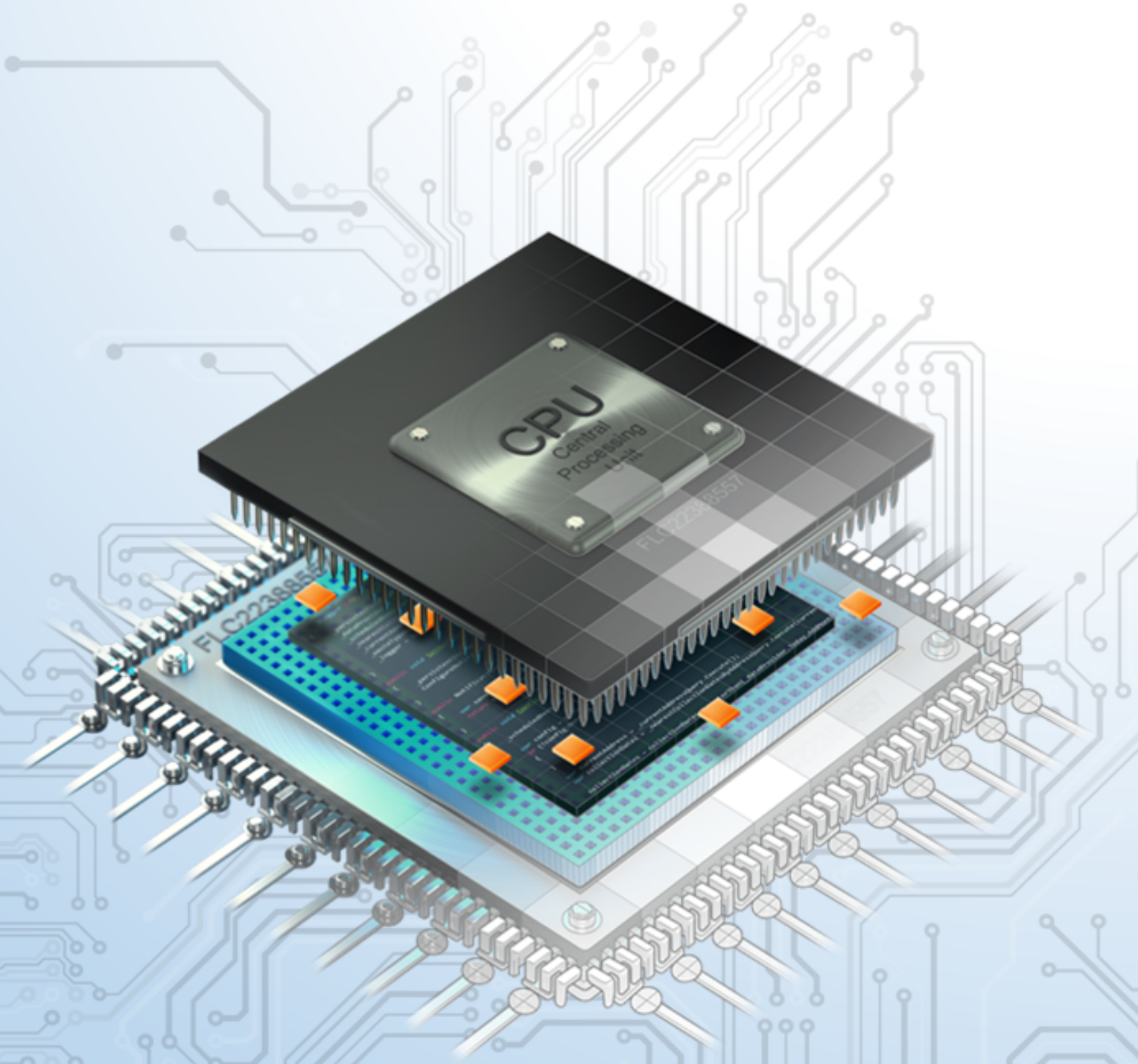
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

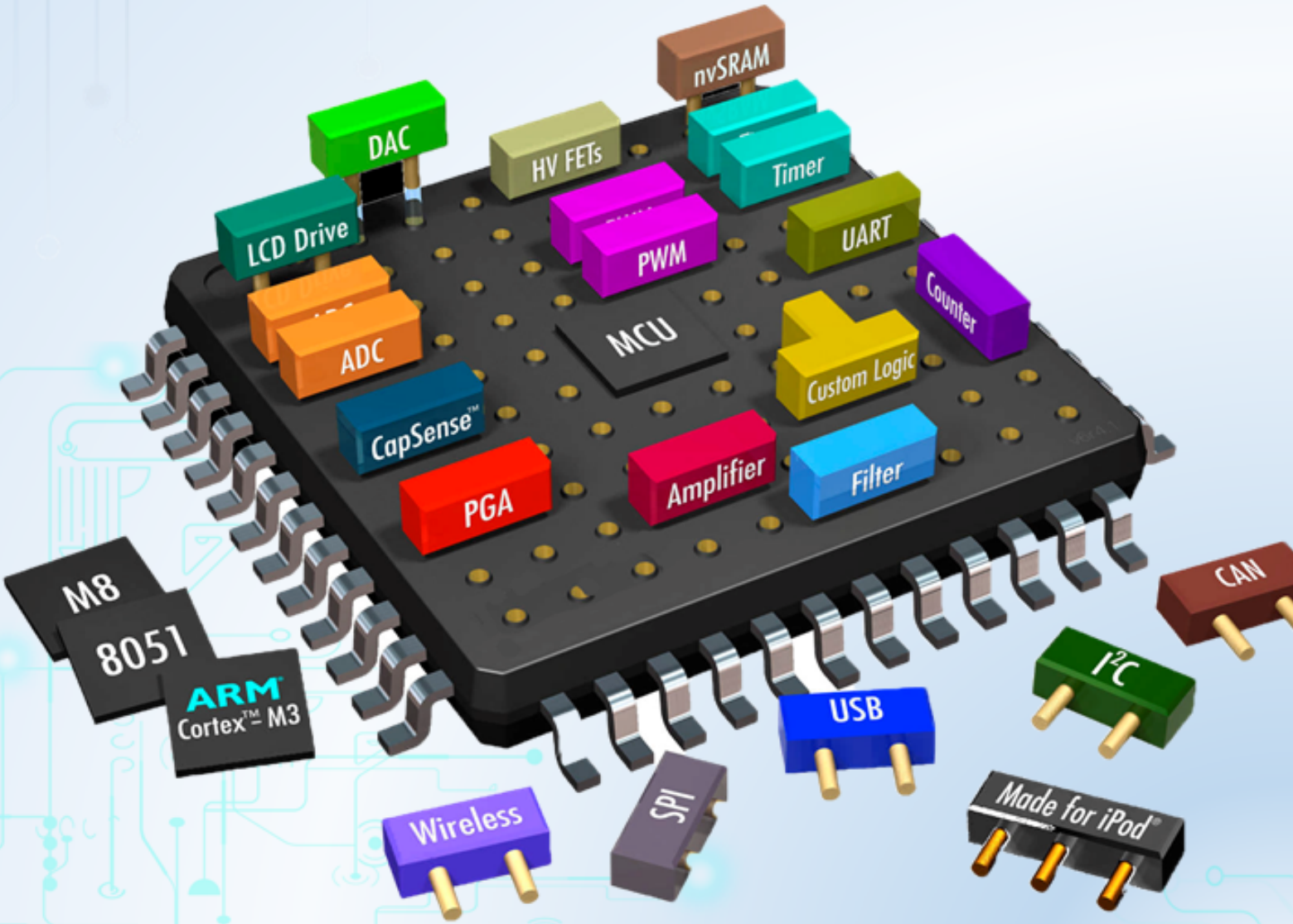


Dr. Le Trong Nhan

Mục lục

CHƯƠNG 1

LED Animations



1 Introduction

In this manual, the STM32CubeIDE is used as an editor to program the ARM microcontroller. STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors.

Hình 1.1: *STM32Cube IDE for STM32 Programming*

The most interest of STM32CubeIDE is that after the selection of an empty STM32 MCU or MPU, or preconfigured microcontroller or microprocessor from the selection of a board, the initialization code generated automatically. At any time during the development, the user can return to the initialization and configuration of the peripherals or middleware and regenerate the initialization code with no impact on the user code. This feature can simplify the initialization process and speedup the development application running on STM32 micro-controller. The software can be downloaded from the link bellow:

https://ubc.sgp1.digitaloceanspaces.com/BKU_Softwares/STM32/stm32cubeide_1.7.0.zip

Moreover, for a hangout class, the program is firstly simulated on Proteus. Students are also supposed to download and install this software as well:

https://ubc.sgp1.digitaloceanspaces.com/BKU_Softwares/STM32/Proteus_8.10_SP0_Pro.exe

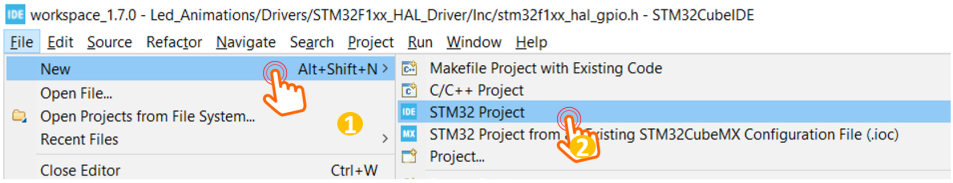
The rest of this manual consists of:

- Create a project on STM32Cube IDE
- Create a project on Proteus
- Simulate the project on Proteus

Finally, students are supposed to finish 10 different projects.

2 First project on STM32Cube

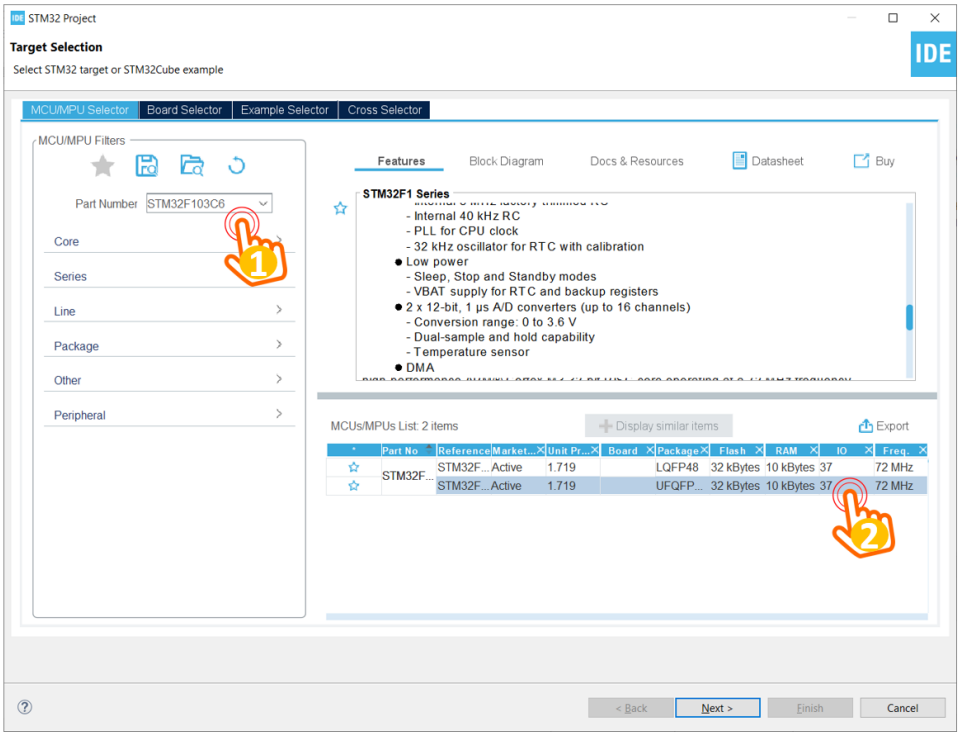
Step 1: Launch STM32CubeIDE, from the menu **File**, select **New**, then chose **STM32 Project**



Hình 1.2: Create a new project on STM32CubeIDE

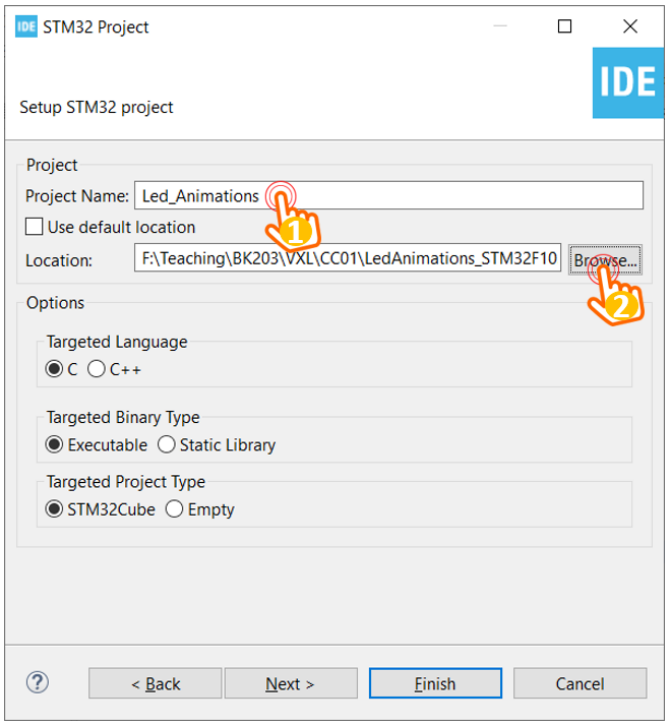
The IDE needs to download some packages, which normally takes time in this first time a project is created.

Step 2: Select the STM32F103C6 in the following dialog, then click on **Next**



Hình 1.3: Select the target device

Step 3: Provide the **Name** and the **Location** for the project.



Hình 1.4: Select the target device

It is important to notice that the **Targeted Project Type** should be **STM32Cube**. In the case this option is disable, step 1 must be repeated. The location path should not contain special characters (e.g. the space). Finally, click on the **Next** button.

Step 4: On the last dialog, just keep the default firmware version and click on **Finish** button.

Step 5: The project is created and the wizard for configuration is display. This utility from CubeIDE can simplify the configuration process for an ARM microcontroller like the STM32.

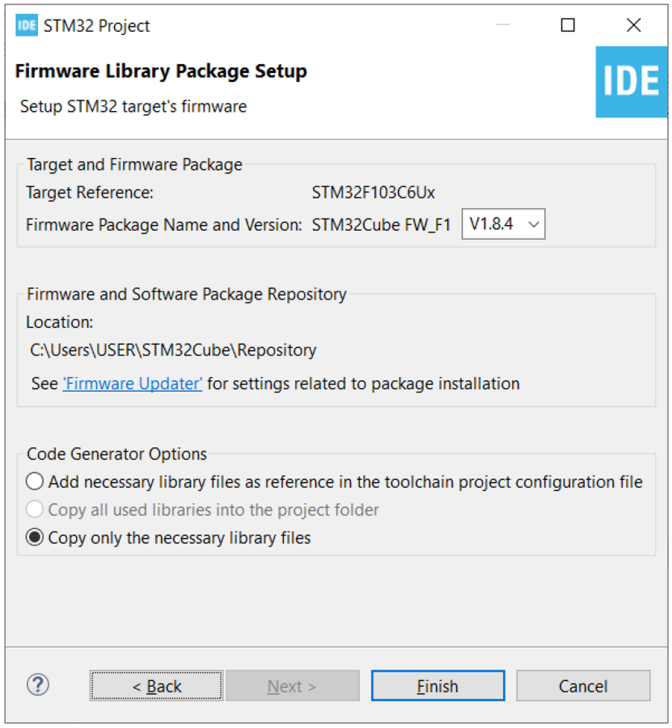
From the configuration windows, select **Pin configuration**, select the pin **PA5** and set to **GPIO Output** mode, since this pin is connected to an LED in the STM32 development kit.

Step 6: Right click on PA5 and select **Enter user lable**, and provide the name for this pin (e.g. **LED_RED**). This step helps programming afterward more memorable.

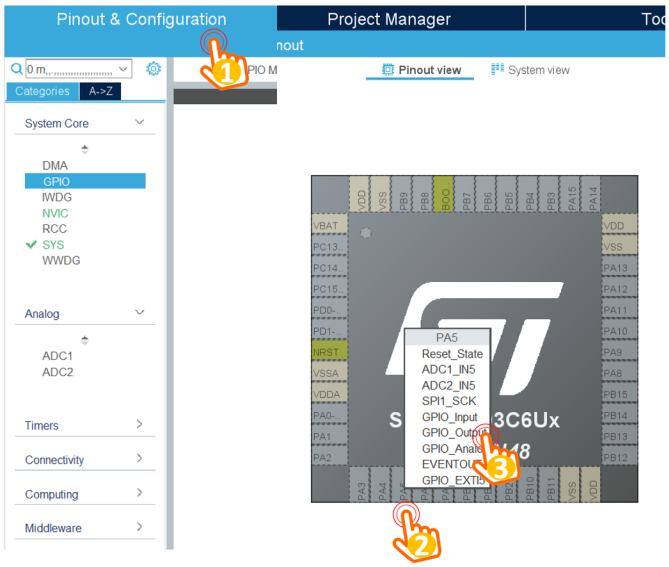
Finally, save the configuration process by pressing **Ctrl + S** and confirm this step by clicking on **OK** button. The code generation is started.

Step 7: Implement the first blinky project in the main function as follow:

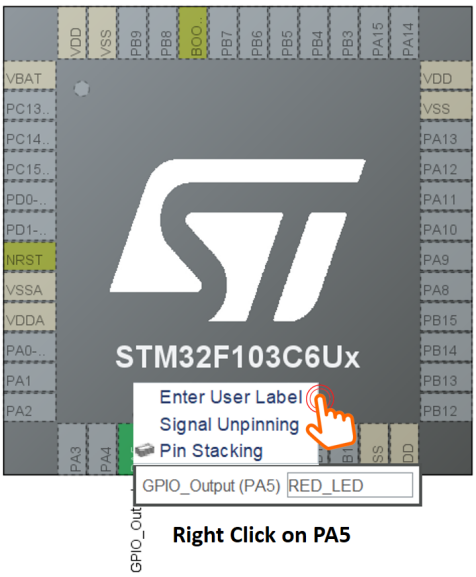
```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
```



Hình 1.5: Keep default firmware version



Hình 1.6: Set PA5 to GPIO Output mode



Hình 1.7: Provide a name for PA5

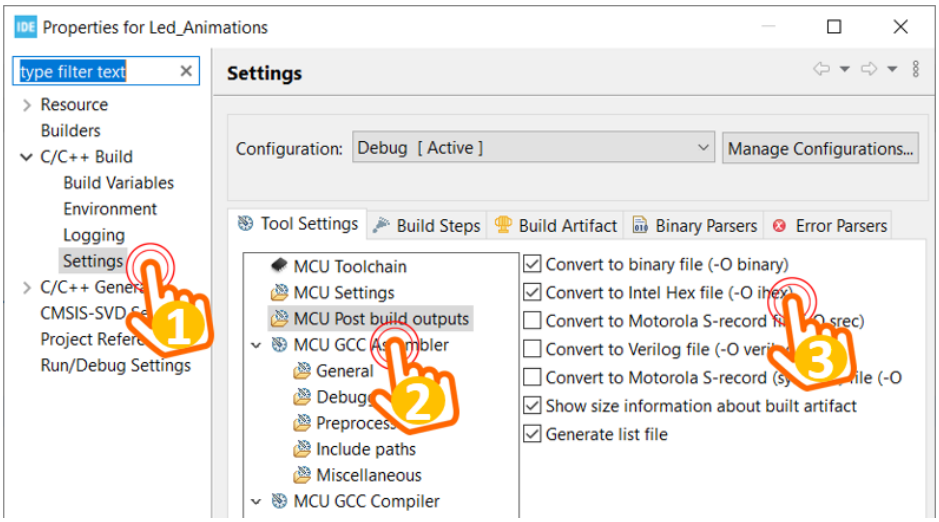
```
6
7   /* MCU Configuration
     ---------------------------------------------------------
     */
8
9   /* Reset of all peripherals, Initializes the Flash
     interface and the Systick. */
10  HAL_Init();
11
12  /* USER CODE BEGIN Init */
13
14  /* USER CODE END Init */
15
16  /* Configure the system clock */
17  SystemClock_Config();
18
19  /* USER CODE BEGIN SysInit */
20
21  /* USER CODE END SysInit */
22
23  /* Initialize all configured peripherals */
24  MX_GPIO_Init();
25  /* USER CODE BEGIN 2 */
26
27  /* USER CODE END 2 */
28
29  /* Infinite loop */
30  /* USER CODE BEGIN WHILE */
31
32  while (1)
33  {
```

```
34      HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
35      HAL_Delay(1000);
36      /* USER CODE END WHILE */
37
38      /* USER CODE BEGIN 3 */
39    }
40    /* USER CODE END 3 */
41  }
```

Program 1.1: First blinky LED project

Actually, what is added to the main function is line number 34 and 35. Please put your code in a right place, otherwise it can be deleted when the code is generated (e.g. change the configuration of the project). When coding, frequently use the suggestions by pressing **Ctrl+Space**.

Step 7: Due to the simulation on Proteus, the hex file should be generated from STM32Cube IDE. From menu **Project**, select **Properties** to open the dialog bellow:



Hình 1.8: Config for hex file output

Navigate to **C/C++ Build**, select **Settings, MCU Post build outputs**, and check to the **Intel Hex file**.

Step 8: Build the project by clicking on menu **Project** and select **Build Project**. Please check on the output console of the IDE to be sure that the hex file is generated, as follow:

```
22:36:06 **** Incremental Build of configuration Debug for project Led_Animations ****
make -j8 all
arm-none-eabi-size   Led_Animations.elf
   text    data     bss     dec     hex filename
   4596      20    1572    6188    182c Led_Animations.elf
Finished building: default.size.stdout


22:36:06 Build Finished. 0 errors, 0 warnings. (took 272ms)
```

Hình 1.9: Compile the project and generate Hex file

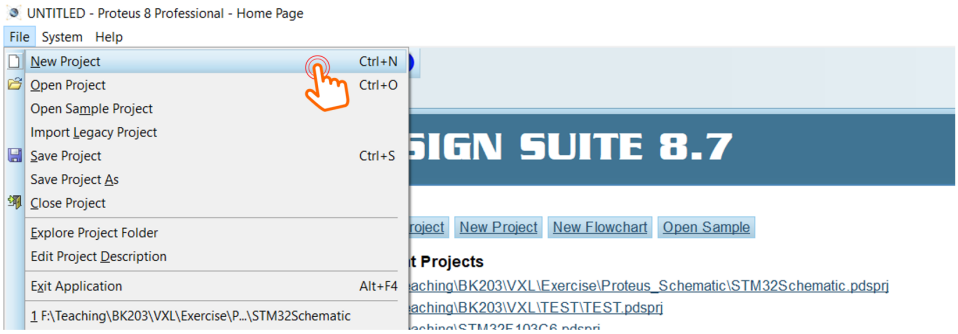
The hex file is located under the **Debug** folder of your project, which is used for the simulation in Proteus afterward. In the case a development kit is connected to your PC, from menu **Run**, select **Run** to download the program to the hardware platform.

In the case there are multiple project in a work-space, double click on the project name to activate this project. Whenever a project is built, check the output files to make sure that you are working in a right project.

3 Simulation on Proteus

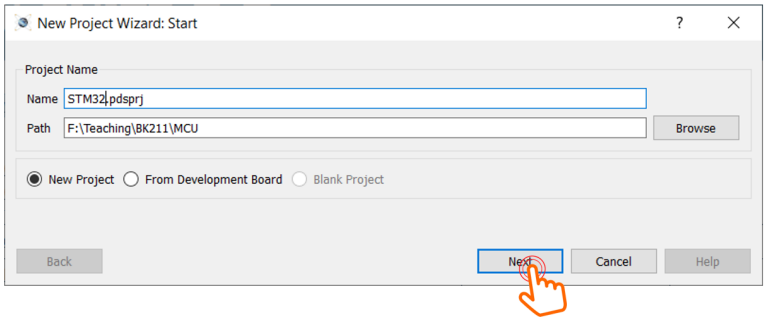
For an online training, a simulation on Proteus can be used. The details to create an STM32 project on Proteus are described bellow.

Step 1: Launch Proteus (**with administration access**) and from menu **File**, select **New Project**.



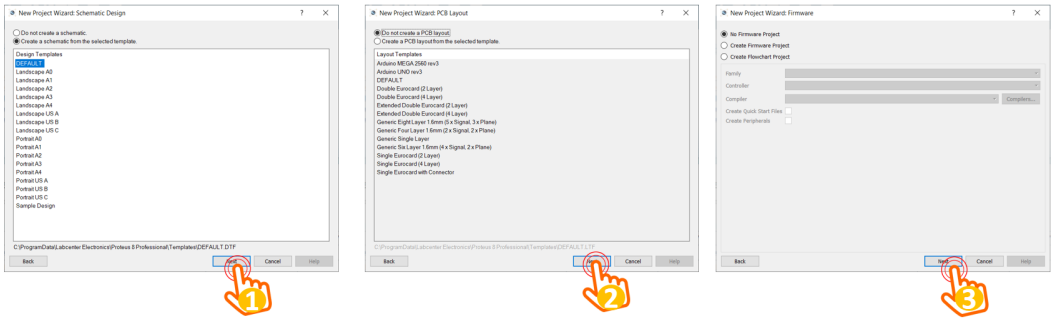
Hình 1.10: Create a new project on Proteus

Step 2: Provide the name and the location of the project, then click on **Next** button.



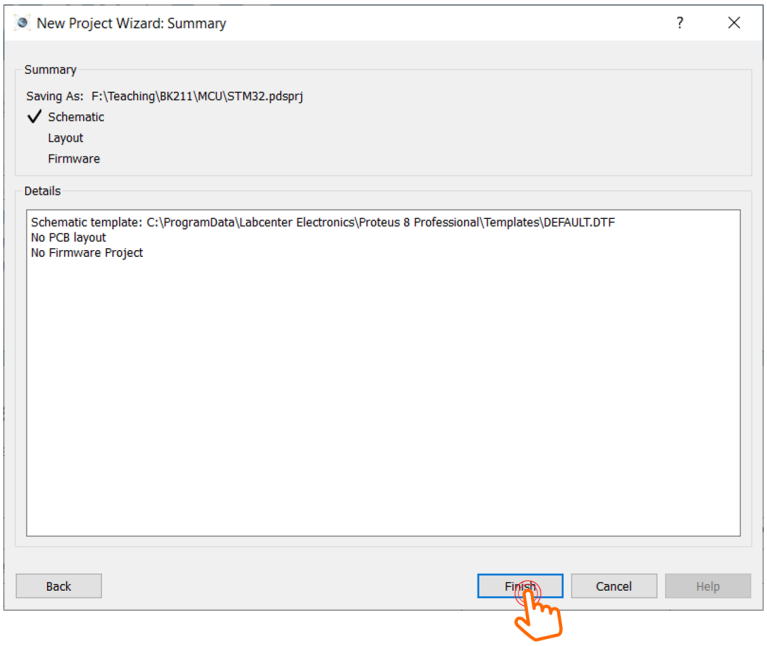
Hình 1.11: Provide project name and location

Step 3: For following dialog, just click on **Next** button as just a schematic is required for the lab.



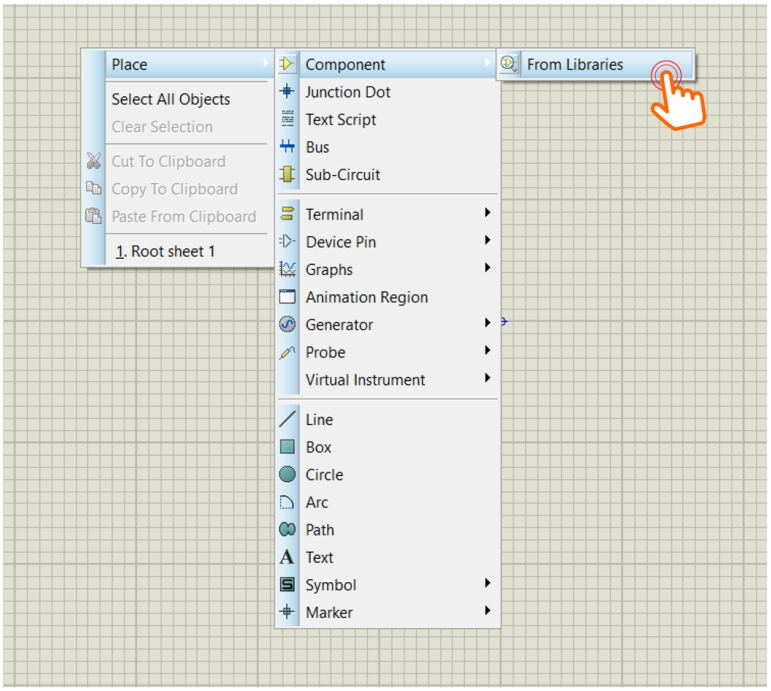
Hình 1.12: Keep the default options by clicking on Next

Step 4: Finally, click on **Finish** button to close the project wizard.



Hình 1.13: Finish the project wizard

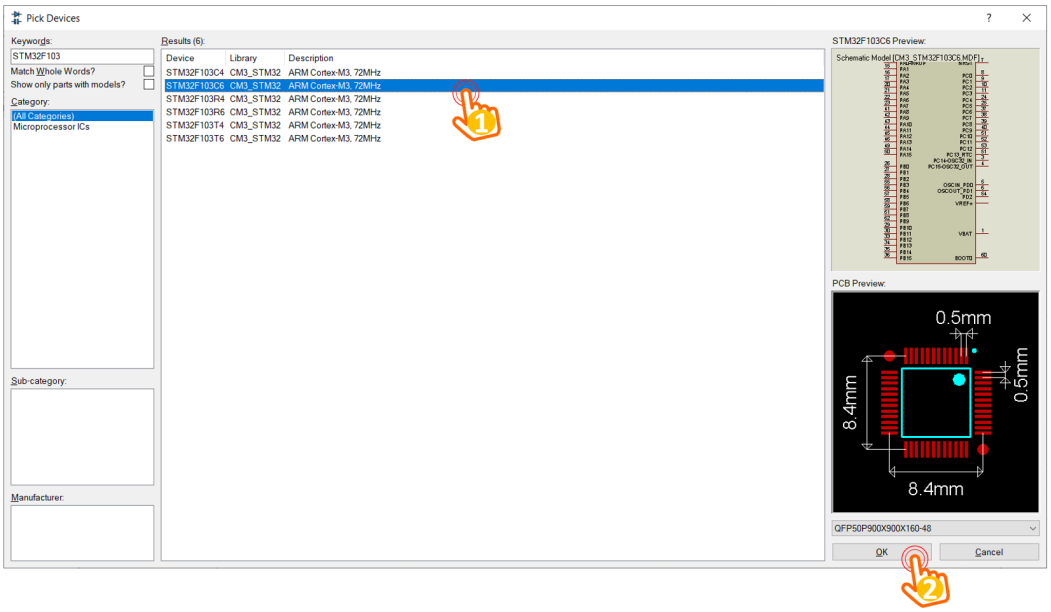
Step 5: On the main page of the project, right click to select **Place, Components, From Libraries**, as follows:



Hình 1.14: Select a component from the library

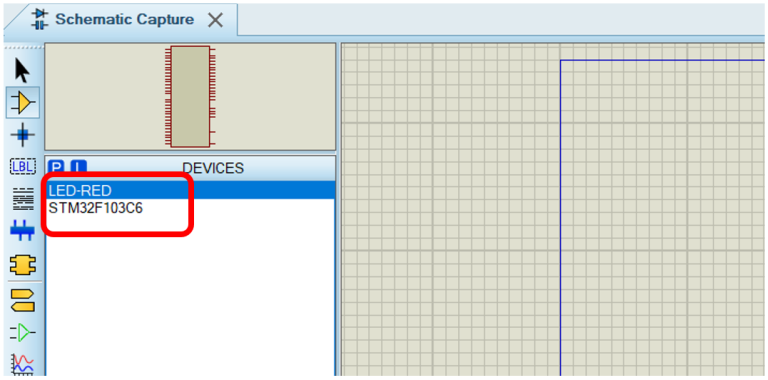
If there is an error with no library found, please restart the Proteus software with Run as administrator option.

Step 6: From the list of components in the library, select STM32F103C6, as follows:



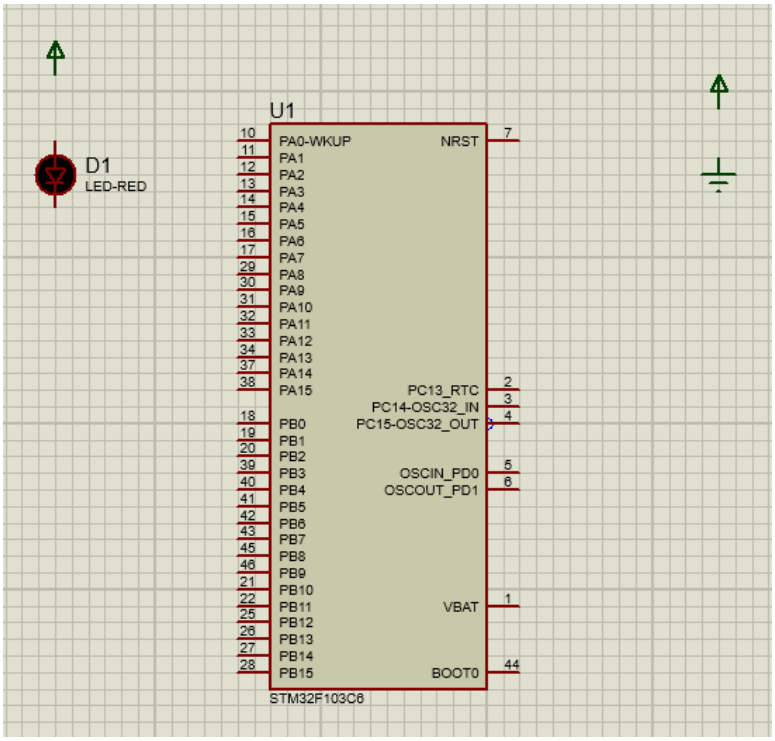
Hình 1.15: Select STM32F103C6

Repeat step 5 and 6 to select an LED, named **LED-RED** in Proteus. Finally, these components are appeared on the DEVICES windows, which is on left hand side as follows:



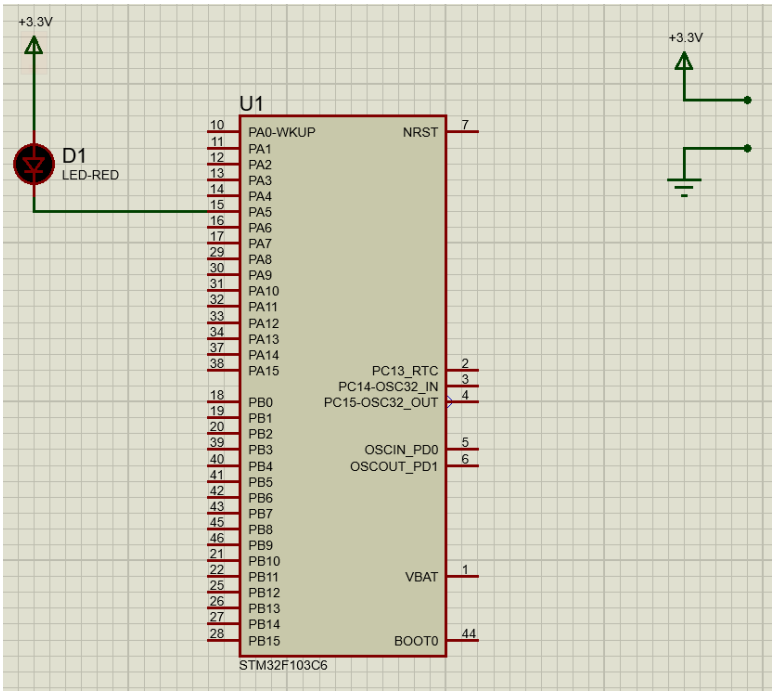
Hình 1.16: STM32 and an LED in the project

Step 7: Place the components to the project: right click on the main page, select on **Place, Component**, and select device added in Step 6. To add the Power and the Ground, right click on the main page, select on **Place, Terminal**. The result in this step is expected as follows:



Hình 1.17: *Place components to the project*

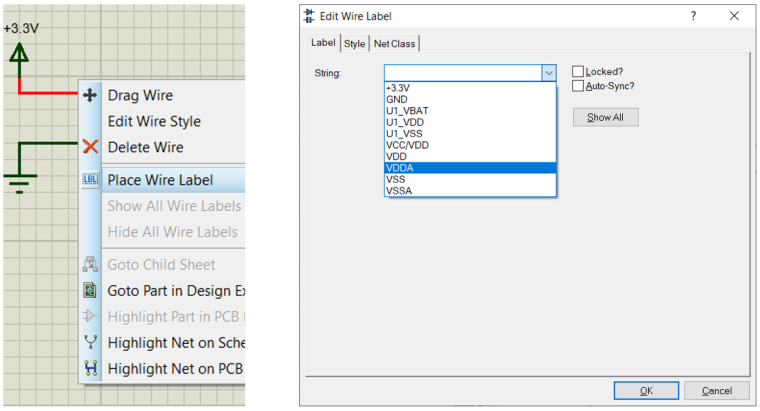
Step 8: Start wiring the circuit. The negative pin of the LED is connected to PA5 while its positive pin is connected to the power supply. For the power and the ground on the right, just make a short wire, which will labeled in the next step.



Hình 1.18: *Connect components and set the power to 3.3V*

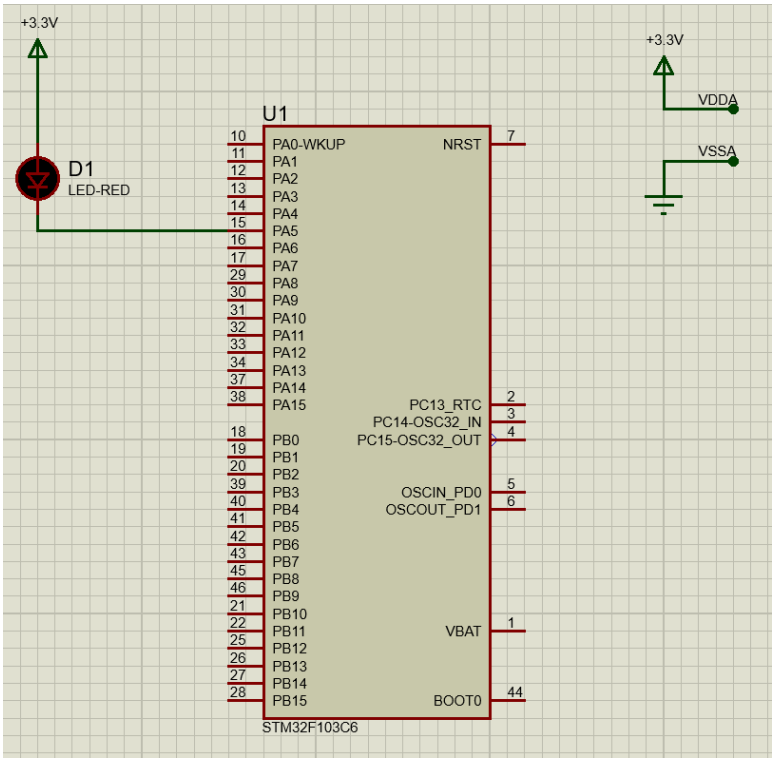
In this step, also double click on the power supply in order to provide the String property to **+3.3V**.

Step 8: Right click on the wire of the power supply and the ground, and select **Place wire Label**



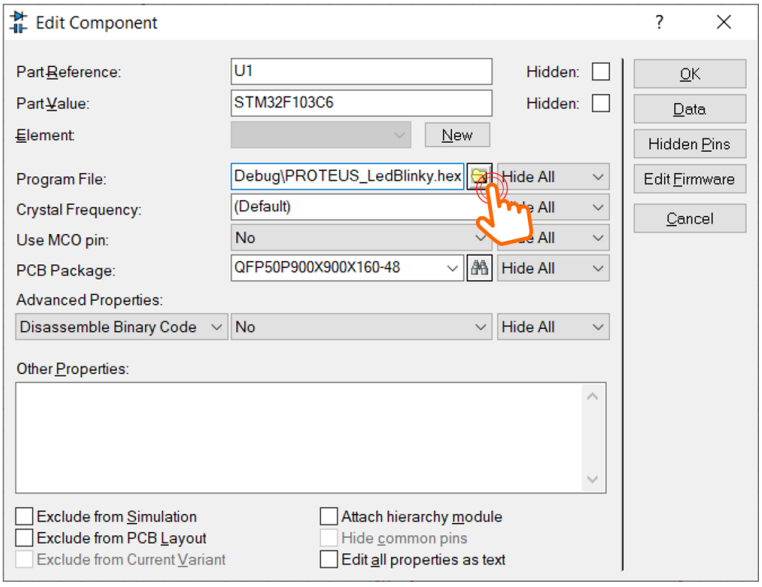
Hình 1.19: *Place label for Power and Ground*

This step is required as VDDA and VSSA of the STM32 must be connected to provide the reference voltage. Therefore, VDDA is connected to 3.3V, while the VSSA is connected to the Ground. Finally, the image of our schematic is shown bellow:



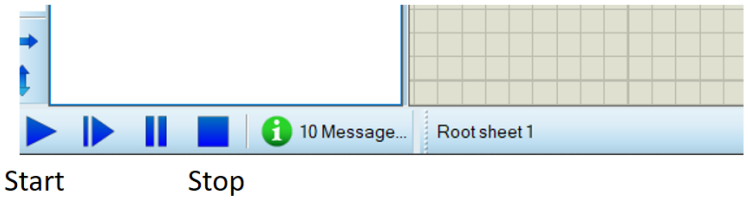
Hình 1.20: *Finalize the schematic*

Step 9: Double click on the STM32, and set the **Program File** to the Hex file, which is generated from Cube IDE, as following:



Hình 1.21: *Set the program of the STM32 to the hex file from Cube IDE*

From now, the simulation is ready to start by clicking on the menu **Debug**, and select on **Run simulation**. To stop the simulation, click on **Debug** and select **Stop VMS Debugging**. Moreover, there are some quick access bottom on the left corner of the Proteus to start or stop the simulation, as shown following:



Hình 1.22: *Quick access buttons to start and stop the simulation*

If everything is success, students can see the LED is blinking every second. Please stop the simulation before updating the project, either in Proteus or STM32Cube IDE. However, the step 9 (set the program file for STM32 in Proteus) is required to do once. Beside the toggle instruction, student can set or reset a pin as following:

```
while (1){
  HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin,
   GPIO_PIN_SET);
  HAL_Delay(1000);
  HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin,
   GPIO_PIN_RESET);
  HAL_Delay(1000);
}
```

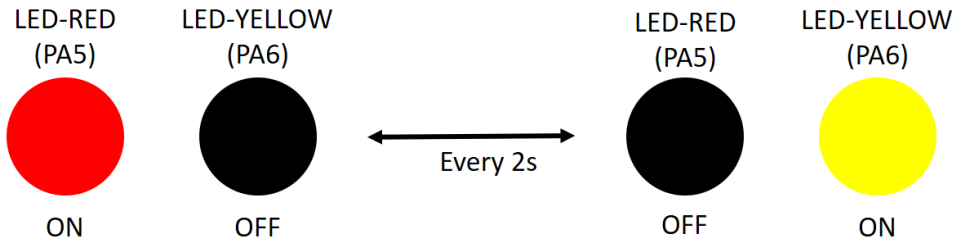
Program 1.2: An example for LED blinky

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 1.23: *State transitions for 2 LEDs*

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

github link: https://github.com/dang-khoa-2k4/MP-MC

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

```
void ex_1_handle()
{
    // Khoi tao bien start de chac chan 2 led sang luan
   phien
    static uint8_t start = 1;
    static uint32_t count_seconds = 0;
    if (start)
    {
        HAL_GPIO_WritePin(GPIOA, LED_RED_Pin, LED_ON);
        start = 0;
    }
    if (count_seconds % 2 == 0)
    {
        HAL_GPIO_TogglePin(GPIOA, LED_RED_Pin);
        HAL_GPIO_TogglePin(GPIOA, LED_YELLOW_Pin);
    }
    count_seconds++;
```

```
}
```

Program 1.3: Exercise 1 handle function

```
while (1)
  {
    ex_1_handle();
    HAL_Delay(1000);

    /* USER CODE END WHILE */

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

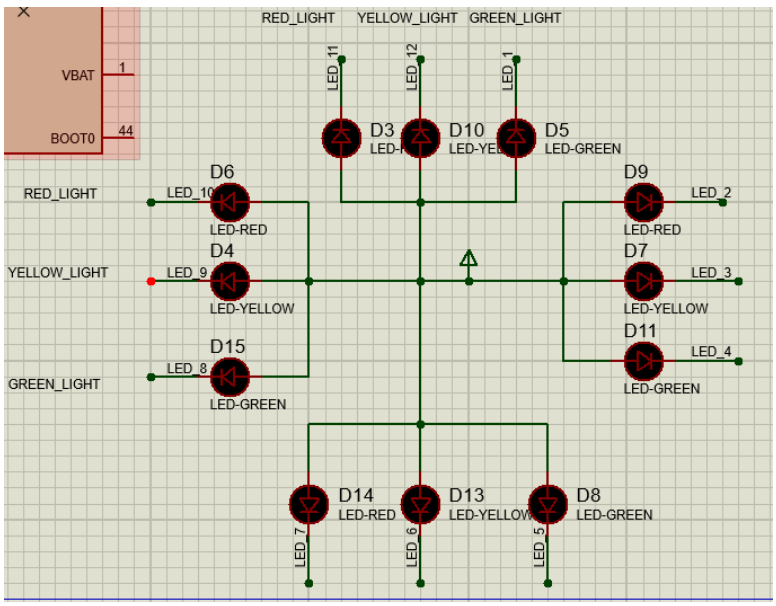
Program 1.4: In while loop

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.

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

Report 1: Present the schematic.



Hình 1.24: The schematic for ex 2

Report 2: Present the source code in while.

```c
static void traffic_light_handle(TrafficLight * light_state
    , uint16_t *leds, uint32_t * count_seconds)
{
     HAL_GPIO_WritePin(GPIOB, leds[0] | leds[1] | leds[2],
    1);
     switch (*light_state)
     {
     case RED_LIGHT:
         HAL_GPIO_WritePin(GPIOB, leds[*light_state], 0);
         if (*count_seconds >= TRAFFIC_LIGHT_TIME[*
    light_state])
         {
             *light_state = GREEN_LIGHT;
             *count_seconds = 0;
         }
         break;
     case YELLOW_LIGHT:
         HAL_GPIO_WritePin(GPIOB, leds[*light_state], 0);
         if (*count_seconds >= TRAFFIC_LIGHT_TIME[*
    light_state])
         {
             *light_state = RED_LIGHT;
             *count_seconds = 0;
         }
         break;
     case GREEN_LIGHT:
         HAL_GPIO_WritePin(GPIOB, leds[*light_state], 0);
         if (*count_seconds >= TRAFFIC_LIGHT_TIME[*
    light_state])
         {
             *light_state = YELLOW_LIGHT;
             *count_seconds = 0;
         }
         break;
     default:
         break;
     }
}
```

Program 1.5: Handle Traffic light function

```c
while (1)
  {
    ex_2_handle();
    HAL_Delay(1000);

    /* USER CODE END WHILE */

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

```
10
11 void ex_2_handle()
12 {
13     static TrafficLight light_state = RED_LIGHT;
14     static uint32_t count_seconds = 0;
15     static uint16_t leds[3] = {LED_8_Pin, LED_9_Pin,
   LED_10_Pin};
16     traffic_light_handle(&light_state, leds, &count_seconds
   );
17     count_seconds++;
18 }
```

Program 1.6: In while loop

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.

```
1 static void traffic_light_handle(TrafficLight * light_state
   , uint16_t *leds, uint32_t * count_seconds)
2 {
3     HAL_GPIO_WritePin(GPIOB, leds[0] | leds[1] | leds[2],
   1);
4     switch (*light_state)
5     {
6     case RED_LIGHT:
7         HAL_GPIO_WritePin(GPIOB, leds[*light_state], 0);
8         if (*count_seconds >= TRAFFIC_LIGHT_TIME[*
   light_state])
9         {
10             *light_state = GREEN_LIGHT;
11             *count_seconds = 0;
12         }
13         break;
14     case YELLOW_LIGHT:
15         HAL_GPIO_WritePin(GPIOB, leds[*light_state], 0);
16         if (*count_seconds >= TRAFFIC_LIGHT_TIME[*
   light_state])
17         {
18             *light_state = RED_LIGHT;
19             *count_seconds = 0;
20         }
21         break;
22     case GREEN_LIGHT:
23         HAL_GPIO_WritePin(GPIOB, leds[*light_state], 0);
24         if (*count_seconds >= TRAFFIC_LIGHT_TIME[*
   light_state])
25         {
```

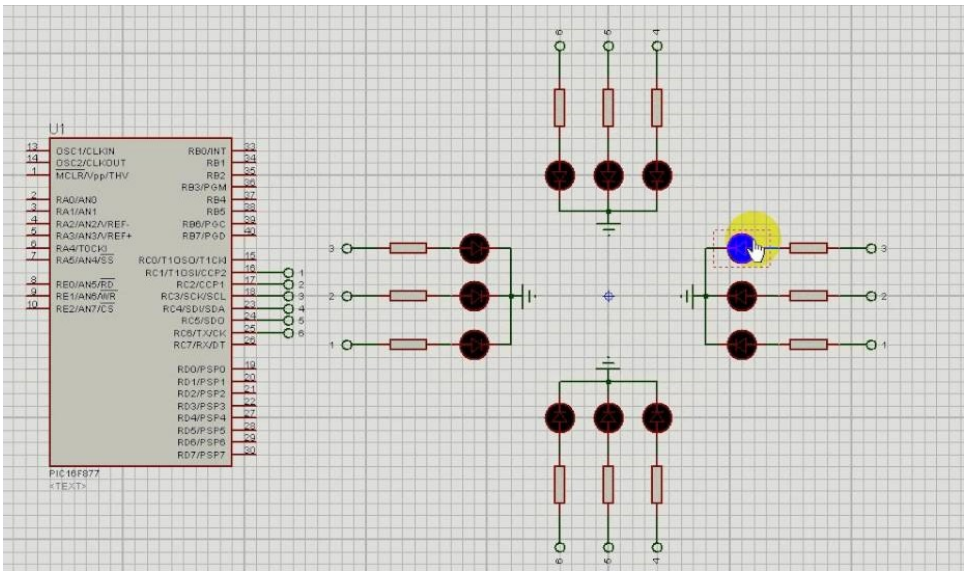
```
            *light_state = YELLOW_LIGHT;
            *count_seconds = 0;
        }
        break;
    default:
        break;
    }
}
```

Program 1.7: Handle traffic light function

```
while (1)
  {
    ex_3_handle();
    HAL_Delay(1000);

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }

void ex_3_handle()
{
    static uint32_t count_seconds_1 = 0;
    static uint32_t count_seconds_2 = 0;
    static uint32_t count_seconds_3 = 0;
    static uint32_t count_seconds_4 = 0;
    static TrafficLight traffic_light_1 = RED_LIGHT;
    static TrafficLight traffic_light_2 = GREEN_LIGHT;
    static TrafficLight traffic_light_3 = RED_LIGHT;
    static TrafficLight traffic_light_4 = GREEN_LIGHT;
    static uint16_t leds_1[3] = {LED_11_Pin, LED_12_Pin,
   LED_1_Pin};
    static uint16_t leds_2[3] = {LED_2_Pin, LED_3_Pin,
   LED_4_Pin};
    static uint16_t leds_3[3] = {LED_7_Pin, LED_6_Pin,
   LED_5_Pin};
    static uint16_t leds_4[3] = {LED_10_Pin, LED_9_Pin,
   LED_8_Pin};

    traffic_light_handle(&traffic_light_1, leds_1, &
   count_seconds_1);
    traffic_light_handle(&traffic_light_2, leds_2, &
   count_seconds_2);
    traffic_light_handle(&traffic_light_3, leds_3, &
   count_seconds_3);
    traffic_light_handle(&traffic_light_4, leds_4, &
   count_seconds_4);
    count_seconds_1++;
    count_seconds_2++;
```

```
32      count_seconds_3++;
33      count_seconds_4++;
34  }
```

Program 1.8: In while loop

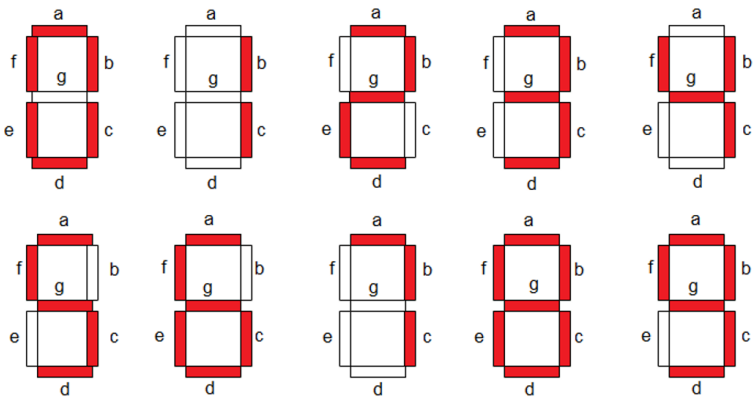


Hình 1.25: *Reference design for a 4 way traffic light*

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).

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



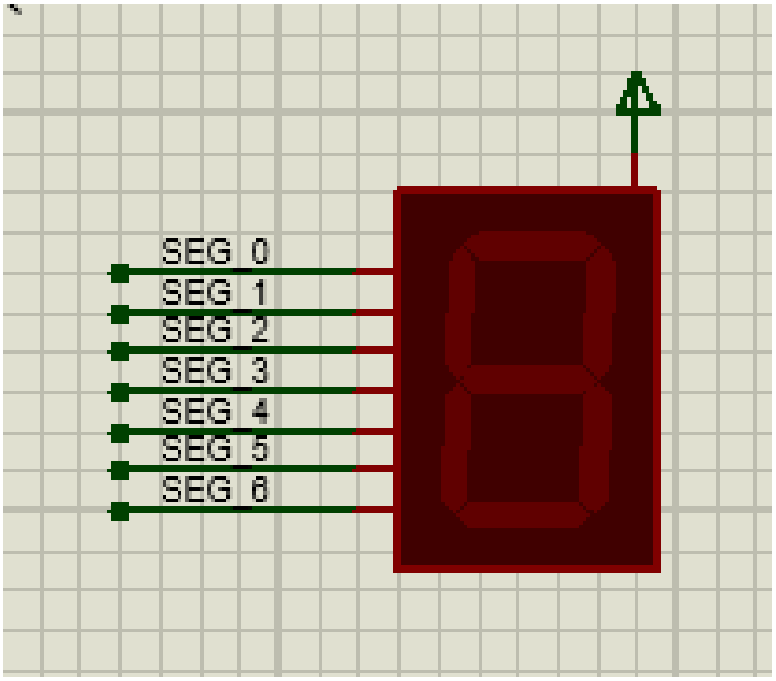
Hình 1.26: *Display a number on 7 segment LED*

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

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

}
```

Program 1.9: An example for your source code

Report 1: Present the schematic.



Hình 1.27: The schematic for 7 segment LED

Report 2: Present the source code for display7SEG function.

```
void display7Seg(uint8_t num)
{
    HAL_GPIO_WritePin(GPIOA, ALL_SEG, LED_OFF);
    switch (num)
    {
    case 0:
        HAL_GPIO_WritePin(GPIOA, NUM_0, LED_ON);
        break;
    case 1:
        HAL_GPIO_WritePin(GPIOA, NUM_1, LED_ON);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOA, NUM_2, LED_ON);
        break;
```

```
    case 3:
        HAL_GPIO_WritePin(GPIOA, NUM_3, LED_ON);
        break;
    case 4:
        HAL_GPIO_WritePin(GPIOA, NUM_4, LED_ON);
        break;
    case 5:
        HAL_GPIO_WritePin(GPIOA, NUM_5, LED_ON);
        break;
    case 6:
        HAL_GPIO_WritePin(GPIOA, NUM_6, LED_ON);
        break;
    case 7:
        HAL_GPIO_WritePin(GPIOA, NUM_7, LED_ON);
        break;
    case 8:
        HAL_GPIO_WritePin(GPIOA, NUM_8, LED_ON);
        break;
    case 9:
        HAL_GPIO_WritePin(GPIOA, NUM_9, LED_ON);
        break;
    default:
        break;
    }
}
```

Program 1.10: display7SEG function

```
// define for 7 segment
#define NUM_0   (SEG_0_Pin | SEG_1_Pin | SEG_2_Pin | SEG_3_Pin | SEG_4_Pin | SEG_5_Pin)
#define NUM_1   (SEG_1_Pin | SEG_2_Pin)
#define NUM_2   (SEG_0_Pin | SEG_1_Pin | SEG_3_Pin | SEG_4_Pin | SEG_6_Pin)
#define NUM_3   (SEG_0_Pin | SEG_1_Pin | SEG_2_Pin | SEG_3_Pin | SEG_6_Pin)
#define NUM_4   (SEG_1_Pin | SEG_2_Pin | SEG_5_Pin | SEG_6_Pin)
#define NUM_5   (SEG_0_Pin | SEG_2_Pin | SEG_3_Pin | SEG_5_Pin | SEG_6_Pin)
#define NUM_6   (SEG_0_Pin | SEG_2_Pin | SEG_3_Pin | SEG_4_Pin | SEG_5_Pin | SEG_6_Pin)
#define NUM_7   (SEG_0_Pin | SEG_1_Pin | SEG_2_Pin)
#define NUM_8   (SEG_0_Pin | SEG_1_Pin | SEG_2_Pin | SEG_3_Pin | SEG_4_Pin | SEG_5_Pin | SEG_6_Pin)
#define NUM_9   (SEG_0_Pin | SEG_1_Pin | SEG_2_Pin | SEG_3_Pin | SEG_5_Pin | SEG_6_Pin)
#define ALL_SEG (SEG_0_Pin | SEG_1_Pin | SEG_2_Pin | SEG_3_Pin | SEG_4_Pin | SEG_5_Pin | SEG_6_Pin)
```

Program 1.11: Full define was be used in display7Seg function

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.

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

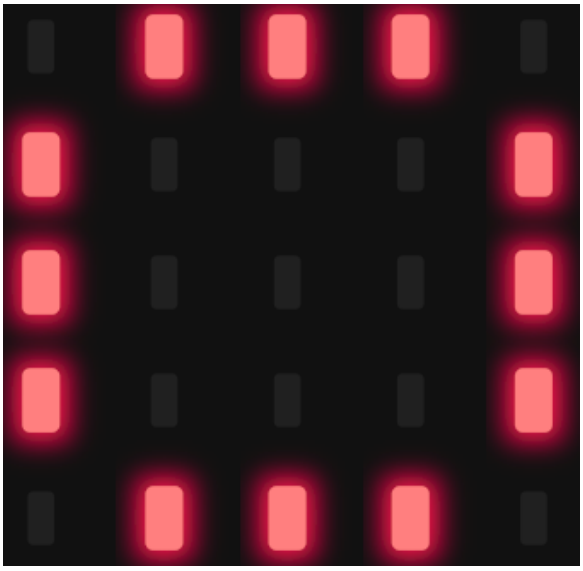
```
void ex_5_handle()
{
    static uint32_t count_seconds_1 = 0;
    static uint32_t count_seconds_2 = 0;
    static uint32_t count_seconds_3 = 0;
    static uint32_t count_seconds_4 = 0;
    static TrafficLight traffic_light_1 = RED_LIGHT;
    static TrafficLight traffic_light_2 = GREEN_LIGHT;
    static TrafficLight traffic_light_3 = RED_LIGHT;
    static TrafficLight traffic_light_4 = GREEN_LIGHT;
    static uint16_t leds_1[3] = {LED_11_Pin, LED_12_Pin,
  LED_1_Pin};
    static uint16_t leds_3[3] = {LED_7_Pin, LED_6_Pin,
  LED_5_Pin};
    static uint16_t leds_2[3] = {LED_2_Pin, LED_3_Pin,
  LED_4_Pin};
    static uint16_t leds_4[3] = {LED_10_Pin, LED_9_Pin,
  LED_8_Pin};

     HAL_GPIO_WritePin(GPIOB,   LED_12_Pin | LED_1_Pin |
  LED_2_Pin | LED_3_Pin |
                               LED_4_Pin  | LED_5_Pin |
  LED_6_Pin | LED_7_Pin |
                               LED_8_Pin  | LED_9_Pin |
  LED_10_Pin| LED_11_Pin, 0);

    display7Seg((TRAFFIC_LIGHT_TIME[traffic_light_1] -
  count_seconds_1) % TRAFFIC_LIGHT_TIME[traffic_light_1]);
    traffic_light_handle(&traffic_light_1, leds_1, &
  count_seconds_1);
    traffic_light_handle(&traffic_light_2, leds_2, &
  count_seconds_2);
    traffic_light_handle(&traffic_light_3, leds_3, &
  count_seconds_3);
    traffic_light_handle(&traffic_light_4, leds_4, &
  count_seconds_4);
    count_seconds_1++;
    count_seconds_2++;
    count_seconds_3++;
    count_seconds_4++;
```

```
29 }
```

Program 1.12: Exercise 5 handle

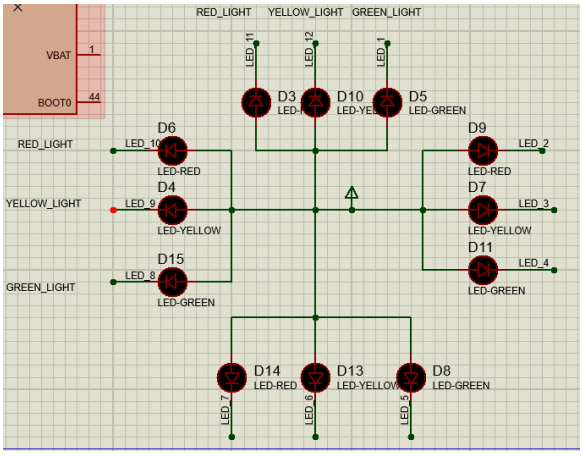
4.6 Exercise 6

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. The arrangement of 12 LEDs is depicted as follows.



Hình 1.28: 12 LEDs for an analog clock

Report 1: Present the schematic.



Hình 1.29: 12 LEDs for an analog clock schematic

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

```
static uint16_t TIME_CLOCK_LED[12] = {   LED_12_Pin,
   LED_1_Pin, LED_2_Pin , LED_3_Pin,
                                        LED_4_Pin ,
   LED_5_Pin, LED_6_Pin , LED_7_Pin,
                                        LED_8_Pin ,
   LED_9_Pin, LED_10_Pin, LED_11_Pin   };
// turn every LED in a sequence
void sequence_leds(uint8_t *index)
{
     HAL_GPIO_WritePin(GPIOB,   LED_12_Pin | LED_1_Pin |
   LED_2_Pin | LED_3_Pin |
                                LED_4_Pin  | LED_5_Pin |
   LED_6_Pin | LED_7_Pin |
                                LED_8_Pin  | LED_9_Pin |
   LED_10_Pin| LED_11_Pin, 0);
    HAL_GPIO_WritePin(GPIOB, TIME_CLOCK_LED[*index], 1);
    *index = (*index + 1) % 12;
}

void ex_6_handle()
{
  // turn every LED in a sequence
    static uint8_t index = 0;
    sequence_leds(&index);
}
```

Program 1.13: sequence turn on led handle

4.7 Exercise 7

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

```
void clearAllClock(){
        HAL_GPIO_WritePin(GPIOB, LED_1_Pin |LED_2_Pin|
   LED_3_Pin| LED_11_Pin
                           |LED_12_Pin|LED_4_Pin|
   LED_5_Pin| LED_6_Pin
                           |LED_7_Pin |LED_8_Pin|
   LED_9_Pin| LED_10_Pin, 1);
}
```

Program 1.14: Clear All Clock Function

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. Present the source code of this function.

```c
void setNumberOnClock(int num)
{
    HAL_GPIO_WritePin(GPIOB, TIME_CLOCK_LED[num % 12], 0);
}
```

Program 1.15: Set Number On Clock

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.

```c
void clearNumberOnClock(int num)
{
    HAL_GPIO_WritePin(GPIOB, TIME_CLOCK_LED[num % 12], 1);
}
```

Program 1.16: clear Number On Clock Function

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.

```c
while (1)
  {
    ex_10_handle();
    HAL_Delay(1000);

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }

void ex_10_handle()
{
    static uint8_t hour = 0;
    static uint8_t minute = 0;
    static uint8_t second = 0;
    if (hour == 0 && minute == 0 && second == 0)
        clearAllClock();
   if (second >= 60)
    {
        second = 0;
        clearNumberOnClock(minute);
        minute++;
        if (minute >= 60)
        {
            minute = 0;
```

```
26              clearNumberOnClock(hour);
27              hour++;
28              if (hour >= 24)
29              {
30                  hour = 0;
31              }
32          }
33      }
34      clearNumberOnClock(second - 1 + 12);
35
36      setNumberOnClock(second);
37      setNumberOnClock(minute);
38      setNumberOnClock(hour);
39
40      second++;
41  }
```

Program 1.17: Clock simulation

All source code and simulation files can be found at: https://github.com/dang-khoa-2k4/MP-