Loyola University

Real Time and Embedded Systems

PRACTICE 2 RESULTS

Authors:
Martyna Baran
Zuzanna Jarlaczynska

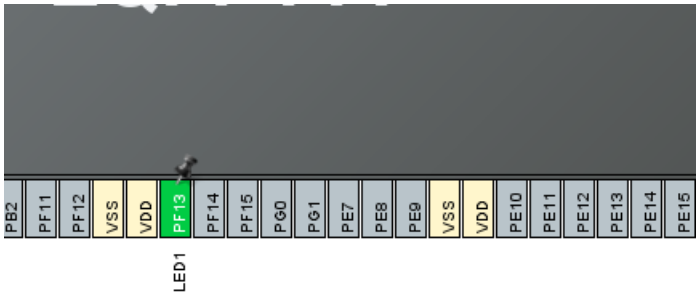
December 12, 2023

INTRODUCTION

The aim of practice 2 was to gain first-hand experience of working with the STM32 microcontroller. It consisted of both installing the necessary software, learning their structure and how they work, as well as connecting the simplest circuits with LED lights. The task required the use of external sources to explain the workflow. An important aspect to bear in mind was that the devices used could be damaged by incorrect connection, so it was necessary to take a longer look at the subject. In conclusion, the task was a great introduction to the work in preparation for the final project.

Problem 1: Connecting the LED

(1) To properly connect the LED to the board we need to remember that it should also be connected to the ground to ensure the close to our circuit. Using the Graphic environment provided by STMCubeIDE we studied the available pins to connect the LED. We have chosen the D7 pin and set it to be "GPIO_Output" because this is the configuration needed for this task. We save and change to the C view.



(2) To connect an LED with a resistor we need to follow the steps:

- Connect the cathode of the LED to the GROUND.
- Connect the LED anode (long leg) to the resistor.
- Connect the other end of the resistor to PF13 pin on the microcontroller board.

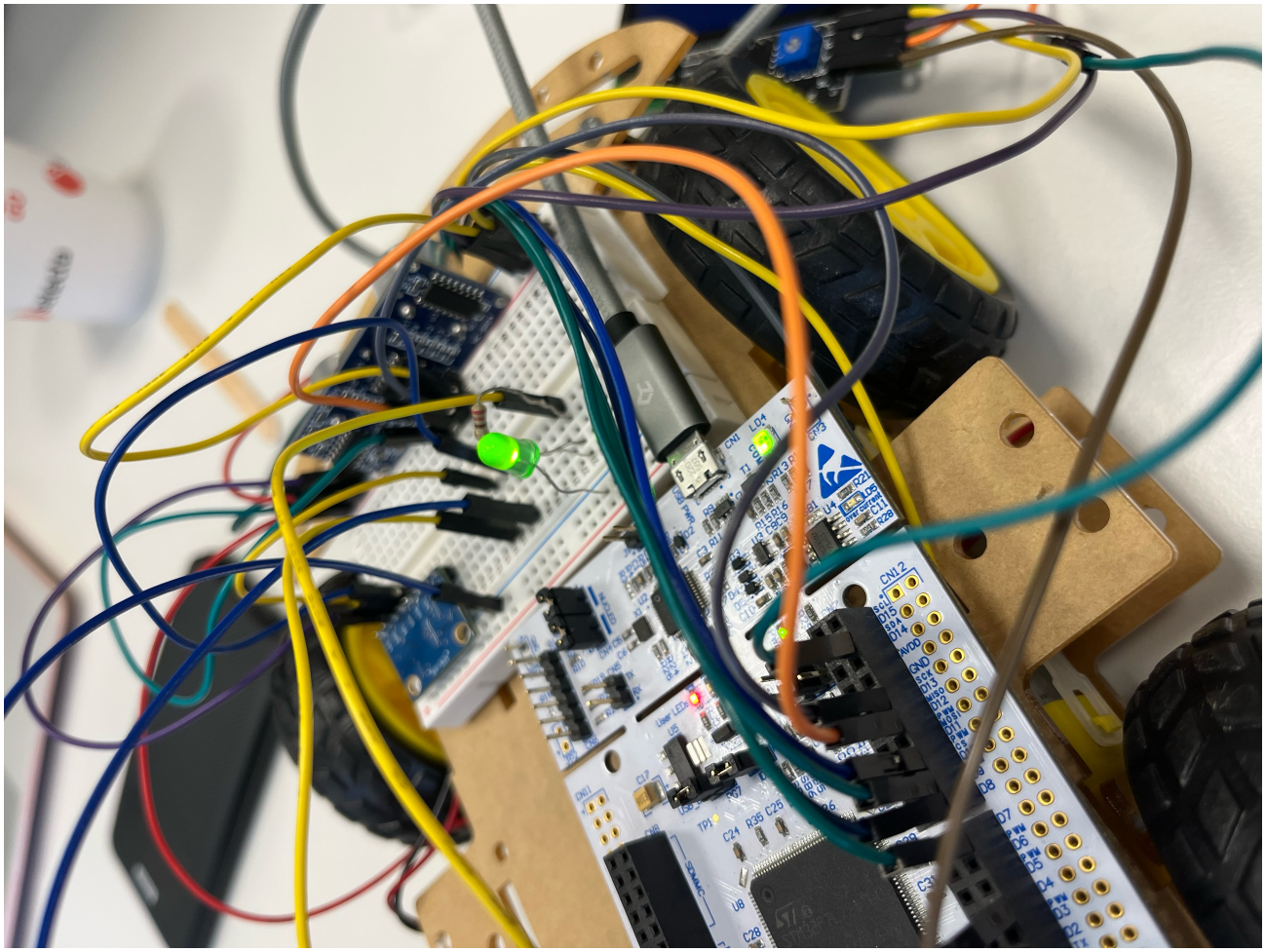
The resistor is added to ensure a safe and stable current for the LED, which is important for its proper operation and longevity.

Problem 2:

Having connected the LED we can set up the code to change the state of the LED. We are using the built-in function from the HAL library that writes the state to the pins. We have also introduced a delay after which the LED is turned on. Then we again wait for 1s and change it to the low. This way our LED switches on and off every 1s.

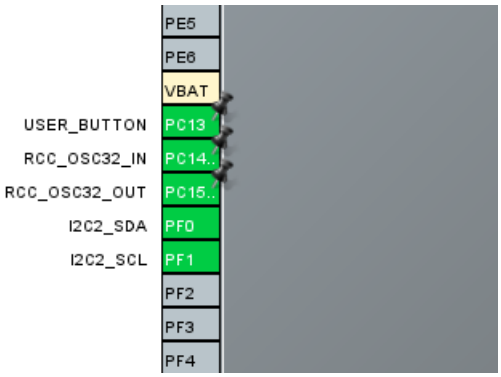
C Code for exercise 2A

```c
void Task1(void *argument)
{
  while(1)
  {
    vTaskDelay(100/ portTICK_PERIOD_MS);
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET); // D7 on
    vTaskDelay(1000/ portTICK_PERIOD_MS);
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET); // D7 off
    vTaskDelay(1000/ portTICK_PERIOD_MS);
  }
}
```



Problem 3:

As an example of the GPIO_Input, we have used the built-in button to control the behavior of the LED. We named the pin PC13 responsible for the button as USER_BUTTON.



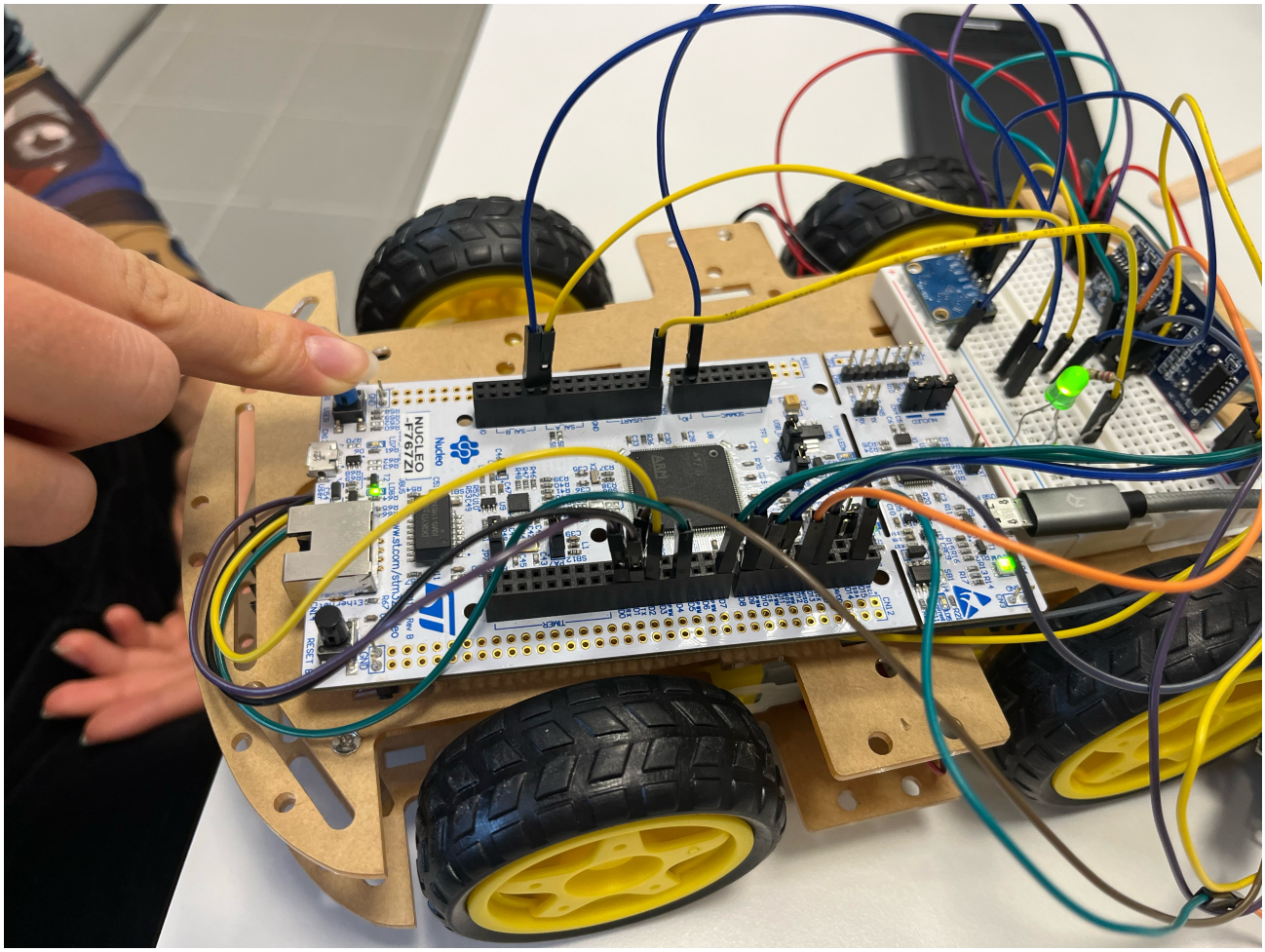
Problem 4:

We decided to use our User button to manipulate the state of the LED. We used the ReadPin function to check its state. If it is pushed, which means that its state is equal to GPIO_PIN_RESET then we turn on the LED. Otherwise, it should stay turned off.

C Code for exercise 2A

```c
while (1)
{

    if (HAL_GPIO_ReadPin(USER_BUTTON_GPIO_Port, USER_BUTTON_Pin) == GPIO_PIN_RESET) {
        HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
    } else {
        HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
    }
}
```



Problem 5: Graphics environment in STM32CubeIDE

STM32CubeIDE offers us a powerful, graphical way to initialize pins and peripherals on the microcontroller – STM32CubeMX perspective. It is a set of windows, panes and visuals that enable the users to configure a particular feature or programming mode. To open this perspective, we simply need to double-click on the .ioc file in our project. After that we can see CubeMX view, which shows us a pinout of the microcontroller. Once we click on a particular pin, we can see a list of peripherals that this pin supports. To enable a chosen peripheral on our pin, we must just click on it. For example, clicking GPIO_Output will turn that pin into an output. Clicking on a chosen peripheral once again will disable it. After saving the file, the program will generate a code compatible with established pinout. This solution makes the initialization process not only clearer but also faster for the user.

Problem 6: PWM signals

(1) What is it? Pulse Width Modulation is a technique used in electronic systems to control the power delivered to a load. It involves varying the width of pulses in a square wave signal while keeping the frequency and amplitude constant.

(2) How does it work? The PWM signal captures a few features. The first of which is the frequency, which is basically a measure of how fast the PWM signal keeps alternating between HIGH and LOW. The frequency is measured in Hz and it's the inverse of the full period time interval. The most important feature is duty cycle. It's a measure of how long the PWM signal stays ON relative to the full PWM's cycle period. It directly affects the PWM's total (average) voltage that most devices respond to. Finally, the PWM resolution is the number of bits that are used to represent the duty cycle value. It can be 8, 10, 12 or even 16 bits. The higher the PWM resolution, the higher number of discrete levels over the entire range of the PWM's duty cycle.

(3) Example As an example, we may think of a simple circuit in which an LED is installed that turns on and off alternately in software. From the moment the diode is turned on, through it is turned off, until it is turned on again, the period is 1 millisecond. The moment when the logic signal "1" is supplied to the diode can be considered "switching on" - power supply with the appropriate voltage value is applied to its output. Turning off is the moment when the diode receives a logical "0" - it is deprived of power. The diode is on for half a millisecond, then off for the second half a millisecond, and this process is repeated. This is what every millisecond looks like. From the perspective of the human eye, you can't even see the blinking - the change is too quick for us to notice, which gives the illusion that the LED is simply dimmer than if it were permanently on. We achieved it by simply changing the width of an impulse.

(4) Usage PWM is commonly used to control the speed of a DC motor, LEDs brightness or loudness of audio signal.

Problem 7: PWM signals in STM32CubeIDE

In STM32CubeIDE PWM signals are usually generated by configuring one of the timers available on our microcontroller. Timer modules can operate a variety of modes one of which is the PWM mode. To use PWM for a chosen pin we need to take the following steps:

- Go to pinout and configuration
- In Timers section select one timer for example TIM2
- Set chosen pin to be TIM2_CH1 which is a PWM source
- In TIM2 Mode and Configuration enable selected timer by setting the Clock Source to 'Internal Clock'
- In the same window select one channel, for example Channel 1, and set it to 'PWM Generation CH1'
- Configure period and duty of a PWM by changing the values in field 'Counter Period' and 'Pulse'
- Save and regenerate the code