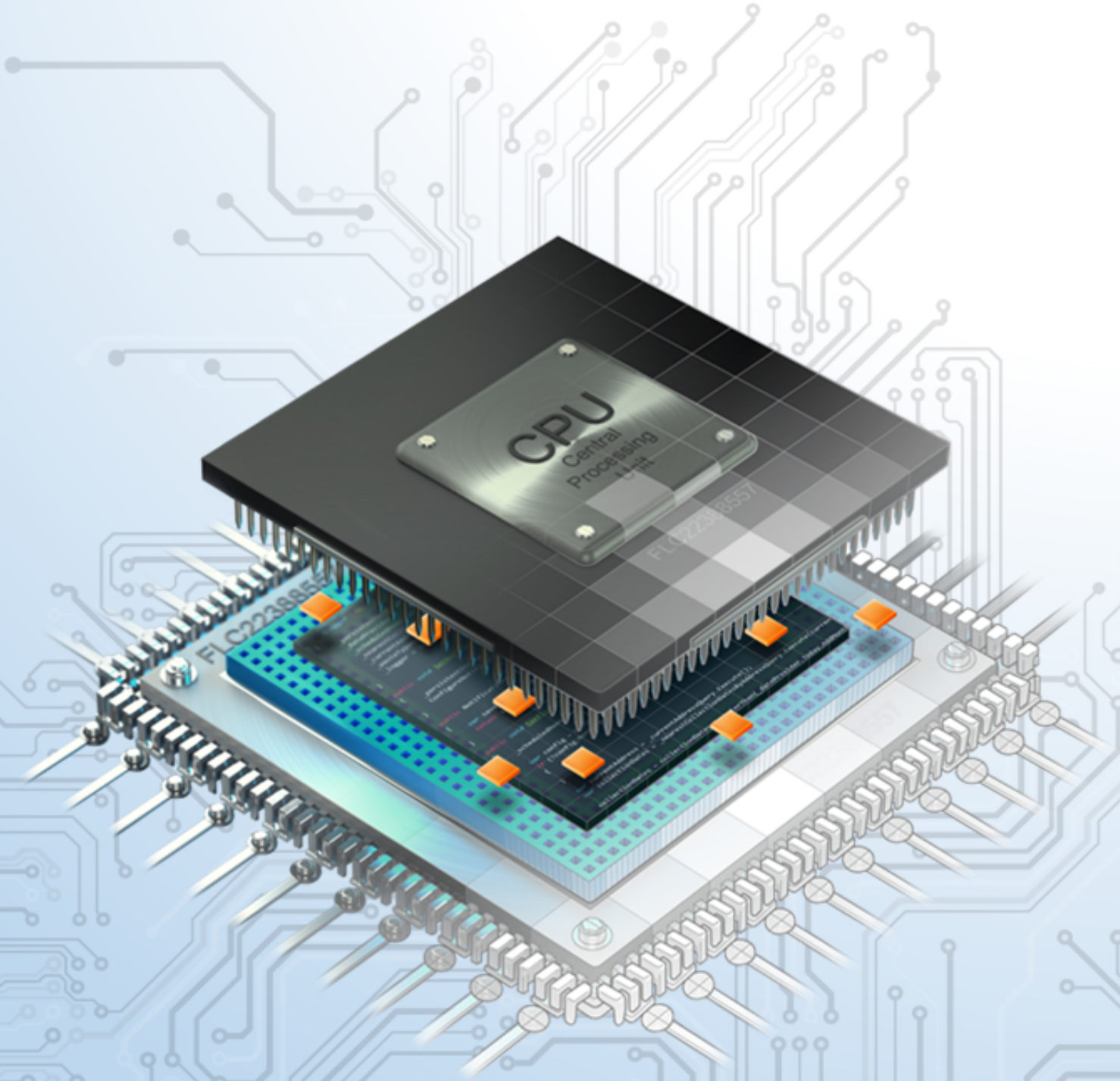
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

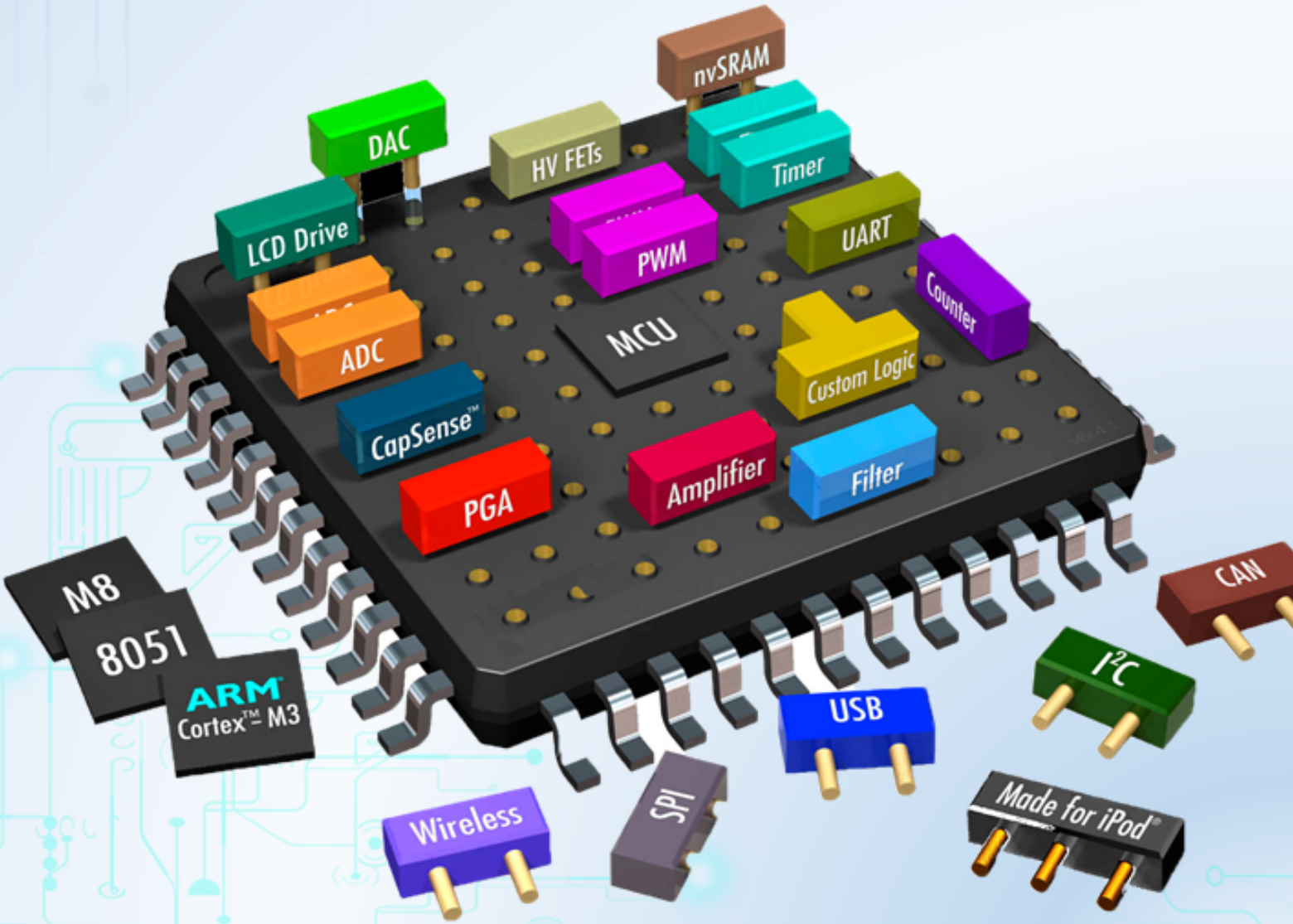


Dr. Le Trong Nhan
Pham Ngoc Manh - 1813045

Contents

CHAPTER 1

LED Animations



1 Introduction

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

Figure 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**

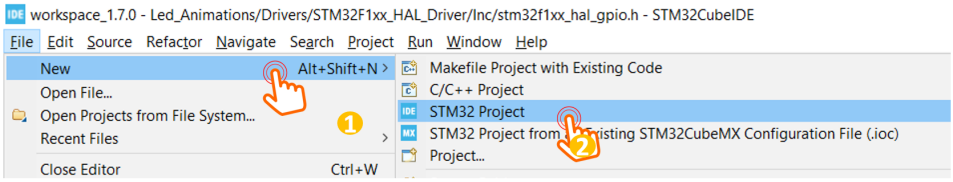


Figure 1.2: Create a new project on STM32CubeIDE

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**

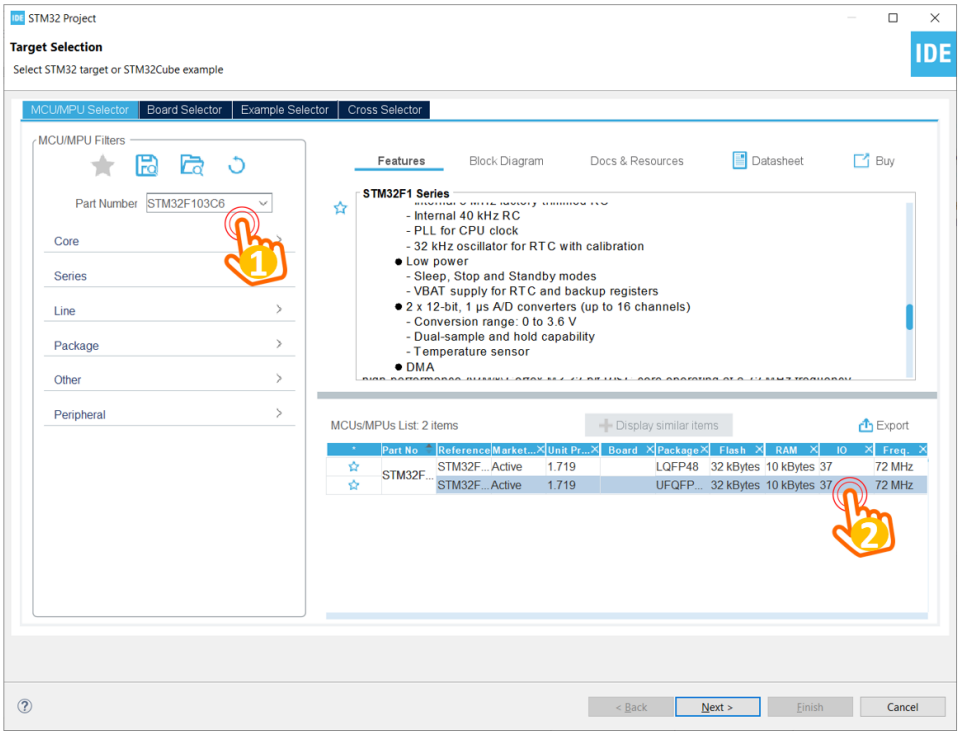


Figure 1.3: Select the target device

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

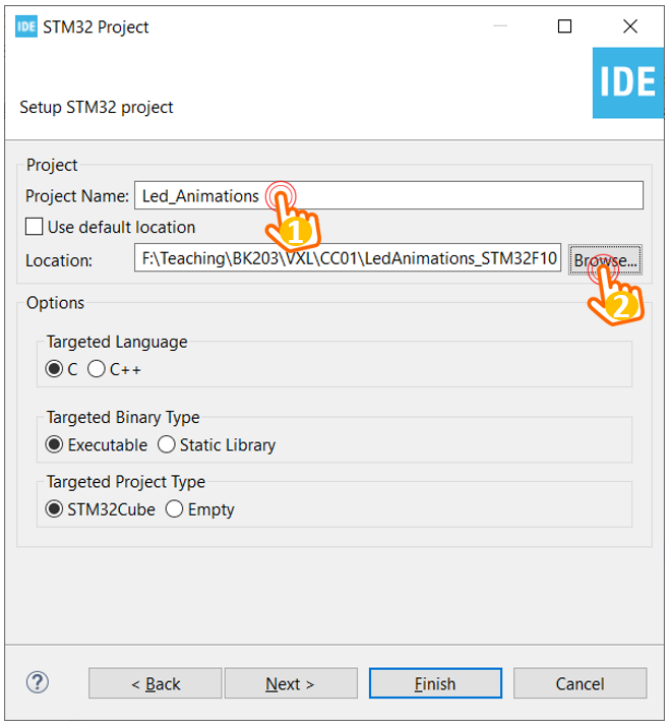


Figure 1.4: Select the target device

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.

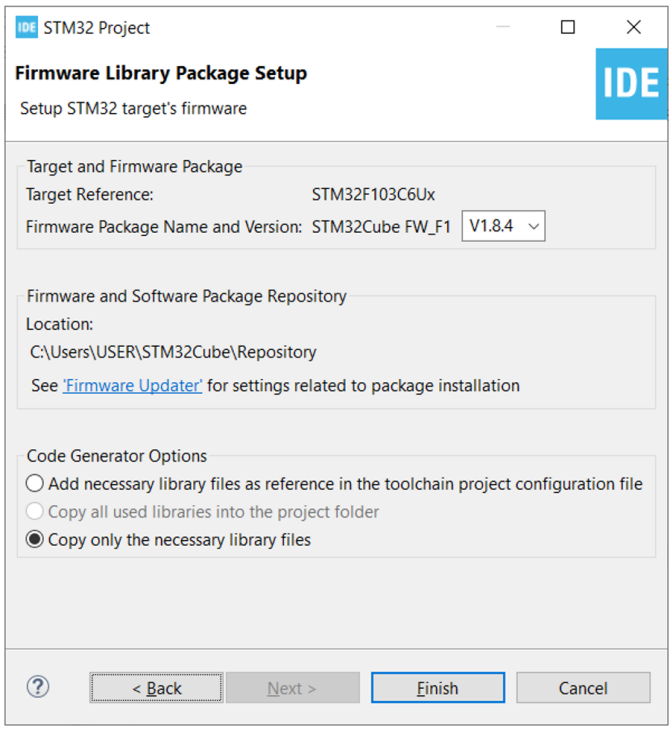


Figure 1.5: Keep default firmware version

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 micro-controller like the STM32.

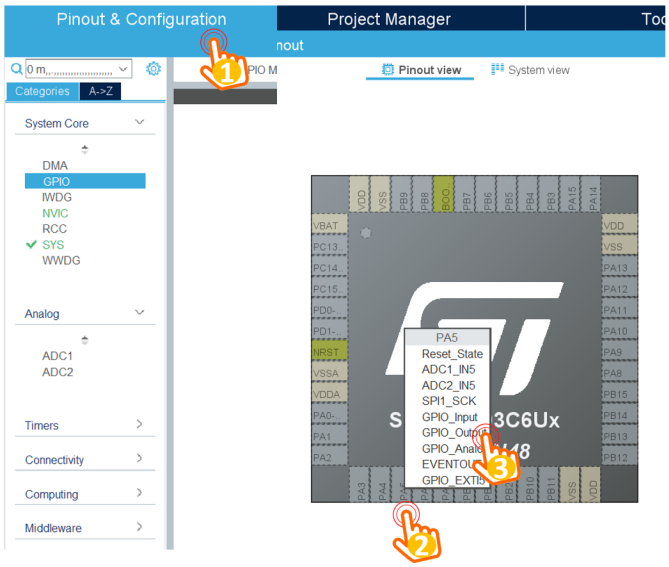


Figure 1.6: *Set PA5 to GPIO Output mode*

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.

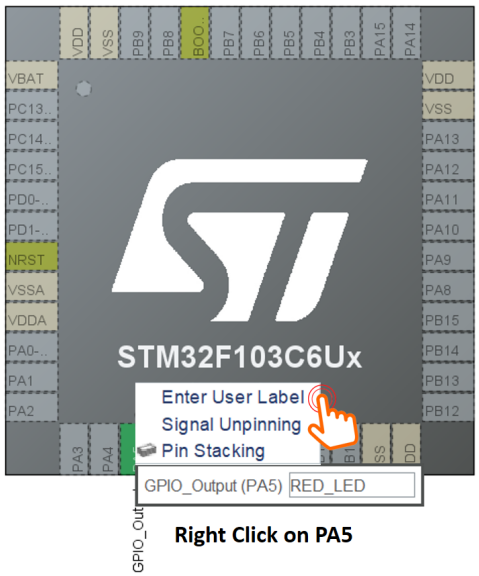


Figure 1.7: *Provide a name for PA5*

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:

```
1  int main(void)
2  {
3    /* USER CODE BEGIN 1 */
4
5    /* USER CODE END 1 */
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:

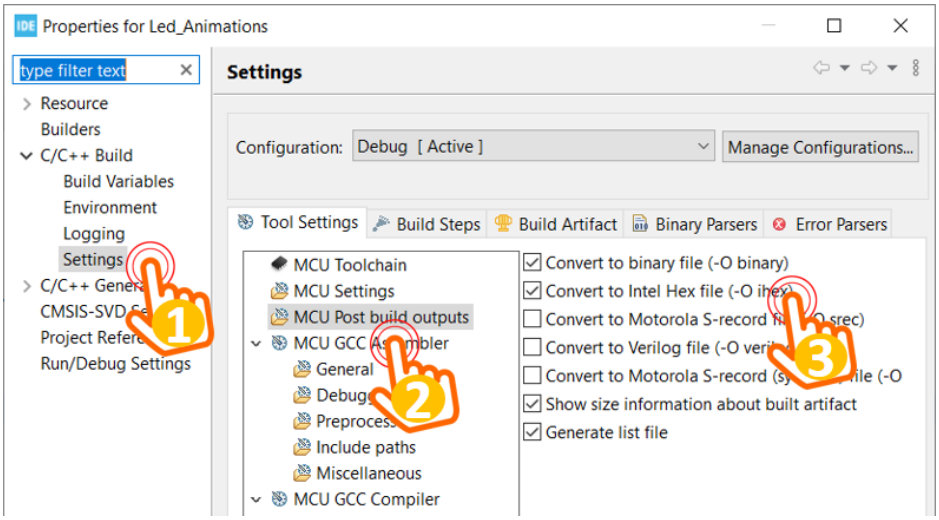


Figure 1.8: Config for hex file output

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)
```

Figure 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**.

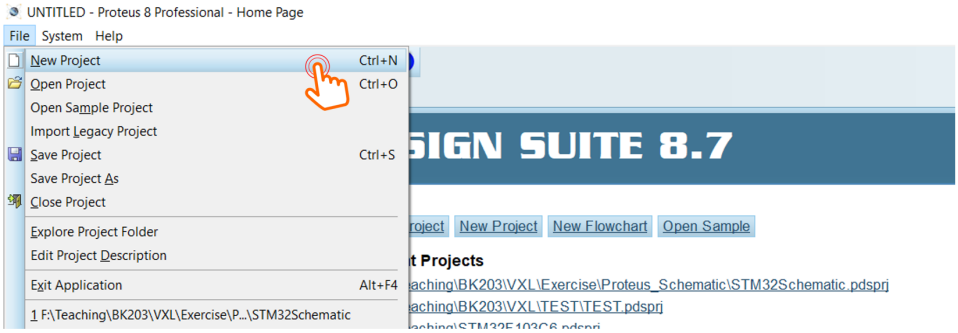


Figure 1.10: Create a new project on Proteus

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

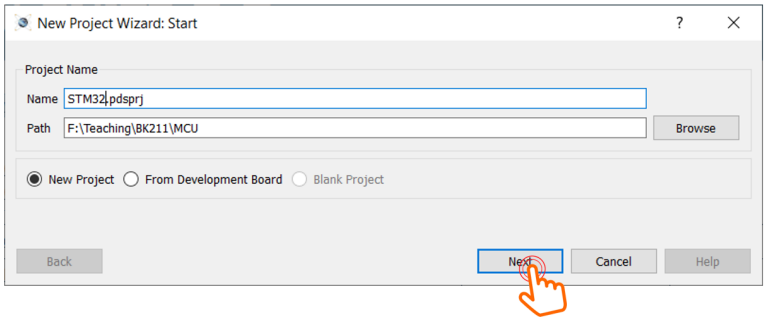


Figure 1.11: Provide project name and location

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

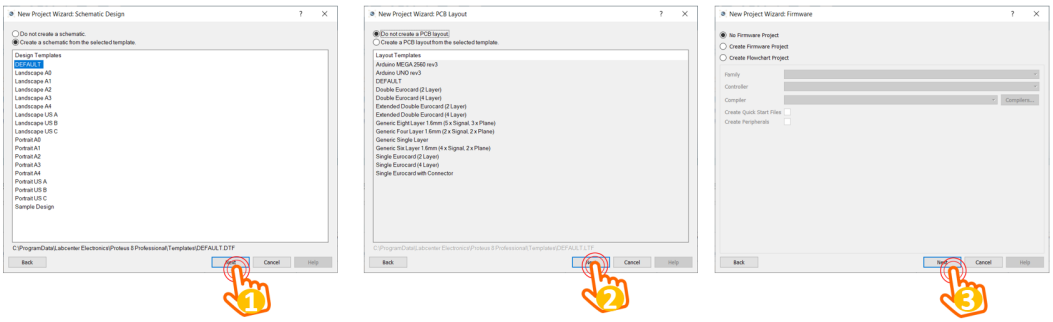


Figure 1.12: Keep the default options by clicking on Next

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

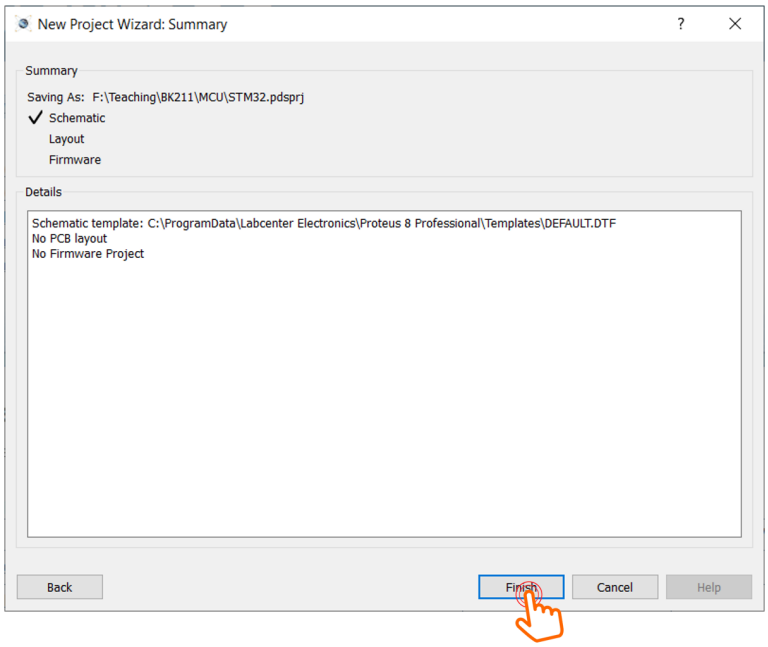


Figure 1.13: Finish the project wizard

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

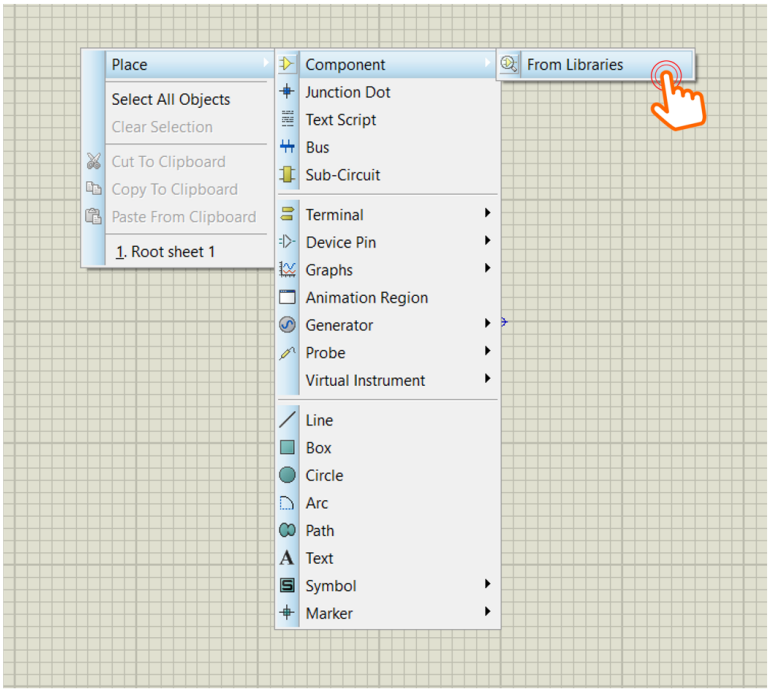


Figure 1.14: Select a component from the library

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:

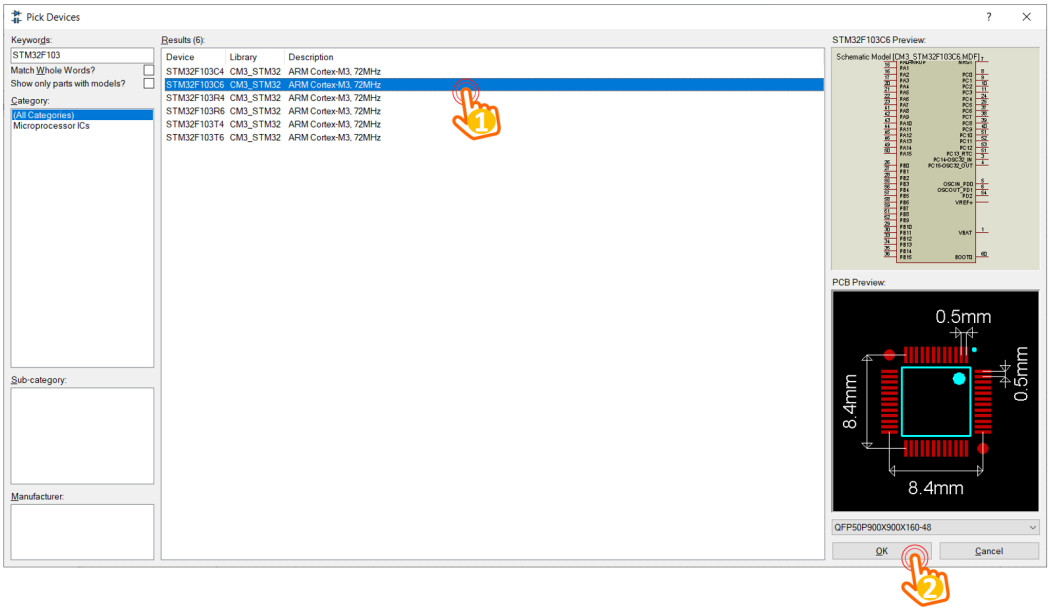


Figure 1.15: Select STM32F103C6

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:

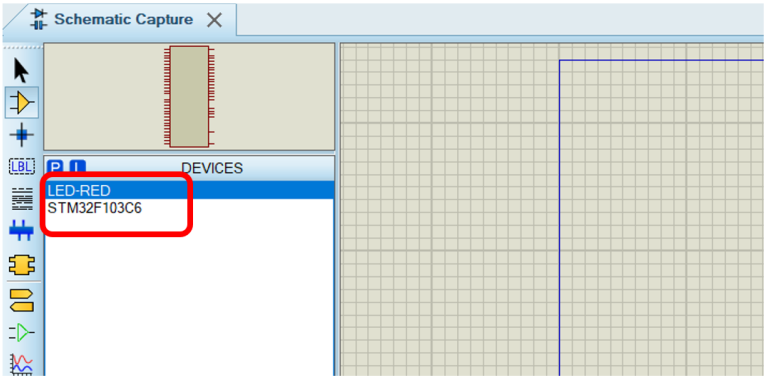


Figure 1.16: STM32 and an LED in the project

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:

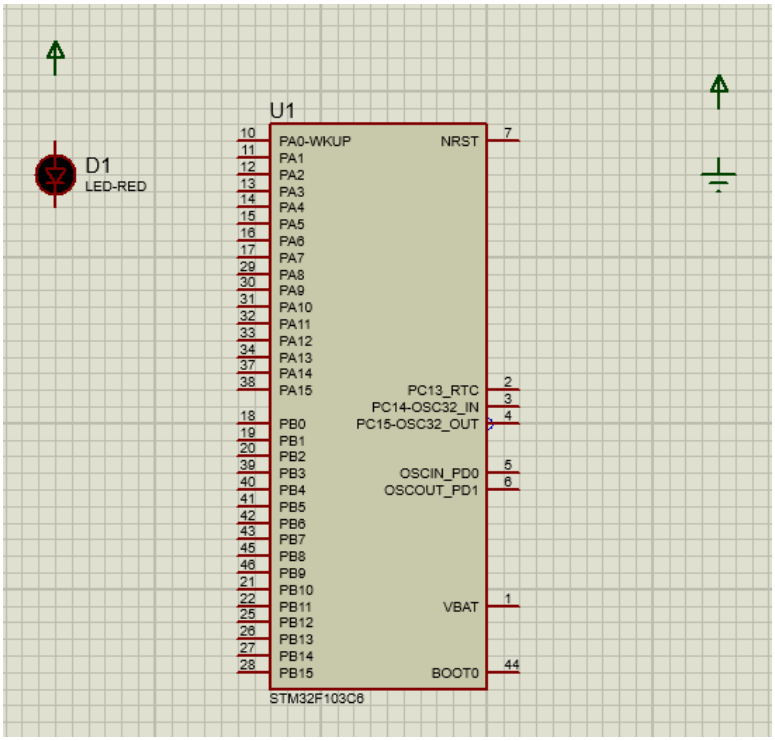


Figure 1.17: *Place components to the project*

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.

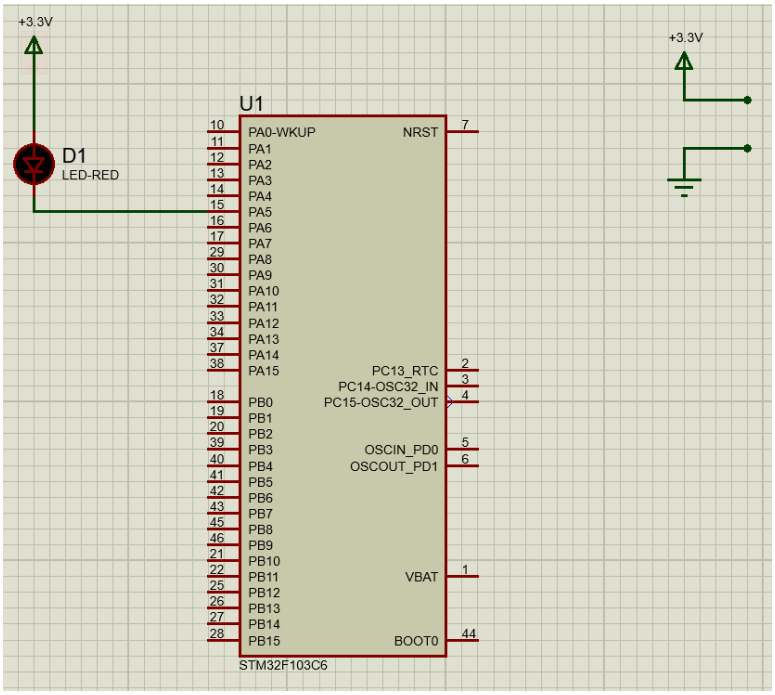


Figure 1.18: *Connect components and set the power to 3.3V*

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**

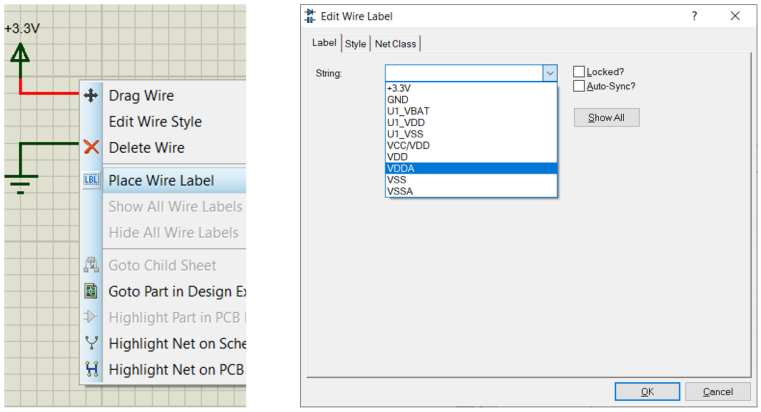


Figure 1.19: Place label for Power and Ground

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:

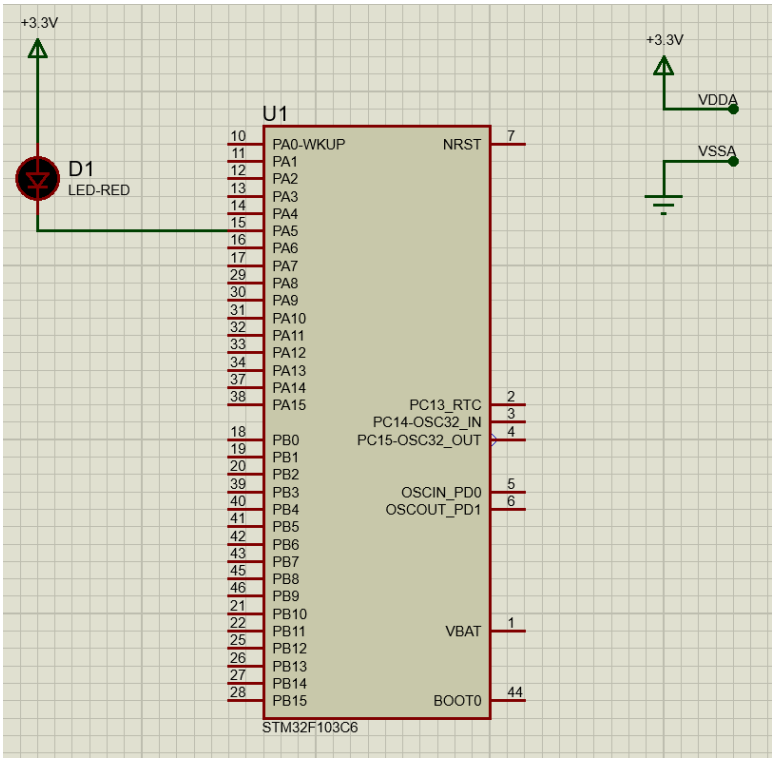


Figure 1.20: Finalize the schematic

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

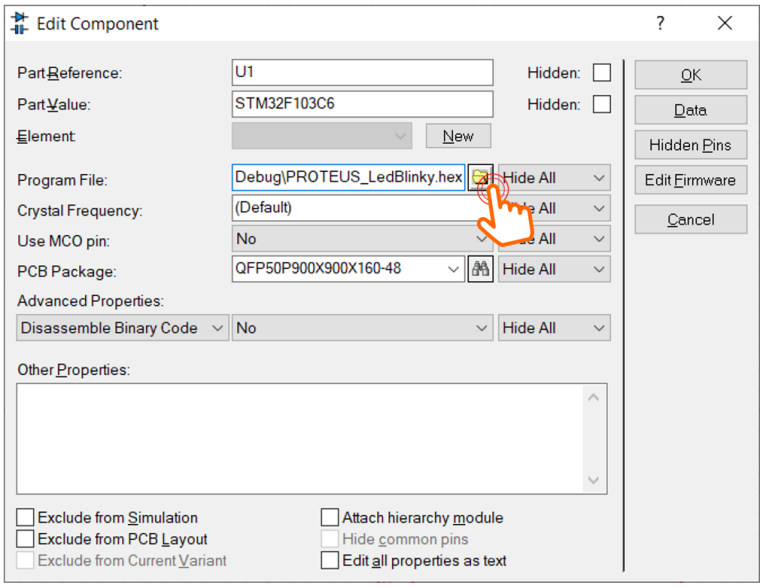


Figure 1.21: Set the program of the STM32 to the hex file from Cube IDE

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:

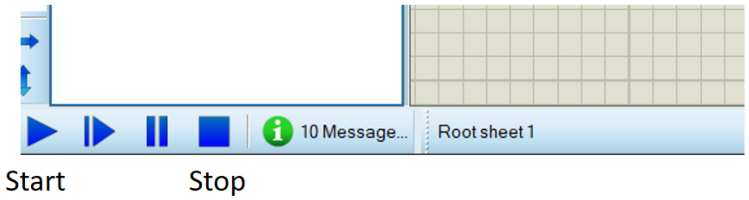


Figure 1.22: Quick access buttons to start and stop the simulation

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.

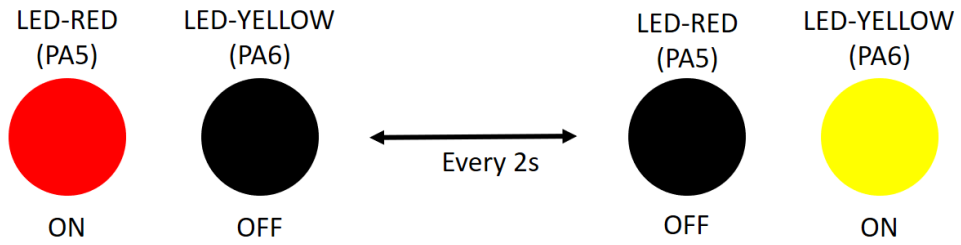


Figure 1.23: State transitions for 2 LEDs

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).

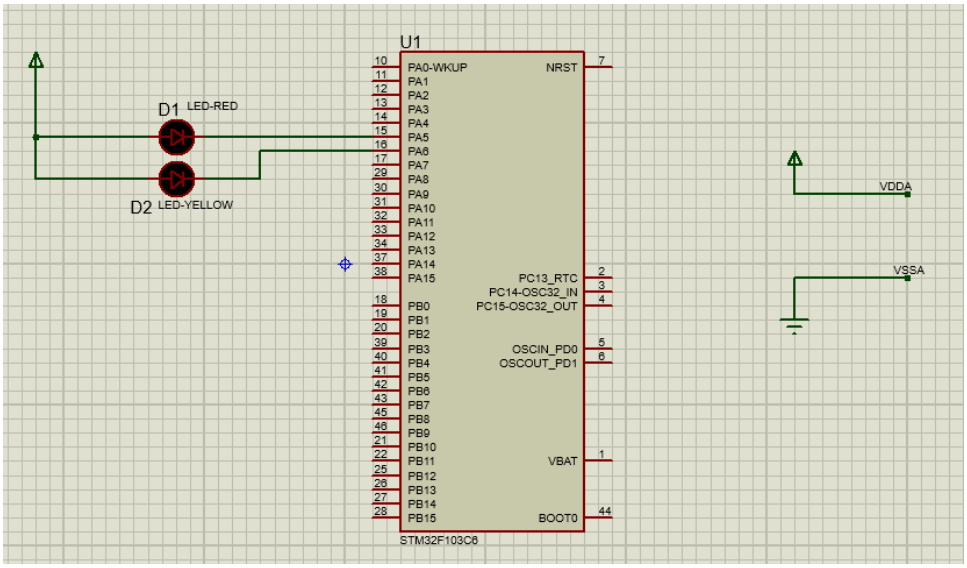


Figure 1.24: The schematic from Proteus simulation.

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).

```
while (1)
  {
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
```

```
  HAL_Delay(2000);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
  HAL_Delay(2000);
  }
```

Program 1.3: the source code in loop while

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.

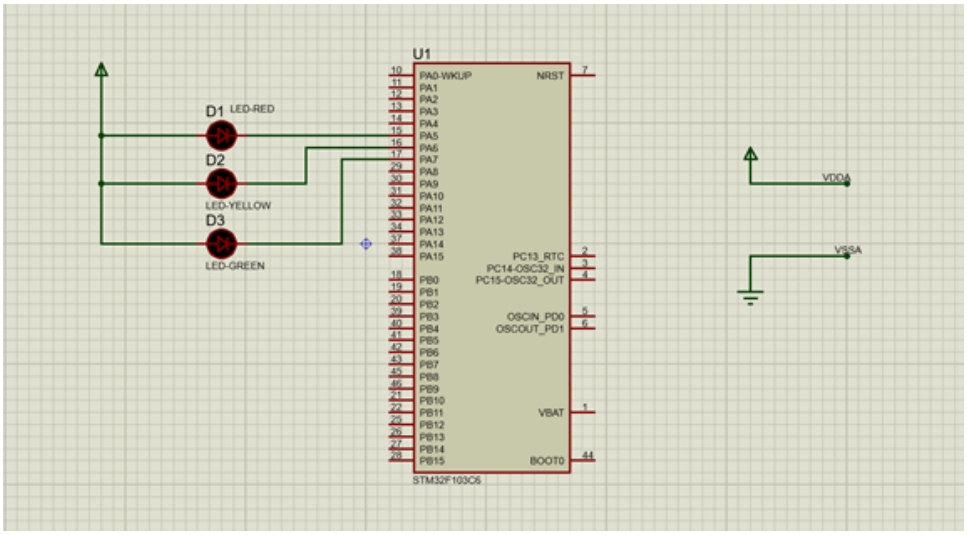


Figure 1.25: The schematic from Proteus simulation.

Report 2: Present the source code in while.

```
while (1)
  {
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET); //
   yellow off
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET); //
   red on
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET); //
   green off
  HAL_Delay(5000);

  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);//
   yellow off
```

```
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);//red
   off
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET); //
   green on
  HAL_Delay(3000);

  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET); //
   yellow on
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);// red
    off
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);//
   green off
  HAL_Delay(2000);
  }
```

Program 1.4: Source in loop while.

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. **Report 1:** Present the schematic.

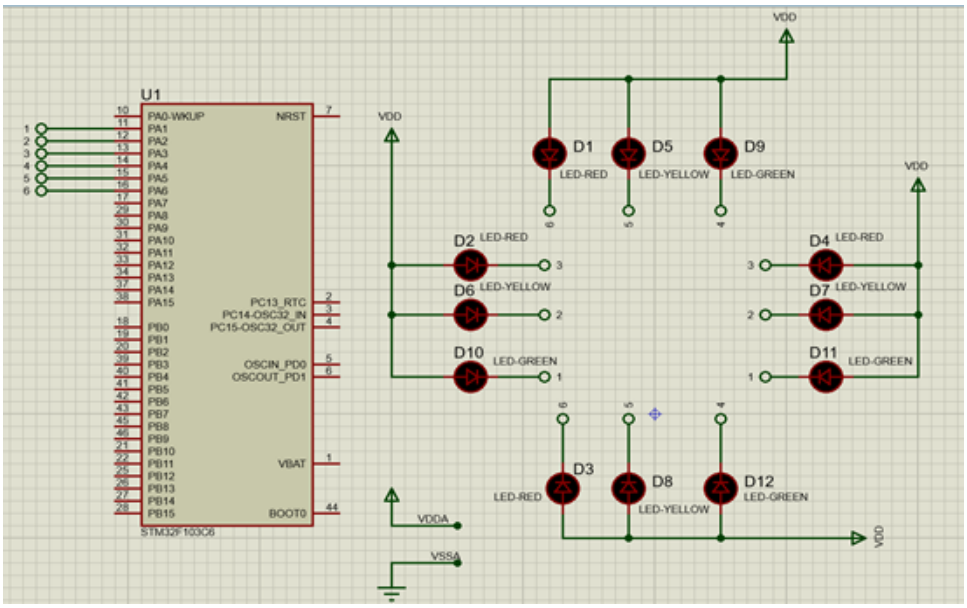


Figure 1.26: A 4 way traffic light.

Report 2: Present the source code in while.

```
int state = 0;
  while (1)
  {
    switch(state){
    case 0:
      for(int i = 5;i > 0;i--){
        // led red 1 on, led green 2 on
```

```
8         // wait 3s then i=2, led green 2 off, led yellow 2
  on
9         // after 2s, led red 1 off -> led green 1 on, led
  yellow 2 off -> led red 2 on
10        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);
   //yellow1
11        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET
  ); //red1
12        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
   //green1
13        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
   //yellow2
14        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
   //red2
15        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET
  ); //green2
16        if(i<=2){
17          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5,
  GPIO_PIN_RESET); //yellow2
18          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET
  ); //red2
19          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET
  ); //green2
20        }
21        HAL_Delay(1000);
22      }
23      state = 1;
24      break;
25    case 1:
26      for(int i = 5;i > 0;i--){
27        // led red 2 on, led green 1 on
28        // wait 3s then i=2, led green 1 off, led yellow 1
  on
29        // after 2s, led red 2 off -> led green 2 on, led
  yellow 1 off -> led red 1 on
30        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);
   //yellow1
31        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);
   //red1
32        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET
  ); // green1
33        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
   //yellow2
34        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET
  ); //red2
35        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
   //green2
36        if(i<=2){
37          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2,
```

```
  GPIO_PIN_RESET); //yellow1
38          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET
  ); //red1
39          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET
  ); // green1
40        }
41        HAL_Delay(1000);
42      }
43      state = 0;
44      break;
45    default:
46      state = 0;
47      break;
48    }
49  }
```

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:

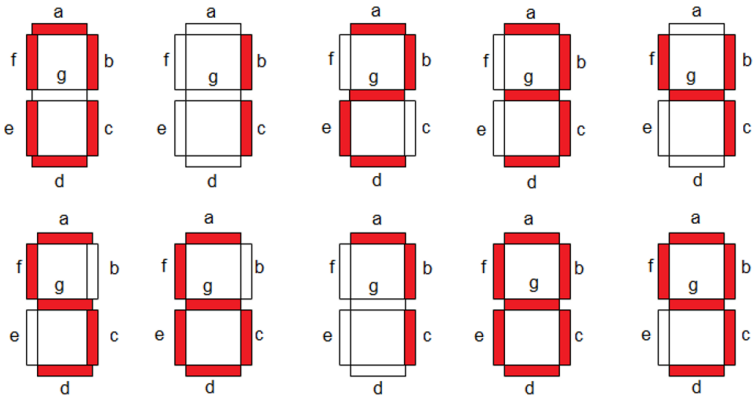


Figure 1.27: Display a number on 7 segment LED

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 1.5: An example for your source code

Report 1: Present the schematic.

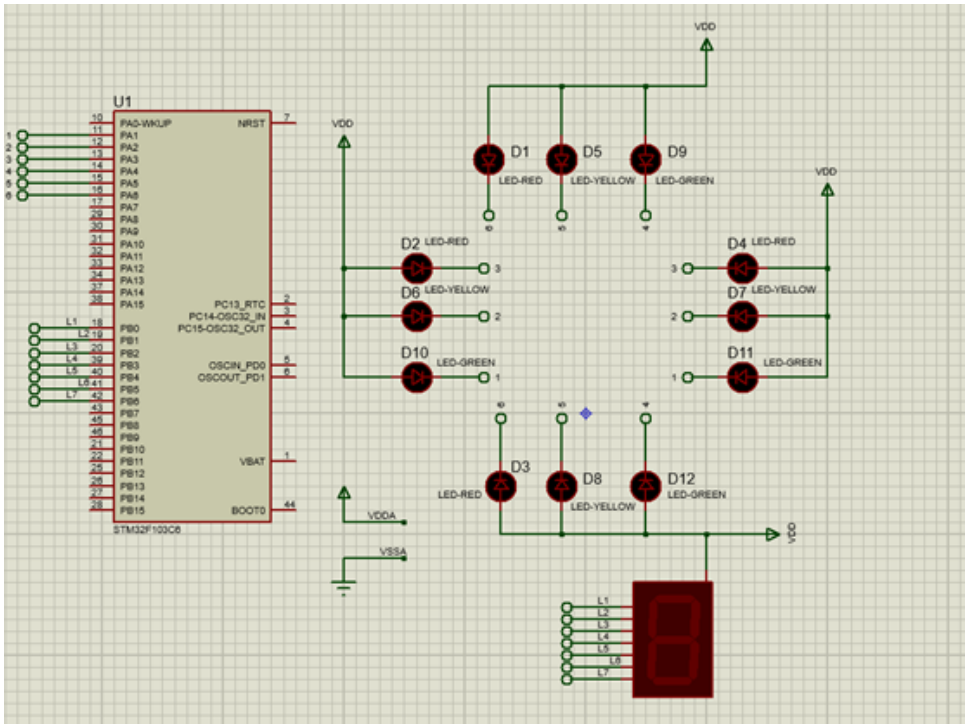


Figure 1.28: The schematic in Proteus.

Report 2: Present the source code for display7SEG function.

```
void display7SEG(int num)
{
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2
   |GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6,
   GPIO_PIN_SET);
  switch (num)
  {
  case 0: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
   GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5,
   GPIO_PIN_RESET);
      break;
  case 1: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1|GPIO_PIN_2,
   GPIO_PIN_RESET);
      break;
```

```
10    case 2: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
      GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_6, GPIO_PIN_RESET);
11        break;
12    case 3: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
      GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_6, GPIO_PIN_RESET);
13        break;
14    case 4: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1|GPIO_PIN_2|
      GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_RESET);
15        break;
16    case 5: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_2|
      GPIO_PIN_3|GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_RESET);
17        break;
18    case 6: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_2|
      GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6,
      GPIO_PIN_RESET);
19        break;
20    case 7: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
      GPIO_PIN_2, GPIO_PIN_RESET);
21        break;
22    case 8: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
      GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6,
      GPIO_PIN_RESET);
23        break;
24    case 9: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
      GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_5|GPIO_PIN_6,
      GPIO_PIN_RESET);
25        break;
26    default: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2|GPIO_PIN_3|
      GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_RESET);
27        break;
28    }
29 }
```

Program 1.6: Source code 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.

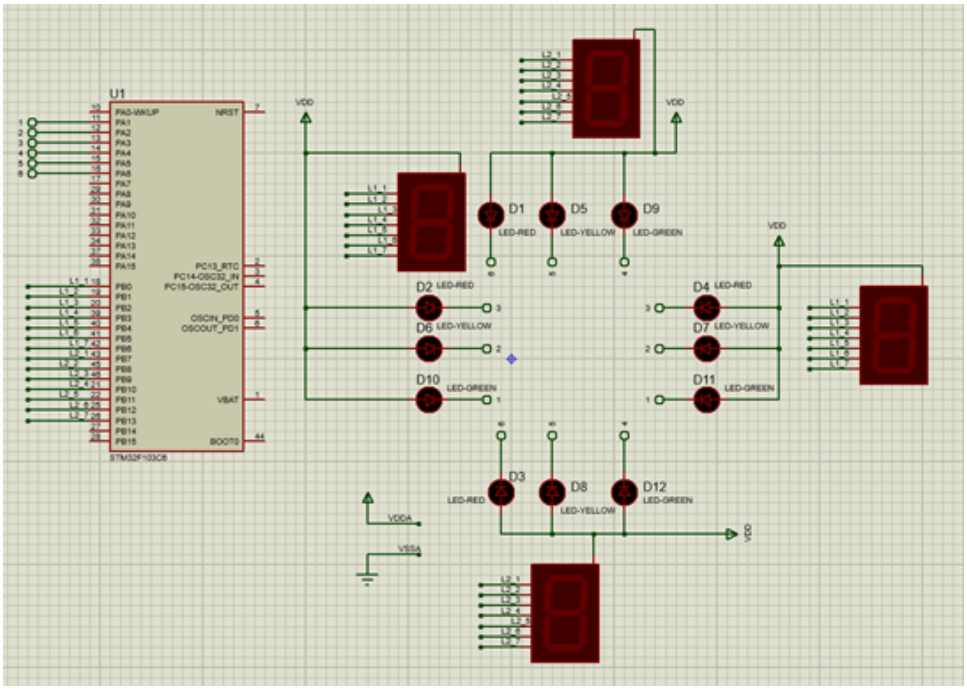


Figure 1.29: The schematic

Idea: Let **i** and **j** count variable to count down the time of traffic lights of 2 streets, **i** street 1 and **j** street 2. Set values for **i** and **j**.
+ State 0: (led red on street 1 on and led green on street 2 on) when led green on street 2 runs out, reset **j** to the time of led yellow on street 2, led yellow on street 2 on and continue counting. When time of led red on street 1 and led led yellow on street 2 out then change to state 1.
+ State 1: (led red on street 2 on and led green on street 1 on) when led green on street 1 runs out, reset **i** to the time of led yellow on street 1, led yellow on street 1 on and continue counting. When time of led red on street 2 and led led yellow on street 1 out then change to state 0.

```
int state=0;
int  i, j;
while (1)
{
  i = 5; j = 3;
  switch(state){
  case 0:
    while(i > 0){
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);
  //yellow1
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET
); //red1
```

```
11          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
     //green1
12          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
     //yellow2
13          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
     //red2
14          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET
    ); //green2
15          if(j == 0) j = 2;
16          if(i<=2){
17            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5,
    GPIO_PIN_RESET); //yellow2
18            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET
    ); //red2
19            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET
    ); //green2
20          }
21          display7SEG(i);
22          display7SEG2(j);
23          HAL_Delay(1000);
24          i--; j--;
25        }
26        state = 1;
27        break;
28      case 1:
29        i = 3; j = 5;
30        while(j > 0){
31          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);
     //yellow1
32          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);
     //red1
33          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET
    ); // green1
34          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
     //yellow2
35          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET
    ); //red2
36          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
     //green2
37          if(i == 0) i = 2;
38          if(i<=2){
39            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2,
    GPIO_PIN_RESET); //yellow1
40            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET
    ); //red1
41            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET
    ); // green1
42          }
43          display7SEG(i);
```

```
        display7SEG2(j);
        HAL_Delay(1000);
        j--; i--;
      }
      state = 0;
      break;
    default:
      state = 0;
      break;
    }
  }
  /* USER CODE END 3 */
}
```

Program 1.7: Code in loop while

```
void display7SEG(int num)
{
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2
   |GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6,
   GPIO_PIN_SET);
  switch (num)
  {
  case 0: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
   GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5,
   GPIO_PIN_RESET);
      break;
  case 1: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1|GPIO_PIN_2,
   GPIO_PIN_RESET);
      break;
  case 2: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
   GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_6, GPIO_PIN_RESET);
      break;
  case 3: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
   GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_6, GPIO_PIN_RESET);
      break;
  case 4: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1|GPIO_PIN_2|
   GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_RESET);
      break;
  case 5: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_2|
   GPIO_PIN_3|GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_RESET);
      break;
  case 6: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_2|
   GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6,
   GPIO_PIN_RESET);
      break;
  case 7: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
   GPIO_PIN_2, GPIO_PIN_RESET);
      break;
  case 8: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
```

```
    GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6,
    GPIO_PIN_RESET);
23      break;
24  case 9: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|
    GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_5|GPIO_PIN_6,
    GPIO_PIN_RESET);
25      break;
26  default: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2|GPIO_PIN_3|
    GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_RESET);
27      break;
28  }
29 }
30
31 void display7SEG2(int num)
32 {
33  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9
    |GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13,
    GPIO_PIN_SET);
34  switch (num)
35  {
36  case 0: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7|GPIO_PIN_8|
    GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12,
    GPIO_PIN_RESET);
37      break;
38  case 1: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8|GPIO_PIN_9,
    GPIO_PIN_RESET);
39      break;
40  case 2: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7|GPIO_PIN_8|
    GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_13, GPIO_PIN_RESET);
41      break;
42  case 3: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7|GPIO_PIN_8|
    GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_13, GPIO_PIN_RESET);
43      break;
44  case 4: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8|GPIO_PIN_9|
    GPIO_PIN_12|GPIO_PIN_13, GPIO_PIN_RESET);
45      break;
46  case 5: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7|GPIO_PIN_9|
    GPIO_PIN_10|GPIO_PIN_12|GPIO_PIN_13, GPIO_PIN_RESET);
47      break;
48  case 6: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7|GPIO_PIN_9|
    GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13,
    GPIO_PIN_RESET);
49      break;
50  case 7: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7|GPIO_PIN_8|
    GPIO_PIN_9, GPIO_PIN_RESET);
51      break;
52  case 8: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7|GPIO_PIN_8|
    GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12|
    GPIO_PIN_13, GPIO_PIN_RESET);
```

```
53          break;
54      case 9: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7|GPIO_PIN_8|
        GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_12|GPIO_PIN_13,
        GPIO_PIN_RESET);
55          break;
56      default: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9|GPIO_PIN_10|
        GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13, GPIO_PIN_RESET);
57          break;
58      }
59  }
```

Program 1.8: Code in function display7SEG

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.

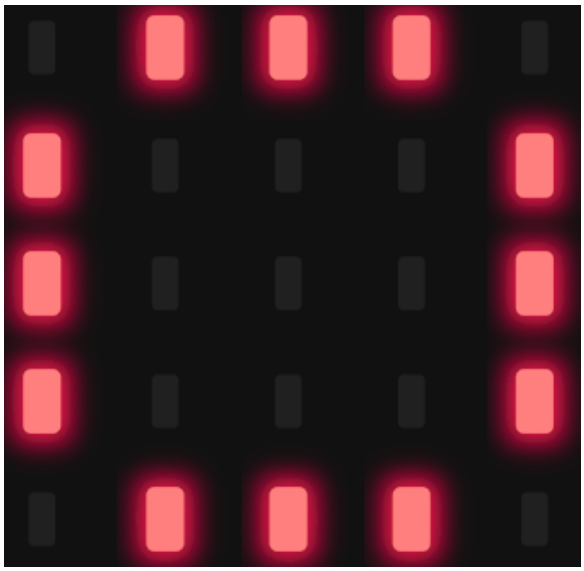
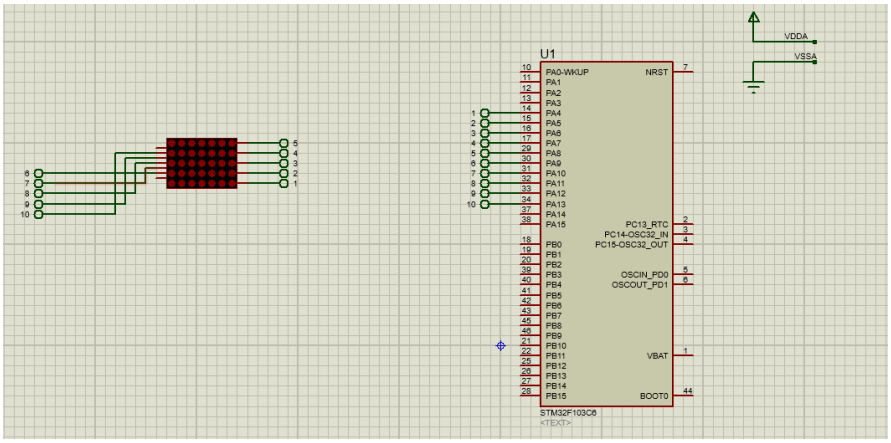


Figure 1.30: 12 LEDs for an analog clock

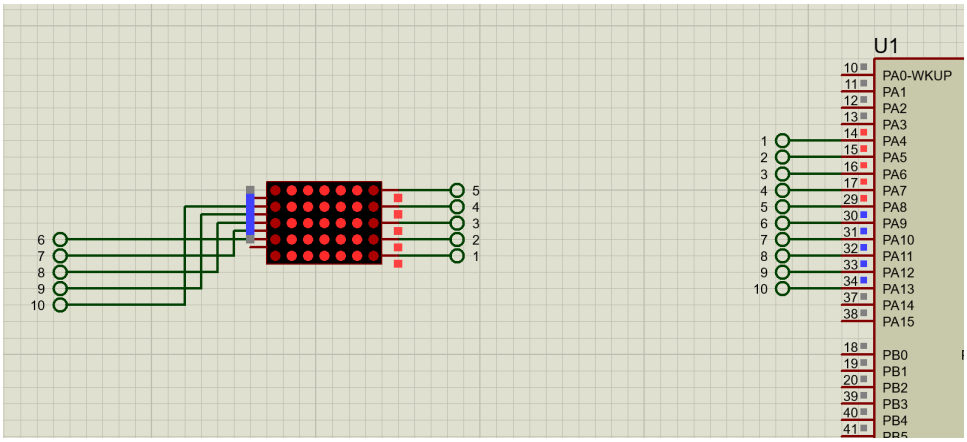
Report 1: Present the schematic.



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

```
while (1)
  {
    /* USER CODE END WHILE */
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, GPIO_PIN_RESET);
    /* USER CODE BEGIN 3 */
  }
```

Output:



4.7 Exercise 7

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

```
void clearAllClock(){
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, GPIO_PIN_SET);
```

```
12 }
```

Program 1.9: Function clearAllClock().

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.

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.

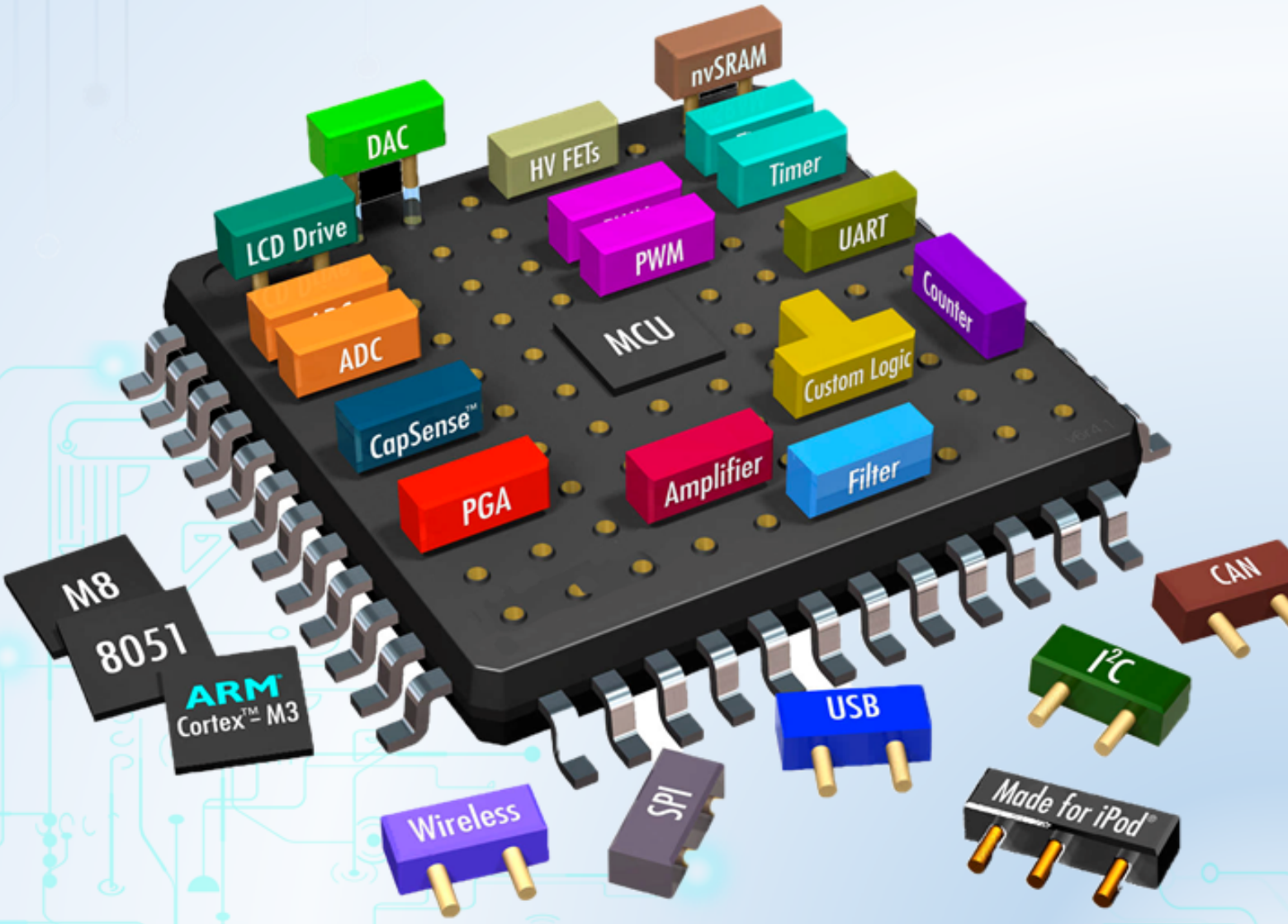
4.10 Exercise 10

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.

Link github: `https://github.com/PNgocmanh/Microcontroller_1813045.git`

CHAPTER 2

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.

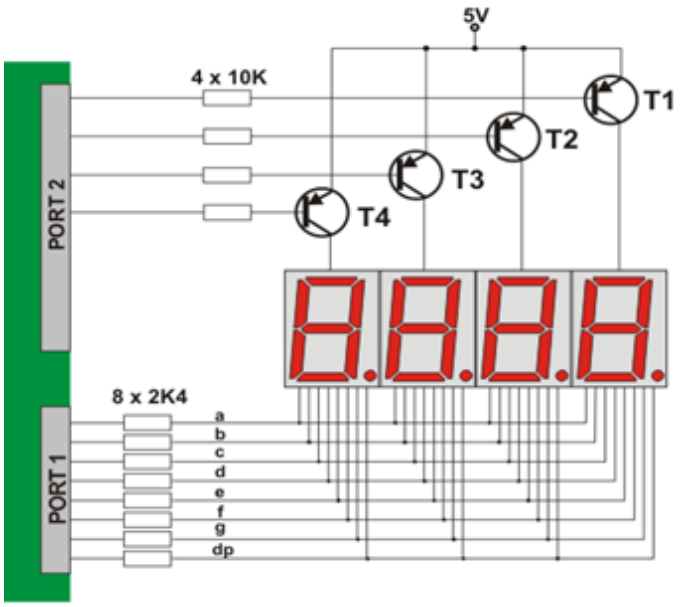


Figure 2.1: *Four seven segment LED interface for a micro-controller*

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.

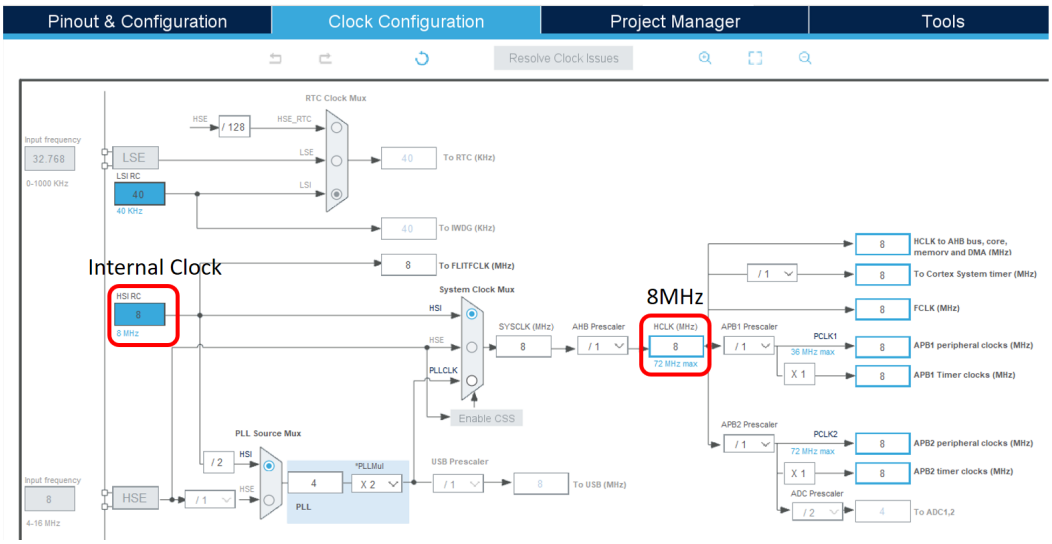


Figure 2.2: Default clock source for the system

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

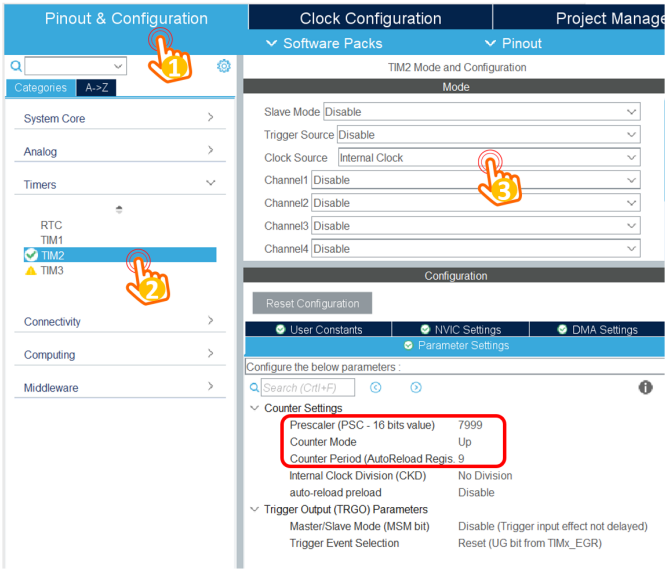


Figure 2.3: Configure for Timer 2

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:

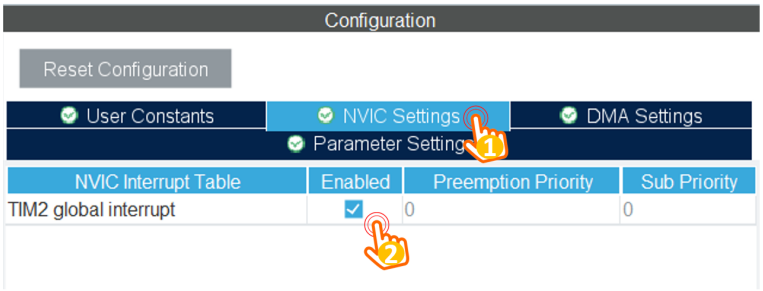


Figure 2.4: Enable timer interrupt

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 */

  while (1){

  }
}
```

Program 2.1: 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:

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

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

Program 2.2: Add an interrupt service routine

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

```
/* 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 2.3: 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

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:

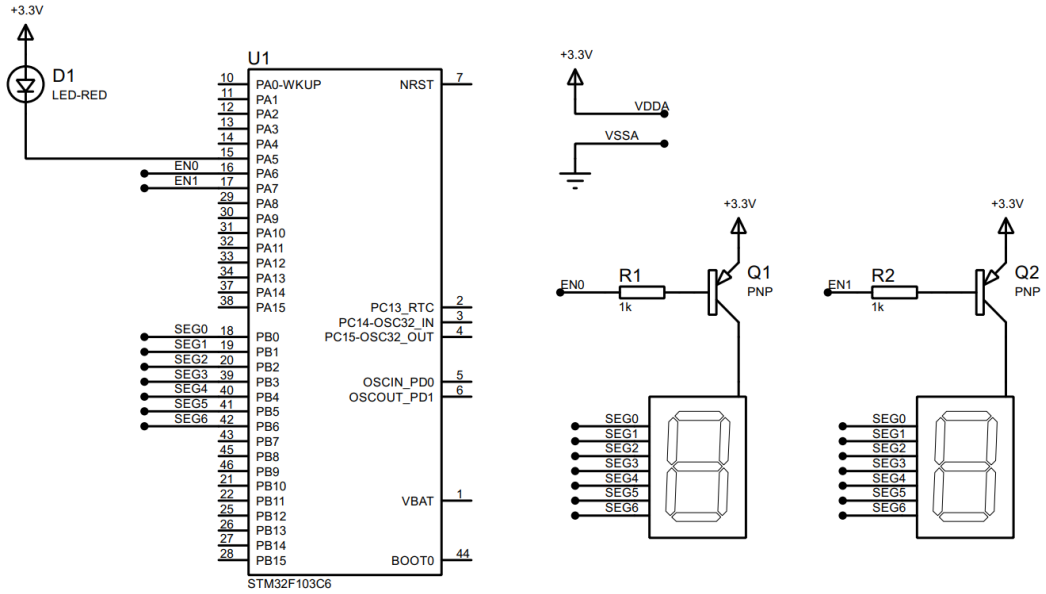


Figure 2.5: Simulation schematic in Proteus

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: My schematic from Proteus:

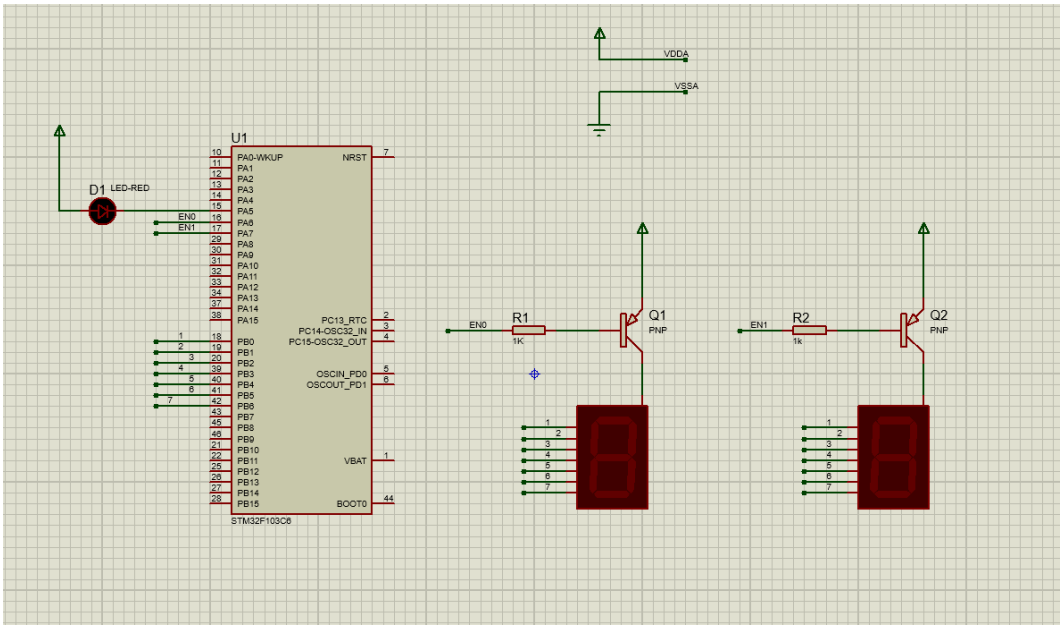


Figure 2.6: Schematic in Protues

Report 2: Present your source code in the **HAL_TIM_PeriodElapsedCallback** function
Ta set Prescaler = 7999, Period = 499.
Ta có

$$f = \frac{8x10^6}{(7999+1)(499+1)} \rightarrow T = 0.5s$$

.

```
uint8_t State = 0;
void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef *
   htim ){
  if(htim->Instance== TIM2){
    switch(State){
    case 0:
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);
      display7SEG(1);
      State = 1;
      break;
    case 1:
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
      display7SEG(2);
      State = 0;
```

```
        break;
      }
    }
}
```

Program 2.4: Funtion HAL_TIM_PeriodElapsedCallBack

Short question: What is the frequency of the scanning process?
Answer: Because switching time between 2 LEDs is half of second(0.5s), so the frequency of the scanning process is 2 Hz.

3.2 Exercise 2

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

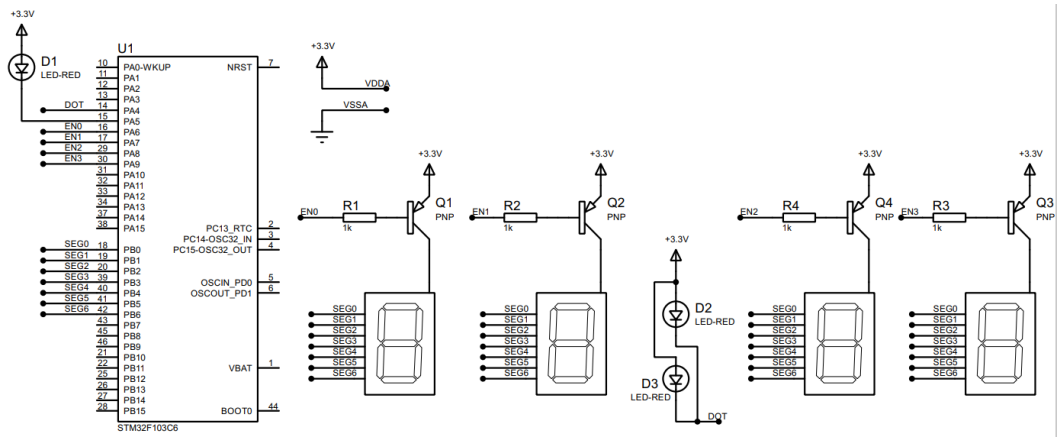


Figure 2.7: Simulation schematic in Proteus

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.

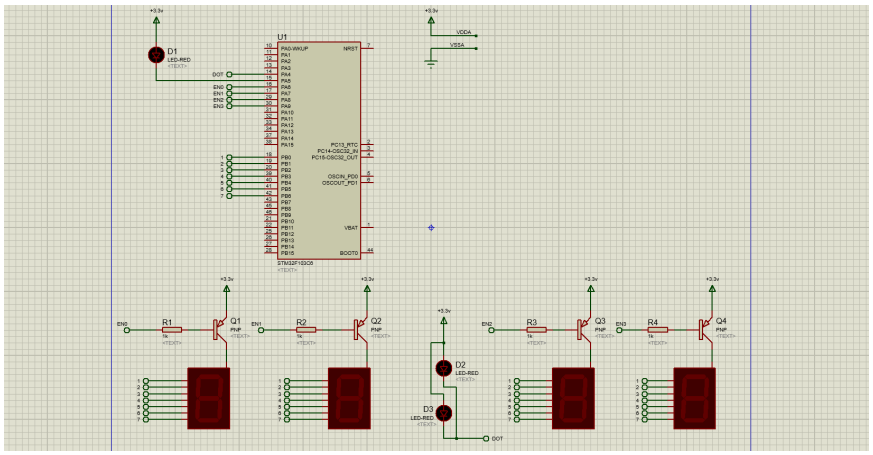


Figure 2.8: Schematic in Proteus

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

```
uint8_t State = 0;
int count = 0;
void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef *
   htim ){
  if(htim->Instance == htim2.Instance){
    count++;
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5); // display led
   every 0.5s
    // FSM
    switch(State){
    case 0:
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7 | GPIO_PIN_8 |
   GPIO_PIN_9, GPIO_PIN_SET);
      display7SEG(1);
      State = 1;
      break;
    case 1:
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6 | GPIO_PIN_8 |
   GPIO_PIN_9, GPIO_PIN_SET);
      display7SEG(2);
      State = 2;
      break;
    case 2:
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6 | GPIO_PIN_7 |
   GPIO_PIN_9, GPIO_PIN_SET);
      display7SEG(3);
      State = 3;
      break;
    case 3:
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6 | GPIO_PIN_7 |
   GPIO_PIN_8, GPIO_PIN_SET);
      display7SEG(0);
      State = 0;
      break;
    }
    // display led DOT every second (1s)
    if(count==2){
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
      count = 0;
    }
  }
}
```

Program 2.5: Funtion HAL_TIM_PeriodElapsedCallBack

Explaining:
- Use the State variable to determine the state every 0.5s.
- Use the count variable to determine the time of the DOT pin blinking (1s) when count = 2.
Short question: What is the frequency of the scanning process?
Answer: Because switching time between 2 LEDs is half of second(0.5s), so the frequency of the scanning process is 2 Hz.

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

```
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 7 SEG with led_buffer [0]
    display7SEG(led_buffer[0]);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7 | GPIO_PIN_8 |
   GPIO_PIN_9, GPIO_PIN_SET);
    break;
  case 1:
    // Display the second 7 SEG with led_buffer [1]
    display7SEG(led_buffer[1]);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6 | GPIO_PIN_8 |
   GPIO_PIN_9, GPIO_PIN_SET);
    break;
  case 2:
    // Display the third 7 SEG with led_buffer [2]
    display7SEG(led_buffer[2]);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6 | GPIO_PIN_7 |
   GPIO_PIN_9, GPIO_PIN_SET);
    break;
  case 3:
    // Display the forth 7 SEG with led_buffer [3]
    display7SEG(led_buffer[3]);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6 | GPIO_PIN_7 |
   GPIO_PIN_8, GPIO_PIN_SET);
    break;
  default:
    break;
  }
```

```
}
```

Program 2.6: 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 2: Present the source code in the HAL_TIM_PeriodElapsedCallback. Students are proposed to change the values in the **led_buffer** array for unit test this function, which is used afterward.

```
int count = 0;
int led_index = 0;
void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef *
   htim ){
  if(htim->Instance == htim2.Instance){
    count++;
    update7SEG (led_index++);
    if(led_index >= MAX_LED) led_index = 0;
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    if(count==2){
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
      count = 0;
    }
  }
}
```

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**.
Ta set Prescaler = 7999, Period = 9.
Ta có

$$f = \frac{8x10^6}{(7999+1)(9+1)} \rightarrow T = 10ms$$

.
Ta set count = 100 cho hàm **HAL_TIM_PeriodElapsedCallBack** ta có T = 1s.

```
int count = 100;
int led_index = 0;
void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef *
   htim ){
  if(htim->Instance == htim2.Instance){
    if(count <= 0) count = 100; // reset count
    if(count %50== 0){
        // display led every 0.5s
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
      //display 7-segement
      if(led_index >= MAX_LED) led_index = 0;
```

```
      update7SEG (led_index++);
    }
    if(count%100==0){
        // display led DOT pin every 1s
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
    }
    count--;
  }
}
```

Program 2.7: **HAL_TIM_PeriodElapsedCallBack**

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 2.8: 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 2.9: **updateClockBuffer** Function

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 2.10: 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 2.11: 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 2.12: 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 infinite loop while(1) do nothing, because line 1 of the code above is missing so timer0_flag = 0 lead to function time_run() is not change timer0_flag from 0 to 1.

Report 2: if in line 1 of the code above is changed to setTimer0(1), what happens after that and why?
Answer: In function setTimer0(int duration) line timer0_counter = duration / TIMER_CYCLE is calculation: timer0_counter = 1/10, because type of timer0_counter is **int**
so timer0_counter = 0. It is 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 = 10/10 = 1; and after function timer_run() is call, timer0_flag = 1 => LED_RED is blink every 2 seconds.

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.

```
  setTimer0(1000);
  while (1)
  {
    if( timer0_flag == 1){
      HAL_GPIO_TogglePin ( GPIOA , GPIO_PIN_4 ); // led DOT
    1s
      second ++;
      if ( second >= 60){
        second = 0;
        minute ++;
      }
      if( minute >= 60){
        minute = 0;
        hour ++;
      }
      if( hour >=24) hour = 0;
    }
    updateClockBuffer();
```

```
18      setTimer0(1000); // HAL_Delay(1000)
19  }
```

Program 2.13: Code in main function

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.

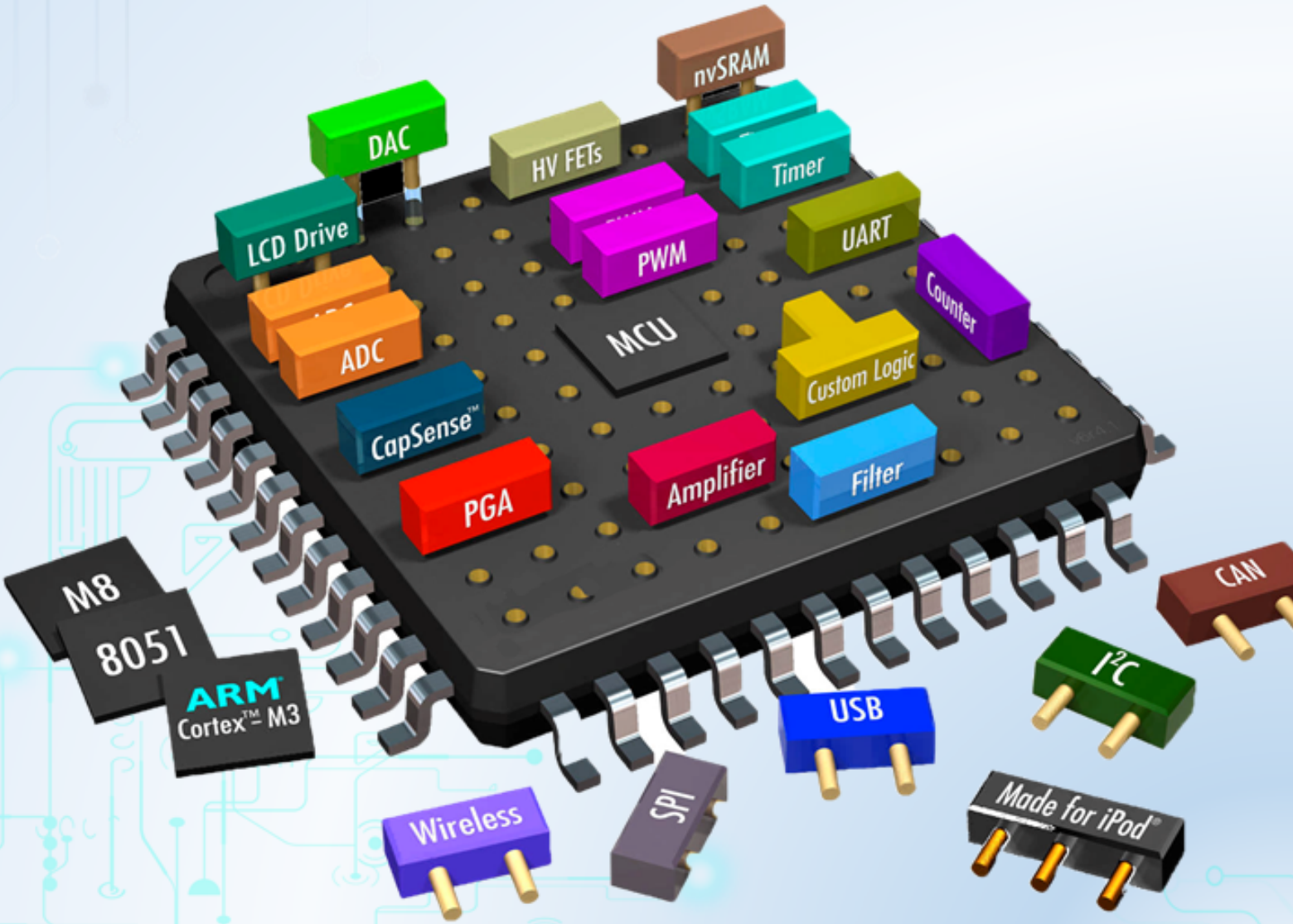
```
1   int led_index=0;
2   setTimer0(1000);
3   while (1)
4   {
5     if ( timer0_counter %50==0){
6       HAL_GPIO_TogglePin (GPIOA , GPIO_PIN_5 ); // LED
     blink every 0.5s
7       if(led_index >= MAX_LED) led_index = 0;
8       update7SEG (led_index++);
9     }
10    if( timer0_flag == 1){
11      HAL_GPIO_TogglePin ( GPIOA , GPIO_PIN_4 ); // led DOT
      1s
12      second ++;
13      if ( second >= 60){
14        second = 0;
15        minute ++;
16      }
17      if( minute >= 60){
18        minute = 0;
19        hour ++;
20      }
21      if( hour >=24) hour = 0;
22    }
23    updateClockBuffer();
24    setTimer0(1000); // HAL_Delay(1000)
25  }
```

Program 2.14: Code in main function

Link github: `https://github.com/PNgocmanh/Microcontroller_1813045.git`

CHAPTER 3

Buttons/Switches



1 Objectives

In this lab, you will

- Learn how to add new C source files and C header files in an STM32 project,
- Learn how to read digital inputs and display values to LEDs using a timer interrupt of a microcontroller (MCU).
- Learn how to debounce when reading a button.
- Learn how to create an FSM and implement an FSM in an MCU.

2 Introduction

Embedded systems usually use buttons (or keys, or switches, or any form of mechanical contacts) as part of their user interface. This general rule applies from the most basic remote-control system for opening a garage door, right up to the most sophisticated aircraft autopilot system. Whatever the system you create, you need to be able to create a reliable button interface.
A button is generally hooked up to an MCU so as to generate a certain logic level when pushed or closed or “active" and the opposite logic level when unpushed or open or “inactive." The active logic level can be either ‘0’ or ‘1’, but for reasons both historical and electrical, an active level of ’0’ is more common.
We can use a button if we want to perform operations such as:

- Drive a motor while a switch is pressed.
- Switch on a light while a switch is pressed.
- Activate a pump while a switch is pressed.

These operations could be implemented using an electrical button without using an MCU; however, use of an MCU may well be appropriate if we require more complex behaviours. For example:

- Drive a motor while a switch is pressed.

  Condition: If the safety guard is not in place, don’t turn the motor. Instead sound a buzzer for 2 seconds.

- Switch on a light while a switch is pressed.

  Condition: To save power, ignore requests to turn on the light during daylight hours.

- Activate a pump while a switch is pressed

  Condition: If the main water reservoir is below 300 litres, do not start the main pump: instead, start the reserve pump and draw the water from the emergency tank.

In this lab, we consider how you read inputs from mechanical buttons in your embedded application using an MCU.

3 Basic techniques for reading from port pins

3.1 The need for pull-up resistors

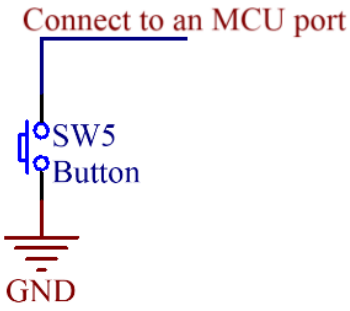


Figure 3.1: Connecting a button to an MCU

Figure 3.1 shows a way to connect a button to an MCU. This hardware operates as follows:

- When the switch is open, it has no impact on the port pin. An internal resistor on the port “pulls up" the pin to the supply voltage of the MCU (typically 3.3V for STM32F103). If we read the pin, we will see the value ‘1’.
- When the switch is closed (pressed), the pin voltage will be 0V. If we read the pin, we will see the value ‘0’.

However, if the MCU does not have a pull-up resistor inside, when the button is pressed, the read value will be ‘0’, but even we release the button, the read value is still ‘0’ as shown in Figure 3.2.

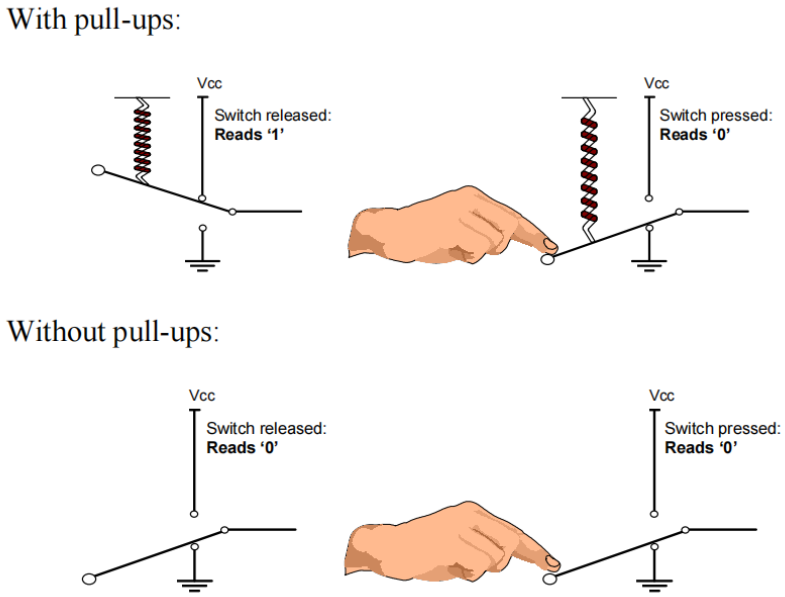


Figure 3.2: The need of pull up resistors

So a reliable way to connect a button/switch to an MCU is that we explicitly use an external pull-up resistor as shown in Figure 3.3.

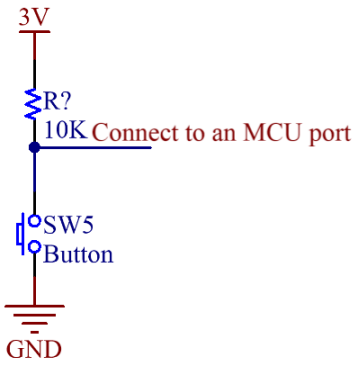


Figure 3.3: A reliable way to connect a button to an MCU

3.2 Dealing with switch bounces

In practice, all mechanical switch contacts bounce (that is, turn on and off, repeatedly, for short period of time) after the switch is closed or opened as shown in Figure 3.4.

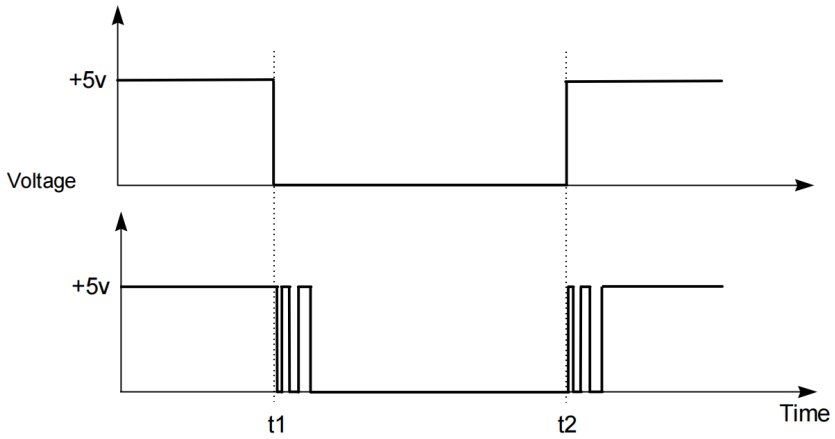


Figure 3.4: Switch bounces

Every system that uses any kind of mechanical switch must deal with the issue of debouncing. The key task is to make sure that one mechanical switch or button action is only read as one action by the MCU, even though the MCU will typically be fast enough to detect the unwanted switch bounces and treat them as separate events. Bouncing can be eliminated by special ICs or by RC circuitry, but in most cases debouncing is done in software because software is “free".
As far as the MCU concerns, each “bounce" is equivalent to one press and release of an “ideal” switch. Without appropriate software design, this can give several problems:

- Rather than reading ‘A’ from a keypad, we may read ‘AAAAA’
- Counting the number of times that a switch is pressed becomes extremely difficult
- If a switch is depressed once, and then released some time later, the ‘bounce’ may make it appear as if the switch has been pressed again (at the time of release).

The key to debouncing is to establish a minimum criterion for a valid button push, one that can be implemented in software. This criterion must involve differences in time - two

button presses in 20ms must be treated as one button event, while two button presses in 2 seconds must be treated as two button events. So what are the relevant times we need to consider? They are these:

- Bounce time: most buttons seem to stop bouncing within 10ms
- Button press time: the shortest time a user can press and release a button seems to be between 50 and 100ms
- Response time: a user notices if the system response is 100ms after the button press, but not if it is 50ms after

Combining all of these times, we can set a few goals

- Ignore all bouncing within 10ms
- Provide a response within 50ms of detecting a button push (or release)
- Be able to detect a 50ms push and a 50ms release

The simplest debouncing method is to examine the keys (or buttons or switches) every N milliseconds, where N > 10ms (our specified button bounce upper limit) and N <= 50ms (our specified response time). We then have three possible outcomes every time we read a button:

- We read the button in the solid '0' state
- We read the button in the solid '1' state
- We read the button while it is bouncing (so we will get either a '0' or a '1')

Outcomes 1 and 2 pose no problems, as they are what we would always like to happen. Outcome 3 also poses no problem because during a bounce either state is acceptable. If we have just pressed an active-low button and we read a '1' as it bounces, the next time through we are guaranteed to read a '0' (remember, the next time through all bouncing will have ceased), so we will just detect the button push a bit later. Otherwise, if we read a '0' as the button bounces, it will still be '0' the next time after all bouncing has stopped, so we are just detecting the button push a bit earlier. The same applies to releasing a button. Reading a single bounce (with all bouncing over by the time of the next read) will never give us an invalid button state. It's only reading multiple bounces (multiple reads while bouncing is occurring) that can give invalid button states such as repeated push signals from one physical push.
So if we guarantee that all bouncing is done by the time we next read the button, we're good. Well, almost good, if we're lucky...
MCUs often live among high-energy beasts, and often control the beasts. High energy devices make electrical noise, sometimes great amounts of electrical noise. This noise can, at the worst possible moment, get into your delicate button-and-high-value-pullup circuit and act like a real button push. Oops, missile launched, sorry!
If the noise is too intense we cannot filter it out using only software, but will need hardware of some sort (or even a redesign). But if the noise is only occasional, we can filter it out in software without too much bother. The trick is that instead of regarding a single button 'make' or 'break' as valid, we insist on N contiguous makes or breaks to mark a valid button event. N will be a factor of your button scanning rate and the amount of

filtering you want to add. Bigger N gives more filtering. The simplest filter (but still a big improvement over no filtering) is just an N of 2, which means compare the current button state with the last button state, and only if both are the same is the output valid.
Note that now we have not two but three button states: active (or pressed), inactive (or released), and indeterminate or invalid (in the middle of filtering, not yet filtered). In most cases we can treat the invalid state the same as the inactive state, since we care in most cases only about when we go active (from whatever state) and when we cease being active (to inactive or invalid). With that simplification we can look at simple N = 2 filtering reading a button wired to STM32 MCU:

```
void button_reading(void){
    static unsigned char last_button;
    unsigned char raw_button;
    unsigned char filtered_button;
    last_button = raw_button;
    raw_button = HAL_GPIO_ReadPin(BUTTON_1_GPIO_Port,
   BUTTON_1_Pin);
    if(last_button == raw_button){
        filtered_button = raw_button;
    }
}
```

Program 3.1: Read port pin and deboucing

The function button_reading() must be called no more often than our debounce time (10ms).
To expand to greater filtering (larger N), keep in mind that the filtering technique essentially involves reading the current button state and then either counting or reseting the counter. We count if the current button state is the same as the last button state, and if our count reaches N we then report a valid new button state. We reset the counter if the current button state is different than the last button state, and we then save the current button state as the new button state to compare against the next time. Also note that the larger our value of N the more often our filtering routine must be called, so that we get a filtered response within our specified 50ms deadline. So for example with an N of 8 we should be calling our filtering routine every 2 - 5ms, giving a response time of 16 - 40ms (>10ms and <50ms).

4 Reading switch input (basic code) using STM32

To demonstrate the use of buttons/switches in STM32, we use an example which requires to write a program that

- Has a timer which has an interrupt in every 10 milliseconds.
- Reads values of button PB0 every 10 milliseconds.
- Increases the value of LEDs connected to PORTA by one unit when the button PB0 is pressed.
- Increases the value of PORTA automatically in every 0.5 second, if the button PB0 is pressed in more than 1 second.

4.1 Input Output Processing Patterns

For both input and output processing, we have a similar pattern to work with. Normally, we have a module named driver which works directly to the hardware. We also have a buffer to store temporarily values. In the case of input processing, the driver will store the value of the hardware status to the buffer for further processing. In the case of output processing, the driver uses the buffer data to output to the hardware.

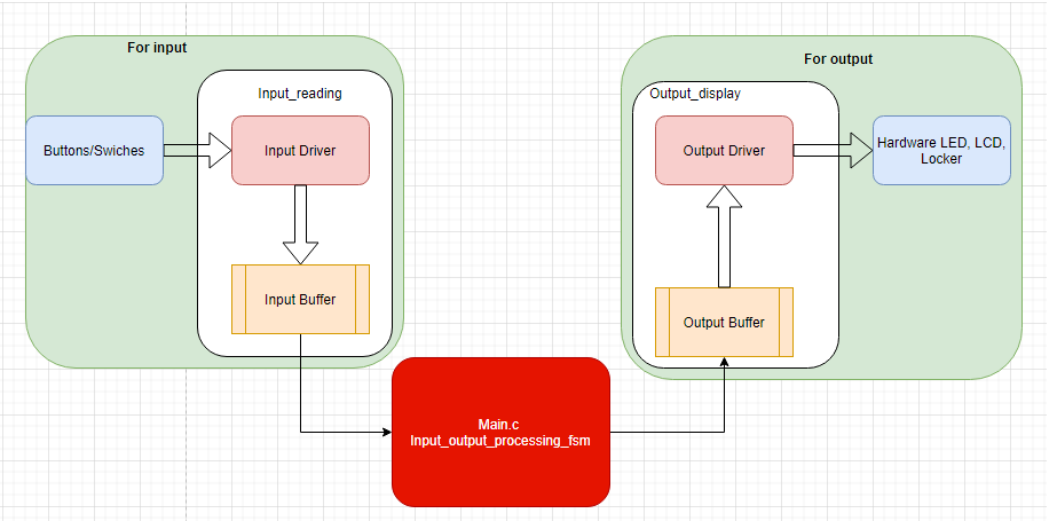


Figure 3.5: Input Output Processing Patterns

Figure 3.5 shows that we should have an *input_reading* module to processing the buttons, then store the processed data to the buffer. Then a module of *input_output_processing_fsm* will process the input data, and update the output buffer. The output driver gets the value from the output buffer to transfer to the hardware.

4.2 Setting up

4.2.1 Create a project

Please follow the instruction in Labs 1 and 2 to create a project that includes:

- PB0 as an input port pin,
- PA0-PA7 as output port pins, and
- Timer 2 10ms interrupt

4.2.2 Create a file C source file and header file for input reading

We are expected to have files for button processing and led display as shown in Figure 3.6.

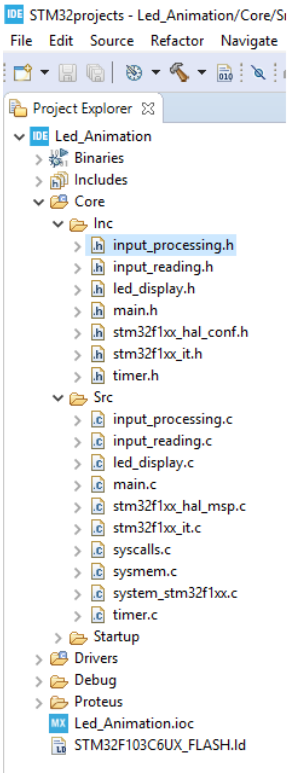


Figure 3.6: File Organization

Steps 1 (Figure 3.7): Right click to the folder **Src**, select **New**, then select **Source File**. There will be a pop-up. Please type the file name, then click **Finish**.
Step 2 (Figure 3.8): Do the same for the C header file in the folder **Inc**.

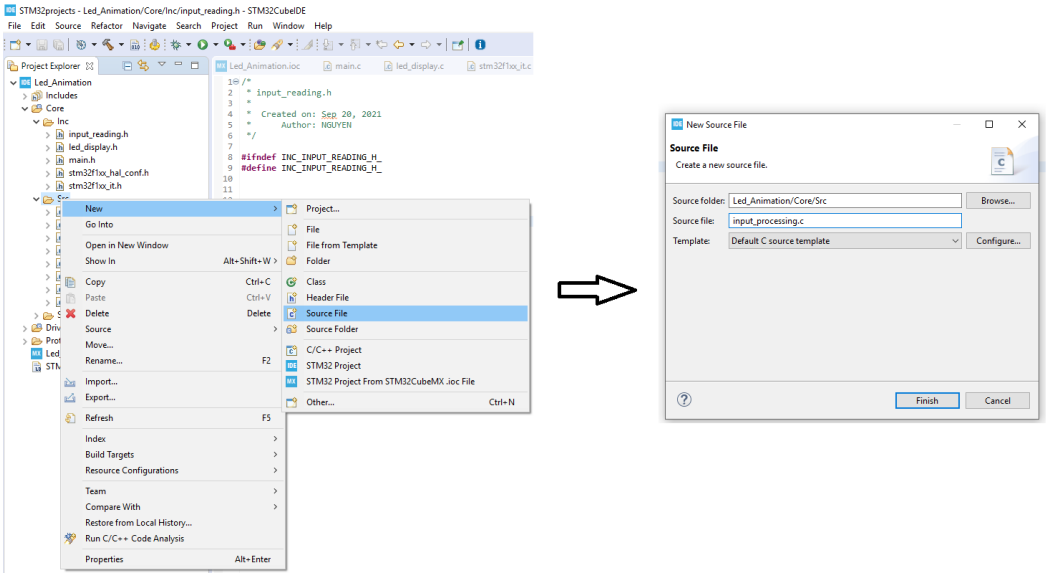


Figure 3.7: Step 1: Create a C source file for input reading

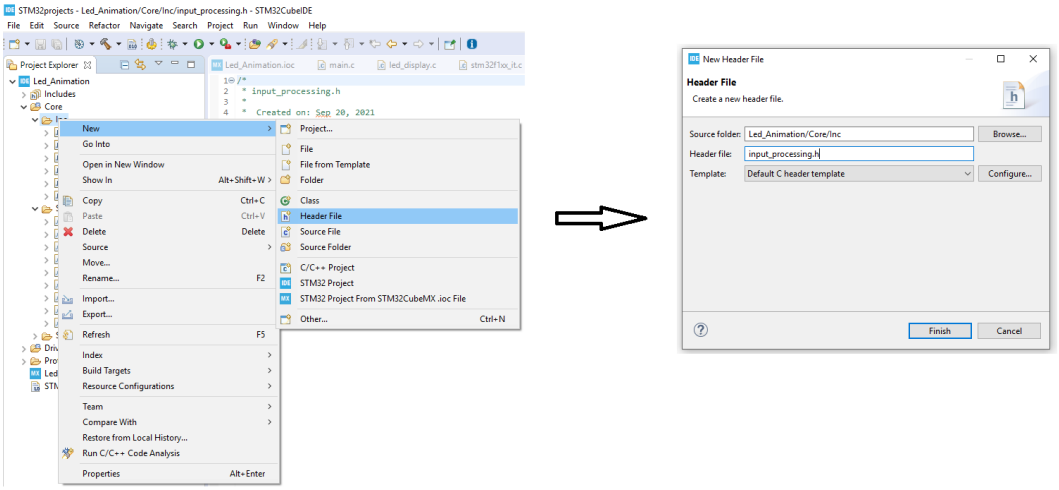


Figure 3.8: Step 2: Create a C header file for input processing

4.3 Code For Read Port Pin and Debouncing

4.3.1 The code in the input_reading.c file

```
#include "main.h"
//we aim to work with more than one buttons
#define NO_OF_BUTTONS                    1
//timer interrupt duration is 10ms, so to pass 1 second,
//we need to jump to the interrupt service routine 100 time
#define DURATION_FOR_AUTO_INCREASING     100
#define BUTTON_IS_PRESSED                  GPIO_PIN_RESET
#define BUTTON_IS_RELEASED                 GPIO_PIN_SET
//the buffer that the final result is stored after
//debouncing
static GPIO_PinState buttonBuffer[NO_OF_BUTTONS];
//we define two buffers for debouncing
static GPIO_PinState debounceButtonBuffer1[NO_OF_BUTTONS];
static GPIO_PinState debounceButtonBuffer2[NO_OF_BUTTONS];
//we define a flag for a button pressed more than 1 second.
static uint8_t flagForButtonPress1s[NO_OF_BUTTONS];
//we define counter for automatically increasing the value
//after the button is pressed more than 1 second.
static uint16_t counterForButtonPress1s[NO_OF_BUTTONS];
void button_reading(void){
  for(char i = 0; i < NO_OF_BUTTONS; i ++){
    debounceButtonBuffer2[i] =debounceButtonBuffer1[i];
    debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(
   BUTTON_1_GPIO_Port, BUTTON_1_Pin);
    if(debounceButtonBuffer1[i] == debounceButtonBuffer2[i
   ])
       buttonBuffer[i] = debounceButtonBuffer1[i];
       if(buttonBuffer[i] == BUTTON_IS_PRESSED){
       //if a button is pressed, we start counting
         if(counterForButtonPress1s[i] <
   DURATION_FOR_AUTO_INCREASING){
           counterForButtonPress1s[i]++;
         } else {
         //the flag is turned on when 1 second has passed
         //since the button is pressed.
           flagForButtonPress1s[i] = 1;
           //todo
         }
       } else {
         counterForButtonPress1s[i] = 0;
         flagForButtonPress1s[i] = 0;
       }
  }
}
```

Program 3.2: Define constants buffers and button_reading function

```c
unsigned char is_button_pressed(uint8_t index){
  if(index >= NO_OF_BUTTONS) return 0;
  return (buttonBuffer[index] == BUTTON_IS_PRESSED);
}
```

Program 3.3: Checking a button is pressed or not

```c
unsigned char is_button_pressed_1s(unsigned char index){
  if(index >= NO_OF_BUTTONS) return 0xff;
  return (flagForButtonPress1s[index] == 1);
}
```

Program 3.4: Checking a button is pressed more than a second or not

4.3.2 The code in the input_reading.h file

```c
#ifndef INC_INPUT_READING_H_
#define INC_INPUT_READING_H_
void button_reading(void);
unsigned char is_button_pressed(unsigned char index);
unsigned char is_button_pressed_1s(unsigned char index);
#endif /* INC_INPUT_READING_H_ */
```

Program 3.5: Prototype in input_reading.h file

4.3.3 The code in the timer.c file

```c
#include "main.h"
#include "input_reading.h"

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if(htim->Instance == TIM2){
    button_reading();
  }
}
```

Program 3.6: Timer interrupt callback function

4.4 Button State Processing

4.4.1 Finite State Machine

To solve the example problem, we define 3 states as follows:

- State 0: The button is released or the button is in the initial state.
- State 1: When the button is pressed, the FSM will change to State 1 that is increasing the values of PORTA by one value. If the button is released, the FSM goes back to State 0.
- State 2: while the FSM is in State 1, the button is kept pressing more than 1 second, the state of FSM will change from 1 to 2. In this state, if the button is kept pressing, the value of PORTA will be increased automatically in every 500ms. If the button is released, the FSM goes back to State 0.

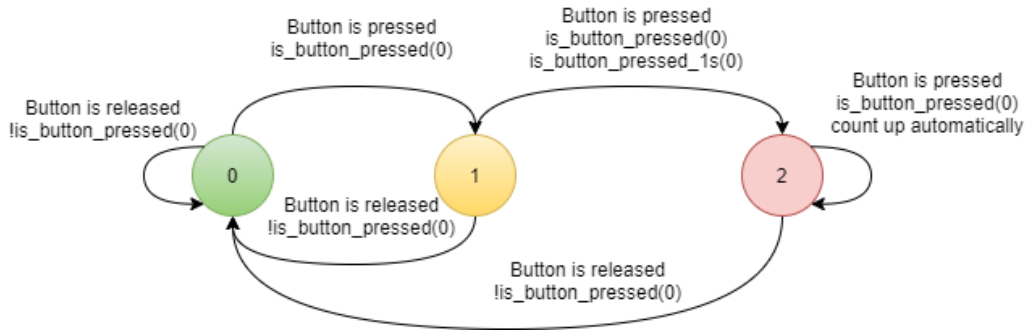


Figure 3.9: An FSM for processing a button

4.4.2 The code for the FSM in the input_processing.c file

Please note that *fsm_for_input_processing* function should be called inside the super loop of the main functin.

```
#include "main.h"
#include "input_reading.h"

enum ButtonState{BUTTON_RELEASED, BUTTON_PRESSED,
   BUTTON_PRESSED_MORE_THAN_1_SECOND} ;
enum ButtonState buttonState = BUTTON_RELEASED;
void fsm_for_input_processing(void){
  switch(buttonState){
  case BUTTON_RELEASED:
    if(is_button_pressed(0)){
      buttonState = BUTTON_PRESSED;
      //INCREASE VALUE OF PORT A BY ONE UNIT
    }
    break;
  case BUTTON_PRESSED:
    if(!is_button_pressed(0)){
      buttonState = BUTTON_RELEASED;
    } else {
      if(is_button_pressed_1s(0)){
        buttonState = BUTTON_PRESSED_MORE_THAN_1_SECOND;
      }
    }
    break;
  case BUTTON_PRESSED_MORE_THAN_1_SECOND:
    if(!is_button_pressed(0)){
      buttonState = BUTTON_RELEASED;
    }
    //todo
    break;
  }
}
```

Program 3.7: The code in the input_processing.c file

4.4.3 The code in the input_processing.h

```
#ifndef INC_INPUT_PROCESSING_H_
#define INC_INPUT_PROCESSING_H_

void fsm_for_input_processing(void);

#endif /* INC_INPUT_PROCESSING_H_ */
```

Program 3.8: Code in the input_processing.h file

4.4.4 The code in the main.c file

```c
#include "main.h"
#include "input_processing.h"
//don't modify this part
int main(void){
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_TIM2_Init();
    while (1)
    {
        //you only need to add the fsm function here
        fsm_for_input_processing();
    }
}
```

Program 3.9: The code in the main.c file

5 Exercises and Report

5.1 Specifications

You are required to build an application of a traffic light in a cross road which includes some features as described below:

- The application has 12 LEDs including 4 red LEDs, 4 amber LEDs, 4 green LEDs.
- The application has 4 seven segment LEDs to display time with 2 for each road. The 2 seven segment LEDs will show time for each color LED corresponding to each road.
- The application has three buttons which are used

  - to select modes,

  - to modify the time for each color led on the fly, and

  - to set the chosen value.

- The application has at least 4 modes which is controlled by the first button. Mode 1 is a normal mode, while modes 2 3 4 are modification modes. You can press the first button to change the mode. Modes will change from 1 to 4 and back to 1 again.

  Mode 1 - Normal mode:

  - The traffic light application is running normally.

  Mode 2 - Modify time duration for the red LEDs: This mode allows you to change the time duration of the red LED in the main road. The expected behaviours of this mode include:

  - All single red LEDs are blinking in 2 Hz.

  - Use two seven-segment LEDs to display the value.

  - Use the other two seven-segment LEDs to display the mode.

  - The second button is used to increase the time duration value for the red LEDs.

  - The value of time duration is in a range of 1 - 99.

  - The third button is used to set the value.

  Mode 3 - Modify time duration for the amber LEDs: Similar for the red LEDs described above with the amber LEDs.

  Mode 4 - Modify time duration for the green LEDs: Similar for the red LEDs described above with the green LEDs.

5.2 Exercise 1: Sketch an FSM

Your task in this exercise is to sketch an FSM that describes your idea of how to solve the problem.
To solve the example problem, I define 4 states as follows:

- State 0(mode 1): The button A is released or the button A is in the initial state.
- State 1(mode 2): When the button A is pressed, the FSM will change to Mode 2, then pressed button B to change value of led red and pressed button C to set this value.
- State 2(mode 3): When the button A is pressed, the FSM will change to Mode 3, then pressed button B to change value of led amber and pressed button C to set this value.
- State 3(mode 4): When the button A is pressed, the FSM will change to Mode 4, then pressed button B to change value of led green and pressed button C to set this value. If button A is pressed one more time, FSM will be change back to Mode 1.

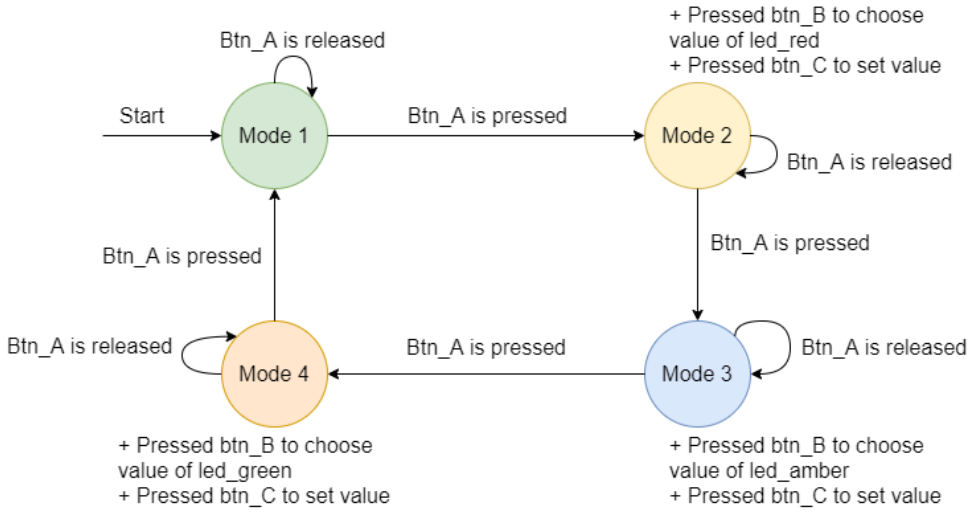


Figure 3.10: A FSM for processing button

5.3 Exercise 2: Proteus Schematic

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

- PA1 - PA6 is output port pin, connect led red, green, amber in two streets.
- PA0, PA14, PA15 is input port pin, connect each button.
- PA7 - PA13 and PB0 is output port pin, connect first led7segment.
- PB1 - PB7 is output port pin, connect second and fourth led7segment.
- PB8 - PB14 is output port pin, connect third led7segment.

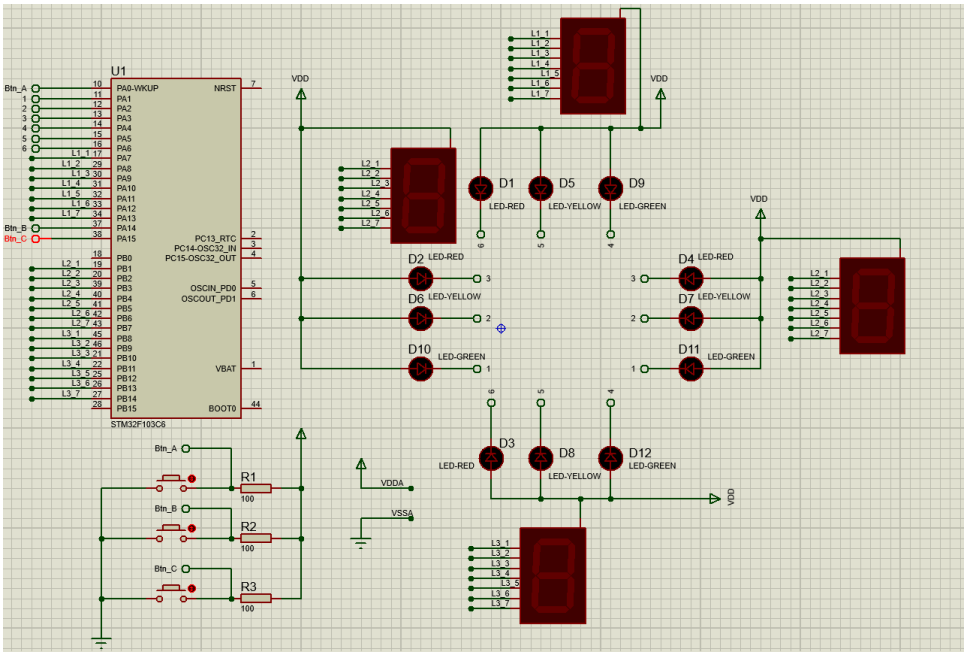


Figure 3.11: The schematic in Proteus.

5.4 Exercise 3: Create STM32 Project

Your task in this exercise is to create a project that has pin corresponding to the Proteus schematic that you draw in previous section. You need to set up your timer interrupt is about 10ms.

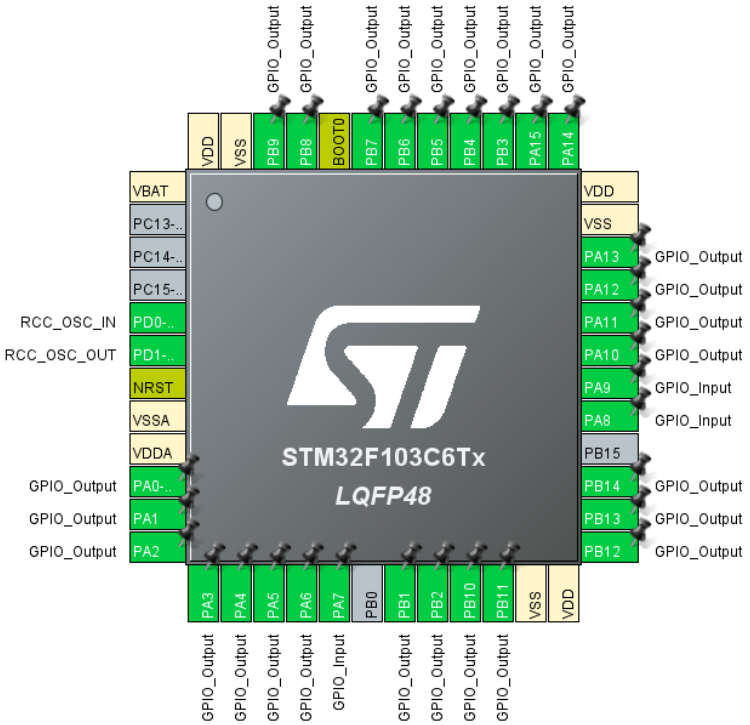


Figure 3.12: Create project STM32 corresponding to the Proteus schematic in exercise 2.

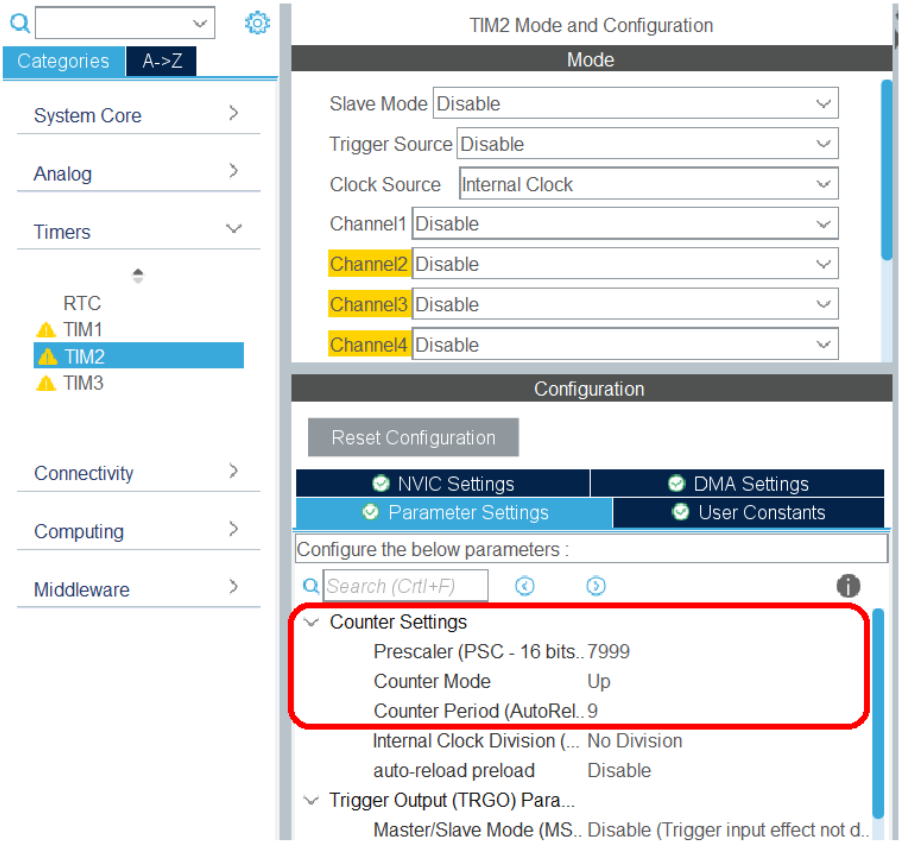


Figure 3.13: Set up timer interrupt is about 10ms.

5.5 Exercise 4: Modify Timer Parameters

Your task in this exercise is to modify the timer settings so that when we want to change the time duration of the timer interrupt, we change it the least and it will not affect the overall system. For example, the current system we have implemented is that it can blink an LED in 2 Hz, with the timer interrupt duration is 10ms. However, when we want to change the timer interrupt duration to 1ms or 100ms, it will not affect the 2Hz blinking LED.

```
#ifndef INC_TIMER_H_
#define INC_TIMER_H_

#include "main.h"
#define TIMER_INTERUPT 10 // 10ms

#define TIMER_1s 1000 // 1000ms = 1s
static int count_1s = TIMER_1s / TIMER_INTERUPT ;

#define TIMER_blinking 500 // 500ms = 0.5s <=> 2Hz
static int count_500ms = TIMER_blinking / TIMER_INTERUPT ;

#endif /* INC_TIMER_H_ */
```

Program 3.10: The code in timer.h

5.6 Exercise 5: Adding code for button debouncing

Following the example of button reading and debouncing in the previous section, your tasks in this exercise are:

- To add new files for input reading and output display,
- To add code for button debouncing,
- To add code for increasing mode when the first button is pressed.

5.6.1 Input reading file

Code in input reading.h

```
#ifndef INC_INPUT_READING_H_
#define INC_INPUT_READING_H_

// the buffer that the final result is stored after
# define NO_OF_BUTTONS 3

# define DURATION_FOR_AUTO_INCREASING 100
# define BUTTON_IS_PRESSED GPIO_PIN_RESET
# define BUTTON_IS_RELEASED GPIO_PIN_SET

void button_reading ( void );
void getButton ( void );
unsigned char is_button_pressed ( unsigned char index );
unsigned char is_button_pressed_1s ( unsigned char index );

#endif /* INC_INPUT_READING_H_ */

#endif /* INC_INPUT_READING_H_ */
```

Program 3.11: The code in input reading.h

Code in input_reading.c

```
#include "main.h"
#include "input_reading.h"
#include "led_display.h"
GPIO_PinState buttonBuffer [ NO_OF_BUTTONS ];
// we define two buffers for
GPIO_PinState debounceButtonBuffer1 [ NO_OF_BUTTONS ];
GPIO_PinState debounceButtonBuffer2 [ NO_OF_BUTTONS ];

uint16_t pinButton [ NO_OF_BUTTONS ];

// we define a flag for a button pressed more than 1 second
static uint8_t flagForButtonPress1s [ NO_OF_BUTTONS ];
// we define counter for automatically increasing the value
// after the button is pressed more than 1 second .
static uint16_t counterForButtonPress1s [ NO_OF_BUTTONS ];
```

```
16
17 void getButton ( void ){
18     pinButton [0] = GPIO_PIN_0 ;
19     pinButton [1] = GPIO_PIN_14 ;
20     pinButton [2] = GPIO_PIN_15 ;
21     for (int i = 0 ; i < NO_OF_BUTTONS ; i++) {
22         debounceButtonBuffer1 [i] = BUTTON_IS_RELEASED ;
23         debounceButtonBuffer2 [i]= BUTTON_IS_RELEASED ;
24         buttonBuffer [i]= BUTTON_IS_RELEASED ;
25     }
26 }
27
28 void button_reading ( void ){
29     for ( char i = 0; i < NO_OF_BUTTONS ; i ++){
30         debounceButtonBuffer2 [i] = debounceButtonBuffer1 [
    i];
31         debounceButtonBuffer1 [i] = HAL_GPIO_ReadPin (GPIOA
     ,
32         pinButton [i]);
33         if( debounceButtonBuffer1 [i] ==
    debounceButtonBuffer2 [i])
34             buttonBuffer [i] = debounceButtonBuffer1 [i];
35             if( buttonBuffer [i] == BUTTON_IS_PRESSED ){
36                 // if a button is pressed , we start
    counting
37                 if( counterForButtonPress1s [i] <
    DURATION_FOR_AUTO_INCREASING ){
38                     counterForButtonPress1s [i ]++;
39                 } else {
40                 // the flag is turned on when 1 second has
    passed
41                 // since the button is pressed .
42                 flagForButtonPress1s [i] = 1;
43                 // todo
44                 }
45             } else {
46                 counterForButtonPress1s [i] = 0;
47                 flagForButtonPress1s [i] = 0;
48             }
49     }
50 }
51
52 unsigned char is_button_pressed ( uint8_t index ){
53     if( index >= NO_OF_BUTTONS ) return 0;
54     return ( buttonBuffer [ index ] == BUTTON_IS_PRESSED );
55 }
56 unsigned char is_button_pressed_1s ( unsigned char index ){
57     if( index >= NO_OF_BUTTONS ) return 0xff;
58     return ( flagForButtonPress1s [ index ] == 1);
```

```
}
```

Program 3.12: The code in input reading.c

Code in input_processing.h

```
#ifndef INC_INPUT_PROCESSING_H_
#define INC_INPUT_PROCESSING_H_

void fsm_for_input_processing ( void );

#endif /* INC_INPUT_PROCESSING_H_ */
```

Program 3.13: Source code in input_processing.h

Code in input_processing.c

```
void increase_mode (){
    if( mode > 4 ) mode = 1;
    else mode ++;
}
```

Program 3.14: Function increase_mode().

```
void reset_mode (){
    // reset time_duration after transform to a mode and
   reset timer each when we didnt complete time s run
    cycle at mode 1
    for ( int i = 0; i < 3 ; i++){
        street2_arr [i] = arr2 [i];
        street1_arr [i] = arr2 [i];
    }
    time_duration = 0;
}
```

Program 3.15: Function reset$_mode$().

```
# include "main.h"
# include "input_processing.h"
# include "input_reading.h"
# include "led_display.h"

enum ButtonState { BUTTON_RELEASED , BUTTON_PRESSED ,
   BUTTON_PRESSED_MORE_THAN_1_SECOND } ;
enum ButtonState buttonState [3] = { BUTTON_RELEASED ,
   BUTTON_RELEASED , BUTTON_RELEASED };
int time_duration = 0;
int mode = 0;
void fsm_for_input_processing ( void ){
    for ( int i = 0 ; i < NO_OF_BUTTONS ; i ++) {
        switch ( buttonState [i] ){
```

```
13              case BUTTON_RELEASED :
14                  if( is_button_pressed (i)){
15                      buttonState [i] = BUTTON_PRESSED ;
16                      // INCREASE VALUE OF PORT A BY ONE UNIT
17                      if(i == 0){ // first button .
18                          increase_mode ();
19                          if( mode > 1) reset_mode ();
20                      }
21                  }
22                  break ;
23              case BUTTON_PRESSED :
24                  if (! is_button_pressed (i)){
25                      buttonState [i] = BUTTON_RELEASED ;
26                  } else {
27                      if( is_button_pressed_1s (i) )
28                          buttonState [i] =
    BUTTON_PRESSED_MORE_THAN_1_SECOND ;
29                  }
30                  break ;
31              case BUTTON_PRESSED_MORE_THAN_1_SECOND :
32                  if (! is_button_pressed (i)){
33                      buttonState [i] = BUTTON_RELEASED ;
34                  }
35                  // todo
36                  break ;
37          }
38      }
39  }
```

Program 3.16: Function fsm_for_input_processing().

5.7 Exercise 6: Adding code for displaying modes

Your tasks in this exercise are:

- To add code for display mode on seven-segment LEDs, and
- To add code for blinking LEDs depending on the mode that is selected.

Code in led_display.h

```
#ifndef INC_LED_DISPLAY_H_
#define INC_LED_DISPLAY_H_

int street1_arr [3];
int street2_arr [3];
int arr2 [3];
void display7SEG ( int num , GPIO_TypeDef * GPIO1 ,
   uint16_t GPIO_PIN0 , uint16_t GPIO_PIN1 , uint16_t
   GPIO_PIN2 , uint16_t GPIO_PIN3 , uint16_t GPIO_PIN4 ,
   uint16_t GPIO_PIN5 , uint16_t GPIO_PIN6 );
void traffic_light ( uint16_t GPIO_Pin1 , uint16_t
   GPIO_Pin2 ,uint16_t GPIO_Pin3 , int count , int arr []);
void traffic_light_4 (int count_vled , int count_hled );
void mode_1 ();
void mode_2 ();
void mode_3 ();
void mode_4 ();

#endif /* INC_LED_DISPLAY_H_ */
```

Program 3.17: Source in led_display.h

Code in led_display.c

```
# include "main.h"
# include "led_display.h"
int street1_arr [3] = {5 ,2 ,3};
int street2_arr [3] = {5 ,2 ,3};
int arr2 [3] = {0 ,0 ,0}; // temp arr


void display7SEG ( int num , GPIO_TypeDef * GPIO1 ,
   uint16_t GPIO_PIN0 ,uint16_t GPIO_PIN1 , uint16_t
   GPIO_PIN2 , uint16_t GPIO_PIN3 , uint16_t GPIO_PIN4 ,
   uint16_t GPIO_PIN5 , uint16_t GPIO_PIN6 )
{
    switch ( num ){
    case 0:
        HAL_GPIO_WritePin (GPIO1 , GPIO_PIN0 , RESET );
        HAL_GPIO_WritePin (GPIO1 , GPIO_PIN1 , RESET );
```

```
14          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN2 , RESET );
15          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN3 , RESET );
16          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN4 , RESET );
17          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN5 , RESET );
18          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN6 , SET );
19          break ;
20      case 1:
21          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN0 , SET );
22          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN1 , RESET );
23          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN2 , RESET );
24          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN3 , SET );
25          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN4 , SET );
26          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN5 , SET );
27          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN6 , SET );
28          break ;
29      case 2:
30          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN0 , RESET );
31          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN1 , RESET );
32          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN2 , SET );
33          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN3 , RESET );
34          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN4 , RESET );
35          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN5 , SET );
36          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN6 , RESET );
37          break ;
38      case 3:
39          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN0 , RESET );
40          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN1 , RESET );
41          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN2 , RESET );
42          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN3 , RESET );
43          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN4 , SET );
44          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN5 , SET );
45          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN6 , RESET );
46          break ;
47      case 4:
48          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN0 , SET );
49          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN1 , RESET );
50          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN2 , RESET );
51          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN3 , SET );
52          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN4 , SET );
53          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN5 , RESET );
54          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN6 , RESET );
55          break ;
56      case 5:
57          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN0 , RESET );
58          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN1 , SET );
59          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN2 , RESET );
60          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN3 , RESET );
61          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN4 , SET );
62          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN5 , RESET );
```

```
63          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN6 , RESET );
64          break ;
65      case 6:
66          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN0 , RESET );
67          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN1 , SET );
68          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN2 , RESET );
69          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN3 , RESET );
70          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN4 , RESET );
71          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN5 , RESET );
72          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN6 , RESET );
73          break ;
74      case 7:
75          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN0 , RESET );
76          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN1 , RESET );
77          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN2 , RESET );
78          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN3 , SET );
79          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN4 , SET );
80          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN5 , SET );
81          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN6 , SET );
82          break ;
83      case 8:
84          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN0 , RESET );
85          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN1 , RESET );
86          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN2 , RESET );
87          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN3 , RESET );
88          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN4 , RESET );
89          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN5 , RESET );
90          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN6 , RESET );
91          break ;
92      case 9:
93          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN0 , RESET );
94          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN1 , RESET );
95          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN2 , RESET );
96          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN3 , RESET );
97          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN4 , SET );
98          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN5 , RESET );
99          HAL_GPIO_WritePin (GPIO1 , GPIO_PIN6 , RESET );
100         break ;
101     }
102 }
103
104 void traffic_light ( uint16_t GPIO_Pin1 , uint16_t
        GPIO_Pin2 , uint16_t GPIO_Pin3 , int count , int arr [])
        {
105     switch ( count ){
106     case 0:
107         HAL_GPIO_WritePin (GPIOB , GPIO_Pin1 , RESET );
108         HAL_GPIO_WritePin (GPIOB , GPIO_Pin2 , SET);
109         HAL_GPIO_WritePin (GPIOB , GPIO_Pin3 , SET);
```

```
110         break ;
111     case 1:
112         HAL_GPIO_WritePin (GPIOB , GPIO_Pin1 , SET);
113         HAL_GPIO_WritePin (GPIOB , GPIO_Pin2 , SET);
114         HAL_GPIO_WritePin (GPIOB , GPIO_Pin3 , RESET );
115         break ;
116     case 2:
117         HAL_GPIO_WritePin (GPIOB , GPIO_Pin1 , SET);
118         HAL_GPIO_WritePin (GPIOB , GPIO_Pin2 , RESET );
119         HAL_GPIO_WritePin (GPIOB , GPIO_Pin3 , SET);
120         break ;
121     }
122     arr [ count ]--;
123 }
124 void traffic_light_4 (int count_vled , int count_hled ){
125 ////////////////////////////// HORIZONTAL LEDS
    ////////////////////////////
126     traffic_light ( GPIO_PIN_7 , GPIO_PIN_8 , GPIO_PIN_9 ,
    count_hled , street2_arr );
127
128     ////////////////////////// VERTICAL LEDS
    ////////////////////////
129     traffic_light ( GPIO_PIN_10 , GPIO_PIN_11 , GPIO_PIN_12
    , count_vled , street1_arr );
130 }
131
132 ////////////////////// MODE
    1//////////////////////////////////////
133 int count_vled = 0;
134 int count_hled = 2;
135 void mode_1 (){
136     if( count_vled >= 3) count_vled = 0;
137     if( count_hled < 0) count_hled = 2;
138         traffic_light_4 ( count_vled , count_hled );
139         display7SEG ( street1_arr [ count_vled ], GPIOB ,
    GPIO_PIN_1 , GPIO_PIN_2 , GPIO_PIN_3 , GPIO_PIN_4 ,
    GPIO_PIN_5 , GPIO_PIN_6, GPIO_PIN_7 );
140         display7SEG ( street2_arr [ count_hled ], GPIOA ,
    GPIO_PIN_7 , GPIO_PIN_8, GPIO_PIN_9, GPIO_PIN_10 ,
    GPIO_PIN_11 , GPIO_PIN_12 , GPIO_PIN_13 );
141         display7SEG ( street2_arr [ count_hled ], GPIOB ,
    GPIO_PIN_8 , GPIO_PIN_9 , GPIO_PIN_10 , GPIO_PIN_11 ,
    GPIO_PIN_12 , GPIO_PIN_13 , GPIO_PIN_14 );
142     if( street1_arr [ count_vled ] <= 1){
143         street1_arr [ count_vled ] = arr2 [ count_vled ];
144         count_vled ++;
145     }
146     if( street2_arr [ count_hled ] <= 1){
147         street2_arr [ count_hled ] = arr2 [ count_hled ];
```

```
148         count_hled --;
149     }
150 }
151 
152 ////////////////////////// MODE 2
    ////////////////////////////////
153 int counter_mode2 = 1;
154 void mode_2 (){
155     if( counter_mode2 == 1){
156         HAL_GPIO_WritePin (GPIOA , GPIO_PIN_3 | GPIO_PIN_6
    , RESET);
157         counter_mode2 = 2;
158     }
159     else {
160         HAL_GPIO_WritePin (GPIOA, GPIO_PIN_3 | GPIO_PIN_6 ,
     SET );
161         counter_mode2 = 1;
162     }
163     HAL_GPIO_WritePin (GPIOA , GPIO_PIN_1 | GPIO_PIN_2 |
    GPIO_PIN_4 | GPIO_PIN_5 , SET );
164     display7SEG (2, GPIOB , GPIO_PIN_1 , GPIO_PIN_2 ,
    GPIO_PIN_3 , GPIO_PIN_4 , GPIO_PIN_5 , GPIO_PIN_6,
    GPIO_PIN_7 );
165 }
166 
167 // ////////////// MODE 3//////////////////
168 int counter_mode3 = 1;
169 void mode_3 (){
170     if( counter_mode3 == 1){
171         HAL_GPIO_WritePin (GPIOA , GPIO_PIN_2 | GPIO_PIN_5
    , RESET);
172         counter_mode3 = 2;
173     }
174     else {
175         HAL_GPIO_WritePin (GPIOA , GPIO_PIN_2 | GPIO_PIN_5
    , SET );
176         counter_mode3 = 1;
177     }
178     HAL_GPIO_WritePin (GPIOA , GPIO_PIN_1 | GPIO_PIN_3 |
    GPIO_PIN_4 | GPIO_PIN_6 , SET );
179     display7SEG (3, GPIOB , GPIO_PIN_1 , GPIO_PIN_2 ,
    GPIO_PIN_3 , GPIO_PIN_4 , GPIO_PIN_5 , GPIO_PIN_6,
    GPIO_PIN_7 );
180 }
181 
182 // ////////////// MODE 4////////////////
183 int counter_mode4 = 1;
184 void mode_4 (){
185     if( counter_mode4 == 1){
```

```
186         HAL_GPIO_WritePin (GPIOA , GPIO_PIN_1 | GPIO_PIN_4
    , RESET);
187         counter_mode4 = 2;
188      }
189      else {
190         HAL_GPIO_WritePin (GPIOA , GPIO_PIN_1 | GPIO_PIN_4
    , SET );
191         counter_mode4 = 1;
192      }
193      HAL_GPIO_WritePin (GPIOA , GPIO_PIN_7 | GPIO_PIN_9 |
    GPIO_PIN_10 | GPIO_PIN_12 , SET );
194      display7SEG (4, GPIOB , GPIO_PIN_1 , GPIO_PIN_2 ,
    GPIO_PIN_3 , GPIO_PIN_4 , GPIO_PIN_5 , GPIO_PIN_6 ,
    GPIO_PIN_7);
195 }
```

Program 3.18: Source code in led_display.c

Explain:
Create three array street1_arr[3], street2_arr[3], arr2[3] to hold time duration Leds,Create 4 functions mode_1(),mode_2(),mode_3(),mode_4() to display four Mode. In mode 1, display normal reuse code in Lab1.In mode 2,3,4 create a global variable counter_mode with value = 1 (the counter_mode = 1 will turn on, counter_mode = 2 will turn off) and turn off remains four Leds.
Create a function void mode_2() to display led red and display value 2 in two street1 LED 7 SEGMENT. To guaranty timer turn on of the led RED is 500ms, create a int counter_mode2 , when int counter_mode2 = 1 ,led RED is will turn all,when int counter_mode2 = 2 led RED will turn off.
Create a function void mode_3() like void mode_2(). Led Amber turn all.
Similarly, create a function void mode_4() like void mode_2().Led Green turn all.

5.8 Exercise 7: Adding code for increasing time duration value for the red LEDs

Your tasks in this exercise are:

- to use the second button to increase the time duration value of the red LEDs
- to use the third button to set the value for the red LEDs.

Please add your report here.

5.9 Exercise 8: Adding code for increasing time duration value for the amber LEDs

Your tasks in this exercise are:

- to use the second button to increase the time duration value of the amber LEDs
- to use the third button to set the value for the amber LEDs.

Please add your report here.

5.10 Exercise 9: Adding code for increasing time duration value for the green LEDs

Your tasks in this exercise are:

- to use the second button to increase the time duration value of the green LEDs
- to use the third button to set the value for the green LEDs.

Please add your report here.

5.11 Exercise 10: To finish the project

Your tasks in this exercise are:

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

Link github: `https://github.com/PNgocmanh/Microcontroller_1813045.git`