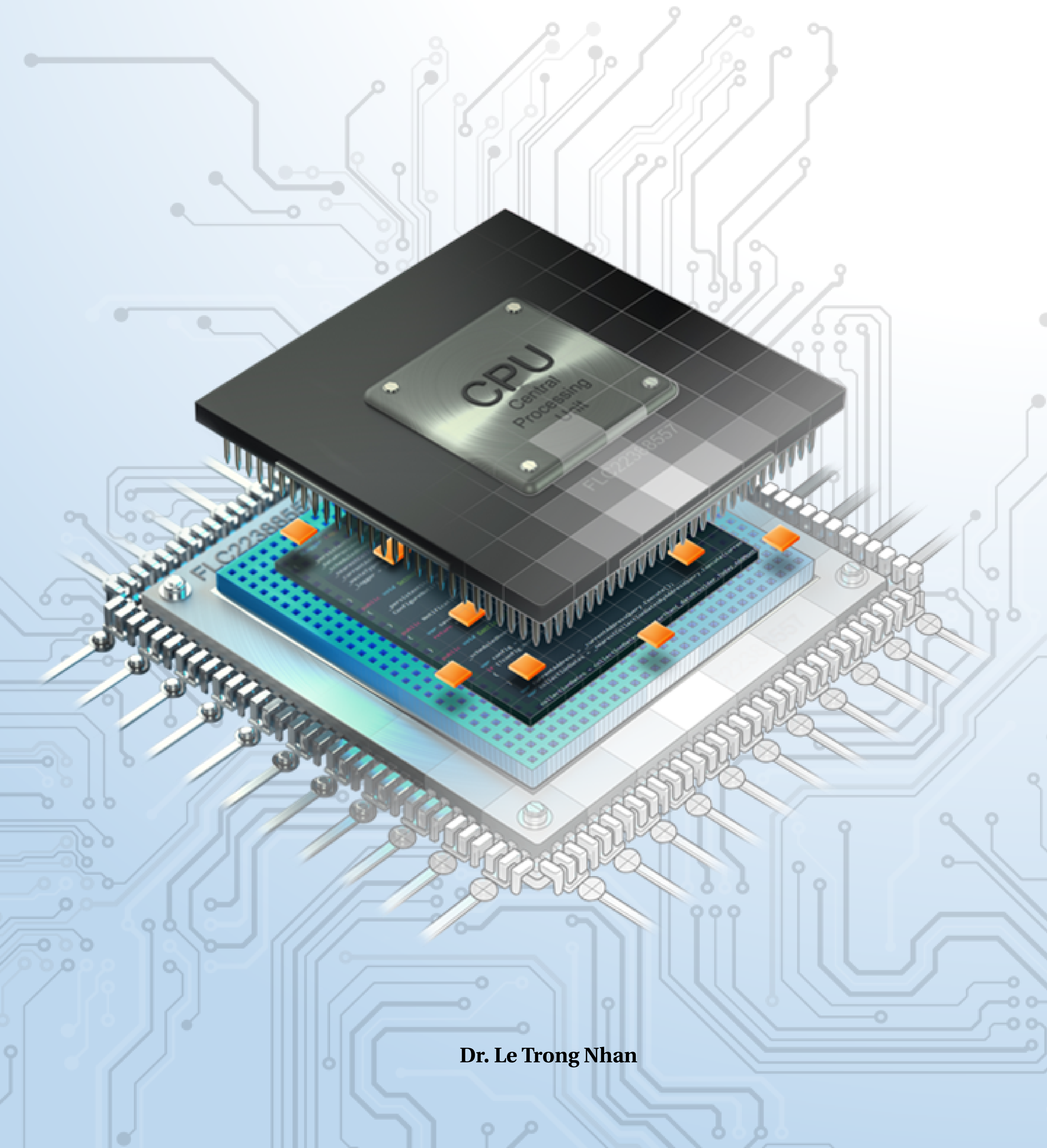
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

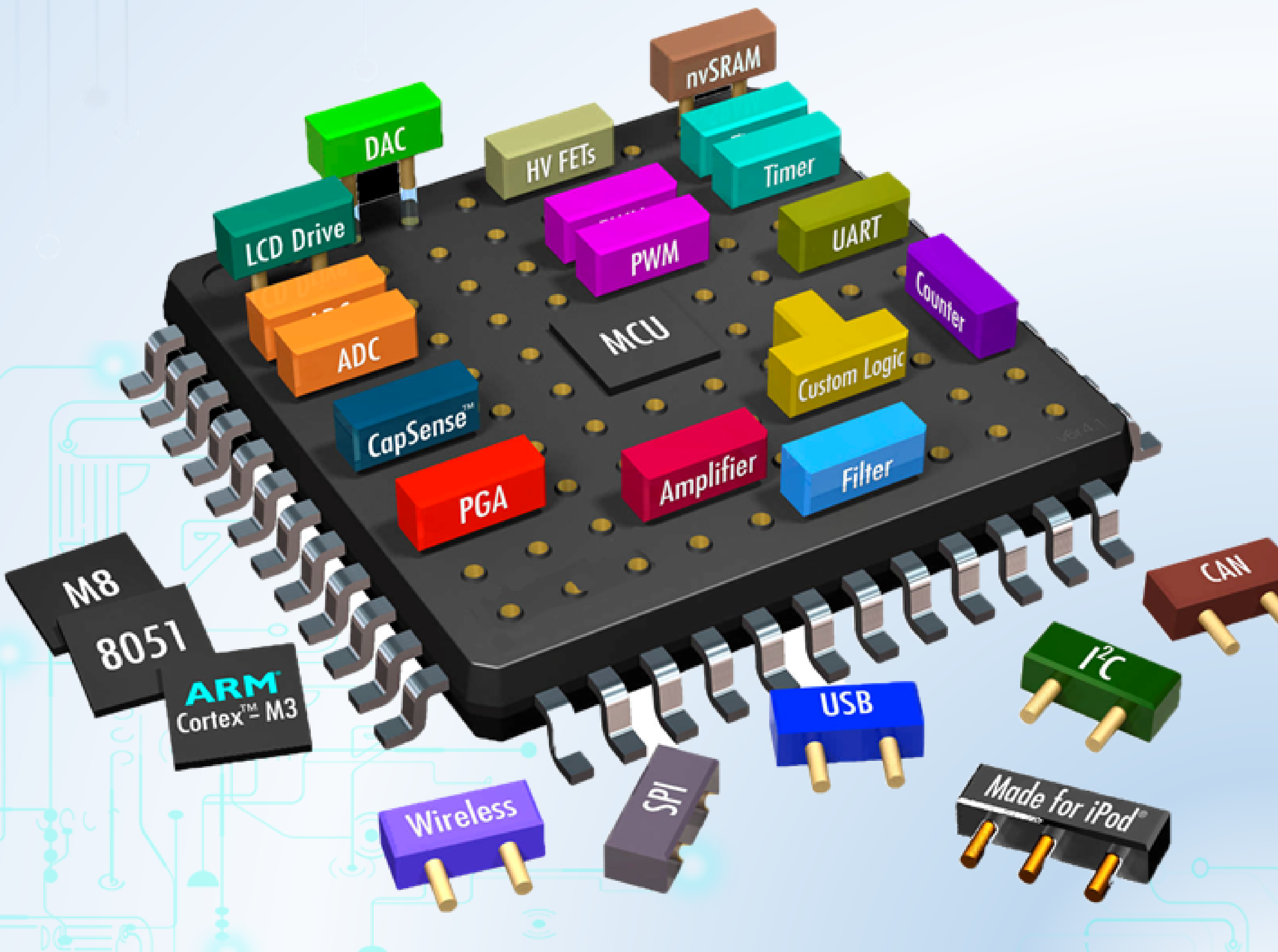


Dr. Le Trong Nhan

Mục lục

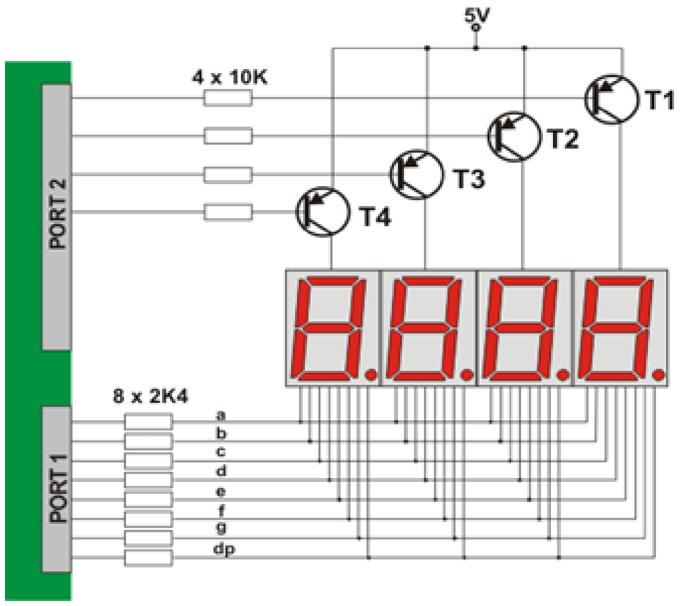
CHƯƠNG 1

Timer Interrupt and LED Scanning



1 Introduction

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



Hình 1.1: Four seven segment LED interface for a micro-controller

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

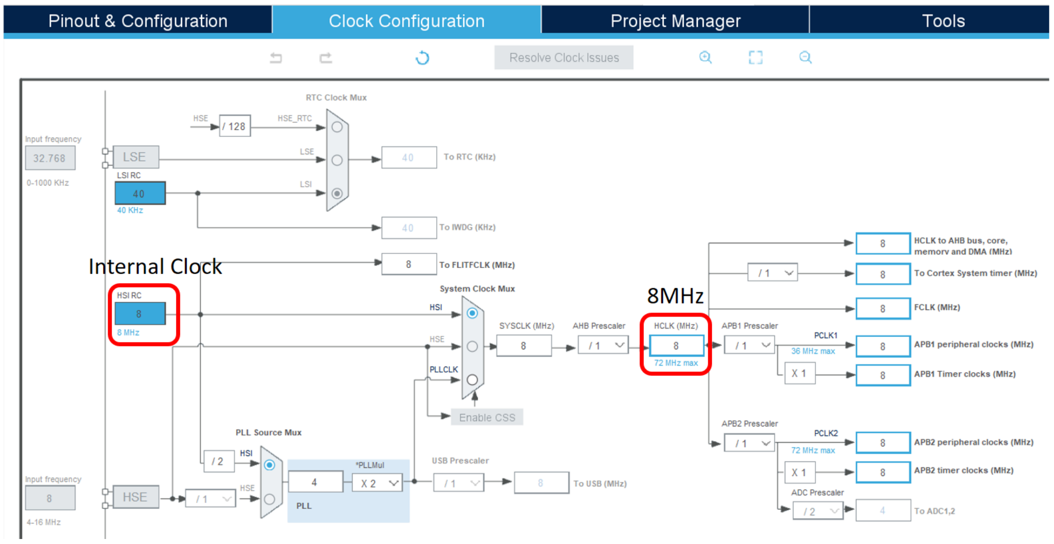
In the above diagram each seven segment display is having 8 internal LEDs, leading to the total number of LEDs is 32. However, not all the LEDs are required to turn ON, but one of them is needed. Therefore, only 12 lines are needed to control the whole 4 seven segment LEDs. By controlling with the micro-controller,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

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

2 Timer Interrupt Setup

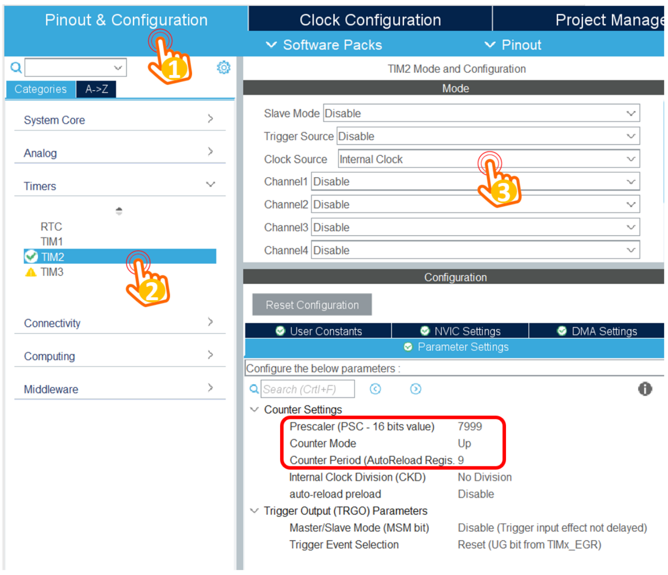
Step 1: Create a simple project, which LED connected to PA5. The manual can be found in the first lab.

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



Hình 1.2: Default clock source for the system

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



TIM2 Mode and Configuration

Mode

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

Configuration

Configure the below parameters :

Counter Settings

Prescaler (PSC - 16 bits value)	7999
Counter Mode	Up
Counter Period (AutoReload Regis	9
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable

Trigger Output (TRGO) Parameters

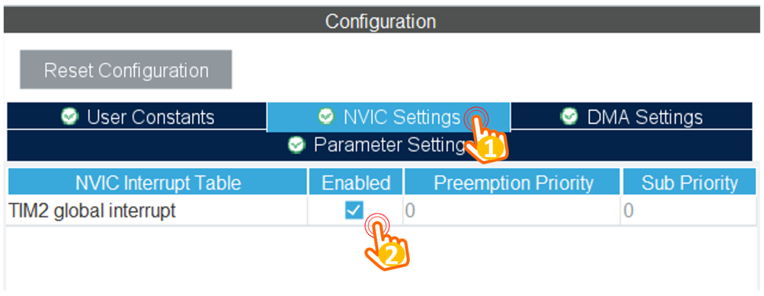
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

Hình 1.3: Configure for Timer 2

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

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

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



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

Hình 1.4: Enable timer interrupt

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

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

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

  MX_GPIO_Init();
  MX_TIM2_Init();

  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */g3

  while (1){

  }
}
```

Program 1.1: Init the timer interrupt in main

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

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

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

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

Program 1.2: Add an interrupt service routine

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

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

Program 1.3: LED Blinky using timer interrupt

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

3 Exercise and Report

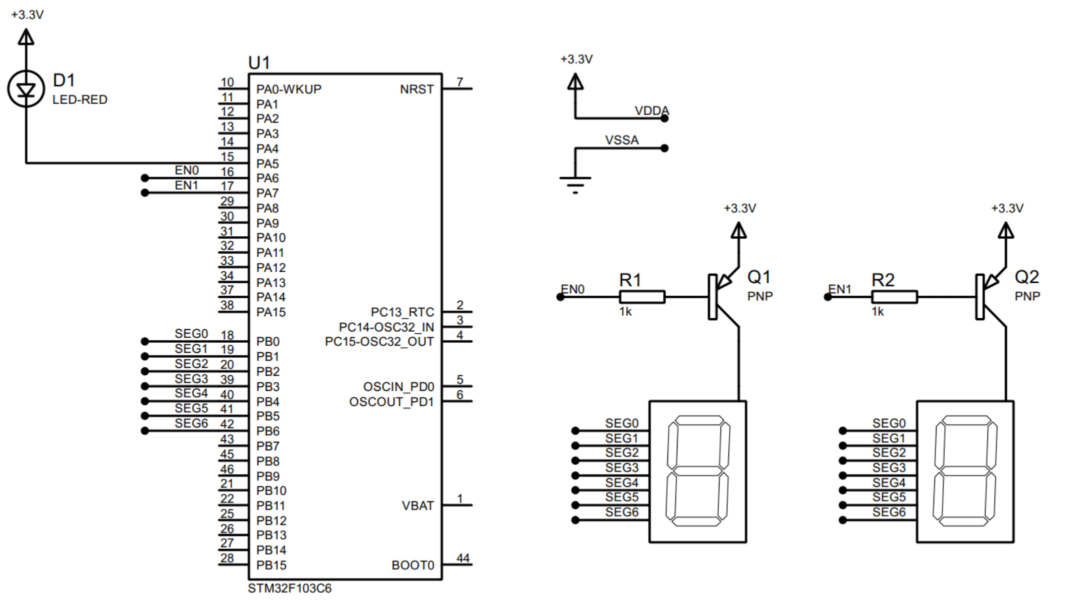
GITHUB LINK: `https://github.com/dongghoul/Microcontroller_Lab/tree/Lab2`

3.1 Exercise 1

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

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

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



Hình 1.5: Simulation schematic in Proteus

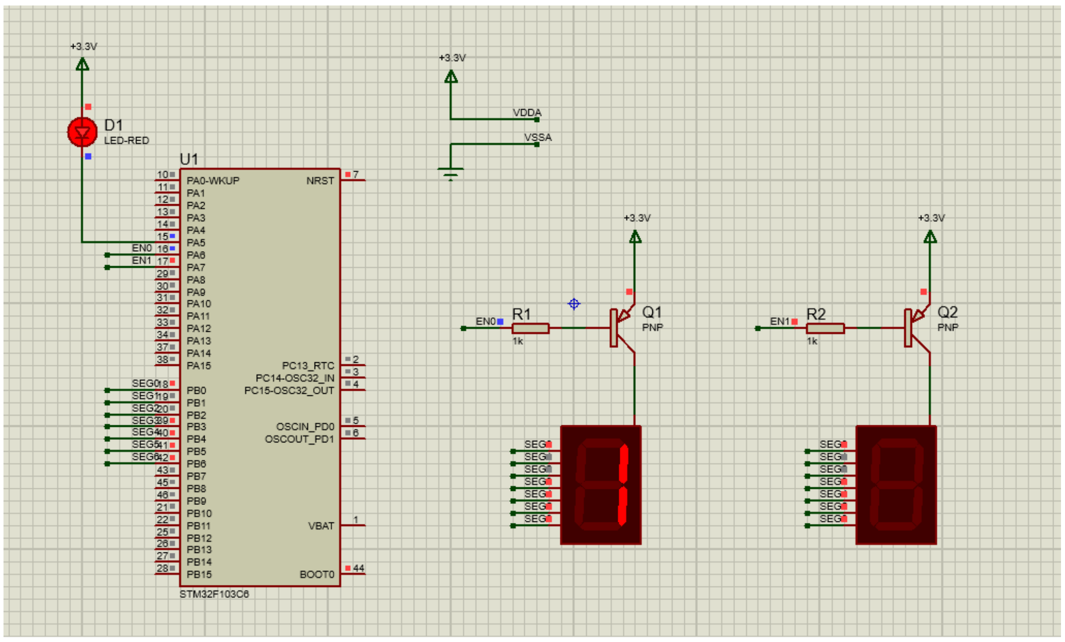
Components used in the schematic are listed bellow:

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

- STM32F103C6

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

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



Hình 1.6: Exercise 1

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

```
int led7seg_counter = 50;
int led7seg_state = 0;
int ledRed_counter = 100;
int ledRed_state = 0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  led7seg_counter--;
  ledRed_counter--;
  if (led7seg_counter <= 0){
    led7seg_counter = 50;
    led7seg_state = 1 - led7seg_state;
  }
  if (ledRed_counter <= 0){
    ledRed_counter = 100;
    ledRed_state = 1 - ledRed_state;
  }

  switch (led7seg_state){
    case 0:
      HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, GPIO_PIN_SET);
      display7SEG(1);
      break;
    case 1:
      HAL_GPIO_WritePin(EN0_GPIO_Port, EN0_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(EN1_GPIO_Port, EN1_Pin, GPIO_PIN_RESET);
      display7SEG(2);
      break;
    default:
      break;
  }
  switch (ledRed_state){
    case 0:
      HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, RESET);
      break;
    case 1:
      HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, SET);
      break;
    default:
      break;
  }
}
```

Program 1.4: An example for your source code

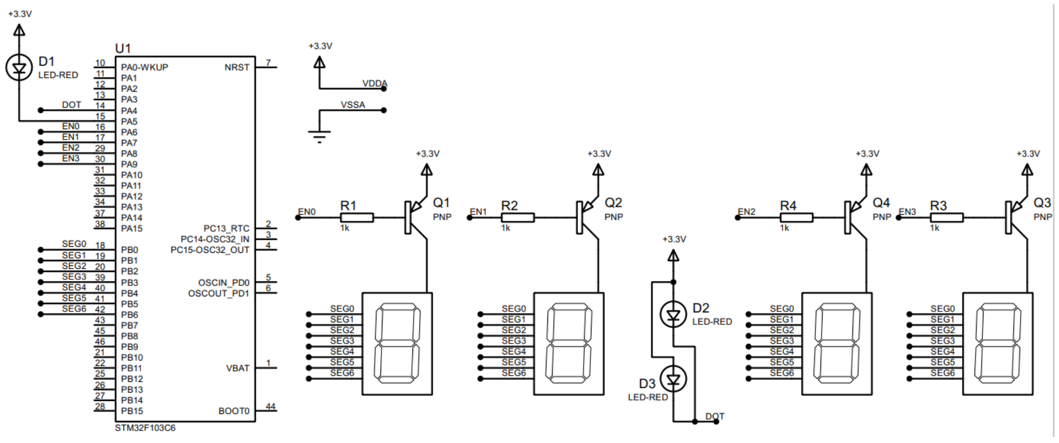
Short question: What is the frequency of the scanning process?

The frequency of the scanning process is 1Hz

3.2 Exercise 2

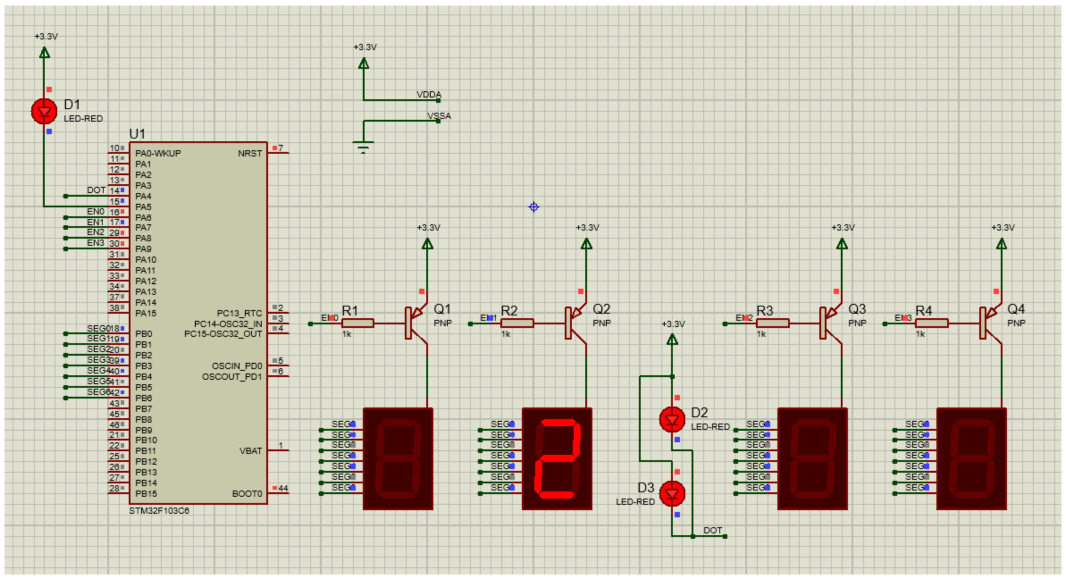
Extend to 4 seven segment LEDs and two LEDs (connected to PA4, labeled as **DOT**) in the middle as following:

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.**



Hình 1.7: Simulation schematic in Proteus

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



Hình 1.8: Exercise 2

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

```c
int led7seg_counter = 50;
int led7seg_state = 0;
int ledDot_counter = 100;
int ledDot_state = 0;
int ledRed_counter = 100;
int ledRed_state = 0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  led7seg_counter--;
  ledDot_counter--;
  ledRed_counter--;
  if (led7seg_counter <= 0){
    led7seg_counter = 50;
    led7seg_state = (led7seg_state + 1) % 4;
  }
  if (ledDot_counter <= 0){
    ledDot_counter = 100;
    ledDot_state = 1 - ledDot_state;
  }
  if (ledRed_counter <= 0){
    ledRed_counter = 100;
    ledRed_state = 1 - ledRed_state;
  }

  switch (led7seg_state){
    case 0:
      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);
      display7SEG(1);
      break;
    case 1:
      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);
      display7SEG(2);
      break;
    case 2:
      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);
      display7SEG(3);
      break;
    case 3:
      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);
        display7SEG(0);
        break;
      default:
        break;
    }
    switch (ledDot_state){
      case 0:
        HAL_GPIO_WritePin(DOT_GPIO_Port, DOT_Pin, RESET);
        break;
      case 1:
        HAL_GPIO_WritePin(DOT_GPIO_Port, DOT_Pin, SET);
        break;
      default:
        break;
    }
    switch (ledRed_state){
      case 0:
        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin,
    RESET);
        break;
      case 1:
        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, SET
    );
        break;
      default:
        break;
    }
}
```

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

The frequency of the scanning process is 0.5Hz

3.3 Exercise 3

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

```c
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 1.5: 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.

```c
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:
      //turn on first 7seg led only
      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:
      //turn on second 7seg led only
      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:
      //turn on third 7seg led only
      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:
      //turn on fourth 7seg led only
      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;
  }
  //display number according to led_buffer and index
  display7SEG(led_buffer[index]);
}
```

Report 2: Present the source code in the HAL_TIM_PeriodElapsedCallback.

```c
int led7seg_counter = 0;
//int led7seg_state = 0;
int ledDot_counter = 100;
int ledDot_state = 0;
int ledRed_counter = 100;
int ledRed_state = 0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
 led7seg_counter--;
 ledDot_counter--;
 ledRed_counter--;
```

```
  if (led7seg_counter <= 0){
    //reset counter
    led7seg_counter = 50;
    //updated index_led to stay in valid range, which is
 from 0 to 3
    index_led %= MAX_LED;
    //update 7seg leds
    update7SEG(index_led++);
  }
  if (ledDot_counter <= 0){
    //reset counter
    ledDot_counter = 100;
    //switch state of DOT leds
    ledDot_state = 1 - ledDot_state;
  }
  if (ledRed_counter <= 0){
    //reset counter
    ledRed_counter = 100;
    //switch state of RED led
    ledRed_state = 1 - ledRed_state;
  }

  switch (ledDot_state){
    case 0:
      //turn DOT leds on
      HAL_GPIO_WritePin(DOT_GPIO_Port, DOT_Pin, RESET);
      break;
    case 1:
      //turn DOT leds off
      HAL_GPIO_WritePin(DOT_GPIO_Port, DOT_Pin, SET);
      break;
    default:
      break;
  }
  switch (ledRed_state){
    case 0:
      //turn RED led on
      HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin,
 RESET);
      break;
    case 1:
      //turn RED led off
      HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, SET
 );
      break;
    default:
      break;
  }
}
```

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

3.4 Exercise 4

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

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

```c
//counter to change the frequency of 4 seven segment leds
    to 1Hz
const int COUNTER_SCAN_1HZ = 25;
int led7seg_counter = 0;
//int led7seg_state = 0;
int ledDot_counter = 100;
int ledDot_state = 0;
int ledRed_counter = 100;
int ledRed_state = 0;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  led7seg_counter--;
  ledDot_counter--;
  ledRed_counter--;
  if (led7seg_counter <= 0){
    //reset counter
    led7seg_counter = COUNTER_SCAN_1HZ;
    //updated index_led to stay in valid range, which is
   from 0 to 3
    index_led %= MAX_LED;
    //update 7seg leds
    update7SEG(index_led++);
  }
  if (ledDot_counter <= 0){
    //reset counter
    ledDot_counter = 100;
    //switch state of DOT leds
    ledDot_state = 1 - ledDot_state;
  }
  if (ledRed_counter <= 0){
    //reset counter
    ledRed_counter = 100;
    //switch state of RED led
    ledRed_state = 1 - ledRed_state;
  }

  switch (ledDot_state){
    case 0:
      //turn DOT leds on
      HAL_GPIO_WritePin(DOT_GPIO_Port, DOT_Pin, RESET);
      break;
    case 1:
      //turn DOT leds off
      HAL_GPIO_WritePin(DOT_GPIO_Port, DOT_Pin, SET);
      break;
    default:
      break;
  }
  switch (ledRed_state){
    case 0:
      //turn RED led on
      HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, RESET);
      break;
    case 1:
      //turn RED led off
      HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, SET);
      break;
    default:
      break;
  }
}
```

3.5 Exercise 5

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

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

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

Program 1.6: 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.

```
int hour = 5, minute = 55, second = 50;

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

3.6 Exercise 6

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

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

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

```
/* USER CODE BEGIN 0 */
int timer0_counter = 0;
int timer0_flag = 0;
int TIMER_CYCLE = 10;
void 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 1.7: 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 1.8: 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:

```
1 setTimer0(1000);
2 while (1){
3     if(timer0_flag == 1){
4         HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
5         setTimer0(2000);
6     }
7 }
```

Program 1.9: Software timer is used in main fuction to blink the LED

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

Answer: The system will not be able to check timer0_counter because the if statement in timer_run() function is never true. Thus the timer0_flag is never 1.

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

Answer: timer0_counter here equals 1/TIMER_CYCLE = 1/10 = 0 (according to how divide works in C). Thus, the system also won't do anything similar to Report 1.

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?

Answer: timer0_counter here equals 10/TIMER_CYCLE = 10/10 = 1. timer_run() function and the system, therefore, works properly.

3.7 Exercise 7

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

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

```c
setTimerClock(10);
setTimerDOT(1000);

int ledDOT_state = 0;
int ledRED_state = 0;

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

  if (timerDOT_flag == 1){
    //1 second blinking
    setTimerDOT(1000);
    ledDOT_state = 1 - ledDOT_state;
  }

  if (timerRED_flag == 1){
    //1 second blinking
    setTimerRED(1000);
    ledRED_state = 1 - ledRED_state;
  }

  switch (ledDOT_state){
    case 0:
      //turn DOT leds on
```

```c
        HAL_GPIO_WritePin(DOT_GPIO_Port, DOT_Pin, RESET);
        break;
      case 1:
        //turn DOT leds off
        HAL_GPIO_WritePin(DOT_GPIO_Port, DOT_Pin, SET);
        break;
      default:
        break;
    }

    switch (ledRED_state){
      case 0:
        //turn RED led on
        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, RESET);
        break;
      case 1:
        //turn RED led off
        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, SET);
        break;
      default:
        break;
    }
  }
```

3.8 Exercise 8

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

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

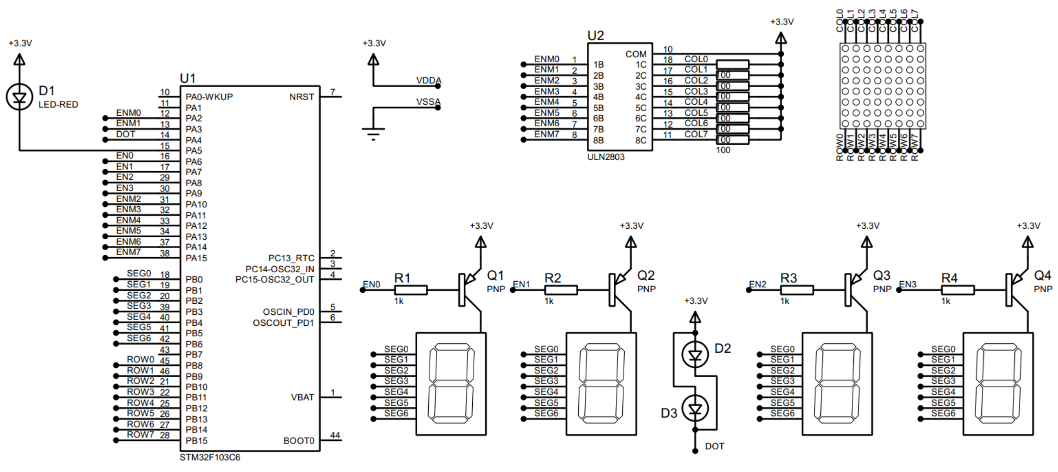
```c
int timer7seg_counter = 0;
int timer7seg_flag = 0;

void setTimer7SEG(int duration){
  timer7seg_counter = duration / TIMER_CYCLE;
  timer7seg_flag = 0;
}
```

Program 1.10: timer7seg

```c
  setTimerClock(10);
  setTimerDOT(1000);
  setTimer7SEG(10);

  int ledDOT_state = 0;
  int ledRED_state = 0;

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

    if (timer7seg_flag == 1){
      //set timer for 1Hz frequency
      setTimer7SEG(250);
      //updated index_led to stay in valid range, which is
   from 0 to 3
      index_led %= MAX_LED;
      //update 7seg leds
      update7SEG(index_led++);
    }

    if (timerDOT_flag == 1){
      //1 second blinking
      setTimerDOT(1000);
      ledDOT_state = 1 - ledDOT_state;
    }

    if (timerRED_flag == 1){
      //1 second blinking
      setTimerRED(1000);
      ledRED_state = 1 - ledRED_state;
    }

    switch (ledDOT_state){
      case 0:
        //turn DOT leds on
        HAL_GPIO_WritePin(DOT_GPIO_Port, DOT_Pin, RESET);
        break;
      case 1:
        //turn DOT leds off
        HAL_GPIO_WritePin(DOT_GPIO_Port, DOT_Pin, SET);
        break;
      default:
        break;
    }

    switch (ledRED_state){
      case 0:
          //turn RED led on
        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin,
  RESET);
        break;
      case 1:
        //turn RED led off
        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin,
  SET);
        break;
      default:
        break;
    }
  }
```
Program 1.11: while loop

3.9 Exercise 9

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

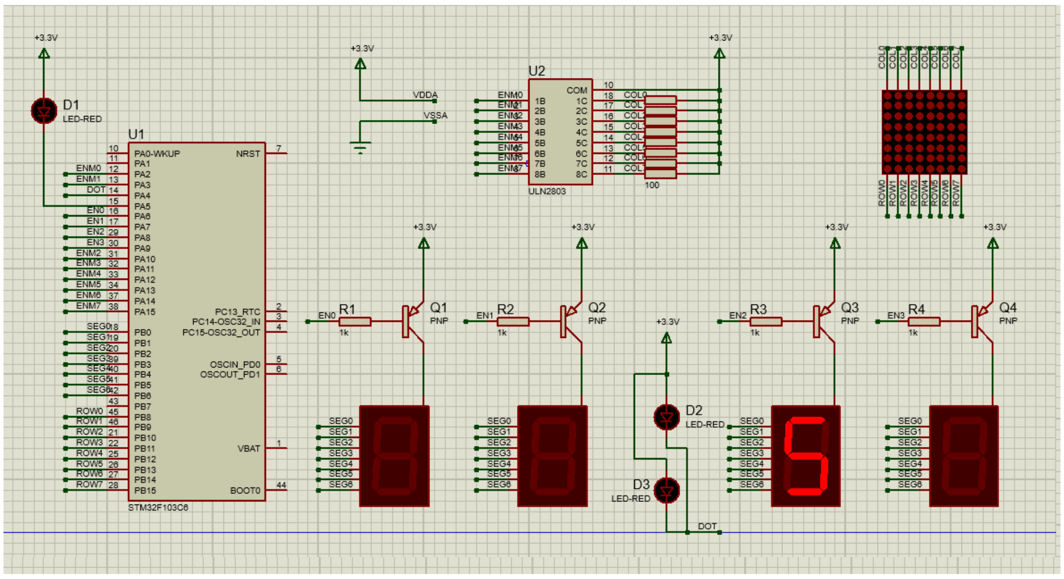


Hình 1.9: LED matrix is added to the simulation

In this schematic, two new components are added, including the **MATRIX-8X8-**

RED and **ULN2803**, which is an NPN transistor array to enable the power supply for a column of the LED matrix. Students can change the enable signal (from ENM0 to ENM7) if needed. Finally, the data signal (from ROW0 to ROW7) is connected to PB8 to PB15.

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



Hình 1.10: LED matrix is added to the simulation

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

```
const int MAX_LED_MATRIX = 8;
int index_led_matrix = 0;
// buffer to display "A" character
uint8_t matrix_buffer[8] = {0xFF, 0x01, 0x00, 0xEC
              , 0xEC, 0x00, 0x01, 0xFF};
void updateLEDMatrix(int index){
  //set 8 high bits of GPIOB's output data register PB8 -
   PB15 to according buffer (ROW7 -> ROW0)
  GPIOB->ODR = (GPIOB->ODR & 0x00FF) | (matrix_buffer[index
   ] << 8);
  //turn each column on one-by-one
  switch (index){
    case 0:
      //display 1st column
      HAL_GPIO_WritePin(GPIOA, ENM1_Pin|ENM2_Pin|ENM3_Pin
                  |ENM4_Pin|ENM5_Pin|ENM6_Pin
                  |ENM7_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(ENM0_GPIO_Port, ENM0_Pin,
   GPIO_PIN_RESET);
      break;
    case 1:
      //display 2nd column
      HAL_GPIO_WritePin(GPIOA, ENM0_Pin|ENM2_Pin|ENM3_Pin
                  |ENM4_Pin|ENM5_Pin|ENM6_Pin
                  |ENM7_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(ENM1_GPIO_Port, ENM1_Pin,
   GPIO_PIN_RESET);
      break;
    case 2:
      //display 3rd column
      HAL_GPIO_WritePin(GPIOA, ENM1_Pin|ENM0_Pin|ENM3_Pin
                  |ENM4_Pin|ENM5_Pin|ENM6_Pin
                  |ENM7_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(ENM2_GPIO_Port, ENM2_Pin,
   GPIO_PIN_RESET);
      break;
    case 3:
      //display 4th column
      HAL_GPIO_WritePin(GPIOA, ENM1_Pin|ENM2_Pin|ENM0_Pin
                  |ENM4_Pin|ENM5_Pin|ENM6_Pin
                  |ENM7_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(ENM3_GPIO_Port, ENM3_Pin,
   GPIO_PIN_RESET);
      break;
    case 4:
      //display 5th column
      HAL_GPIO_WritePin(GPIOA, ENM1_Pin|ENM2_Pin|ENM3_Pin
                  |ENM0_Pin|ENM5_Pin|ENM6_Pin
                  |ENM7_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(ENM4_GPIO_Port, ENM4_Pin,
   GPIO_PIN_RESET);
      break;
    case 5:
      //display 6th column
      HAL_GPIO_WritePin(GPIOA, ENM1_Pin|ENM2_Pin|ENM3_Pin
                  |ENM4_Pin|ENM0_Pin|ENM6_Pin
                  |ENM7_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(ENM5_GPIO_Port, ENM5_Pin,
   GPIO_PIN_RESET);
      break;
    case 6:
      //display 7th column
      HAL_GPIO_WritePin(GPIOA, ENM1_Pin|ENM2_Pin|ENM3_Pin
                  |ENM4_Pin|ENM5_Pin|ENM0_Pin
                  |ENM7_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(ENM6_GPIO_Port, ENM6_Pin,
   GPIO_PIN_RESET);
      break;
    case 7:
      //display 8th column
      HAL_GPIO_WritePin(GPIOA, ENM1_Pin|ENM2_Pin|ENM3_Pin
                  |ENM4_Pin|ENM5_Pin|ENM6_Pin
                  |ENM0_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(ENM7_GPIO_Port, ENM7_Pin,
   GPIO_PIN_RESET);
      break;
    default:
      break;
  }
}
```

Program 1.12: 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"**.

3.10 Exercise 10

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

Report 1: Briefly describe your solution and present your source code in the report.
Answer: Scan the led matrix similar to Exercise 9 but after scanning all 8 columns, we shift left the matrix_buffer by 1 index and so on.

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

Program 1.13: shiftLeftMatrixBuffer function

```
setTimerMaxtrix(10);
int scan_counter = 0;

while (1)
{
  if (timerMatrix_flag == 1){
    setTimerMaxtrix(50);
    //increase scan counter for each column scanned
    scan_counter++;
    index_led_matrix %= MAX_LED_MATRIX;
    updateLEDMatrix(index_led_matrix++);
  }

  if (scan_counter >= 8){
    //shift left matrix_buffer after scanning all 8 columns
    shiftLeftMatrixBuffer();
    //reset scan counter
    scan_counter = 0;
  }
}
```

Program 1.14: while loop