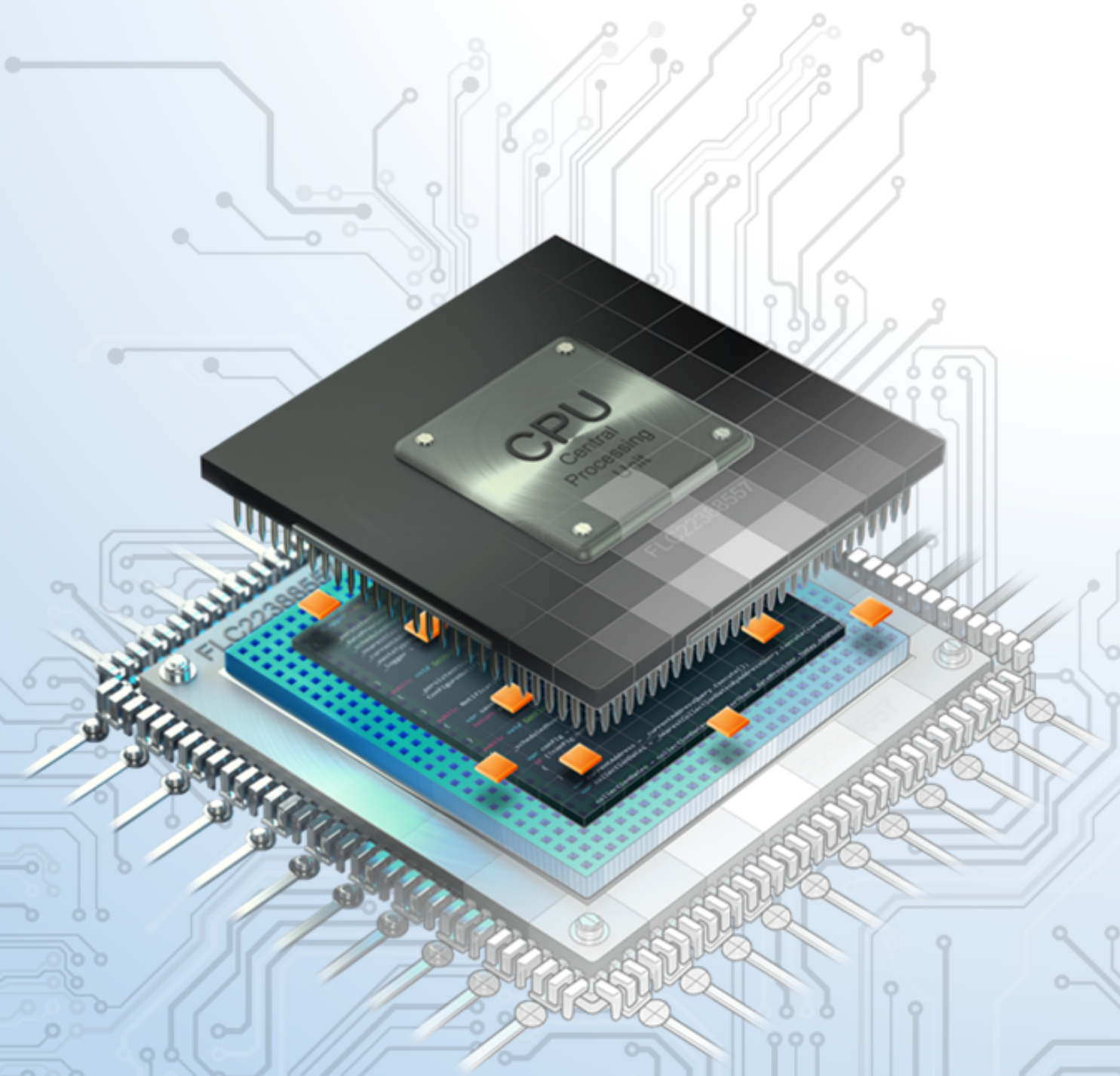
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller



Dr. Le Trong Nhan

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

Microprocessor - Microcontroller (CO3009)

Laboratory Report

LAB 2

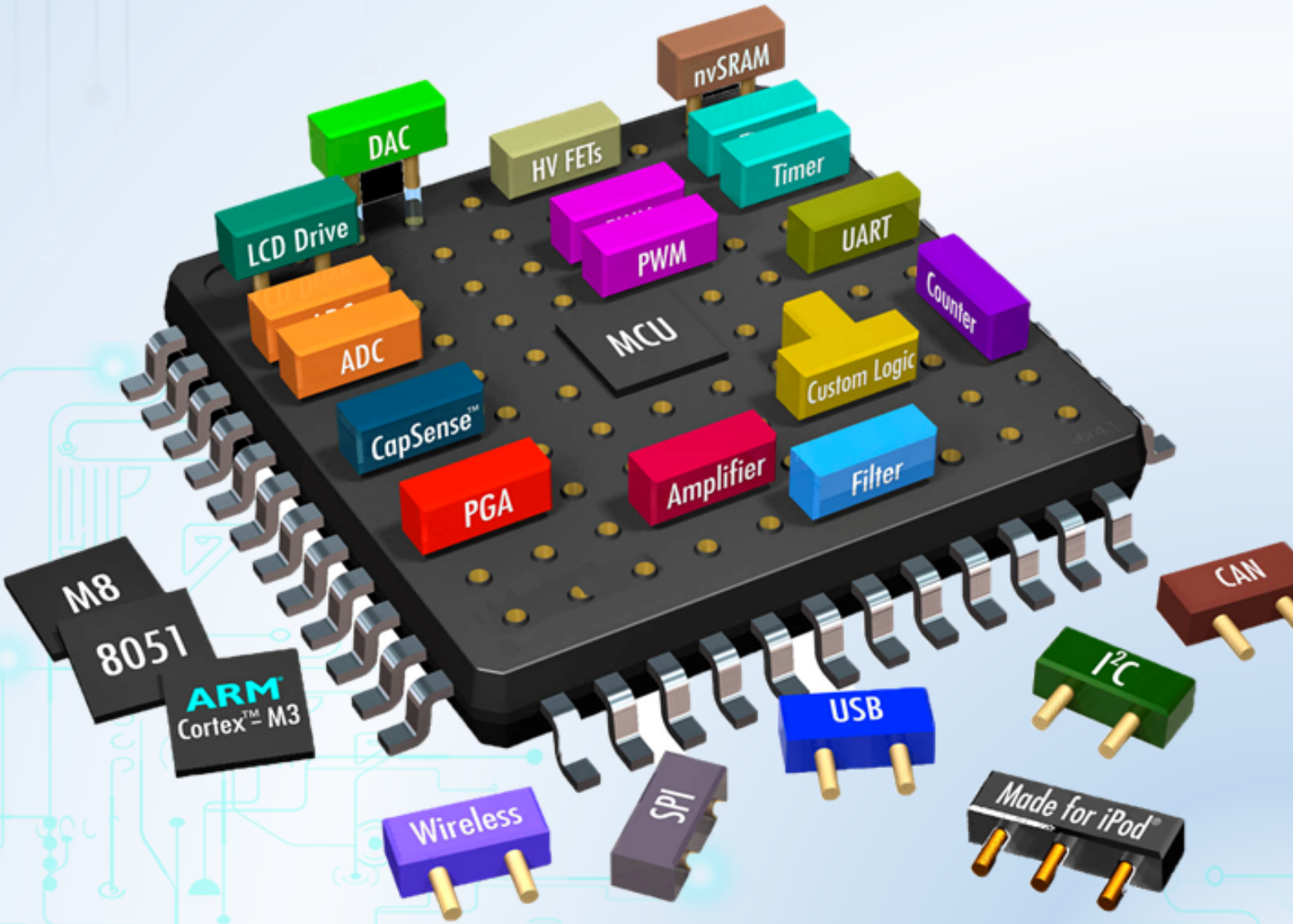
Teacher: Le Trong Nhan
Huynh Phuc Nghi
Class: L03
Student: Pham Van Nhat Vu (2110676)

HO CHI MINH CITY, 2023

Contents

CHAPTER 1

Timer Interrupt and LED Scanning



1 Exercise and Report

1.1 Exercise 1

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

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

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

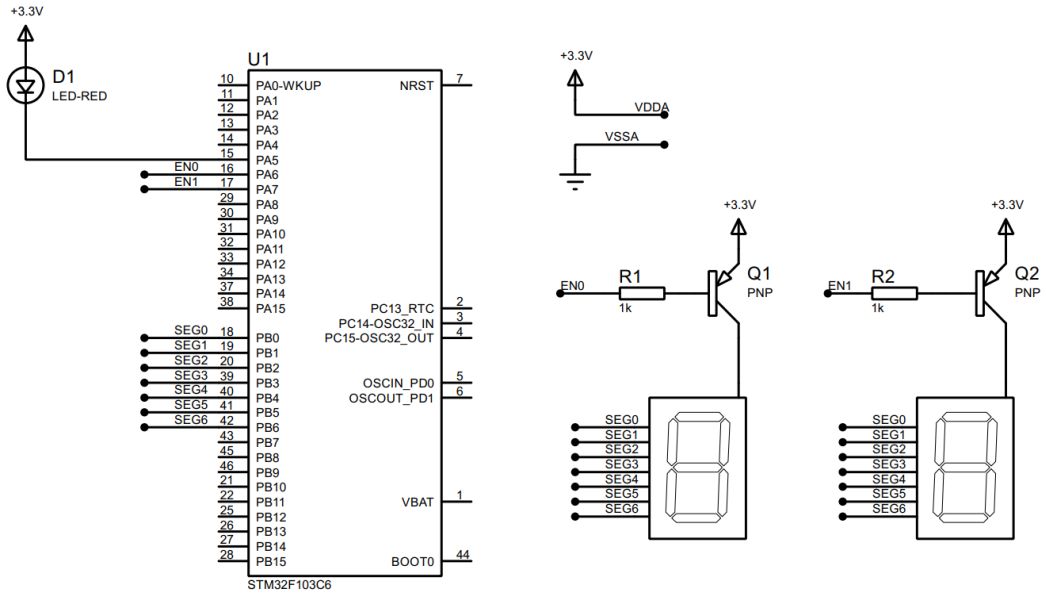


Figure 1.1: *Simulation schematic in Proteus*

Components used in the schematic are listed bellow:

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

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

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

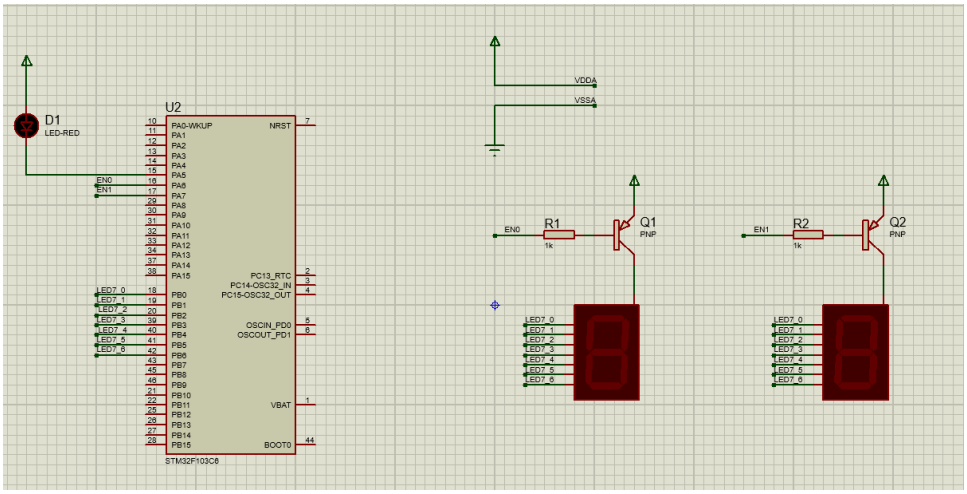


Figure 1.2: Schematic of Proteus simulation project

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

Answer: From this exercise, I move all processing (or complex computation) to the while loop, the ISR is only used to handle software timer.

```
#define LED7_DURATION 50
...
int led_test_counter = 100;
int led_test_flag = 0;
int scanning_counter = LED7_DURATION;
int scanning_flag = 0;
int led_enabled = 0;
...
int main(void)
{
  ...
  HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, GPIO_PIN_RESET)
   ;
  HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, GPIO_PIN_SET);
  while (1)
  {
    if (led_test_flag == 1)
    {
        HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
        led_test_flag = 0;
        led_test_counter = 100;
    }
```

```
22
23      switch (led_enabled)
24      {
25      case 0:
26          HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin,
    GPIO_PIN_RESET);
27          display7SEG(1);
28          if (scanning_flag == 1)
29          {
30              display7SEG(2);
31              scanning_flag = 0;
32              scanning_counter = LED7_DURATION;
33              led_enabled = 1;
34              HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin,
    GPIO_PIN_SET);
35              HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin,
    GPIO_PIN_RESET);
36          }
37          break;
38      case 1:
39          HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin,
    GPIO_PIN_RESET);
40          display7SEG(2);
41          if (scanning_flag == 1)
42          {
43              display7SEG(1);
44              scanning_flag = 0;
45              scanning_counter = LED7_DURATION;
46              led_enabled = 0;
47              HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin,
    GPIO_PIN_SET);
48              HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin,
    GPIO_PIN_RESET);
49          }
50          break;
51      }
52    }
53    ...
54  }
55  ...
56  void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef *
    htim )
57  {
58      --led_test_counter;
59      if (led_test_counter <= 0)
60      {
61          led_test_flag = 1;
62      }
63
```

```
64      --scanning_counter;
65      if (scanning_counter <= 0)
66      {
67          scanning_flag = 1;
68      }
69  }
```

Program 1.1: Source code for exercise 1

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

The frequency of the scanning process is $\frac{1}{2*0.5} = 1\text{Hz}$

1.2 Exercise 2

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

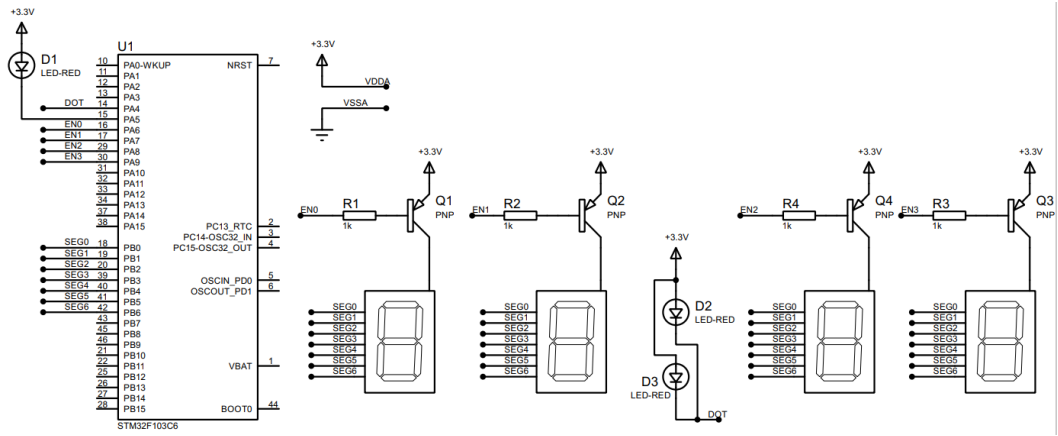


Figure 1.3: Simulation schematic in Proteus

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

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

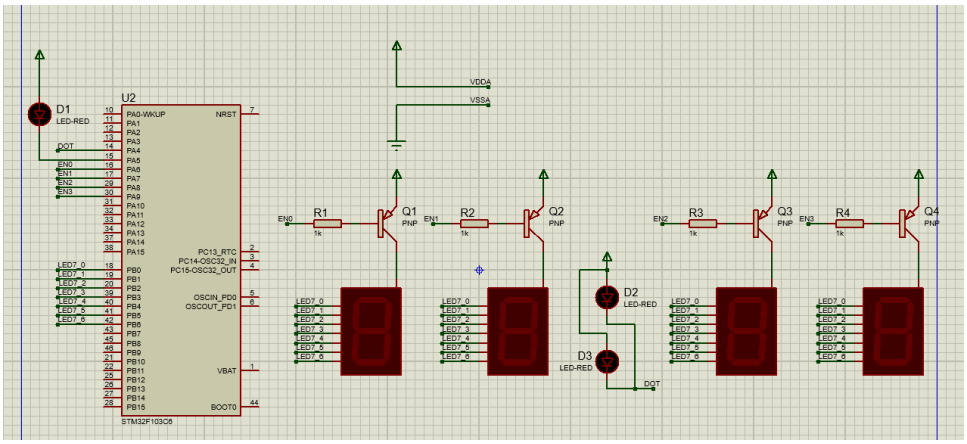


Figure 1.4: Schematic of Proteus simulation project

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

```
#define LED7_DURATION 50
#define DOT_DURATION 100
...
int led_test_counter = 100;
int led_test_flag = 0;
```

```
int dot_counter = DOT_DURATION;
int dot_flag = 0;
int scanning_counter = LED7_DURATION;
int scanning_flag = 0;
int led_enabled = 0;
...
int main(void)
{
  ...
  HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, GPIO_PIN_RESET)
   ;
  HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin, GPIO_PIN_SET);
  while (1)
  {
    if (led_test_flag == 1)
    {
        HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
        led_test_flag = 0;
        led_test_counter = 100;
    }

    if (dot_flag == 1)
    {
        HAL_GPIO_TogglePin(DOT_GPIO_Port, DOT_Pin);
        dot_flag = 0;
        dot_counter = DOT_DURATION;
    }

    switch (led_enabled)
    {
    case 0:
        HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin,
  GPIO_PIN_RESET);
        display7SEG(1);
        if (scanning_flag == 1)
        {
            scanning_flag = 0;
            scanning_counter = LED7_DURATION;
            led_enabled = 1;
            HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin,
  GPIO_PIN_SET);
            display7SEG(2);
            HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin,
  GPIO_PIN_RESET);
        }
        break;
    case 1:
```

```
51        HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin,
  GPIO_PIN_RESET);
52        display7SEG(2);
53        if (scanning_flag == 1)
54        {
55            scanning_flag = 0;
56            scanning_counter = LED7_DURATION;
57            led_enabled = 2;
58            HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin,
  GPIO_PIN_SET);
59            display7SEG(3);
60            HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin,
  GPIO_PIN_RESET);
61        }
62        break;
63     case 2:
64        HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin,
  GPIO_PIN_RESET);
65        display7SEG(3);
66        if (scanning_flag == 1)
67        {
68            scanning_flag = 0;
69            scanning_counter = LED7_DURATION;
70            led_enabled = 3;
71            HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin,
  GPIO_PIN_SET);
72            display7SEG(0);
73            HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin,
  GPIO_PIN_RESET);
74        }
75        break;
76     case 3:
77        HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin,
  GPIO_PIN_RESET);
78        display7SEG(0);
79        if (scanning_flag == 1)
80        {
81            scanning_flag = 0;
82            scanning_counter = LED7_DURATION;
83            led_enabled = 0;
84            HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin,
  GPIO_PIN_SET);
85            display7SEG(1);
86            HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin,
  GPIO_PIN_RESET);
87        }
88        break;
89     }
90  }
```

```
91    ...
92  }
93  ...
94  void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef *
        htim )
95  {
96      --led_test_counter;
97      if (led_test_counter <= 0)
98      {
99          led_test_flag = 1;
100     }
101
102     --dot_counter;
103     if (dot_counter <= 0)
104     {
105         dot_flag = 1;
106     }
107
108     --scanning_counter;
109     if (scanning_counter <= 0)
110     {
111         scanning_flag = 1;
112     }
113 }
```

Program 1.2: Source code for exercise 2.

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

The frequency of the scanning process is $\frac{1}{4*0.5} = 0.5\text{Hz}$

1.3 Exercise 3

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

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

Program 1.3: An example for your source code

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

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

```
void update7SEG(int index)
{
    switch (index)
    {
    case 0:
        display7SEG(led_buffer[index]);
        HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin,
  GPIO_PIN_RESET);
        HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin,
  GPIO_PIN_SET);
        HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin,
  GPIO_PIN_SET);
        HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin,
  GPIO_PIN_SET);
        break;
    case 1:
```

```
        display7SEG(led_buffer[index]);
        HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin, GPIO_PIN_SET);
        break;
    case 2:
        display7SEG(led_buffer[index]);
        HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin, GPIO_PIN_SET);
        break;
    case 3:
        display7SEG(led_buffer[index]);
        HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(EN2_GPIO_Port, EN2_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(EN3_GPIO_Port, EN3_Pin, GPIO_PIN_RESET);
        break;
    default:
        break;
    }
}
```

Program 1.4: Source code of the update7SEG function.

Report 2: Present the source code in the HAL_TIM_PeriodElapsedCallback.

```
#define LED7_DURATION 50
#define DOT_DURATION 100
...
int led_test_counter = 100;
int led_test_flag = 0;
int dot_counter = DOT_DURATION;
int dot_flag = 0;
int scanning_counter = LED7_DURATION;
int scanning_flag = 0;
```

```
int led_enabled = 0;
...
const int MAX_LED = 4;
int index_led = 0;
int led_buffer[4] = {6, 2, 8, 9};
...
int main(void)
{
  ...
  update7SEG(index_led);
  while (1)
  {
    ...
    if (led_test_flag == 1)
    {
        HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
        led_test_flag = 0;
        led_test_counter = 100;
    }

    if (dot_flag == 1)
    {
        HAL_GPIO_TogglePin(DOT_GPIO_Port, DOT_Pin);
        dot_flag = 0;
        dot_counter = DOT_DURATION;
    }

    if (scanning_flag == 1)
    {
        ++index_led;
        if (index_led >= MAX_LED)
        {
            index_led = 0;
        }
        update7SEG(index_led);
        scanning_flag = 0;
        scanning_counter = LED7_DURATION;
    }
  }
  ...
}
...
void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef *
   htim )
{
    --led_test_counter;
    if (led_test_counter <= 0)
    {
        led_test_flag = 1;
```

```
58      }
59
60      --dot_counter;
61      if (dot_counter <= 0)
62      {
63          dot_flag = 1;
64      }
65
66      --scanning_counter;
67      if (scanning_counter <= 0)
68      {
69          scanning_flag = 1;
70      }
71  }
```

Program 1.5: Source code of the `HAL_TIM_PeriodElapsedCallback` function.

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

1.4 Exercise 4

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

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

```
#define LED7_DURATION 25
#define DOT_DURATION 100
...
int led_test_counter = 100;
int led_test_flag = 0;
int dot_counter = DOT_DURATION;
int dot_flag = 0;
int scanning_counter = LED7_DURATION;
int scanning_flag = 0;
int led_enabled = 0;
...
const int MAX_LED = 4;
int index_led = 0;
int led_buffer[4] = {6, 2, 8, 9};
...
int main(void)
{
  ...
  update7SEG(index_led);
  while (1)
  {
    if (led_test_flag == 1)
    {
        HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
        led_test_flag = 0;
        led_test_counter = 100;
    }

    if (dot_flag == 1)
    {
        HAL_GPIO_TogglePin(DOT_GPIO_Port, DOT_Pin);
        dot_flag = 0;
        dot_counter = DOT_DURATION;
    }

    if (scanning_flag == 1)
    {
        ++index_led;
        if (index_led >= MAX_LED)
        {
            index_led = 0;
        }
```

```
43          update7SEG(index_led);
44          scanning_flag = 0;
45          scanning_counter = LED7_DURATION;
46      }
47    }
48    ...
49  }
50  ...
51  void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef *
        htim )
52  {
53      --led_test_counter;
54      if (led_test_counter <= 0)
55      {
56          led_test_flag = 1;
57      }
58
59      --dot_counter;
60      if (dot_counter <= 0)
61      {
62          dot_flag = 1;
63      }
64
65      --scanning_counter;
66      if (scanning_counter <= 0)
67      {
68          scanning_flag = 1;
69      }
70  }
```

Program 1.6: Source code of the `HAL_TIM_PeriodElapsedCallback` function.

The switching time is reduced to 250ms so the frequency of 4 seven segment LEDs is $\frac{1}{4*250} = 1\text{Hz}$

1.5 Exercise 5

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

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

while(1){
    second++;
    if (second >= 60){
        second = 0;
        minute++;
    }
    if(minute >= 60){
        minute = 0;
        hour++;
    }
    if(hour >=24){
        hour = 0;
    }
    updateClockBuffer();
    HAL_Delay(1000);
}
```

Program 1.7: An example for your source code

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

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

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

Program 1.8: Source code in the **updateClockBuffer** function.

1.6 Exercise 6

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

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

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

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

Program 1.9: Software timer based timer interrupt

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

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

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
   {

  timer_run();

  //YOUR OTHER CODE
}
```

Program 1.10: Software timer based timer interrupt

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

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

Program 1.11: Software timer is used in main fuction to blink the LED

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

Answer: If line 1 of the code above is miss, variable `timer0_counter` is always equal to 0, the condition of if-statement in function `timer_run()` is always false, therefore variable `timer0_counter` and `timer0_flag` is never updated. Consequently, `HAL_GPIO_TogglePin` function for `LED_RED` in the while loop is not invoked and the LED connected to PA5 is always on, suppose that led's cathode pin is connected to PA5 and its anode pin is connected to power.

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

Answer: If line 1 of the code above is changed to setTimer0(1), variable `timer0_counter` is set to $\lfloor \frac{1}{\text{TIMER\_CYCLE}} \rfloor = \lfloor \frac{1}{10} \rfloor = 0$, therefore variable `timer0_counter` is always equal to 0 and the LED connected to PA5 is always on as is the case of Report 1.

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

Answer: If line 1 of the code above is changed to setTimer0(10), variable `timer0_counter` is set to $\lfloor \frac{10}{\text{TIMER\_CYCLE}} \rfloor = \lfloor \frac{10}{10} \rfloor = 1$. As a result, after about 10ms, the LED connected to PA5 is toggled and become OFF, and `timer0_counter` is set to 2000. After that, the LED connected to PA5 is toggled every 2000ms.

1.7 Exercise 7 and Exercise 8

Upgrade the source code in Exercise 5 (update values for hour, minute and second) by using the software timer and remove the HAL_Delay function at the end. Moreover, the DOT (connected to PA4) of the digital clock is also moved to main function. Move also the update7SEG() function from the interrupt timer to the main. Finally, the timer interrupt only used to handle software timers. All processing (or complex computations) is move to an infinite loop on the main function, optimizing the complexity of the interrupt handler function.

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

```
#define LED7_DURATION 25
#define DOT_DURATION 100
...
int led_test_counter = 100;
int led_test_flag = 0;
int dot_counter = DOT_DURATION;
int dot_flag = 0;
int scanning_counter = LED7_DURATION;
int scanning_flag = 0;
int led_enabled = 0;
int hour = 15, minute = 8, second = 50;
...
void display7SEG(int);
const int MAX_LED = 4;
int index_led = 0;
int led_buffer[4] = {6, 2, 8, 9};
void update7SEG(int);
void updateClockBuffer();
...
int timer0_counter = 0;
int timer0_flag = 0;
int TIMER_CYCLE = 10;
...
void setTimer0(int duration)
{
    timer0_counter = duration / TIMER_CYCLE;
    timer0_flag = 0;
}
...
void timer_run()
{
    if (timer0_counter > 0)
    {
        timer0_counter--;
        if (timer0_counter == 0) timer0_flag = 1;
    }
}
```

```
...
int main(void)
{
  ...
  setTimer0(1000);
  updateClockBuffer();
  update7SEG(index_led);
  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();
        setTimer0(1000);
    }

    if (led_test_flag == 1)
    {
        led_test_flag = 0;
        led_test_counter = 100;
        HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
    }

    if (dot_flag == 1)
    {
        dot_flag = 0;
        dot_counter = DOT_DURATION;
        HAL_GPIO_TogglePin(DOT_GPIO_Port, DOT_Pin);
    }

    if (scanning_flag == 1)
    {
        ++index_led;
        if (index_led >= MAX_LED)
        {
```

```
87              index_led = 0;
88          }
89          update7SEG(index_led);
90          scanning_flag = 0;
91          scanning_counter = LED7_DURATION;
92      }
93    }
94    ...
95 }
```

Program 1.12: Source code for exercise 7 and exercise 8.

1.8 Exercise 9

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

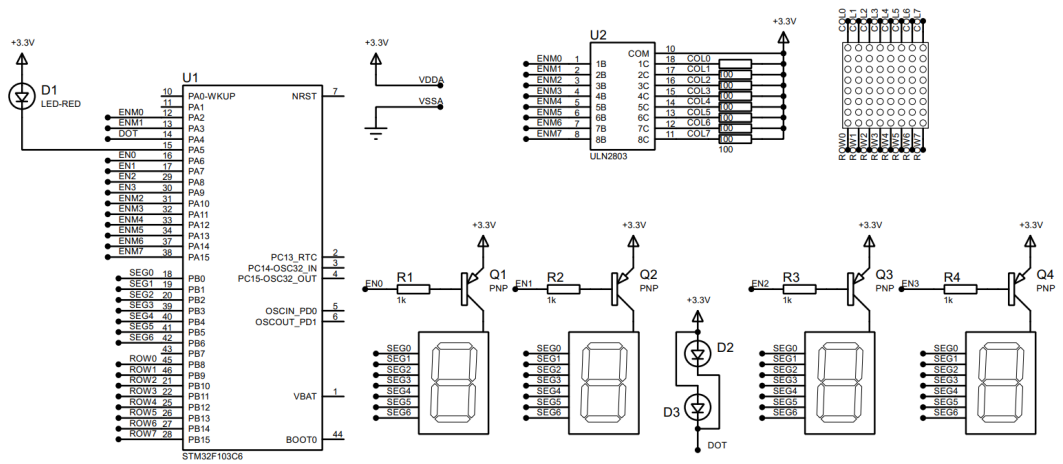


Figure 1.5: LED matrix is added to the simulation

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

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

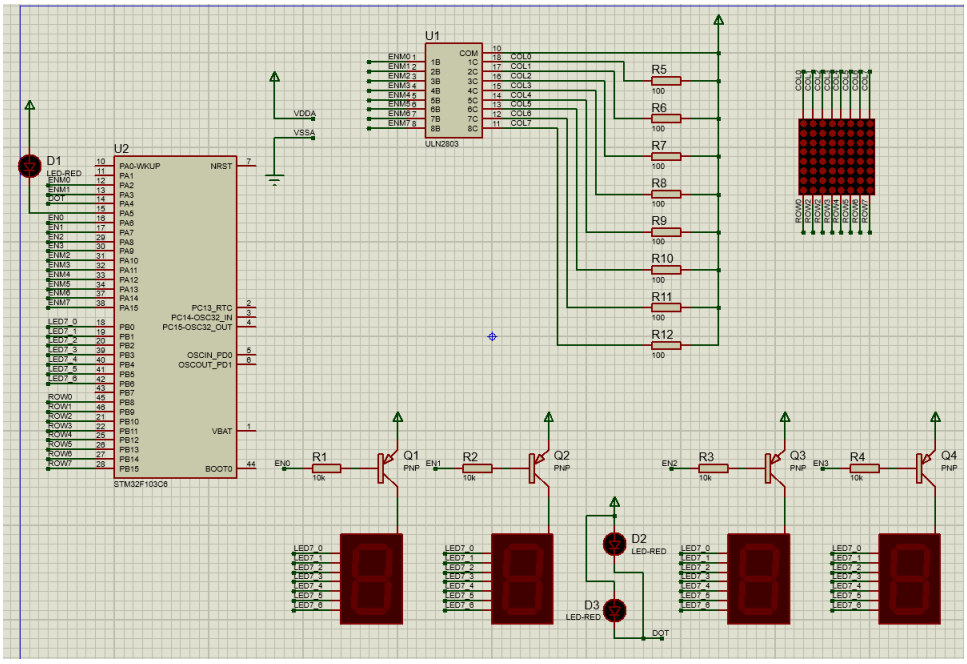


Figure 1.6: Schematic of Proteus simulation project

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

seven led segments.

Answer: The matrix LED will activate the LED columns with their respective COL pins set to the HIGH-level logic. The arrangement of these columns is determined by the status of the ROW pins: LEDs corresponding to rows with their respective ROW pins configured to a HIGH-level logic will be turned off, and conversely, they will be illuminated when the ROW pins are at a LOW-level logic. As an illustration, if the ROW0 through ROW7 pins are configured as 10101111, and only COL3, COL5, and COL6 are set to the HIGH-level logic, the matrix LED states can be described as follows (the columns is indexed from left to right while the rows are indexed from top to bottom):

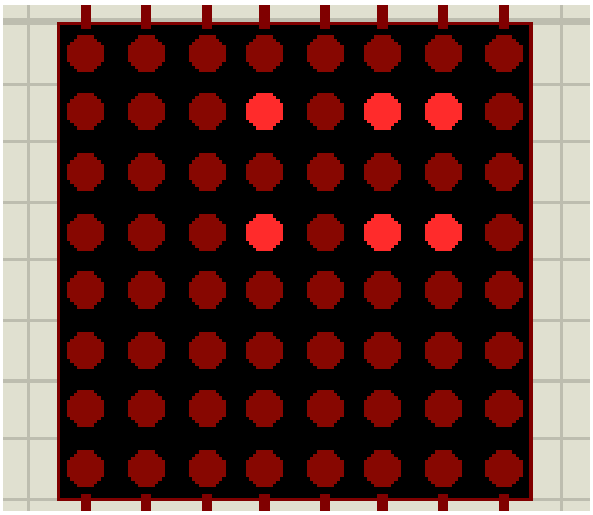


Figure 1.7: *An example of how the matrix LED works*

Furthermore, a COL pin is configured with the HIGH-level logic state when its corresponding ENM pin in module ULN2803 is set to the LOW-level logic. I use an array of 1-byte numbers to store the desired matrix LED pattern, structured as follows:

```
const int MAX_LED_MATRIX = 8;
int index_led_matrix = 0;
uint8_t matrix_buffer[8] = {0x00, 0xfc, 0x12, 0x11, 0x1, 0
   x12, 0xfc, 0x00};
void updateLEDMatrix(int index)
{
    HAL_GPIO_WritePin(ROW0_GPIO_Port, ROW0_Pin, ((
   matrix_buffer[index_led_matrix] >> 0) & 1) ?
   GPIO_PIN_RESET : GPIO_PIN_SET);
    HAL_GPIO_WritePin(ROW1_GPIO_Port, ROW1_Pin, ((
   matrix_buffer[index_led_matrix] >> 1) & 1) ?
   GPIO_PIN_RESET : GPIO_PIN_SET);
    HAL_GPIO_WritePin(ROW2_GPIO_Port, ROW2_Pin, ((
   matrix_buffer[index_led_matrix] >> 2) & 1) ?
   GPIO_PIN_RESET : GPIO_PIN_SET);
    HAL_GPIO_WritePin(ROW3_GPIO_Port, ROW3_Pin, ((
   matrix_buffer[index_led_matrix] >> 3) & 1) ?
   GPIO_PIN_RESET : GPIO_PIN_SET);
    HAL_GPIO_WritePin(ROW4_GPIO_Port, ROW4_Pin, ((
```

```
    matrix_buffer[index_led_matrix] >> 4) & 1) ?
    GPIO_PIN_RESET : GPIO_PIN_SET);
11   HAL_GPIO_WritePin(ROW5_GPIO_Port, ROW5_Pin, ((
    matrix_buffer[index_led_matrix] >> 5) & 1) ?
    GPIO_PIN_RESET : GPIO_PIN_SET);
12   HAL_GPIO_WritePin(ROW6_GPIO_Port, ROW6_Pin, ((
    matrix_buffer[index_led_matrix] >> 6) & 1) ?
    GPIO_PIN_RESET : GPIO_PIN_SET);
13   HAL_GPIO_WritePin(ROW7_GPIO_Port, ROW7_Pin, ((
    matrix_buffer[index_led_matrix] >> 7) & 1) ?
    GPIO_PIN_RESET : GPIO_PIN_SET);
14
15   switch (index_led_matrix)
16   {
17   case 0:
18       HAL_GPIO_WritePin(ENM0_GPIO_Port, ENM0_Pin,
    GPIO_PIN_RESET);
19       HAL_GPIO_WritePin(ENM1_GPIO_Port, ENM1_Pin,
    GPIO_PIN_SET);
20       HAL_GPIO_WritePin(ENM2_GPIO_Port, ENM2_Pin,
    GPIO_PIN_SET);
21       HAL_GPIO_WritePin(ENM3_GPIO_Port, ENM3_Pin,
    GPIO_PIN_SET);
22       HAL_GPIO_WritePin(ENM4_GPIO_Port, ENM4_Pin,
    GPIO_PIN_SET);
23       HAL_GPIO_WritePin(ENM5_GPIO_Port, ENM5_Pin,
    GPIO_PIN_SET);
24       HAL_GPIO_WritePin(ENM6_GPIO_Port, ENM6_Pin,
    GPIO_PIN_SET);
25       HAL_GPIO_WritePin(ENM7_GPIO_Port, ENM7_Pin,
    GPIO_PIN_SET);
26       break;
27   case 1:
28       HAL_GPIO_WritePin(ENM0_GPIO_Port, ENM0_Pin,
    GPIO_PIN_SET);
29       HAL_GPIO_WritePin(ENM1_GPIO_Port, ENM1_Pin,
    GPIO_PIN_RESET);
30       HAL_GPIO_WritePin(ENM2_GPIO_Port, ENM2_Pin,
    GPIO_PIN_SET);
31       HAL_GPIO_WritePin(ENM3_GPIO_Port, ENM3_Pin,
    GPIO_PIN_SET);
32       HAL_GPIO_WritePin(ENM4_GPIO_Port, ENM4_Pin,
    GPIO_PIN_SET);
33       HAL_GPIO_WritePin(ENM5_GPIO_Port, ENM5_Pin,
    GPIO_PIN_SET);
34       HAL_GPIO_WritePin(ENM6_GPIO_Port, ENM6_Pin,
    GPIO_PIN_SET);
35       HAL_GPIO_WritePin(ENM7_GPIO_Port, ENM7_Pin,
    GPIO_PIN_SET);
```

```
36          break;
37      case 2:
38          HAL_GPIO_WritePin(ENM0_GPIO_Port, ENM0_Pin,
    GPIO_PIN_SET);
39          HAL_GPIO_WritePin(ENM1_GPIO_Port, ENM1_Pin,
    GPIO_PIN_SET);
40          HAL_GPIO_WritePin(ENM2_GPIO_Port, ENM2_Pin,
    GPIO_PIN_RESET);
41          HAL_GPIO_WritePin(ENM3_GPIO_Port, ENM3_Pin,
    GPIO_PIN_SET);
42          HAL_GPIO_WritePin(ENM4_GPIO_Port, ENM4_Pin,
    GPIO_PIN_SET);
43          HAL_GPIO_WritePin(ENM5_GPIO_Port, ENM5_Pin,
    GPIO_PIN_SET);
44          HAL_GPIO_WritePin(ENM6_GPIO_Port, ENM6_Pin,
    GPIO_PIN_SET);
45          HAL_GPIO_WritePin(ENM7_GPIO_Port, ENM7_Pin,
    GPIO_PIN_SET);
46          break;
47      case 3:
48          HAL_GPIO_WritePin(ENM0_GPIO_Port, ENM0_Pin,
    GPIO_PIN_SET);
49          HAL_GPIO_WritePin(ENM1_GPIO_Port, ENM1_Pin,
    GPIO_PIN_SET);
50          HAL_GPIO_WritePin(ENM2_GPIO_Port, ENM2_Pin,
    GPIO_PIN_SET);
51          HAL_GPIO_WritePin(ENM3_GPIO_Port, ENM3_Pin,
    GPIO_PIN_RESET);
52          HAL_GPIO_WritePin(ENM4_GPIO_Port, ENM4_Pin,
    GPIO_PIN_SET);
53          HAL_GPIO_WritePin(ENM5_GPIO_Port, ENM5_Pin,
    GPIO_PIN_SET);
54          HAL_GPIO_WritePin(ENM6_GPIO_Port, ENM6_Pin,
    GPIO_PIN_SET);
55          HAL_GPIO_WritePin(ENM7_GPIO_Port, ENM7_Pin,
    GPIO_PIN_SET);
56          break;
57      case 4:
58          HAL_GPIO_WritePin(ENM0_GPIO_Port, ENM0_Pin,
    GPIO_PIN_SET);
59          HAL_GPIO_WritePin(ENM1_GPIO_Port, ENM1_Pin,
    GPIO_PIN_SET);
60          HAL_GPIO_WritePin(ENM2_GPIO_Port, ENM2_Pin,
    GPIO_PIN_SET);
61          HAL_GPIO_WritePin(ENM3_GPIO_Port, ENM3_Pin,
    GPIO_PIN_SET);
62          HAL_GPIO_WritePin(ENM4_GPIO_Port, ENM4_Pin,
    GPIO_PIN_RESET);
63          HAL_GPIO_WritePin(ENM5_GPIO_Port, ENM5_Pin,
```

```
  GPIO_PIN_SET);
64        HAL_GPIO_WritePin(ENM6_GPIO_Port, ENM6_Pin,
  GPIO_PIN_SET);
65        HAL_GPIO_WritePin(ENM7_GPIO_Port, ENM7_Pin,
  GPIO_PIN_SET);
66        break;
67     case 5:
68        HAL_GPIO_WritePin(ENM0_GPIO_Port, ENM0_Pin,
  GPIO_PIN_SET);
69        HAL_GPIO_WritePin(ENM1_GPIO_Port, ENM1_Pin,
  GPIO_PIN_SET);
70        HAL_GPIO_WritePin(ENM2_GPIO_Port, ENM2_Pin,
  GPIO_PIN_SET);
71        HAL_GPIO_WritePin(ENM3_GPIO_Port, ENM3_Pin,
  GPIO_PIN_SET);
72        HAL_GPIO_WritePin(ENM4_GPIO_Port, ENM4_Pin,
  GPIO_PIN_SET);
73        HAL_GPIO_WritePin(ENM5_GPIO_Port, ENM5_Pin,
  GPIO_PIN_RESET);
74        HAL_GPIO_WritePin(ENM6_GPIO_Port, ENM6_Pin,
  GPIO_PIN_SET);
75        HAL_GPIO_WritePin(ENM7_GPIO_Port, ENM7_Pin,
  GPIO_PIN_SET);
76        break;
77     case 6:
78        HAL_GPIO_WritePin(ENM0_GPIO_Port, ENM0_Pin,
  GPIO_PIN_SET);
79        HAL_GPIO_WritePin(ENM1_GPIO_Port, ENM1_Pin,
  GPIO_PIN_SET);
80        HAL_GPIO_WritePin(ENM2_GPIO_Port, ENM2_Pin,
  GPIO_PIN_SET);
81        HAL_GPIO_WritePin(ENM3_GPIO_Port, ENM3_Pin,
  GPIO_PIN_SET);
82        HAL_GPIO_WritePin(ENM4_GPIO_Port, ENM4_Pin,
  GPIO_PIN_SET);
83        HAL_GPIO_WritePin(ENM5_GPIO_Port, ENM5_Pin,
  GPIO_PIN_SET);
84        HAL_GPIO_WritePin(ENM6_GPIO_Port, ENM6_Pin,
  GPIO_PIN_RESET);
85        HAL_GPIO_WritePin(ENM7_GPIO_Port, ENM7_Pin,
  GPIO_PIN_SET);
86        break;
87     case 7:
88        HAL_GPIO_WritePin(ENM0_GPIO_Port, ENM0_Pin,
  GPIO_PIN_SET);
89        HAL_GPIO_WritePin(ENM1_GPIO_Port, ENM1_Pin,
  GPIO_PIN_SET);
90        HAL_GPIO_WritePin(ENM2_GPIO_Port, ENM2_Pin,
  GPIO_PIN_SET);
```

```
91          HAL_GPIO_WritePin(ENM3_GPIO_Port, ENM3_Pin,
    GPIO_PIN_SET);
92          HAL_GPIO_WritePin(ENM4_GPIO_Port, ENM4_Pin,
    GPIO_PIN_SET);
93          HAL_GPIO_WritePin(ENM5_GPIO_Port, ENM5_Pin,
    GPIO_PIN_SET);
94          HAL_GPIO_WritePin(ENM6_GPIO_Port, ENM6_Pin,
    GPIO_PIN_SET);
95          HAL_GPIO_WritePin(ENM7_GPIO_Port, ENM7_Pin,
    GPIO_PIN_RESET);
96          break;
97      }
98  }
```

Program 1.13: Function to display data on LED Matrix

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

Answer: I have configured the switching time between columns to be 100ms as the following manner.

```
1  ...
2  int timer_matrix_counter = 0;
3  int timer_matrix_flag = 0;
4  void setTimerMatrix(int duration)
5  {
6      timer_matrix_counter = duration / TIMER_CYCLE;
7      timer_matrix_flag = 0;
8  }
9
10 void timerMatrixRun()
11 {
12     if (timer_matrix_counter > 0)
13     {
14         --timer_matrix_counter;
15         if (timer_matrix_counter == 0) timer_matrix_flag =
    1;
16     }
17 }
18 ...
19 setTimerMatrix(100);
20 ...
21 while (1)
22 {
23     ...
24     if (timer_matrix_flag == 1)
25     {
26         updateLEDMatrix(index_led_matrix);
27         setTimerMatrix(100);
28         ++index_led_matrix;
```

```
29          if (index_led_matrix >= MAX_LED_MATRIX)
30          {
31              index_led_matrix = 0;
32          }
33      }
34      ...
35  }
36  ...
37  void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef *
        htim )
38  {
39      ...
40      timerMatrixRun();
41      ...
42  }
```

Program 1.14: Source code for scanning matrix LED

There are 8 columns in total, so the frequency of the scanning process is $\frac{1}{8*0.1} = 1.25$Hz.

1.9 Exercise 10

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

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

Answer: I have implemented a rightward shift animation on the LED matrix by shifting the elements in the `matrix_buffer` array to the right (with the last element being moved to the first index) every 1600ms (equivalent to 2 cycles of the scanning process) as stated below.

```
...
int timer_shift_counter = 0;
int timer_shift_flag = 0;
void setTimerShift(int duration)
{
    timer_shift_counter = duration / TIMER_CYCLE;
    timer_shift_flag = 0;
}

void timerShiftRun()
{
    if (timer_shift_counter > 0)
    {
        --timer_shift_counter;
        if (timer_shift_counter == 0) timer_shift_flag = 1;
    }
}
...
setTimerMatrix(100);
setTimerShift(1600);
...
while (1)
{
    ...
    if (timer_shift_flag == 1)
    {
          int matrix_buffer7 = matrix_buffer[7];
          for (int i = 6; i >= 0; --i)
          {
              matrix_buffer[i + 1] = matrix_buffer[i];
          }
          matrix_buffer[0] = matrix_buffer7;
          setTimerShift(1600);
    }
    if (timer_matrix_flag == 1)
    {
        updateLEDMatrix(index_led_matrix);
        setTimerMatrix(100);
        ++index_led_matrix;
        if (index_led_matrix >= MAX_LED_MATRIX)
```

```
41          {
42              index_led_matrix = 0;
43          }
44      }
45      ...
46  }
47  ...
48  void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef *
        htim )
49  {
50      ...
51      timerShiftRun();
52      timerMatrixRun();
53      ...
54  }
```

Program 1.15: Source code for exercise 10