see, tick_time is increase over and over, this may cause overflow if we let the system run too long. So, instead of continuously increasing the tick_time, we'll reset the tick_time when tick_time >= min_delay ,when some task can be scheduled to run. Not only reset tick_time, we need to update all the task in Sch_task with new delay value by decreasing them with tick_time. All these works is implemented inside SCH_Check_Ready_Task() function

3.2.3 SCH_Check_Ready_Task()

```
void SCH_Check_Ready_Task() {
    if (tick_time >= min_delay) {
    uint32_t new_min = 0xffffffff;
    for(int i = 0; i < MAX_TASK; i++) {
      if (sch_Task[i].pFunc != 0) {
        sch_Task[i].delay -= tick_time;
        if (sch_Task[i].delay <= 0) {
          //schedule the task to run
          sch_Task[i].ready += 1;
          if (sch_Task[i].period != 0) {
            sch_Task[i].delay += sch_Task[i].period;
            if (sch_Task[i].delay <= 0) {
              //make the task wait for "period" and run
   again
              int temp = sch_Task[i].delay * (-1);
              int distance = temp / sch_Task[i].period;
              sch_Task[i].delay += sch_Task[i].period * (
   distance + 1);
              sch_Task[i].ready += distance;
            }
          } //if period == 0, we don't want it to execute
   again, so we do nothing here
        }
        if (sch_Task[i].delay < new_min && sch_Task[i].
   delay > 0) {
          new_min = sch_Task[i].delay;
        }
      }
    }
    min_delay = new_min;
    tick_time = 0;
  }
}
```

Program 4.4: SCH_Check_Ready_Task()

When running scheduler, we have 2 cases to notice:

- **tick_time == min_delay**
  - The scheduler run normally, there's no task that run too long which cause other task can't run on scheduled time. when tick_time == min_delay, this mean that there one or more task can be scheduled to run, so we will scan for the entire task list and decrease every task's delay by tick_time (in this case, tick_time == min_delay). If a task's delay after decrease is 0, we schedule that task to run by increasing its ready attribute by 1. If that task have period (it's a repeatable task) and its delay is 0, we have to add the task's period to task's delay. In this case, we're guaranteed that if a task have period and can be scheduled later, its delay is always > 0. While decreasing each task's delay with tick_time, we also finding a new

min_delay of the updated task list. At the end of the function, we reset tick_time to 0. tick_time will then continue to count up, reach the new min_delay value, and we'll search through task list again to decide which task to run.

- **tick_time > min_delay**
  - There's a task that run too long inside dispatch task that make the other task can't run on time, because while we're stuck inside dispatch task, update function is still called regularly every 10ms and increasing tick_time, which may cause tick_time become much larger than min_delay. When tick_time > min_delay, this mean that one or more tasks are supposed to be scheduled before, and there may be some task that would have scheduled multiple times while we're stuck inside dispatch task.
  - In the function, we still scan through the entire task list and decrease each task's delay by tick_time. If after decreasing, a task's delay is <= 0, we'll increase its ready by 1. If that task have period and its delay is < 0 after decreasing, that task is supposed to run once or more while we're stuck inside dispatch task. We'll try to further increase that task's delay by adding the task's period till the task's delay is greater than 0, and every time we add task's delay with task's period, we also increase ready variable of that task. The ready variable represent the how many times that task is supposed to run while we're stuck inside dispatch task.
  - Inside for loop, we also finding a new min_delay. At the end of the function, we set tick_time to 0, and let it count up again.

3.2.4 SCH_Add_Task()

We can add task inside our task list as follow.

```
uint32_t SCH_Add_Task(void(*pFunction)(), uint32_t DELAY,
   uint32_t PERIOD) {

  //find an empty slot to put the task in.
  uint32_t index = 0;
  while ((sch_Task[index].pFunc != NULL) && (index <
   MAX_TASK)) {
    index++;
  }
  //if the queue is full
  if (index >= MAX_TASK) {
    //Task list is full
    // Set the global error variable
    Error_code_G = ERROR_SCH_TOO_MANY_TASKS;
    //also return an error code
    return MAX_TASK;
  }
  //if a slot is found (queue still have some space)
  if (tick_time + DELAY < min_delay) {
    min_delay = tick_time + DELAY;
```

```c
  }
  sch_Task[index].delay = tick_time + DELAY;
  sch_Task[index].pFunc = pFunction;
  sch_Task[index].period = PERIOD;
  sch_Task[index].ready = 0;

  return (index);
}
```

Program 4.5: SCH_Add_Task()

When adding task, we also check if the new task delay + tick_time is < min_delay, so we'll update min_delay to an appropriate value.

3.2.5 SCH_Dispactch_Tasks()

```c
void SCH_Dispactch_Tasks(void) {
  //for loop: present FIFO, the first task will execute
   first.
  for(int i = 0; i < MAX_TASK; i++) {
    if (sch_Task[i].pFunc != NULL && sch_Task[i].ready > 0)
    {
      //run the task
      //dereference it first, and then run.
      (*sch_Task[i].pFunc)();
      sch_Task[i].ready -= 1;
      //if period = 0, we don't want it to execute again
      if (sch_Task[i].period == 0) {
        SCH_Delete_Task(i);
      }
    }
  }
  // Report system status
  SCH_Report_Status();
}
```

Program 4.6: SCH_Dispactch_Tasks()

Dispatch Task is the function that "run" the task. If a task's ready is > 0, it run the function pointer of that task and decrease ready by one. It can also report system status and error through LED with SCH_Report_Status().

3.2.6 SCH_Delete_Task()

```c
uint8_t SCH_Delete_Task(uint32_t taskID) {
  unsigned char Return_code;
  if (taskID < 0 || taskID >= MAX_TASK || sch_Task[taskID].
   pFunc == NULL) {
    // No task at this location...
    //
    // Set the global error variable
    Error_code_G = ERROR_SCH_CANNOT_DELETE_TASK;

    // ...also return an error code
    Return_code = RETURN_ERROR;
  } else {
    Return_code = RETURN_NORMAL;
  }

  sch_Task[taskID].delay = 0;
  sch_Task[taskID].pFunc = NULL;
  sch_Task[taskID].period = 0;
  sch_Task[taskID].ready = 0;

  return Return_code;
}
```

Program 4.7: SCH_Delete_Task()

We have a delete function that delete the task in task list. If there no valid task to delete, we return an Error and assign some error code to display outside.

3.2.7 SCH_Report_Status()

```
static uint32_t Error_tick_count_G;

void SCH_Report_Status(void) {
#ifdef SCH_REPORT_ERRORS
  //ONLY APPLIES IF WE ARE REPORTING ERRORS
  // Check for a new error code
  if (Error_code_G != Last_error_code_G) {
    //B(SET)RR: 32 BIT, 16bit low can use to set a pin to
   high if a bit is 1.
    Error_port->BSRR = ((uint16_t)Error_code_G << 8);
    Last_error_code_G = Error_code_G;
    if (Error_code_G != 0) {
      Error_tick_count_G = 60000;
    } else {
      Error_tick_count_G = 0;
      Error_port->BRR = ((uint16_t)0xff << 8);    //have to
     manually RESET error pin.
    }
  } else {
    if (Error_tick_count_G != 0) {
      if (--Error_tick_count_G == 0) {
        Error_code_G = 0; //Reset error code.
      }
    }
  }
#endif
}
```

Program 4.8: SCH_Report_Status()

This function help us report error through LED with Error_code_G. The error_code will disappear after some time.

3.3 main.c

Firstly, we create 5 task to toggle 5 different LEDs

```
void task1(void) {
  HAL_GPIO_TogglePin(GPIOB, T1_Pin);
}
void task2(void) {
  HAL_GPIO_TogglePin(GPIOB, T2_Pin);
}
void task3(void) {
  HAL_GPIO_TogglePin(GPIOB, T3_Pin);
}
void task4(void) {
  HAL_GPIO_TogglePin(GPIOB, T4_Pin);
}
void task5(void) {
  HAL_GPIO_TogglePin(GPIOB, T5_Pin);
}
```

Program 4.9: tasks

Inside our main.c file, we have the main function as follow:

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_USART1_UART_Init();
  MX_TIM3_Init();
  MX_IWDG_Init();
  HAL_TIM_Base_Start_IT(&htim2);
  HAL_TIM_Base_Start_IT(&htim3);
  static char timeformat[30];
  uint32_t time_ms = HAL_GetTick();
  int strlength = sprintf(timeformat, "init: %ld\r\n",
   time_ms);
  HAL_UART_Transmit_IT(&huart1, (uint8_t*)timeformat,
   strlength);


  HAL_UART_Transmit(&huart1, tData, sizeof(tData), 100);
  //NO RECEIVE
```

```c
  //HAL_UART_Receive_IT(&huart1, &rData, 1);
  SCH_Init();
  SCH_Add_Task(task1, 600, 0);
  SCH_Add_Task(task2, 1000, 0);
  SCH_Add_Task(task3, 1500, 0);
  SCH_Add_Task(task4, 2000, 0);
  SCH_Add_Task(task5, 2500, 0);
  while (1)
  {
    //HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
    //HAL_Delay(1000);
    SCH_Check_Ready_Task();
    SCH_Dispactch_Tasks();
  }

}
```

Program 4.10: main function

We first add 5 tasks with the following delay: 600ms, 1000ms, 1500ms, 2000ms and 2500ms, these are all "one-shot" tasks. And in the callback function, we have the following codes:

```c

int counter = 100;

int more_task_counter = 0;
int divider = 60; //60 la boi chung nho nhat cua 2, 3, 4 va 5.
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  if (htim->Instance == TIM3) {

    //tai thoi diem ham nay duoc goi, 500ms da troi qua.
    more_task_counter = (more_task_counter + 1) % divider;     //60 (divider) la boi chung nho nhat cua 2, 3, 4 va 5.

    if (SCH_Add_Task(task1, 10, 0) == MAX_TASK) {
      static uint8_t max_queue[] = {"Queue is full\r\n"};
      HAL_UART_Transmit_IT(&huart1, max_queue, sizeof(max_queue));
    }

    if (more_task_counter % 2 == 0) {   // <-- 2
      if (SCH_Add_Task(task2, 10, 0) == MAX_TASK) {
        static uint8_t max_queue[] = {"Queue is full\r\n"};
        HAL_UART_Transmit_IT(&huart1, max_queue, sizeof(max_queue));
      }
    }
    if (more_task_counter % 3 == 0) {   // <-- 3
      if (SCH_Add_Task(task3, 10, 0) == MAX_TASK) {
        static uint8_t max_queue[] = {"Queue is full\r\n"};
        HAL_UART_Transmit_IT(&huart1, max_queue, sizeof(max_queue));
      }
    }
    if (more_task_counter % 4 == 0) {   // <-- 4
      if (SCH_Add_Task(task4, 10, 0) == MAX_TASK) {
        static uint8_t max_queue[] = {"Queue is full\r\n"};
        HAL_UART_Transmit_IT(&huart1, max_queue, sizeof(max_queue));
      }
    }
    if (more_task_counter % 5 == 0) {   // <-- 5
      if (SCH_Add_Task(task5, 10, 0) == MAX_TASK) {
        static uint8_t max_queue[] = {"Queue is full\r\n"};
        HAL_UART_Transmit_IT(&huart1, max_queue, sizeof(max_queue));
      }
    }


  }

  if (htim->Instance == TIM2) {
  SCH_Update();

  counter--;
  if (counter == 0) {
    counter = 100;
    HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
  }
  //RTC_TimeTypeDef sTime;

  }
}
```
Program 4.11: HAL_TIM_PeriodElapsedCallback

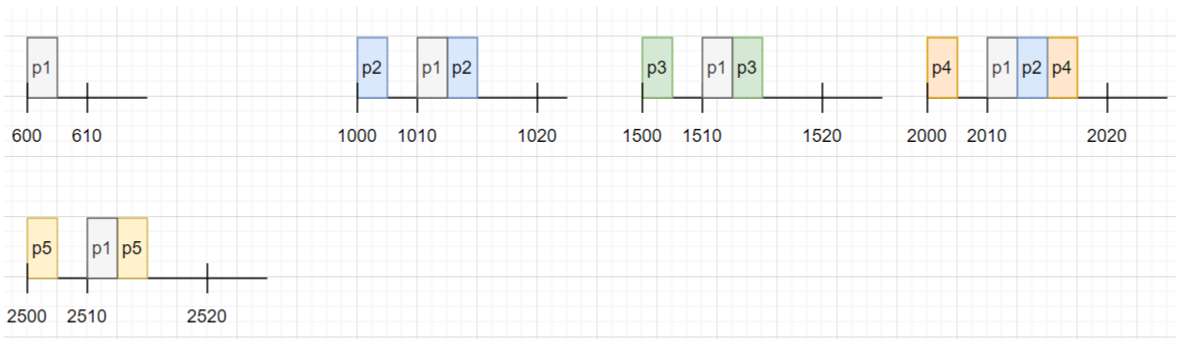
Timer2 (TIM2) is used for calling SCH_Update() and flashing a LED every 1 second to ensure that the system run normally. Timer3 (TIM3) is used to add some task every 500ms (timer3 is set to call an interrupt every 500ms, prescaller is 7999 and period is 499). We also add some code to printout to the terminal if the task queue is full and we can't add more task.

Using TIM3, we'll have task1 will be added every 500ms, task2 is added every 1 second, task3 is 1.5 second, task 4 is 2 second and task 5 is 2.5 second. When adding these task, we also give it a small delay (10ms) and a period of 0 (one-shot task)

3.3.1 Grant chart

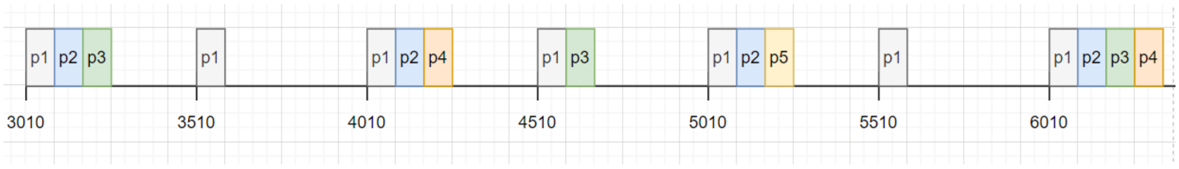
Since our task is just simply toggle a LED, its run time is small compare to the 10ms interrupt period, so we can assume that if all 5 tasks run at the same time, their execution time is still less than 10ms.

In the first 2.5 second, we can draw the task run time as following.



Hình 4.2: Grant chart in the first 2.5 second

After 2.5s, TIMER3 will add task normally: task1 will be added every 0.5 second, task2 is 1 second, task3 is 1.5 second, task4 is 2 second and task5 is added every 2.5 second. In TIMER3, tasks are added with a period of 0 (one-shot task) and a delay of 10, so our tasks will run 10ms after we add them inside the task array.



Hình 4.3: Grant chart after 2.5 second