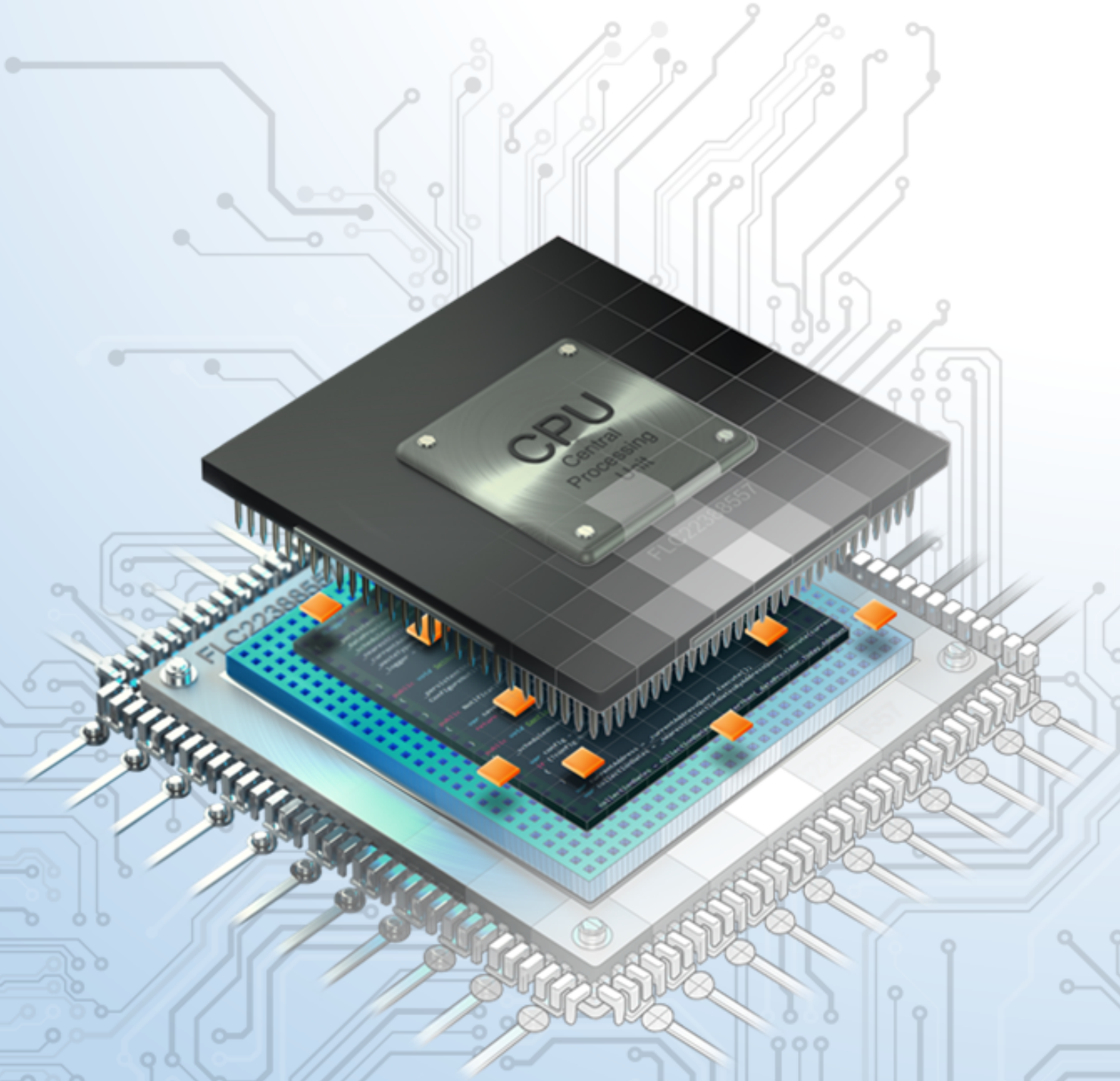
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller



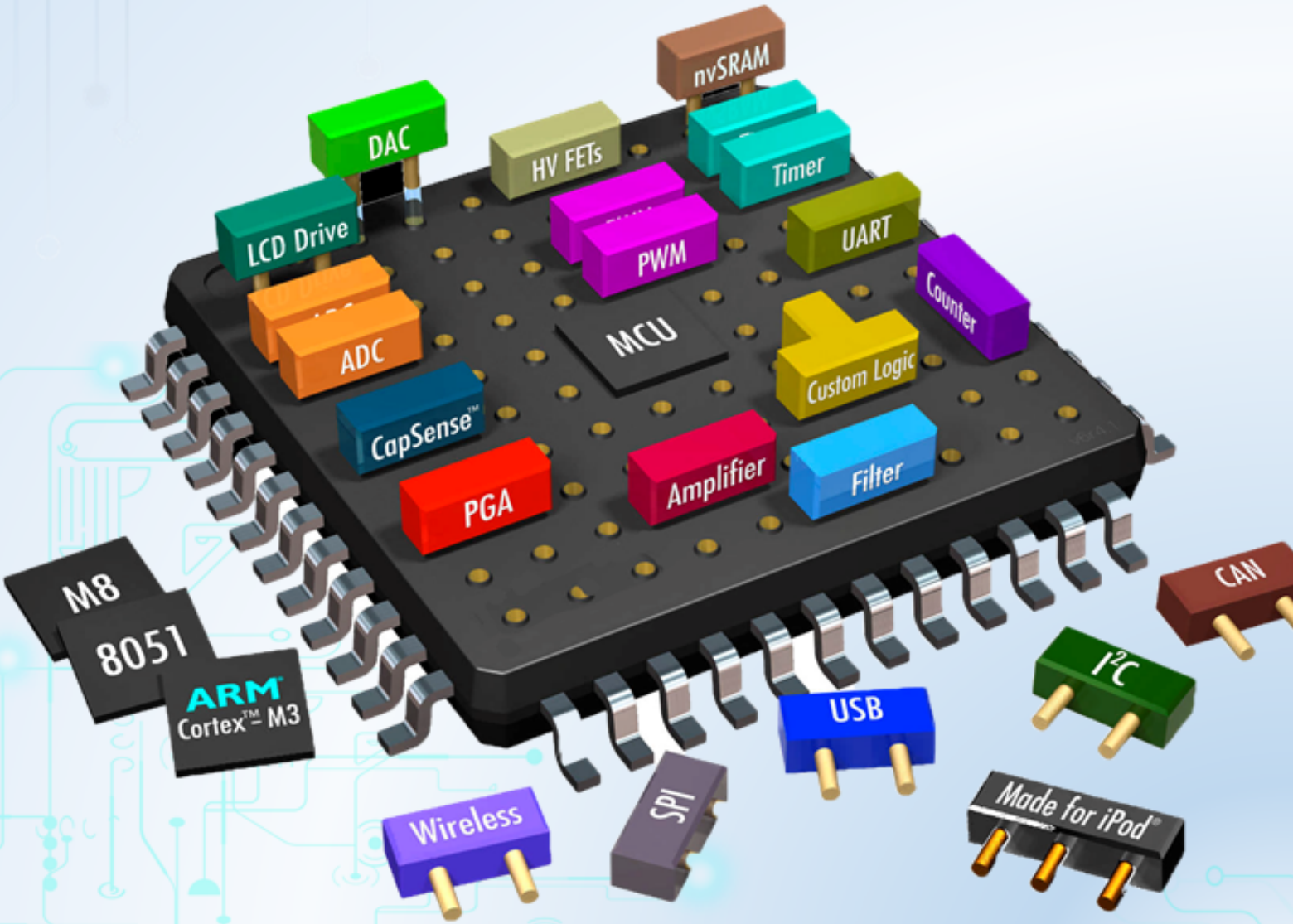
Lecturer: Dr. Le Trong Nhan

Contents

List of Figures

CHAPTER 1

LED Animations



1 Exercise 1

Report 1: Depict the schematic from Proteus simulation.

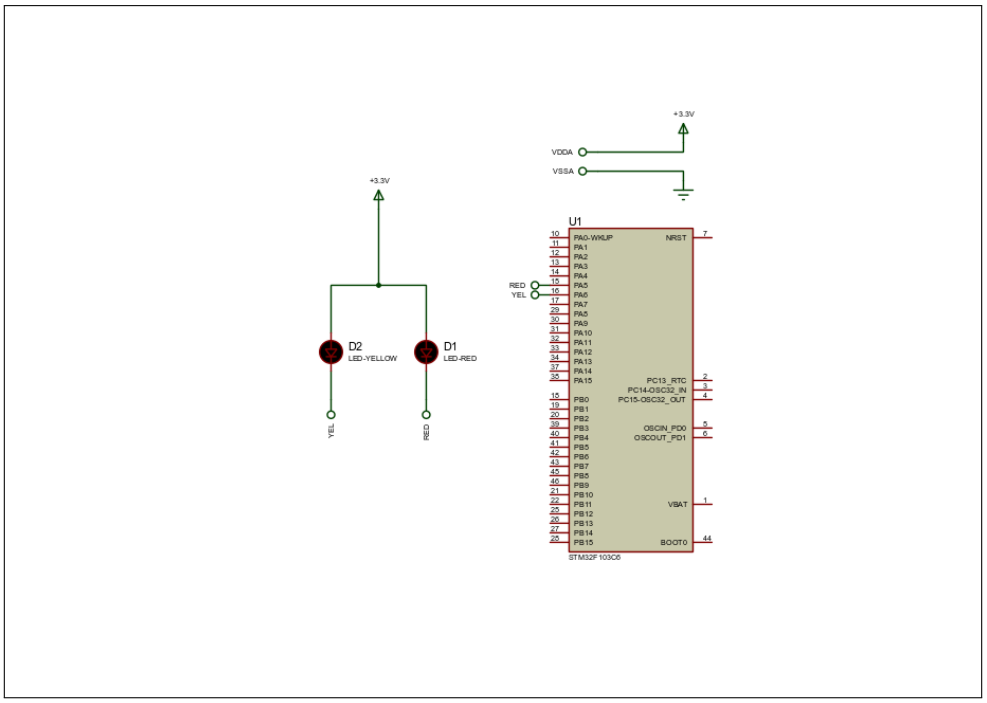


Figure 1.1: Schematic design of exercise 1 - lab 1

Report 2: Present the source code in the infinite while loop.

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
while (1)
{
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5|GPIO_PIN_6);
  HAL_Delay(2000);
}
```

Program 1.1: Source code of exercise 1 - lab 1

2 Exercise 2

Report 1: Present the schematic.

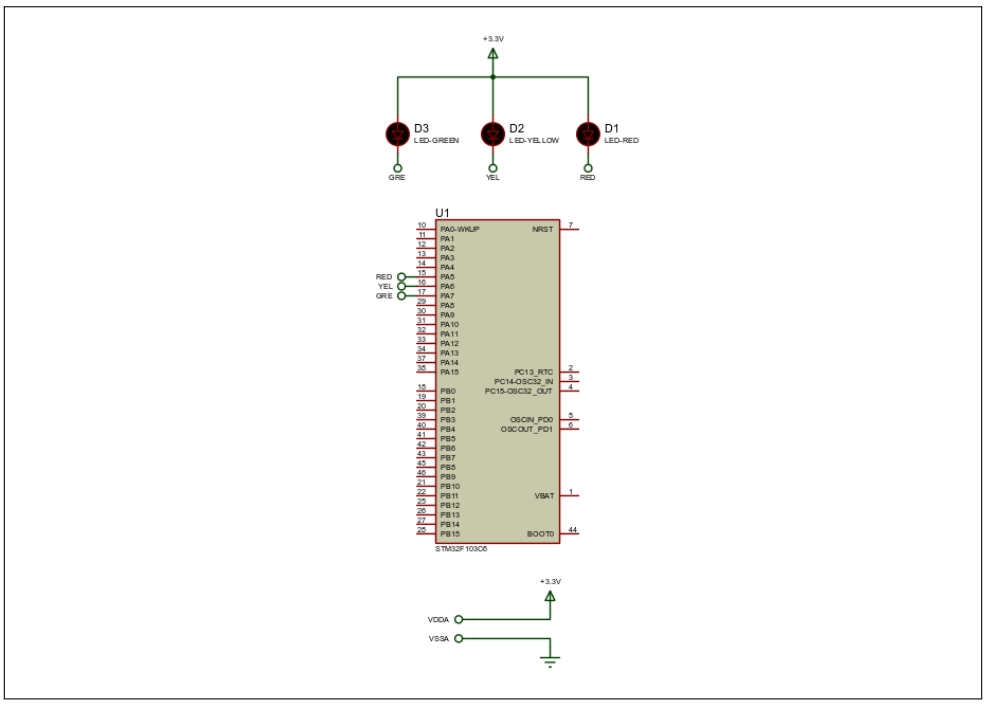


Figure 1.2: Schematic design of exercise 2 - lab 1

Report 2: Present the source code in while loop.

```
while (1)
{
 // RED
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_SET);
 HAL_Delay(5000);
 // YELOW
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
 HAL_Delay(3000);
 // GREEN
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);
 HAL_Delay(2000);
}
```

Program 1.2: Source code of exercise 2 - lab 1

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.

4 Exercise 4

Report 1: Present the schematic.

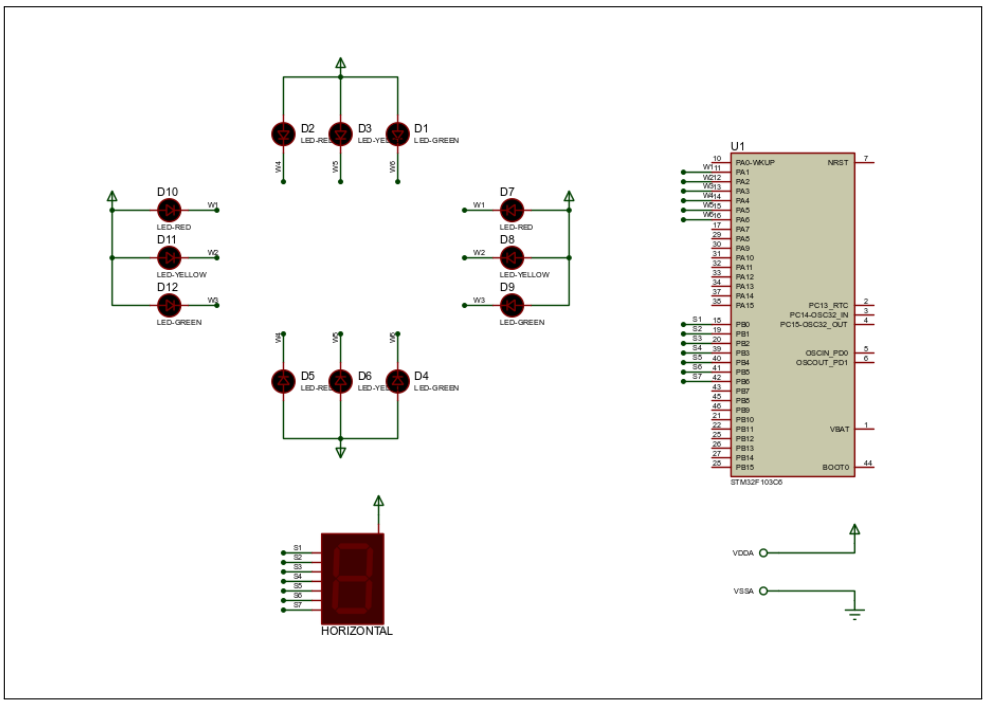


Figure 1.3: Schematic design of exercise 3-5 - lab 1

Report 2: Present the source code for display7SEG function.

```
void display7SEG(int num) {
    char led7seg[10] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82,
    0xF8, 0x80, 0x90};
    for (int i = 0; i < 7; i++) {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 << i, (led7seg[num]
   >> i) & 1);
    }
}
```

Program 1.3: Source code in display7SEG(int)

5 Exercise 5

Report: Present the source code.

```
  while (1)
  {
  switch(counter) {
    case 9: //1RED-2GREEN
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);

      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_RESET);
      display7SEG(4);
```

```
      HAL_Delay(500);
      counter--;
      break;
    case 8:
      display7SEG(3);
      HAL_Delay(500);
      counter--;
      break;
    case 7:
      display7SEG(2);
      HAL_Delay(500);
      counter--;
      break;
    case 6: //2YEL
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
      display7SEG(1);
      HAL_Delay(500);
      counter--;
      break;
    case 5:
      display7SEG(0);
      HAL_Delay(500);
      counter--;
      break;
    case 4: //1GREEN-2RED
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);

      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
      display7SEG(2);
      HAL_Delay(500);
      counter--;
      break;
    case 3:
      display7SEG(1);
      HAL_Delay(500);
      counter--;
      break;
    case 2:
      display7SEG(0);
      HAL_Delay(500);
      counter--;
      break;
    case 1: //1YEL
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);
      display7SEG(1);
```

```
      HAL_Delay(500);
      counter--;
      break;
    case 0:
      display7SEG(0);
      HAL_Delay(500);
      counter += 9;
      break;
  }
  }
}
```

Program 1.4: Source code in while loop

6 Exercise 6

Report 1: Present the schematic.

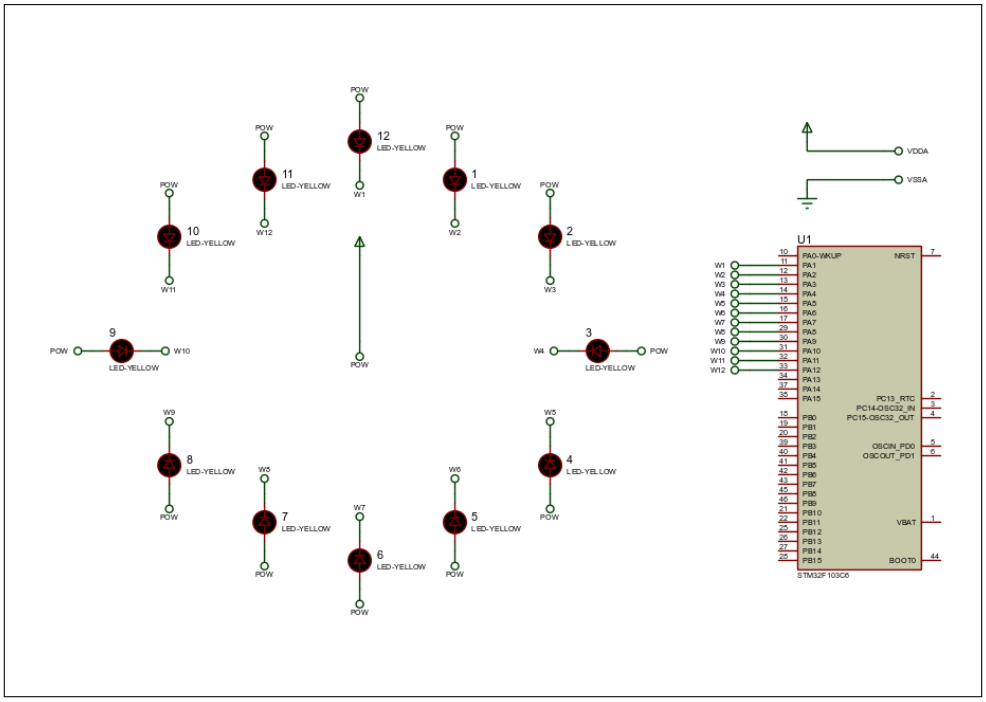


Figure 1.4: Schematic design of exercise 6-10 - lab 1

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

```
uint16_t LEDpin[12] = {GPIO_PIN_1,GPIO_PIN_2,GPIO_PIN_3,
                       GPIO_PIN_4,GPIO_PIN_5,GPIO_PIN_6,
                       GPIO_PIN_7,GPIO_PIN_8,GPIO_PIN_9,
                       GPIO_PIN_10,GPIO_PIN_11,GPIO_PIN_12};
void testLED(int nth) {
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1|GPIO_PIN_2
                       |GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
                       |GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8
```

```
                        |GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11
                          |GPIO_PIN_12, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, LEDpin[nth], GPIO_PIN_RESET);
}
while(1) {
    for (int i = 0; i < 12; i++) {
      testLED(i);
      HAL_Delay(250);
    }
}
```

Program 1.5: Source code of testLED(int) and while loop

7 Exercise 7

Report: Implement 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_1|GPIO_PIN_2
                        |GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
                        |GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8
                        |GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11
                        |GPIO_PIN_12, GPIO_PIN_SET);
}
```

Program 1.6: Source code of clearAllClock()

8 Exercise 8

Report: 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.

```
uint16_t LEDpin[12] = {GPIO_PIN_1,GPIO_PIN_2,GPIO_PIN_3,
                       GPIO_PIN_4,GPIO_PIN_5,GPIO_PIN_6,
                       GPIO_PIN_7,GPIO_PIN_8,GPIO_PIN_9,
                       GPIO_PIN_10,GPIO_PIN_11,GPIO_PIN_12};
void setNumberOnClock(int num) {
    HAL_GPIO_WritePin(GPIOA, LEDpin[num], GPIO_PIN_RESET);
}
```

Program 1.7: Source code of setNumberOnClock(int)

9 Exercise 9

Report: Implement a function named **clearNumberOnClock(int num)**. The input for this function is from **0 to 11** and an appropriate LED is turn off.

```
uint16_t LEDpin[12] = {GPIO_PIN_1,GPIO_PIN_2,GPIO_PIN_3,
                       GPIO_PIN_4,GPIO_PIN_5,GPIO_PIN_6,
                       GPIO_PIN_7,GPIO_PIN_8,GPIO_PIN_9,
                       GPIO_PIN_10,GPIO_PIN_11,GPIO_PIN_12};
void clearNumberOnClock(int num) {
    HAL_GPIO_WritePin(GPIOA, LEDpin[num], GPIO_PIN_SET);
}
```

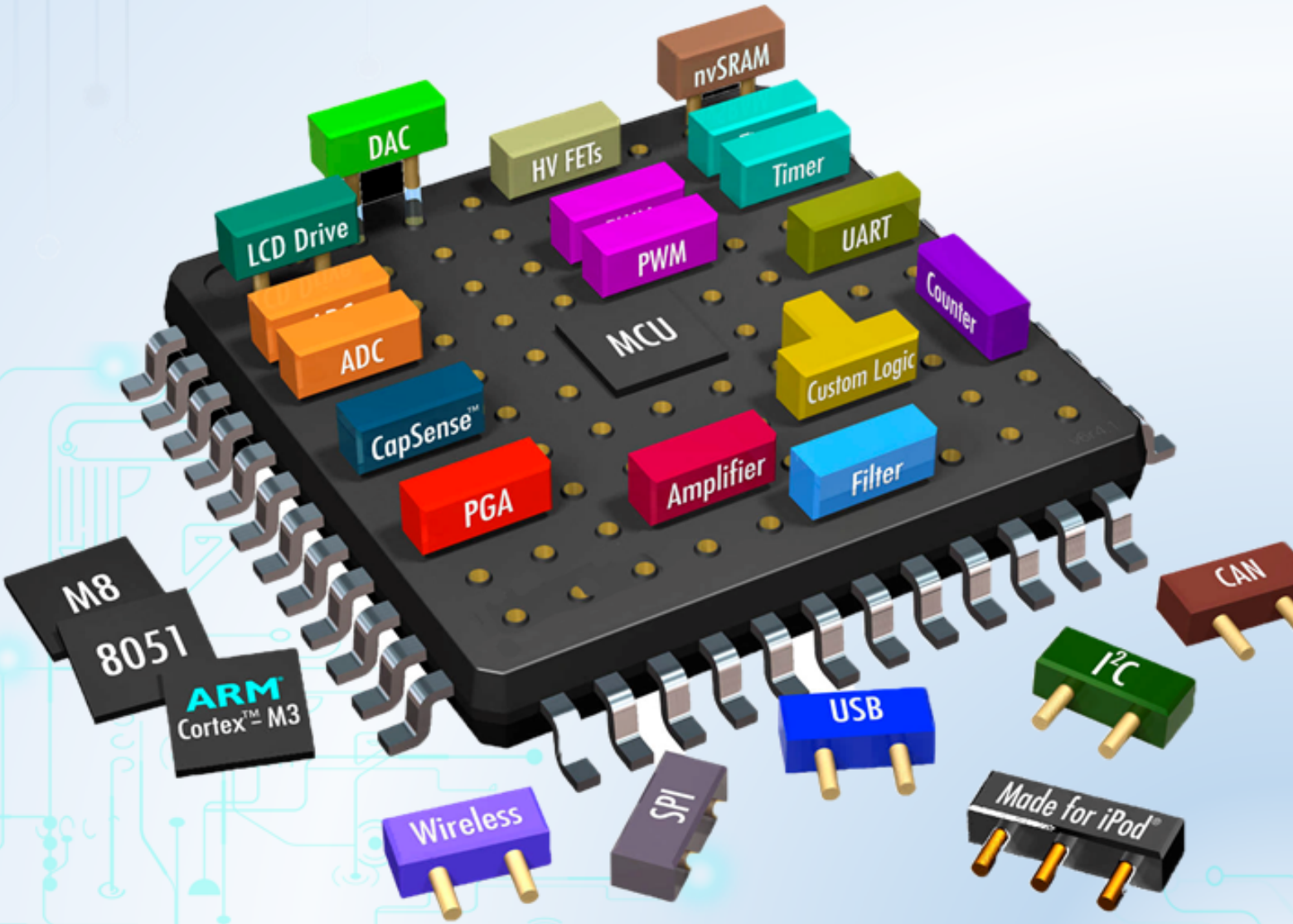
Program 1.8: Source code of clearNumberOnClock(int)

10 Exercise 10

Report: 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.
> STM32 project of exercise 10 - lab 1

CHAPTER 2

Timer Interrupt and LED Scanning



1 Exercise 1

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

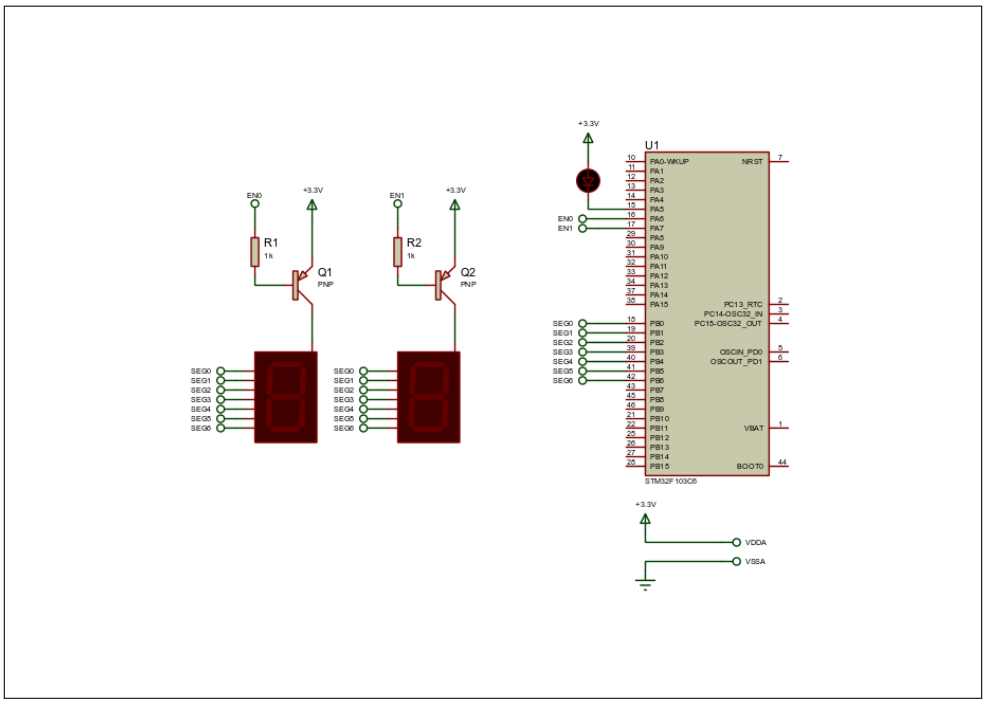


Figure 2.1: Schematic design of exercise 1 - lab 2

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

```
int counter = 100, nth = 0;
int buffer[2] = {1, 2};
void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef * htim ) {
  counter--;
  if (counter == 50 || counter == 0) {
    if (counter == 0) {
      counter = 100;
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    }
    if (nth == 0) nth = 1;
    else nth = 0;
    clearLED();
    display7seg(nth, buffer[nth]);
  }
}
```

Program 2.1: Source code in HAL_TIM_PeriodElapsedCallback()

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

2 Exercise 2

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

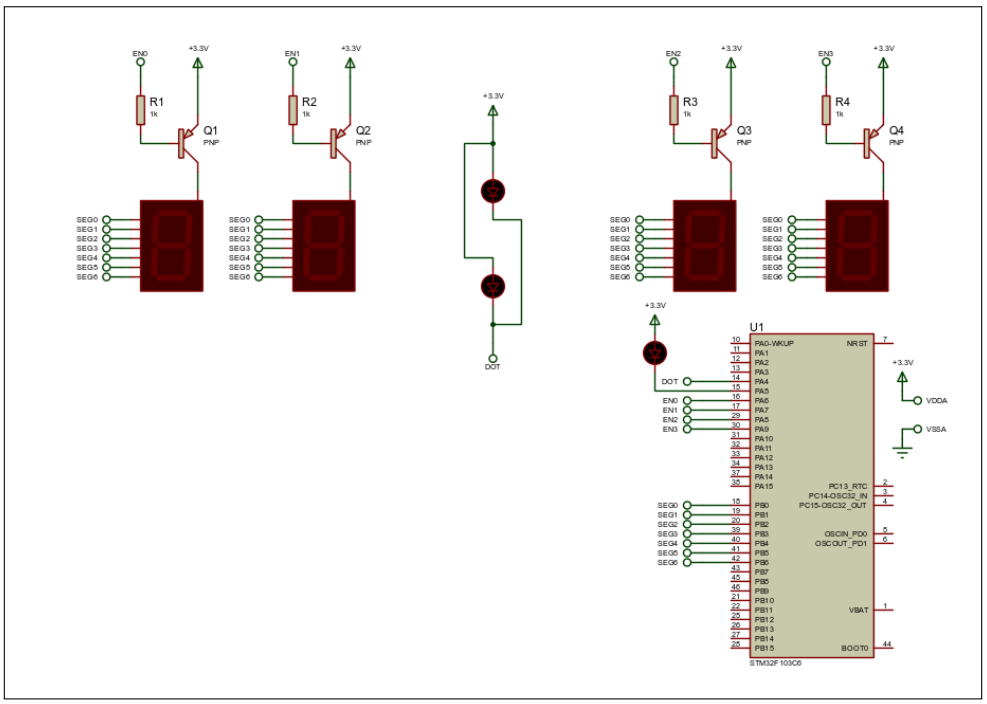


Figure 2.2: Schematic design of exercise 2 - lab 2

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

```
int counter = 100, idx = 0;
int buffer[4] = {1, 2, 3, 0};
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim) {
  counter--;
  if (counter == 50 || counter == 0) {
    if (counter == 0) {
      counter = 100;
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4|GPIO_PIN_5);
    }
    clearLED();
    display7seg(idx, buffer[idx]);
    idx++;
    if (idx >= 4) idx = 0;
  }
}
```

Program 2.2: Source code in HAL_TIM_PeriodElapsedCallback()

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

3 Exercise 3

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

```
uint16_t pin7seg[4] = {GPIO_PIN_6, GPIO_PIN_7, GPIO_PIN_8,
   GPIO_PIN_9};
void clearLED() {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|
```

```
                        GPIO_PIN_9, GPIO_PIN_SET);
    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);
}
void display7seg(int idx, int num) {
  HAL_GPIO_WritePin(GPIOA, pin7seg[idx], GPIO_PIN_RESET);
    char led7seg[10] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82,
    0xF8, 0x80, 0x90};
    for (int i = 0; i < 7; i++) {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 << i, (led7seg[num]
   >> i) & 1);
    }
}
void update7seg (int idx) {
  clearLED();
  display7seg(idx, buffer[idx]);
}
```

Program 2.3: Source code in update7SEG() and its supporting functions

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

```
int counter = 100, idx = 0;
int buffer[4] = {1, 2, 3, 4};
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim) {
  counter--;
  if (counter == 50 || counter == 0) {
    if (counter == 0) {
      counter = 100;
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4|GPIO_PIN_5);
    }
    clearLED();
    display7seg(idx, buffer[idx]);
    idx++;
    if (idx >= 4) idx = 0;
  }
}
```

Program 2.4: Source code in HAL_TIM_PeriodElapsedCallback()

4 Exercise 4

Report: Present the source code in the **HAL_TIM_PeriodElapsedCallback**.
In order to set the frequency of 4 seven-segment LEDs to 1Hz, the display period of each one must be set to 250ms.

```
int counter = 50, idx = 0;
int buffer[4] = {1, 2, 3, 4};
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim) {
  counter--;
```

```
  if (counter == 25 || counter == 0) {
    if (counter == 0) {
      counter = 50;
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4|GPIO_PIN_5);
    }
    clearLED();
    display7seg(idx, buffer[idx]);
    idx++;
    if (idx >= 4) idx = 0;
  }
}
```

Program 2.5: Source code in HAL_TIM_PeriodElapsedCallback()

5 Exercise 5

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

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

Program 2.6: Source code in updateClockBuffer()

6 Exercise 6

Report 1: If line 1 of the code is missed, what happens after that and why?
If line 1 of the code is missed, the value timer0_flag is kept at 0 and can not be set to 1, so the LED will not blink.
Report 2: If line 1 of the code is changed to setTimer0(1), what happens after that and why?
If line 1 of the code is changed to setTimer0(1), the LED will not blink, because if duration = 1, we get timer0_counter = 0 (since timer0_counter is of type int), then when executing timer_run(), the value of timer0_counter can not satisfy the if condition, thus timer0_flag is kept at 0 and can not be set to 1.
Report 3: If line 1 of the code is changed to setTimer0(10), what is changed compared to 2 first questions and why?
If line 1 of the code is changed to setTimer0(10), we get timer0_counter = 1, this value satisfy the if condition in timer_run() and the timer0_flag is set to 1 right away, so the LED will be invoked and start blinking properly.

7 Exercise 7

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

```
  while (1)
  {
```

```
  // LED & DOT
  if (timer0_flag == 1) {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    setTimer0(tvalLED);
  }
  // TIME
  switch (TIMEstate) {
  case 0: // act as HAL_Delay
    sec++;
    setTimer1(tval);
    TIMEstate = 1;
    break;
  case 1: // comparison
    if (timer1_flag == 1) {
      if (sec >= 20) {
        sec = 0;
        min++;
      }
      if (min >= 60) {
        min = 0;
        hour++;
      }
      if (hour >= 24) {
        hour = 0;
      }
      TIMEstate = 0;
    }
  }
  updateClockBuffer();
  }
```

Program 2.7: Source code

8 Exercise 8

Report: Present the source code in main function. In case extra functions are used, present them in the report as well.

```
  while (1)
  {
  // LED & DOT
  if (timer0_flag == 1) {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    setTimer0(tvalLED);
  }
  // 7SEG
  if (timer2_flag == 1) {
```

```
      clearLED();
      display7seg(SEGindex, buffer[SEGindex]);
    SEGindex++;
    if (SEGindex == 4) SEGindex = 0;
    setTimer2(tval7SEG);
  }
  // TIME
  switch (TIMEstate) {
  case 0: // act as HAL_Delay
    sec++;
    setTimer1(tval);
    TIMEstate = 1;
    break;
  case 1: // comparison
    if (timer1_flag == 1) {
      if (sec >= 20) {
        sec = 0;
        min++;
      }
      if (min >= 60) {
        min = 0;
        hour++;
      }
      if (hour >= 24) {
        hour = 0;
      }
      TIMEstate = 0;
    }
  }
  updateClockBuffer();
  }
```

Program 2.8: Source code in main function

9 Exercise 9

Report 1: Present the schematic of your system.

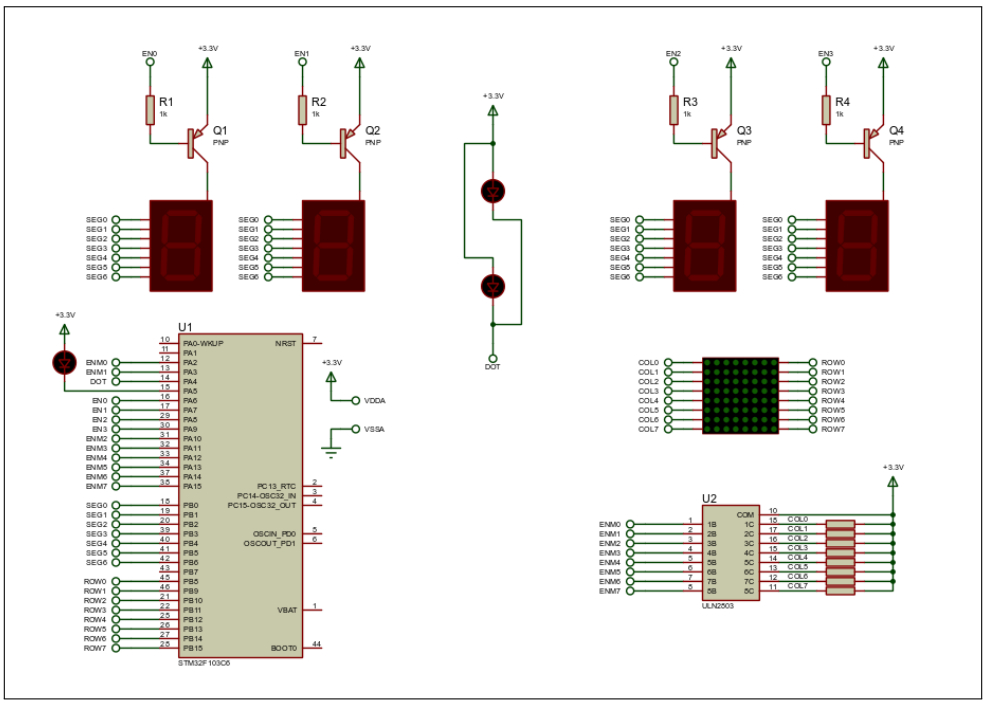


Figure 2.3: Schematic design of exercise 9 - lab 2

Report 2: Present the source code in the **updateLEDmatrix(int index)** function.

Program 2.9: Source code in updateLEDmatrix(int)

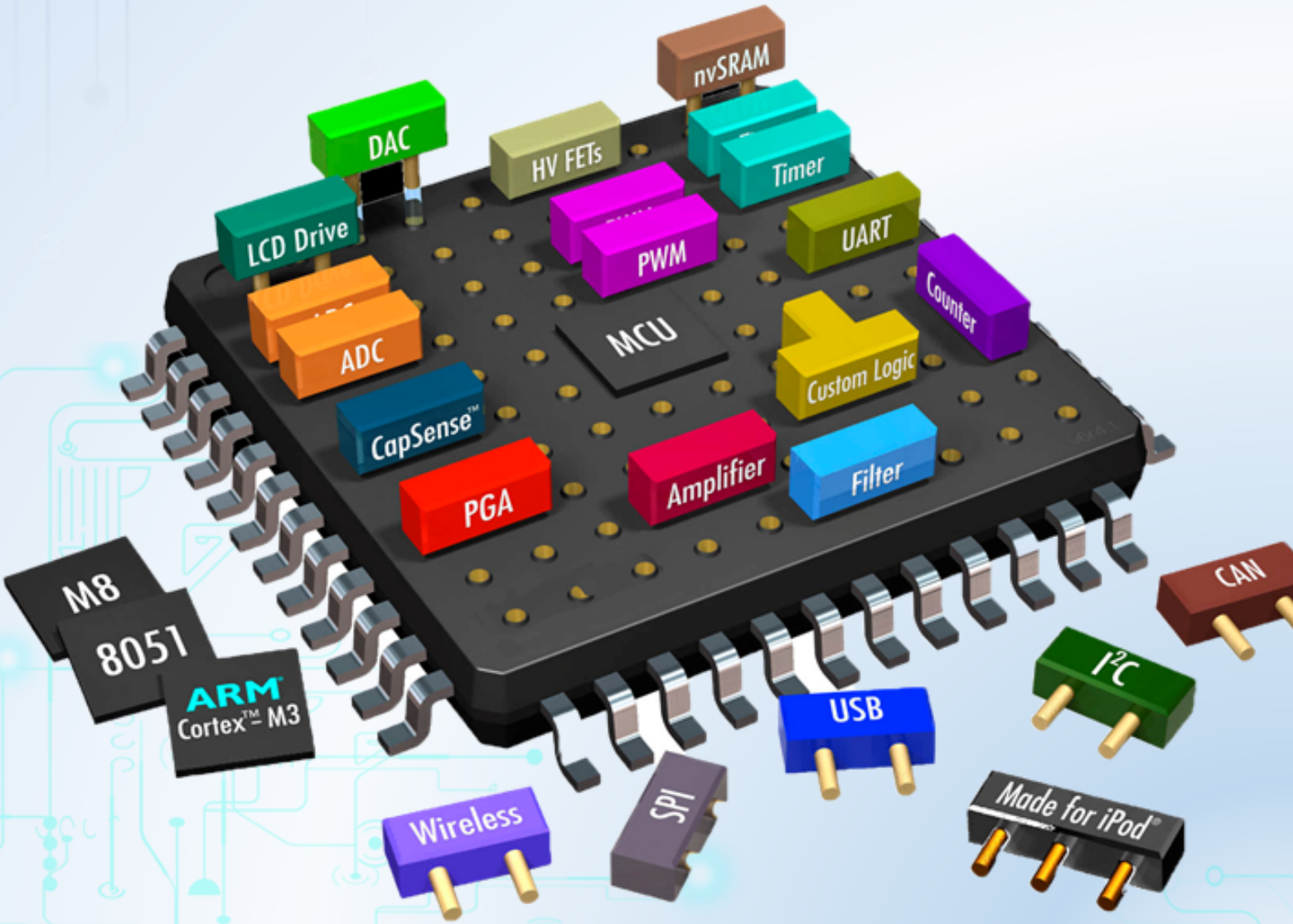
10 Exercise 10

Report: Briefly describe your solution and present your source code.

Program 2.10: Source code

CHAPTER 3

Buttons/Switches



1 Exercise 1: Sketch an FSM

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

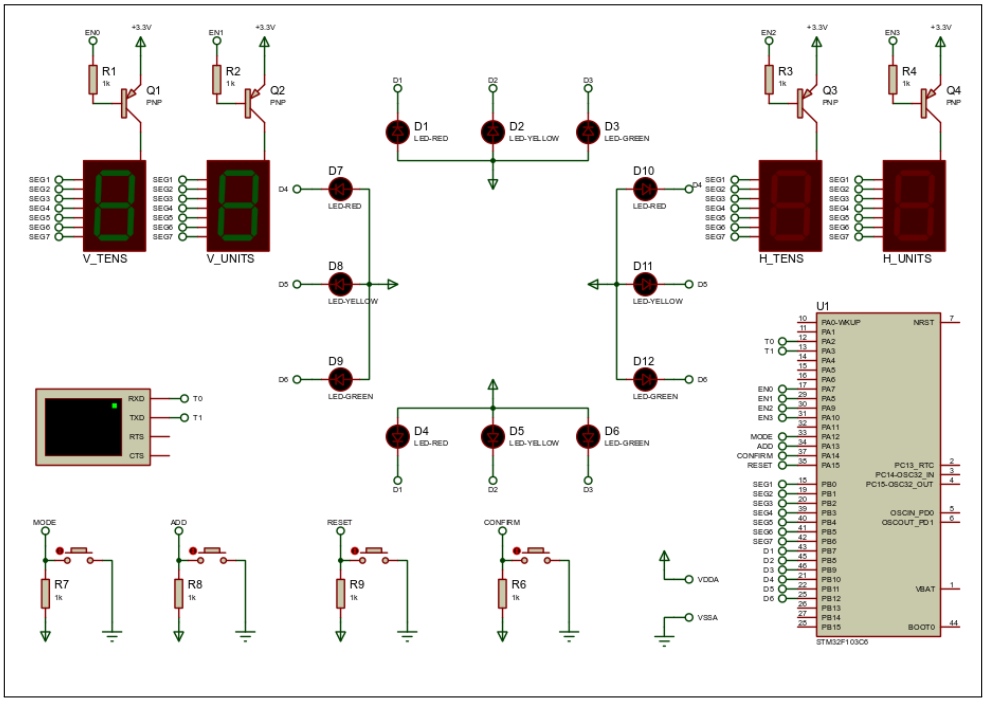


Figure 3.1: FSM design of lab 3

2 Exercise 2: Proteus Schematic

The schematic consists of these components below:

- STM32F103C6 - LQFP48.
- 4 seven-segment LEDs to display time duration with 2 for each road.
- 12 LEDs (4 red, 4 amber, 4 green).
- 4 buttons (namely MODE, ADD, RESET and CONFIRM).
- 1 terminal.

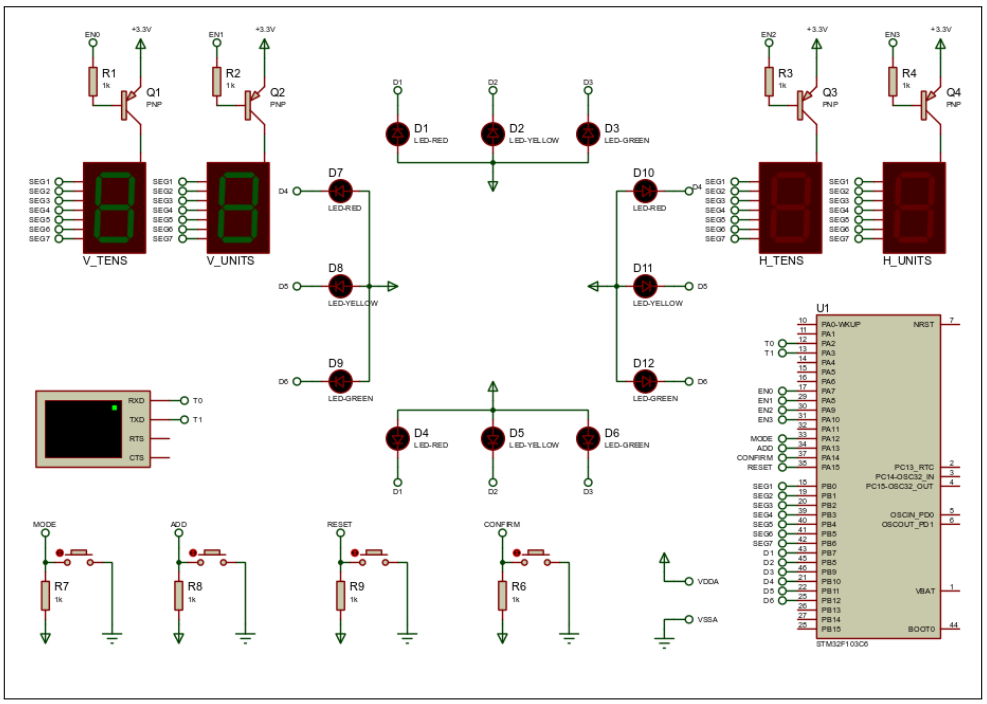


Figure 3.2: Schematic design of lab 3

3 Exercise 3: Create STM32 Project

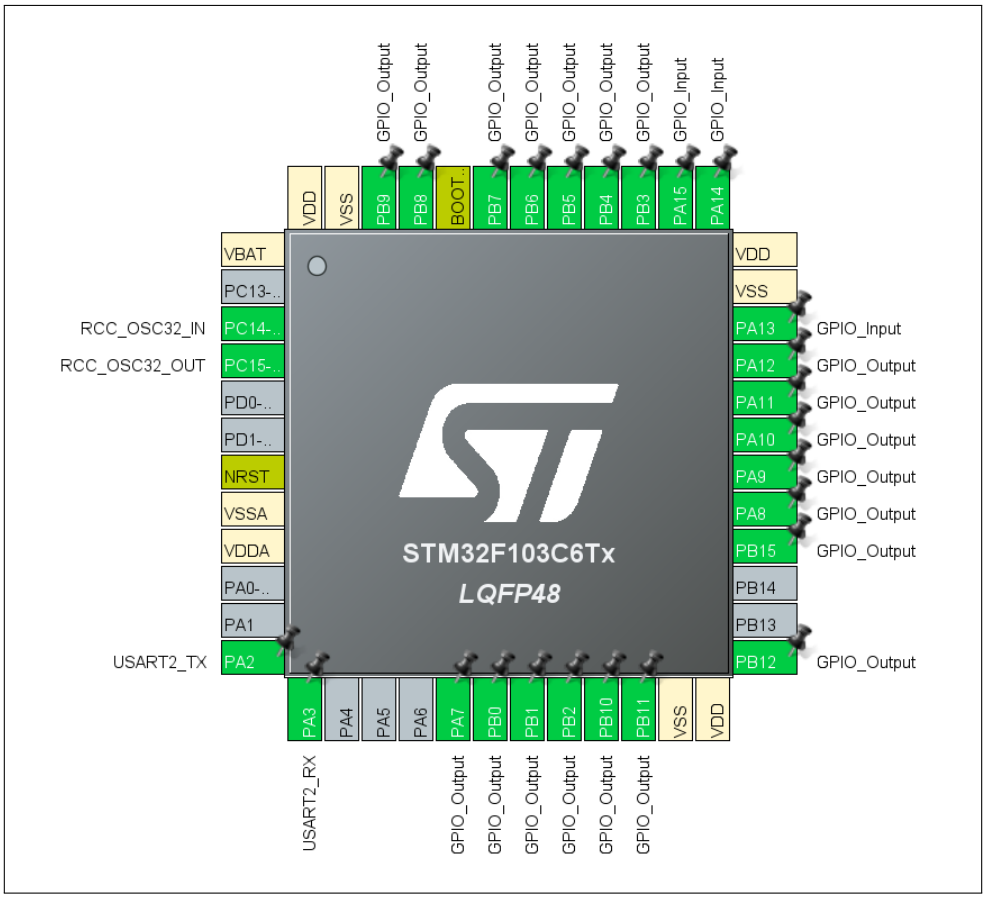


Figure 3.3: Pinout view of STM32 project - lab 3

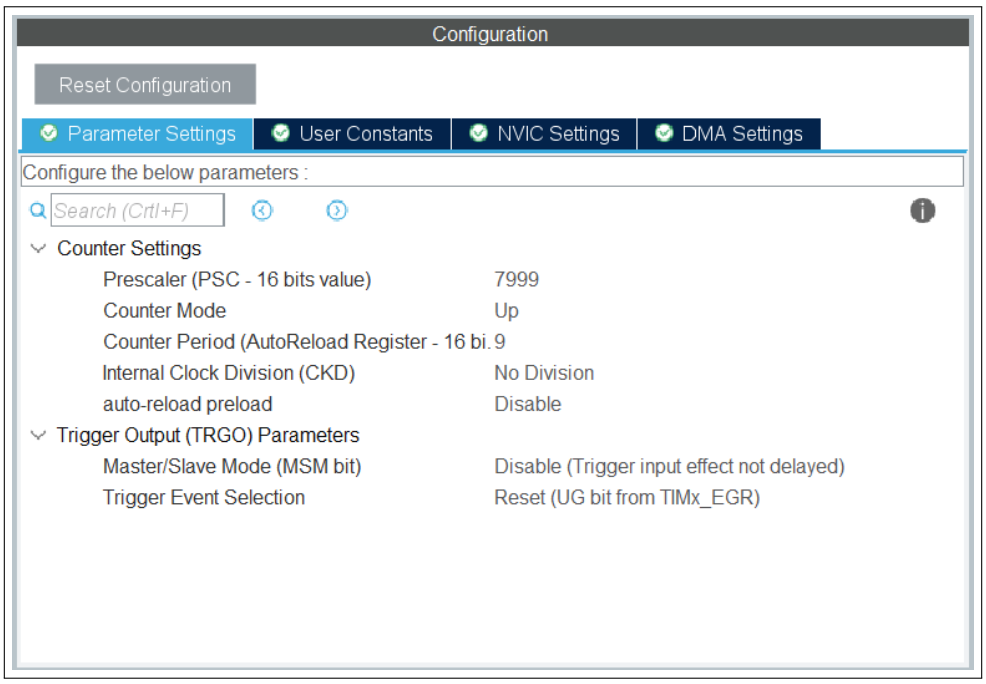


Figure 3.4: TIM2 configuration of STM32 project - lab 3

4 Exercise 4: Modify Timer Parameters

For this exercise, I use a variable named TIMER_CYCLE in dealing with software timers. Below is an example of a setTimerx(int) function:

```
void setTimer_vertical(int duration) {
  timer_vertical_counter = duration / TIMER_CYCLE;
  timer_vertical_flag = 0;
}
```

Program 3.1: Source code of setTimer_vertical(int)

The TIMER_CYCLE is set to be equal to the timer interrupt of TIM2, which is 10(ms). In case there are changes in this timer interrupt value, I only have to modify TIMER_CYCLE to match with the new value.

5 Exercise 5: Adding code for button debouncing

Note: The printf function used in this project is implemented by overwriting the _write() function in file **syscalls.c**.

```
..
#include  <sys/unistd.h>
#include "main.h"
..
extern UART_HandleTypeDef huart2;
..
__attribute__((weak)) int _write(int file, char *ptr, int len)
{
    if ((file != STDOUT_FILENO) && (file != STDERR_FILENO))
```

```
    {
        errno = EBADF;
        return -1;
    }
    HAL_StatusTypeDef status = HAL_UART_Transmit(&huart2, (
  uint8_t*)ptr, len, 1000);
    return (status == HAL_OK ? len : 0);
}
```

Program 3.2: Source code of _write()

There are 4 buttons in total.

- Button MODE is responsible for controlling the modes of the system, its valid value consists of integers from 0 to the predefined MAX_MODE value.
- Button ADD is for increasing value of variable TERM, which will be added to the time duration of corresponding LED later.
- Button RESET, as its name implies, is for reset the MODE and TERM values (set its to 0).
- Button CONFIRM is for confirming the changes, when this button is pressed, the TERM value will be examined, if it is not zero, the system starts updating, else if it is zero, the system returns to mode 0.

```
#include <stdio.h>
#include "main.h"
#include "button.h"

GPIO_PinState DEBOUNCE_BUFFER_1[N_BUTTON] =
{GPIO_PIN_SET, GPIO_PIN_SET, GPIO_PIN_SET, GPIO_PIN_SET};
GPIO_PinState DEBOUNCE_BUFFER_2[N_BUTTON] =
{GPIO_PIN_SET, GPIO_PIN_SET, GPIO_PIN_SET, GPIO_PIN_SET};
GPIO_PinState BUTTON_BUFFER[N_BUTTON]     =
{GPIO_PIN_SET, GPIO_PIN_SET, GPIO_PIN_SET, GPIO_PIN_SET};
uint16_t BUTTON_PIN[N_BUTTON] = {GPIO_PIN_12, GPIO_PIN_13,
                                 GPIO_PIN_14, GPIO_PIN_15};
int DEB_TIME = 50;

void button_reading() {
  for (int idx = 0; idx < N_BUTTON; idx++) {
    DEBOUNCE_BUFFER_2[idx] = DEBOUNCE_BUFFER_1[idx];
    DEBOUNCE_BUFFER_1[idx] = HAL_GPIO_ReadPin(GPIOA, BUTTON_PIN[
   idx]);
    if (DEBOUNCE_BUFFER_1[idx] == DEBOUNCE_BUFFER_2[idx]) {
      BUTTON_BUFFER[idx] = DEBOUNCE_BUFFER_1[idx];
    }
  }
}
unsigned char is_pressed(int idx) {
  if (idx >= N_BUTTON) return 0;
  return (BUTTON_BUFFER[idx] == GPIO_PIN_RESET);
```

```
}
void button_processing() {
  // BUTTON 'MODE'
  if (is_pressed(0) && timer_mode_debounce_flag == 1) {
    MODE++;
    if (MODE >= MAX_MODE) MODE = 0;
    printf("Button 'MODE' is pressed\r\n");
    printf("The system is in MODE %d\r\n", MODE);
    setTimer_mode_debounce(DEB_TIME);
  }
  // BUTTON 'ADD'
  if (MODE != 0 && is_pressed(1) && timer_add_debounce_flag == 1)
    {
    TERM++;
    printf("Button 'ADD' is pressed\r\n");
    setTimer_add_debounce(DEB_TIME);
  }
  // BUTTON 'CONFIRM'
  if (is_pressed(2)) {
    printf("Button 'CONFIRM' is pressed\r\n");
    printf("The system has returned to MODE 0\r\n");
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
    if (TERM != 0) {
      printf("Changes will be applied immediately\r\n");
      UPDATE_LED_TIME(MODE);
      RESTART();
      CLRSCR();
      HALT_DISPLAY();
    } else MODE = 0;
  } else HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_SET);
  // BUTTON 'RESET'
  if (is_pressed(3)) {
    MODE = 0;
    TERM = 0;
    printf("Button 'RESET' is pressed\r\n");
    printf("The system is in MODE %d\r\n", MODE);
  }
}
```

Program 3.3: Source code of button.c

6 Exercise 6: Adding code for displaying mode

Below is the code in display_led.c, which contains the DISPLAY_LED() and its helping functions. Two software timers, timer_vertical and timer_horiz, are used to control the LEDs. I also use a state_update(int) function to update the value of CURR_TIME[idx], which then can be used to set the timers. Value of CURR_TIME[idx] is retrieved from LED_TIME[3], a shared integer array that stores the time duration of red/amber/green LEDs.

```
void state_update(int idx) {
  CURR_TIME[idx] = LED_TIME[CURR_STATE[idx]];
  CURR_STATE[idx] = CURR_STATE[idx] + 1;
  if (CURR_STATE[idx] >= 3) CURR_STATE[idx] = 0;
}
```

Program 3.4: Source code in state_update(int)

```
#include "main.h"
#include "display_led.h"

uint16_t RED[2] = {GPIO_PIN_7, GPIO_PIN_10};
uint16_t YEL[2] = {GPIO_PIN_8, GPIO_PIN_11};
uint16_t GRE[2] = {GPIO_PIN_9, GPIO_PIN_12};

void clear_vertical() {
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9,
   GPIO_PIN_SET);
}
void clear_horizontal() {
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12,
    GPIO_PIN_SET);
}
void clearLED() {
  clear_vertical();
  clear_horizontal();
}
void vertical_processing() {
  clear_vertical();
  switch(CURR_STATE[0]) {
  case RED_IDX:
    HAL_GPIO_WritePin(GPIOB, RED[0], GPIO_PIN_RESET);
    break;
  case GRE_IDX:
    HAL_GPIO_WritePin(GPIOB, GRE[0], GPIO_PIN_RESET);
    break;
  case YEL_IDX:
    HAL_GPIO_WritePin(GPIOB, YEL[0], GPIO_PIN_RESET);
    break;
  }
}
void horizontal_processing() {
  clear_horizontal();
  switch(CURR_STATE[1]) {
  case GRE_IDX:
    HAL_GPIO_WritePin(GPIOB, GRE[1], GPIO_PIN_RESET);
    break;
  case YEL_IDX:
    HAL_GPIO_WritePin(GPIOB, YEL[1], GPIO_PIN_RESET);
    break;
```

```
  case RED_IDX:
    HAL_GPIO_WritePin(GPIOB, RED[1], GPIO_PIN_RESET);
    break;
  }
}
void DISPLAY_LED() {
    if (timer_vertical_flag == 1) {
      vertical_processing();
      state_update(0);
      setTimer_vertical(CURR_TIME[0]);
    }
    if (timer_horiz_flag == 1) {
      horizontal_processing();
      state_update(1);
      setTimer_horiz(CURR_TIME[1]);
    }
}
```

Program 3.5: Source code for displaying 12 LEDs (display_led.c)

As for displaying seven-segment LEDs, the two software timers timer_7seg_buffer and timer_7seg_display are used to control the value updating and displaying processes. Both update and display are performed on a shared integer array DISPLAY_7SEG[4].

```
#include "main.h"
#include "display_7seg.h"

const int MAX_7SEG = 4;
int index_7SEG = 0;
int TVAL_7SEG[2] = {0, 0};
int index_STATE[2] = {0, 0};
int DISPLAY_7SEG[4] = {0, 0, 0, 0};
uint16_t TIME_7SEG[4] = {GPIO_PIN_7, GPIO_PIN_8,
                         GPIO_PIN_9, GPIO_PIN_10};

void update_tval() {
  DISPLAY_7SEG[0] = TVAL_7SEG[0] / 10;
  DISPLAY_7SEG[1] = TVAL_7SEG[0] % 10;
  DISPLAY_7SEG[2] = TVAL_7SEG[1] / 10;
  DISPLAY_7SEG[3] = TVAL_7SEG[1] % 10;
}
void clear7SEG() {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9|
                      GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12,
                      GPIO_PIN_SET);

    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);
}
void display7SEG(int idx, int num) {
```

```
28    HAL_GPIO_WritePin(GPIOA, TIME_7SEG[idx], GPIO_PIN_RESET);
29      char led7seg[10] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82,
      0xF8, 0x80, 0x90};
30      for (int i = 0; i < 7; i++) {
31          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 << i, (led7seg[num]
     >> i) & 1);
32      }
33  }
34
35  void check_zero() {
36    if (TVAL_7SEG[0] == 0) {
37      index_STATE[0] = index_STATE[0] + 1;
38      if (index_STATE[0] >= 3) index_STATE[0] = 0;
39      TVAL_7SEG[0] = LED_TIME[index_STATE[0]] / 1000;
40    }
41    if (TVAL_7SEG[1] == 0) {
42      index_STATE[1] = index_STATE[1] + 1;
43      if (index_STATE[1] >= 3) index_STATE[1] = 0;
44      TVAL_7SEG[1] = LED_TIME[index_STATE[1]] / 1000;
45    }
46  }
47  void display_update() {
48    if (timer_7seg_buffer_flag == 1) {
49      TVAL_7SEG[0] = TVAL_7SEG[0] - 1;
50      TVAL_7SEG[1] = TVAL_7SEG[1] - 1;
51      update_tval();
52      check_zero();
53      setTimer_7seg_buffer(1000);
54    }
55  }
56  void num_display() {
57    if (timer_7seg_display_flag == 1) {
58      clear7SEG();
59      display7SEG(index_7SEG, DISPLAY_7SEG[index_7SEG]);
60      index_7SEG++;
61      if (index_7SEG >= MAX_7SEG) index_7SEG = 0;
62      setTimer_7seg_display(250);
63    }
64  }
65
66  void INIT_BUFFER() {
67      // assign initial value
68    index_STATE[0] = 1;    // RED
69    index_STATE[1] = 2;    // GREEN
70    TVAL_7SEG[0] = LED_TIME[index_STATE[0]] / 1000;
71    TVAL_7SEG[1] = LED_TIME[index_STATE[1]] / 1000;
72  }
73  void RESET_7SEG() {
74    index_7SEG = 0;
```

```
75    INIT_BUFFER();
76  }
77  void DISPLAY_TVAL() {
78    display_update();
79    num_display();
80  }
```

Program 3.6: Source code for displaying seven-segment LEDs (display_7seg.c)

7 Exercise 7-9: Adding code for increasing time duration of LEDs

To increase the time duration of a specific LED, I update the corresponding value in the LED_TIME array. Given that this is a traffic light system, the time duration of red light must be equal to the sum of the others' time duration. Thus, the implementation of the function is proposed as below:

```
1  void UPDATE_LED_TIME(int mode) {
2    switch(mode) {
3      case 0:           // if MODE = 0, do nothing
4        return;
5      case 1:       // RED
6      case 3:       // GREEN
7        LED_TIME[RED_IDX] = LED_TIME[RED_IDX] + (TERM * 1000);
8        LED_TIME[GRE_IDX] = LED_TIME[GRE_IDX] + (TERM * 1000);
9        break;
10     case 2:       // YELLOW
11       LED_TIME[YEL_IDX] = LED_TIME[YEL_IDX] + (TERM * 1000);
12       LED_TIME[RED_IDX] = LED_TIME[RED_IDX] + (TERM * 1000);
13       break;
14     }
15 }
```

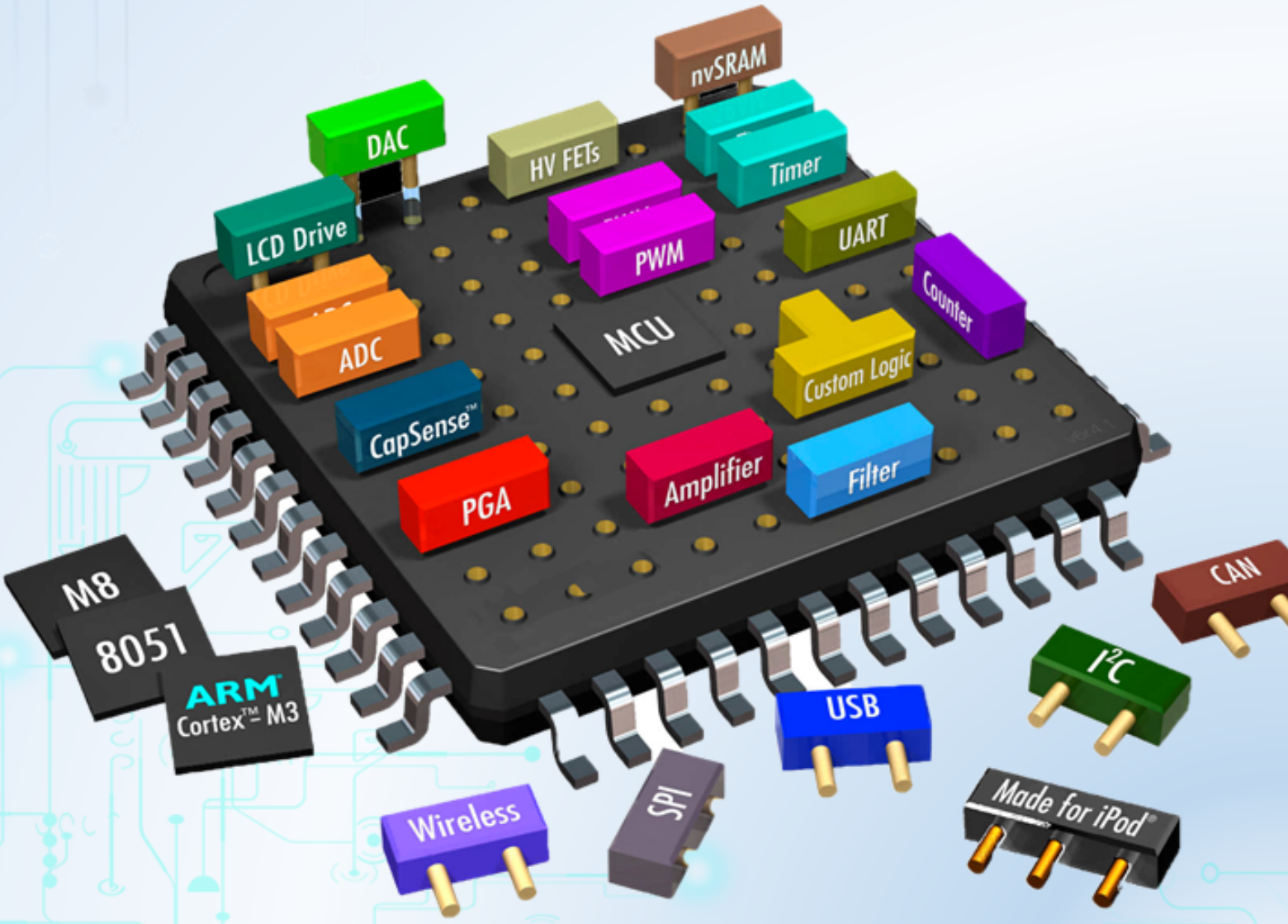
Program 3.7: Source code for increasing time duration

8 Exercise 10: Finish the project

> Download STM32 project of lab 3

CHAPTER 4

A cooperative scheduler



1 Proteus schematic

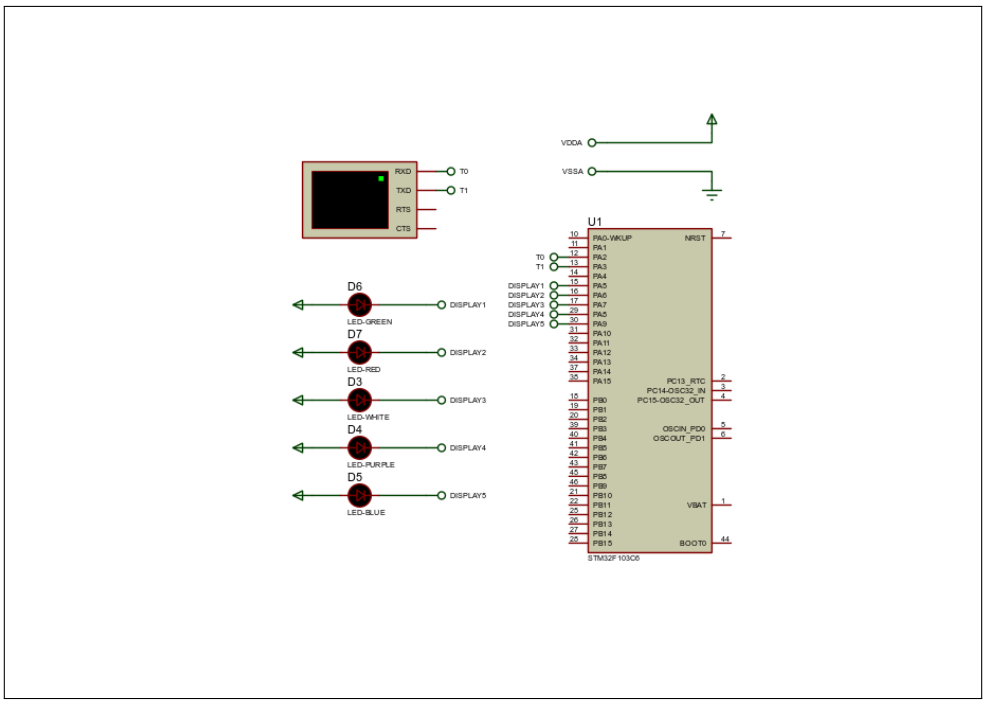


Figure 4.1: Schematic design of lab 4

2 STM32 code

> Download STM32 project of lab 4

The required functions, SCH_Update, SCH_Dispatch_Tasks, SCH_Add_Task and SCH_Delete_Task are presented as follow.

```
void SCH_Update() {
  if (!SCH_TASK_LIST.TASK_QUEUE[0].pTask) {
      // ERROR: NO TASK AVAILABLE!
      ERROR_CODE = 3;
  }
  else if (SCH_TASK_LIST.numofTask == DESIRED_NUMBER_OF_TASKS) {
    if (SCH_TASK_LIST.TASK_QUEUE[0].Delay == 0) {
      SCH_TASK_LIST.TASK_QUEUE[0].RunMe += 1;
      if (SCH_TASK_LIST.TASK_QUEUE[0].Period) {
        SCH_TASK_LIST.TASK_QUEUE[0].Delay = SCH_TASK_LIST.
   TASK_QUEUE[0].Period;
      }
    }
    else SCH_TASK_LIST.TASK_QUEUE[0].Delay -= 1;
  }
}
```

Program 4.1: Source code of SCH_Update

```
void SCH_Dispatch() {
  uint8_t index = 0;
  for (; index < SCH_MAX_TASKS; index++) {
    if (SCH_TASK_LIST.TASK_QUEUE[index].RunMe > 0) {
      printf("\r\n> Run task (ID %d) at index = %d\r\n",
   SCH_TASK_LIST.TASK_QUEUE[index].TaskID, index);
      (*SCH_TASK_LIST.TASK_QUEUE[index].pTask)();
      SCH_TASK_LIST.TASK_QUEUE[index].RunMe -= 1;
      SCH_Delete(index);
    }
  }
  SCH_Report();
}
```

Program 4.2: Source code of SCH_Dispatch_Tasks

```
void SCH_Add_Task(uint8_t id, void (*pFunction)(), uint32_t delay
   , uint32_t period) {
  if (SCH_TASK_LIST.numofTask >= SCH_MAX_TASKS) {
    ERROR_CODE = 1;
    return;
  }
  sTask temp;
  temp.pTask = pFunction;
  temp.Delay = delay;
  temp.Period = period;
  temp.RunMe = 0;
  temp.TaskID = id;
  insert_to_list(temp);
}
```

Program 4.3: Source code of SCH_Add_Task

The above function deploys a helping function, insert_to_list. Source code of this function is presented below.

```
void insert_to_list(sTask task) {
  // NO TASK IN LIST
  if (SCH_TASK_LIST.numofTask == 0) {
    SCH_TASK_LIST.TASK_QUEUE[0] = task;
    SCH_TASK_LIST.numofTask += 1;
    return;
  } else {
  // 1+ TASK(S) IN LIST
    int pos = 0;
    int prev = SCH_TASK_LIST.TASK_QUEUE[0].Delay;
    int sum = SCH_TASK_LIST.TASK_QUEUE[0].Delay;
    // loop until find the right position
    while (pos < SCH_TASK_LIST.numofTask && task.Delay > sum) {
      pos += 1;
      if (pos < SCH_TASK_LIST.numofTask) {
        prev = sum;
```

```
        sum += SCH_TASK_LIST.TASK_QUEUE[pos].Delay;
      }
    }
    if (pos == SCH_TASK_LIST.numofTask) {
      // ADD TO REAR -> update delay of new task only
      task.Delay -= sum;
      SCH_TASK_LIST.TASK_QUEUE[pos] = task;
      SCH_TASK_LIST.numofTask += 1;
      return;
    }
    else {
      // shift elements
      for (int i = SCH_TASK_LIST.numofTask; i > pos; i--) {
        SCH_TASK_LIST.TASK_QUEUE[i] = SCH_TASK_LIST.TASK_QUEUE[i
   - 1];
      }
      if (pos == 0) {
        // ADD TO FRONT -> update the whole array
        SCH_TASK_LIST.TASK_QUEUE[pos] = task;
        sum = 0;
        prev = task.Delay;
      } else {
        // ADD TO MIDDLE -> update from [pos + 1]
        task.Delay -= prev;
        sum = prev;
        prev += task.Delay;
        SCH_TASK_LIST.TASK_QUEUE[pos] = task;
      }
      SCH_TASK_LIST.numofTask += 1;
      // update delay
      for (int i = pos + 1; i < SCH_TASK_LIST.numofTask; i++) {
        sum += SCH_TASK_LIST.TASK_QUEUE[i].Delay;
        SCH_TASK_LIST.TASK_QUEUE[i].Delay = sum - prev;
        prev += SCH_TASK_LIST.TASK_QUEUE[i].Delay;
      }
    }
  }
  //taskList_display();
}
```

Program 4.4: Source code of insert_to_list

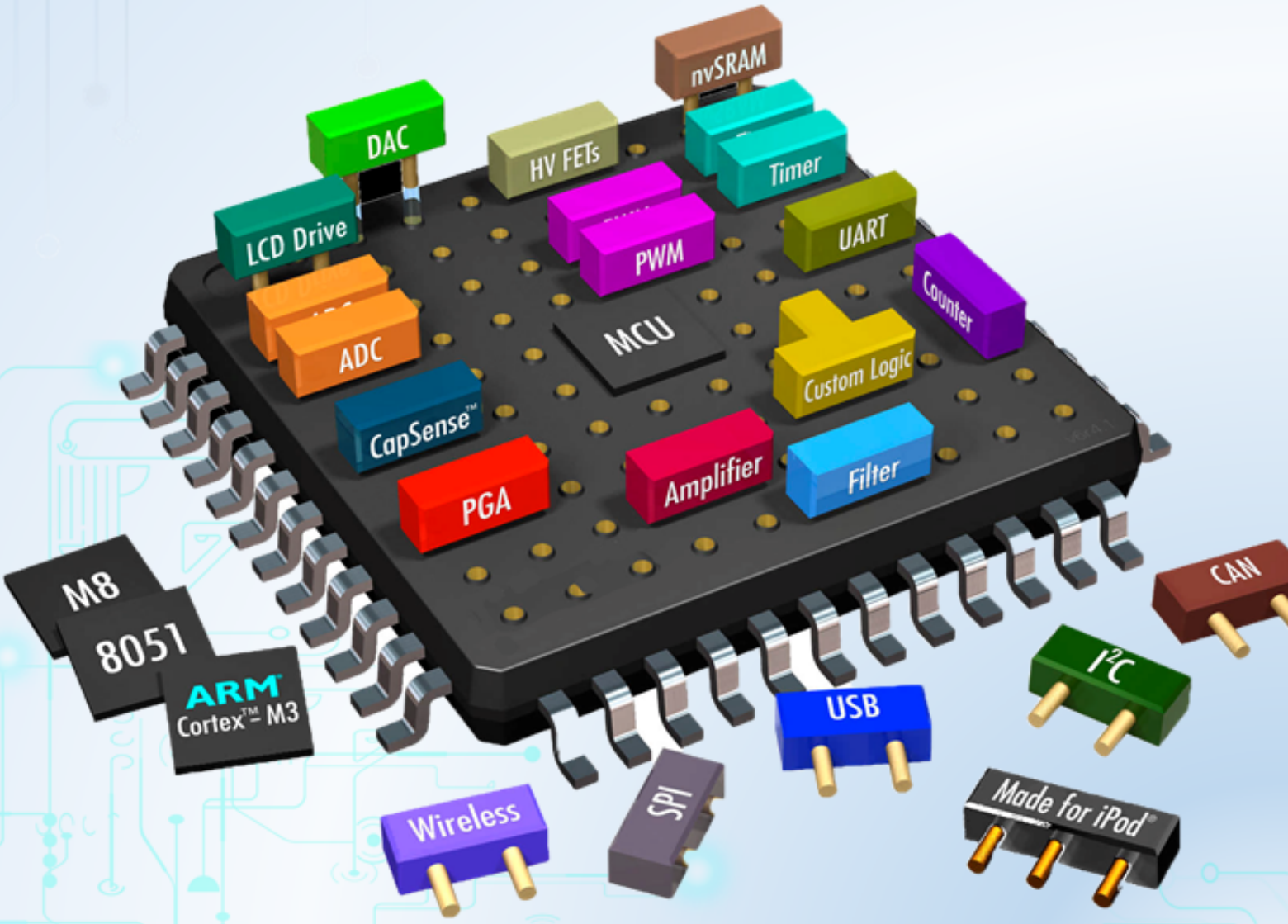
```
void SCH_Delete() {
  int index = 0;
  int add_back_flag = 0;
  sTask temp;
  if (SCH_TASK_LIST.TASK_QUEUE[index].Period) {
    add_back_flag = 1;
    temp = SCH_TASK_LIST.TASK_QUEUE[index];
  }
```

```
9   for (; index < (SCH_TASK_LIST.numofTask - 1); index++) {
10    SCH_TASK_LIST.TASK_QUEUE[index] = SCH_TASK_LIST.TASK_QUEUE[
     index + 1];
11  }
12  SCH_TASK_LIST.TASK_QUEUE[index].pTask = 0x0000;
13  SCH_TASK_LIST.TASK_QUEUE[index].Delay = 0;
14  SCH_TASK_LIST.TASK_QUEUE[index].Period = 0;
15  SCH_TASK_LIST.TASK_QUEUE[index].RunMe = 0;
16  SCH_TASK_LIST.numofTask -= 1;
17  if (add_back_flag == 1) {
18    insert_to_list(temp);
19  }
20 }
```

Program 4.5: Source code of SCH_Delete_Task

CHAPTER 5

Flow and Error Control in Communication



1 FSM

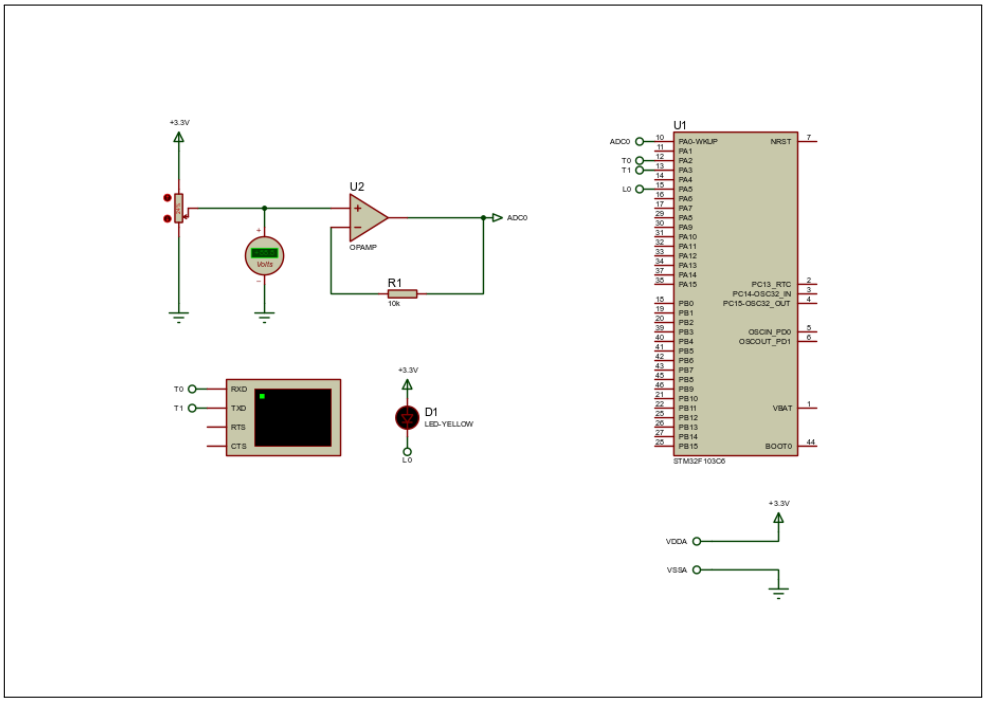


Figure 5.1: FSM design

2 Proteus schematic

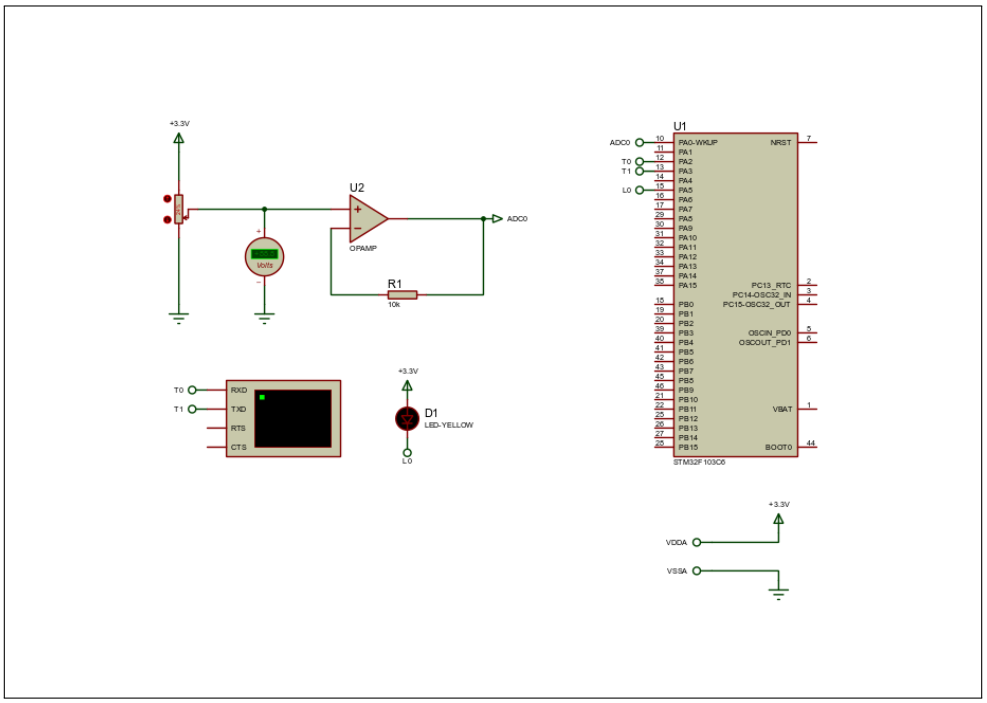


Figure 5.2: Schematic design of lab 5

3 STM32 code

> Download STM32 project of lab 5

Some important parts of this project are presented below. First is the main() function.

```
  /* USER CODE BEGIN 2 */
  // start ADC, TIM, UART and software timer
  HAL_ADC_Start_IT(&hadc1);
  HAL_TIM_Base_Start_IT(&htim2);
  HAL_UART_Receive_IT (&huart2, &rxBuffer, 1);
  timerInit(); // initialize the software timer
  /* USER CODE END 2 */
  while (1)
  {
  if (buffer_flag == 1) {
    command_parser_fsm();
    buffer_flag = 0;
  }
  uart_communiation_fsm();
  }
```

Program 5.1: Source code in the main function

In order to print the entered character onto the terminal, save that character into buffer for later processing, then start receiving from UART again, the function HAL_UART_RxCpltCallback is employed. This function is called whenever the system receive an input from the terminal.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
  if (huart->Instance == USART2) {
      // print the entered character onto the terminal
    HAL_UART_Transmit(&huart2, &rxBuffer, 1, 10);
    // save that character into buffer for later processing
    if (cmd_length >= CMD_MAX_LENGTH && rxBuffer != '\r') {
      ERROR_CODE = BUFFER_IS_FULL;
      UARTstate = ERROR;
    } else {
        // process some special characters
      switch(rxBuffer) {
      case '\b': // backspace
        cmdBuffer[cmd_length - 1] = 0;
        cmd_length--;
        break;
      case ' ': // space
        ERROR_CODE = INVALID_INPUT;
        UARTstate = ERROR;
        break;
      case '\r': // enter
        buffer_flag = 1;
        break;
      default:
        cmdBuffer[cmd_length] = rxBuffer;
```

```
        cmd_length++;
        break;
      }
    }
    setTimer(TIMEOUT_VALUE);
    // start receiving from UART again
    HAL_UART_Receive_IT(&huart2, &rxBuffer, 1);
  }
}
```

Program 5.2: Source code of HAL_UART_RxCpltCallback function

As proposed above, a valid input command must at least has its length <= CMD_MAX_LENGTH. When Enter is pressed, the buffer_flag is set to 1, and the system proceeds to examine the content of cmdBuffer[]. The code for that is as below.

```
#include <string.h>
#include "main.h"
#include "fsm.h"

const uint8_t cmd_RST[] = "!RST#";
const uint8_t cmd_OK[] = "!OK#";
uint8_t UARTstate = IDLE;
uint8_t buffer_flag = 0;

void uart_communiation_fsm() {
  switch(UARTstate) {
  case IDLE:
    break;
  case RESPONSE_ADC:
    printADC(GET_NEW_ADC);
    setTimer(TIMEOUT_VALUE);
    UARTstate = WAIT_FOR_REPLY;
    break;
  case WAIT_FOR_REPLY:
    if (timeout_flag == 1) {
      resetBuffer();
      printADC(GET_OLD_ADC);
      setTimer(TIMEOUT_VALUE);
    }
    break;
  case ERROR:
    printERROR();
    UARTstate = IDLE;
    break;
  }
}
void command_parser_fsm() {
  switch(cmd_length) {
  case 0:
  case 1:
```

```
  case 2:
    ERROR_CODE = CMD_NOT_EXISTED;
    UARTstate = ERROR;
    break;
  case 4:   // "!OK#"
    if (strcmp((void*)cmdBuffer, (void*)cmd_OK) == 0) UARTstate =
    IDLE;
    else {
      ERROR_CODE = CMD_NOT_EXISTED;
      UARTstate = ERROR;
    }
    break;
  case 5:   // "!RST#"
    if (strcmp((void*)cmdBuffer, (void*)cmd_RST) == 0) UARTstate
   = RESPONSE_ADC;
    else {
      ERROR_CODE = CMD_NOT_EXISTED;
      UARTstate = ERROR;
    }
    break;
  }
  resetBuffer();
}
```

Program 5.3: Source code of fsm.c

Besides, an error.c file is included for printing out the error(s). When error occurs, the system will print the error to the terminal, then reset the state back to IDLE.

```
#include "main.h"
#include "error.h"

uint8_t ERROR_CODE = 0;
uint8_t error_1[] = "ERROR: CMD_NOT_EXISTED\r\n";
uint8_t error_2[] = "\r\nERROR: BUFFER_IS_FULL\r\n";
uint8_t error_3[] = "\r\nERROR: INVALID_INPUT\r\n";

void printERROR() {
  switch(ERROR_CODE) {
  case CMD_NOT_EXISTED:
    HAL_UART_Transmit(&huart2, error_1, sizeof(error_1), 100);
    break;
  case BUFFER_IS_FULL:
    HAL_UART_Transmit(&huart2, error_2, sizeof(error_2), 100);
    break;
  case INVALID_INPUT:
    HAL_UART_Transmit(&huart2, error_3, sizeof(error_3), 100);
    break;
  }
}
```

Program 5.4: Source code of error.c

There are two helping functions, namely resetBuffer() and printADC(int).

- resetBuffer() is for clearing out the cmdBuffer[] and set its length to zero.
- printADC(int) receives an integer input and use that input as an indicator to determine whether the ADC value should be printed is the old (in case the system is waiting for !OK#) or new one (in case the system just receives !RST#).

```
#include "main.h"
#include "helper.h"

uint32_t ADC_value = 0;
char ADC_print[4];
uint8_t response_format_1_old[] = "\r\n!ADC=";
uint8_t response_format_1_new[] = "!ADC=";
uint8_t response_format_2[] = "#\r\n";

void resetBuffer() {
  memset(cmdBuffer, 0, CMD_MAX_LENGTH);
  cmd_length = 0;
}
void printADC(int indicator) {
  if (indicator == GET_OLD_ADC)
    HAL_UART_Transmit(&huart2, response_format_1_old, sizeof(
   response_format_1_old), 100);
  else if (indicator == GET_NEW_ADC) {
    HAL_UART_Transmit(&huart2, response_format_1_new, sizeof(
   response_format_1_new), 100);
    ADC_value = HAL_ADC_GetValue(&hadc1);
    sprintf(ADC_print, "%d", (int)ADC_value);
  }
  HAL_UART_Transmit(&huart2, (uint8_t*)ADC_print, sizeof(
   ADC_print), 100);
  HAL_UART_Transmit(&huart2, response_format_2, sizeof(
   response_format_2), 100);
}
```

Program 5.5: Source code of helper.c