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

Microprocessors-Microcontrollers

COURSE ID: CO3010 - HK251 CLASS: CC02

Lab 2 Report

Lecturer: Phan Văn Sỹ
Students: Nguyễn Thế Khang - 2352490

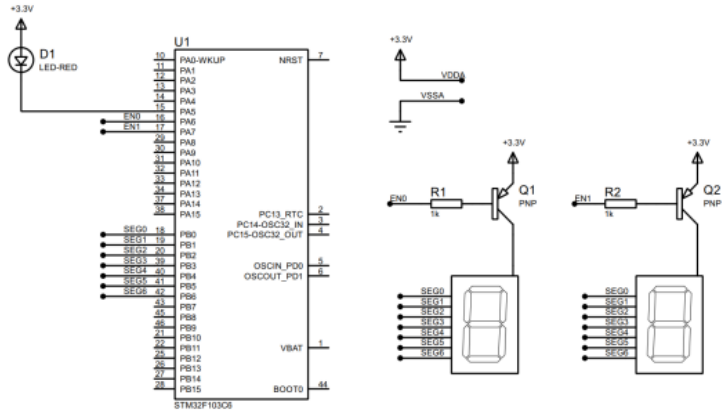
HỒ CHÍ MINH CITY, September 2025

Contents

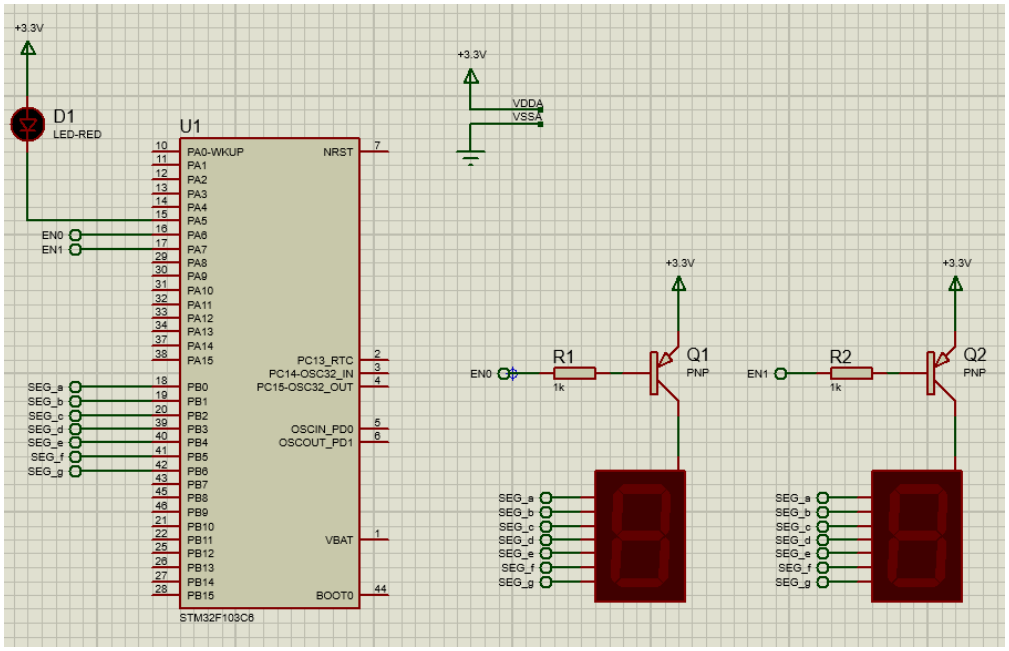
1 Exercise and Report

1.1 Exercise 1

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.



1.1.1 Schematic



1.1.2 Source Code

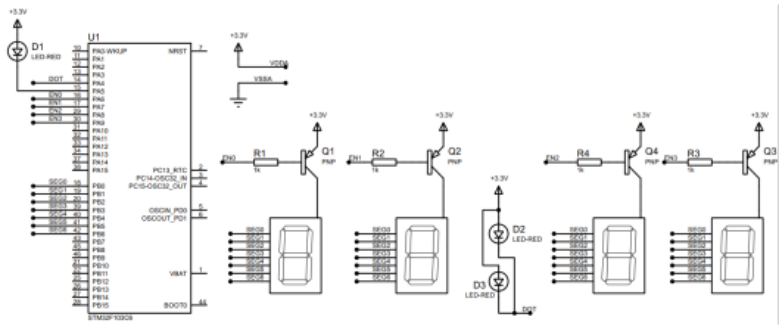
```
int counter = 100;
int seg_counter = 50;
int led_status = 0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  counter--;
  if (counter <= 0) {
     counter = 100;
     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
  }

  seg_counter--;
  if (seg_counter <= 0) {
     seg_counter = 50;

     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);

     if (led_status == 0) {
        //7-SEG 1
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, RESET);
        display7SEG(1);
        led_status = 1;
     } else {
        //7-SEG 2
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, RESET);
        display7SEG(2);
        led_status = 0;
     }
  }
}
```

1.1.3 GitHub Repository

- **Schematic:** Lab2Ex1.pdsprj
- **Source:** Exercise 1.h

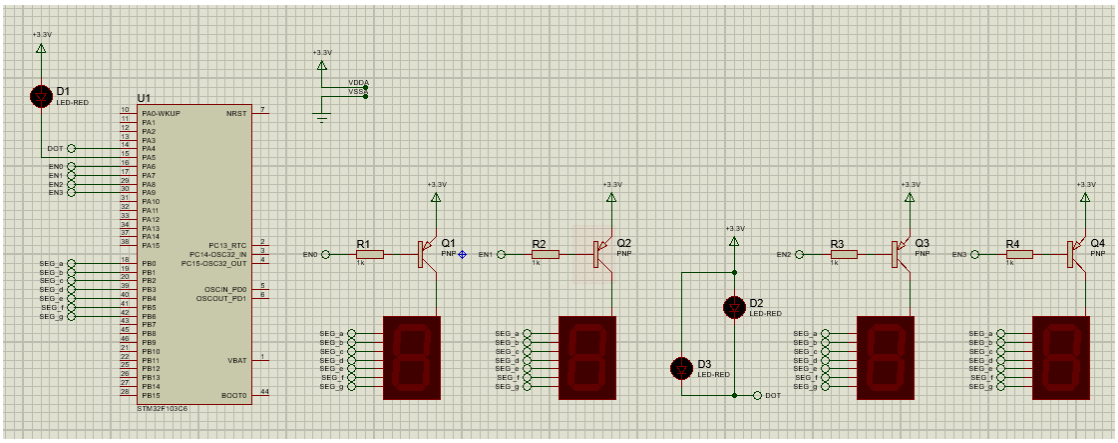
1.2 Exercise 2

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



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.

1.2.1 Schematic



1.2.2 Source Code

```
int counter = 100;
int seg_counter = 50;
int led_status = 0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
```

```
    counter--;
    if (counter <= 0) {
       counter = 100;
       HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
       HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
    }

    seg_counter--;
    if (seg_counter <= 0) {
       seg_counter = 50;

       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, RESET);
       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, RESET);
       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, RESET);
       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, RESET);

       if (led_status == 0) {
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
          display7SEG(1);
          led_status = 1;
       } else if (led_status == 1) {
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
          display7SEG(2);
          led_status = 2;
       } else if (led_status == 2) {
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
          display7SEG(3);
          led_status = 3;
       } else {
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
          display7SEG(0);
          led_status = 0;
       }
    }
}
```

1.2.3 GitHub Repository

- **Schematic:** Lab2Ex2 to 8.pdsprj
- **Source:** Exercise 2.h

1.3 Exercise 3

Implement a function named update7SEG(int index). An array of 4 integer numbers are declared in this case. This function should be invoked in the timer interrupt, e.g update7SEG(indexled++). The variable indexled is updated to stay in a valid range, which is from 0 to 3.

```
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 1.4: An example for your source code

1.3.1 Source Code

update7SEG function

```
1 const int MAX_LED = 4;
2 int index_led = 0;
3 int led_buffer[4] = {1, 2, 3, 4};
4 void update7SEG(int index) {
5     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, RESET);
6     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, RESET);
7     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, RESET);
8     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, RESET);
9
10    switch (index) {
11        case 0:
```

```
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
            display7SEG(led_buffer[0]);
            break;
        case 1:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
            display7SEG(led_buffer[1]);
            break;
        case 2:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, SET);
            display7SEG(led_buffer[2]);
            break;
        case 3:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, SET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, SET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, SET);
            display7SEG(led_buffer[3]);
            break;
        default:
            break;
    }
}
```

HAL_TIM_PeriodElapsedCallback function

```
int counter = 100;
int led_status = 0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    counter--;
    if (counter <= 0) {
        counter = 100;
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    }

    if (counter == 100 || counter == 50) {
        update7SEG(index_led);
        index_led = (index_led + 1) % MAX_LED;
    }
}
```

1.3.2 GitHub Repository

- **Source:** Exercise 3.h

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.

1.4.1 Source Code

```
int counter = 100;
int led_status = 0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    counter--;
    if (counter <= 0) {
        counter = 100;
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    }

    if (counter == 100 || counter == 25) {
        update7SEG(index_led);
        index_led = (index_led + 1) % MAX_LED;
    }
}
```

1.4.2 GitHub Repository

- **Source:** Exercise 4.h

1.5 Exercise 5

Implement a digital clock with hour and minute information displayed by two seven-segment LEDs. 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.

1.5.1 updateClockBuffer function

```
void updateClockBuffer() {
    extern int led_buffer[4];

    // hour
    led_buffer[0] = hour / 10;   // tens
    led_buffer[1] = hour % 10;   // units

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

1.5.2 GitHub Repository

- **Source:** Exercise 5.h

1.6 Exercise 6

1.6.1 Questions

The main goal of this exercise is to reduce the complexity (or code processing) within the timer interrupt. The time consumed in the interrupt can lead to nested interrupt issues, which may crash the whole system. A simple solution is to disable the timer whenever the interrupt occurs, and then enable it again. However, this approach compromises real-time processing guarantees.

In this exercise, a software timer is created, whose counter is decremented every time the timer interrupt is raised (every 10 ms). By using this software timer, the `Hal_Delay(1000)` function call in the main program is removed. In an MCU system, non-blocking delays are preferable over blocking delays. The details of creating the software timer are presented below. The source code is added to your current program; do not delete the code from Exercise 5.

Report 1: If line 1 of the code is missing, what happens and why?
Report 2: If line 1 is changed to `setTimer0(1)`, what happens and why?
Report 3: If line 1 is changed to `setTimer0(10)`, how does it differ from the previous cases, and why?

1.6.2 Answers

1. If line 1, `setTimer0(1000)`, is missing, the timer is not initialized. The `timer0_counter` variable remains unassigned, and the timer interrupt will not decrement it. As a result, `timer0_flag` is never set to 1, preventing the LED toggle condition in the main program from executing. The program will appear to do nothing because the timer never triggers.

2. If line 1 is changed to `setTimer0(1)`, the timer duration is set to 1 ms. Since the timer interrupt occurs every 10 ms (defined by `TIMER_CYCLE = 10`), the `timer0_counter` is calculated as `1 / 10 = 0` (integer division). This means the timer expires immediately without any countdown, and `timer0_flag` is never set to 1. Consequently, the LED toggle condition is not met, and the program does not execute as intended.

3. If line 1 is changed to `setTimer0(10)`, the timer duration is set to 10 ms, resulting in `timer0_counter = 10 / 10 = 1`. The timer interrupt decrements `timer0_counter` to 0 in one cycle (10 ms), setting `timer0_flag` to 1. This triggers the LED toggle condition in the main program after a 10 ms initial delay. Subsequent toggles depend on the program re-setting the timer, typically every 1000 ms (as per Exercise 5), leading to consistent LED toggling every 1 second, unlike the immediate expiration in Report 2 or the non-execution in Report 1.

1.7 Exercise 7

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

1.7.1 Source Code

main Function

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

  MX_GPIO_Init();
  MX_TIM2_Init();

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

 setTimer0(1000);
 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(GPIOA, GPIO_PIN_4);
       HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
       setTimer0(1000);
    }
  }
}
```

1.7.2 GitHub Repository

- **Source:** main.c

1.8 Exercise 8

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

1.8.1 Source Code

main Function

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

  MX_GPIO_Init();
  MX_TIM2_Init();

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

 setTimer0(1000);
 setTimer1(250);
 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(GPIOA, GPIO_PIN_4);
       HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
       setTimer0(1000);
    }
    if (timer1_flag == 1) {
        update7SEG(index_led);
        index_led = (index_led + 1) % MAX_LED;
        setTimer1(250);
    }
  }
```

```
}
```

updateClockBuffer Function

```
int hour = 15, minute = 12, second = 15;
void updateClockBuffer(void) {
    led_buffer[0] = hour / 10; // Tens of hours
    led_buffer[1] = hour % 10; // Units of hours
    led_buffer[2] = minute / 10; // Tens of minutes
    led_buffer[3] = minute % 10; // Units of minutes
}
```

Timer Functions

```
int timer0_counter = 0;
int timer0_flag = 0;
int timer1_counter = 0;
int timer1_flag = 0;
int TIMER_CYCLE = 10;
void setTimer0(int duration) {
   timer0_counter = duration / TIMER_CYCLE;
   timer0_flag = 0;
}

void setTimer1(int duration) {
   timer1_counter = duration / TIMER_CYCLE;
   timer1_flag = 0;
}

void timer_run() {
   if (timer0_counter > 0) {
      timer0_counter--;
      if (timer0_counter == 0) timer0_flag = 1;
   }
   if (timer1_counter > 0) {
      timer1_counter--;
      if (timer1_counter == 0) timer1_flag = 1;
   }
}

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

1.8.2 GitHub Repository

- **Main:** main.c
- **Header File:** Exercise 6–8.h

1.9 Exercise 9

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

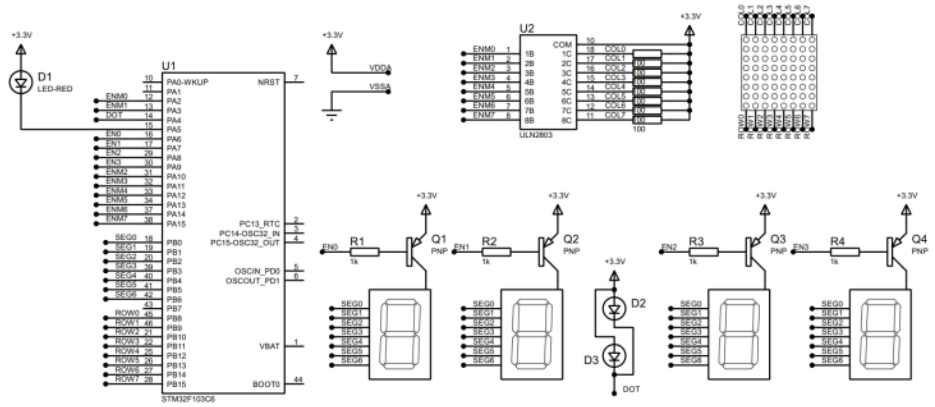
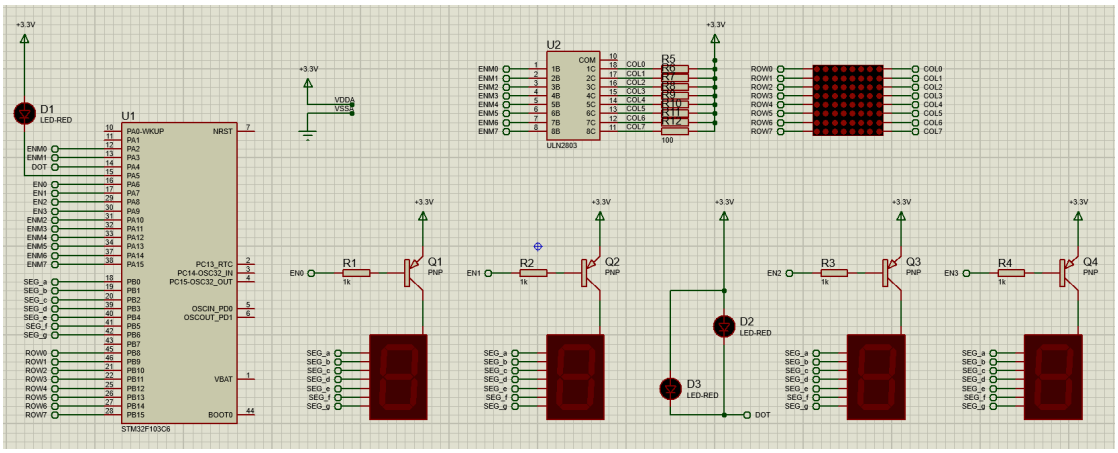


Figure 1.7: LED matrix is added to the simulation

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.

1.9.1 Schematic



1.9.2 Source Code

```
const int MAX_LED_MATRIX = 8;
int index_led_matrix = 0;
uint8_t matrix_buffer[8] = {~0x18, ~0x24, ~0x42, ~0x7E, ~0x42, ~0x42, ~0x42, ~0x00};
void updateLEDMatrix(int index) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_10 | GPIO_PIN_11 |
                            GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15, SET);

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, (matrix_buffer[index] & 0x01) ? SET : RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, (matrix_buffer[index] & 0x02) ? SET : RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, (matrix_buffer[index] & 0x04) ? SET : RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, (matrix_buffer[index] & 0x08) ? SET : RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, (matrix_buffer[index] & 0x10) ? SET : RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, (matrix_buffer[index] & 0x20) ? SET : RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, (matrix_buffer[index] & 0x40) ? SET : RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, (matrix_buffer[index] & 0x80) ? SET : RESET);

    switch (index) {
        case 0:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, RESET);
            break;
        case 1:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, RESET);
            break;
        case 2:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, RESET);
            break;
        case 3:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, RESET);
            break;
        case 4:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, RESET);
            break;
        case 5:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, RESET);
            break;
        case 6:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, RESET);
            break;
        case 7:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, RESET);
            break;
        default:
            break;
    }
}
```

1.9.3 GitHub Repository

- **Schematic:** Lab2Ex9 to 10.pdsprj
- **Source:** Exercise 9.h

1.10 Exercise 10

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

1.10.1 Source Code

```
void shiftLeftMatrix(void) {
    for (int i = 0; i < MAX_LED_MATRIX; i++) {
        uint8_t lsb = (matrix_buffer[i] & 0x01) ? 1 : 0;
        matrix_buffer[i] = (matrix_buffer[i] << 1) | (lsb >> 7);
    }
}
```

Explanation: I intend to use LSB and MSB for shifting, first iterating through each bytes and storing it in variable **temp**. It shifts bits right (temp » 1) and places the MSB in the LSB (temp « 7) using a bitwise OR to produce a left-shifting visual effect.

1.10.2 GitHub Repository

- **Source:** Exercise 10.h