Lab 0: Introductory to STM32 Programming and Remote Lab

Lab Objectives

This lab introduces how to program STM32 development boards using the Keil Studio Cloud and the STM32 remote laboratory.

If you have any questions, please post them on Ed.

Background

STM32

STM32 is a family of 32-bit microcontroller integrated circuits from STMicroelectronics, built around the ARM Cortex-M processor core. Known for balancing high performance, high superiority, and a rich set of peripherals, these chips are widely used in applications ranging from industrial control and the Internet of Things (IoT) to consumer electronics and automotive systems. Their popularity is further supported by a comprehensive development ecosystem, including the graphical STM32CubeMX development tool and extensive documentation, making them accessible to both professional engineers and hobbyists.

The STM32 development boards we will use are the STM32WB Nucleo-64 boards, specifically the NUCLEO-WB55RG model, as shown in Figure 1. This board features an ultra-low-power microcontroller with integrated Bluetooth® Low Energy (BLE) wireless technology, compliant with the Bluetooth SIG specification v5.2. In addition, the board supports programming through the Arduino IDE, making it accessible and versatile for embedded development.

Figure 1

STM32 Remote Laboratory

In most of the lab assignments, we will use an STM32 remote laboratory available on LabsLand, a global network of educational remote laboratories.

To get access to the STM32 remote lab,

1. Please contact RHLab or LabsLand to receive an access link to the remote lab. After you click the given link, you should see a similar page to that in Figure 2.

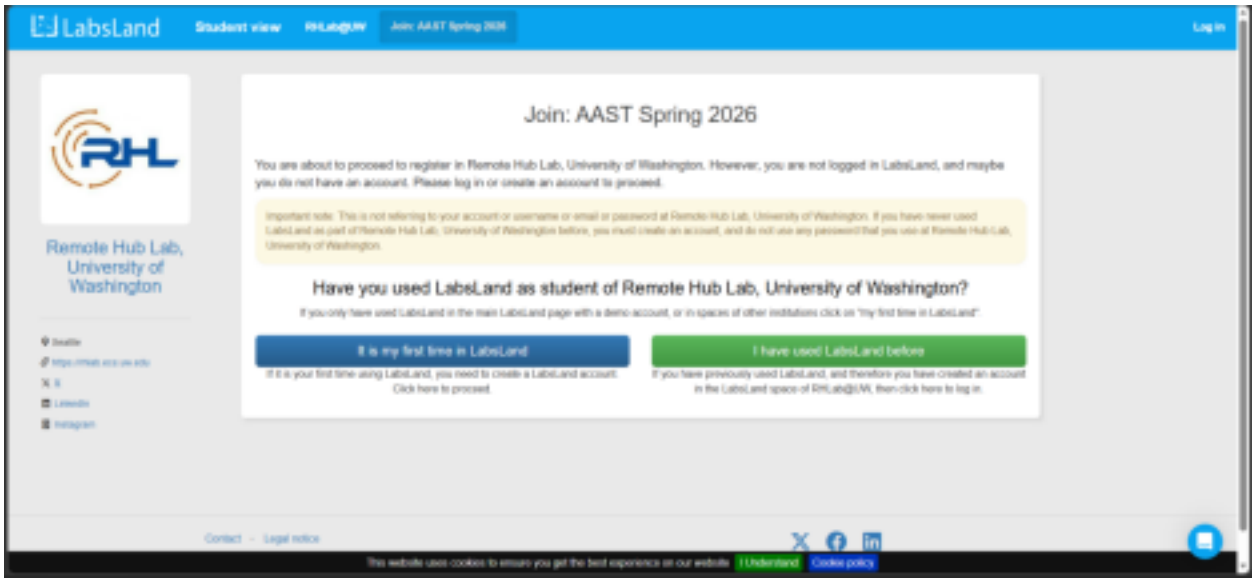

Figure 2

2. If you already have a LabsLand account, sign in by clicking the green icon “I have used LabsLand before”. If not, create a LabsLand account by clicking the blue icon “It is my first time in LabsLand”.
3. After you sign up, you should see an invitation to join the course group in Figure 3. Click the blue icon in the middle of the page, “Yes, add me to your course name”.

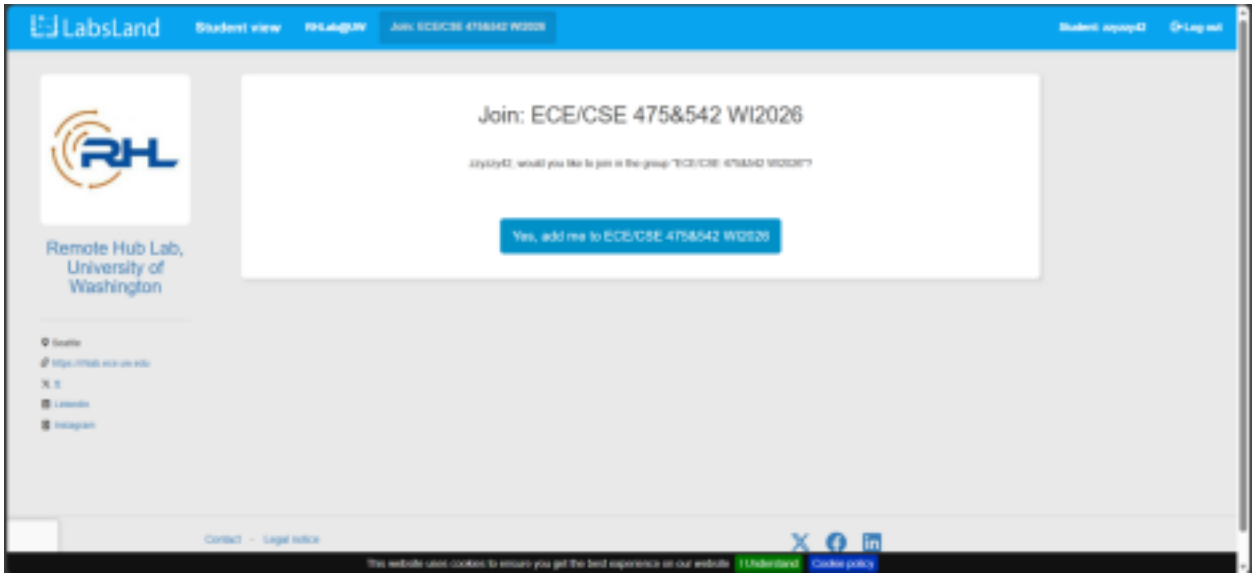

Figure 3

4. You are now a part of the course group, as shown in Figure 4. Now, click the blue icon “Access this lab” to access the STM32 remote lab.

Figure 4

5. For this assignment, we only need to use the basic STM32 remote lab. Click the orange icon indicated by the red arrow in Figure 5.

Figure 5

6. Now we can see the STM32 development board assigned to us in this 3-minute session. Click “Choose File” to upload binary files, as indicated by the red arrow in Figure 6. Then, click the blue Upload button to program the STM32 board. Do not worry about the binary file now. We will create one together later.

Figure 6

7. Figure 7 shows the remote lab after a program is uploaded. Important sections (toggle switches, pushbuttons, potentiometers, and console window) are highlighted.

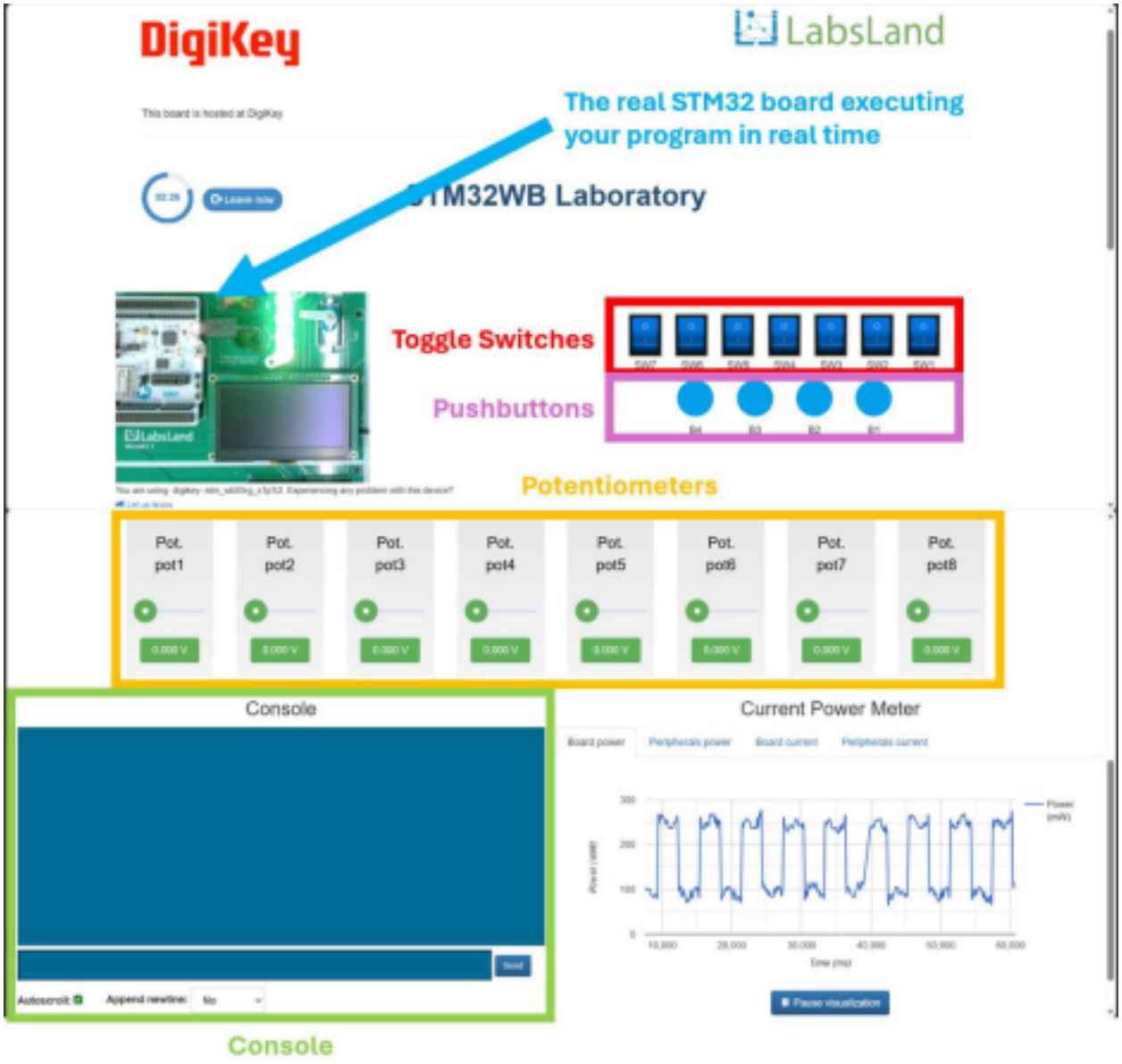

Figure 7

Hardware Connections

The STM32 remote lab has a hardware connection shown in Figure 8. We can find how peripherals, such as LEDs and pushbuttons, are connected to the GPIO pins. We will reference this diagram when we write our programs.

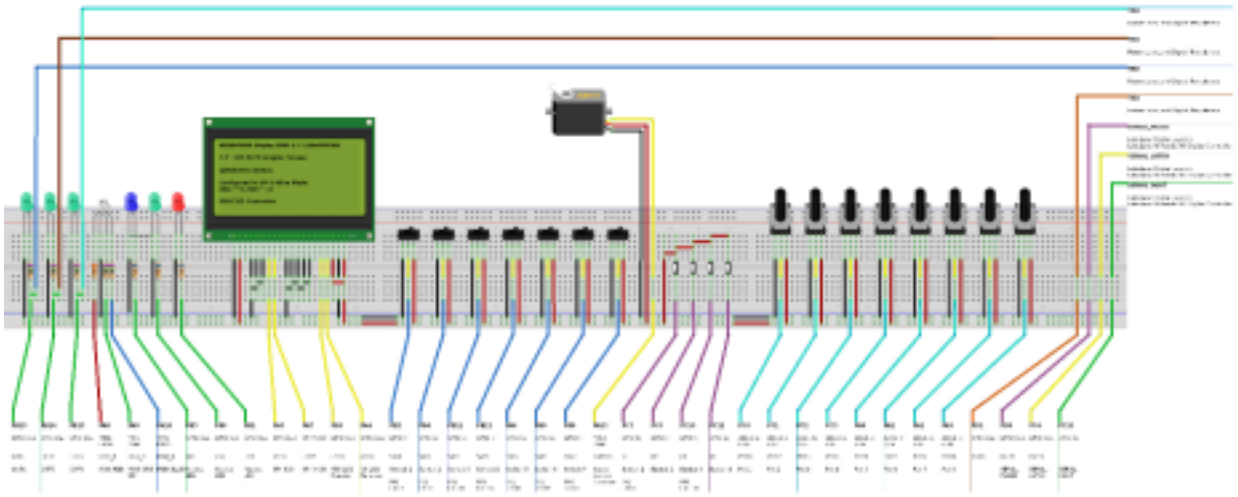

Figure 8

Keil Studio Cloud

In the lab assignments for this course, we will use Keil Studio Cloud to create programs for our STM32 development boards. Keil Studio Cloud is a free, browser-based integrated development environment (IDE) designed for developing microcontroller applications. Developed by ARM, this platform supports a wide range of microcontroller models, providing a flexible and accessible tool for our coursework.

To get access to the Keil Studio Cloud,

1. Go to https://studio.keil.arm.com/
2. You will need to sign in to use Keil Studio Cloud. If you don't have one, create an account first.
3. After you log in and enter the IDE interface, create a new Mbed project by clicking File -> New… -> Mbed Project, as shown in Figure 9.

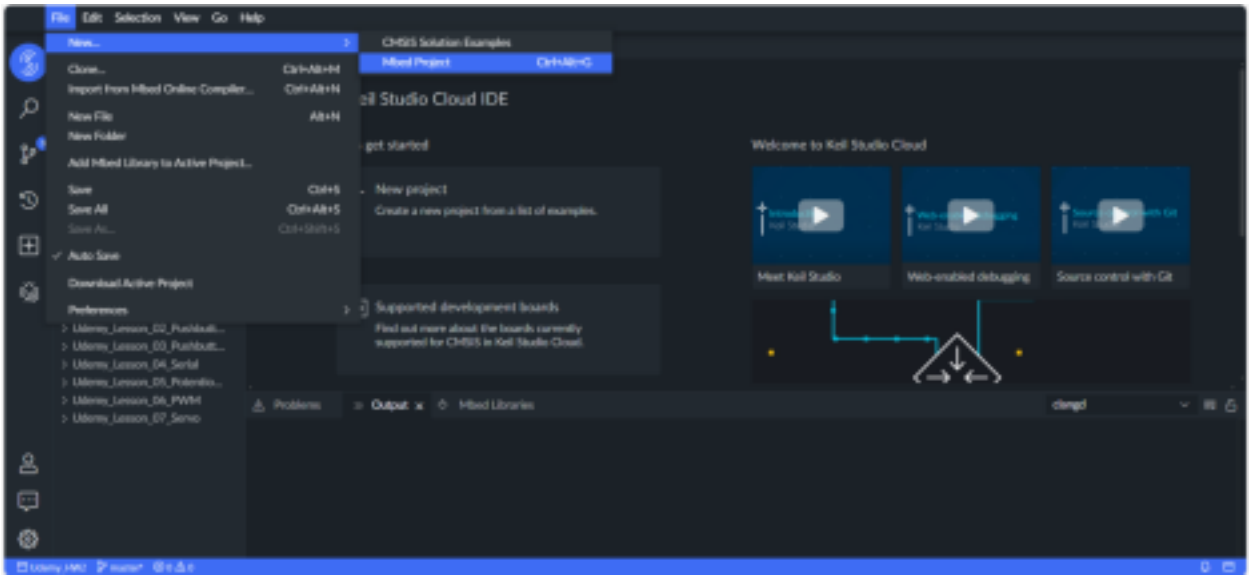

Figure 9

4. We are now prompted to use a starter code/example project and enter a project name. Choose an empty Mbed OS project and enter a project name you like. Next, click Add project.

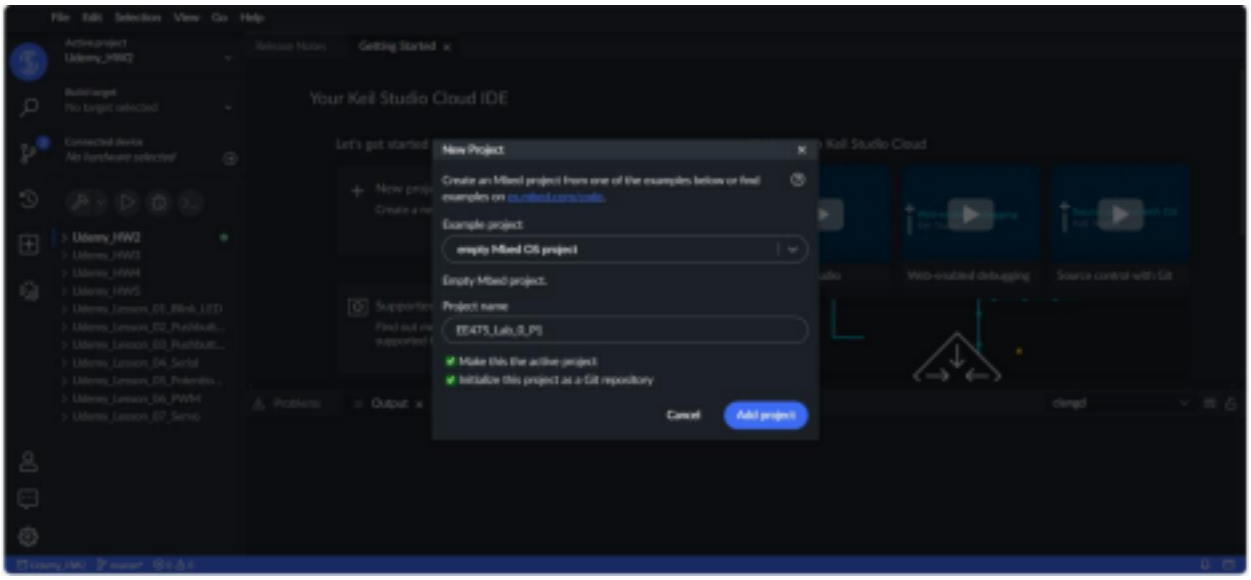

Figure 10

5. Follow the instructions in Figure 11 to ensure the new project is set to the active project. Set the Build target to NUCLEO-WB55RG.

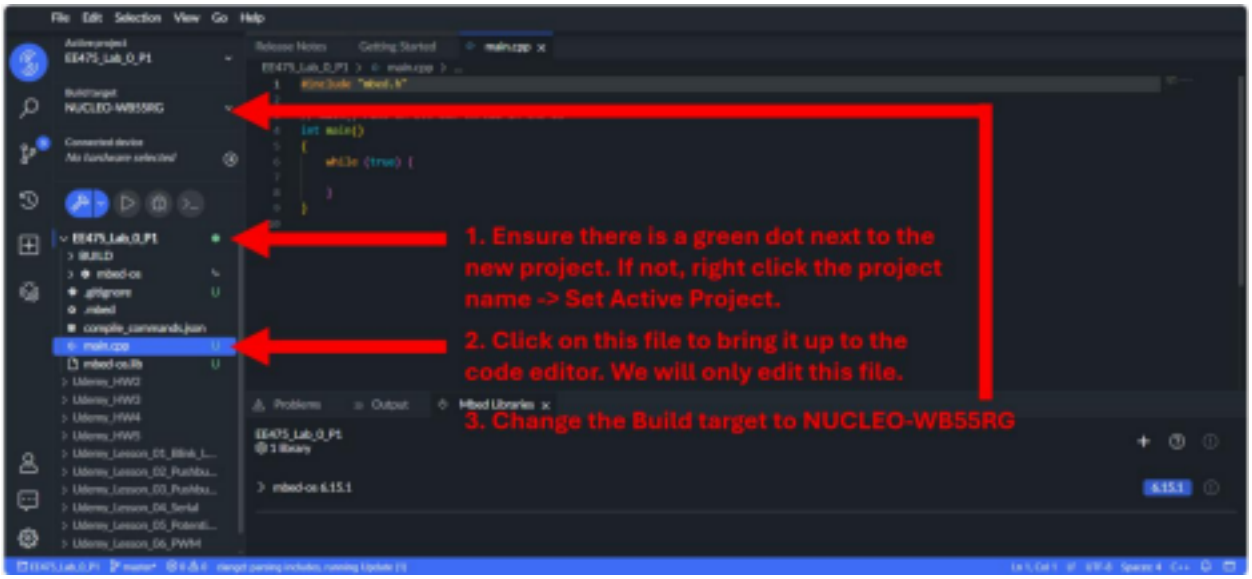

Figure 11

6. In our exploratory assignments, we will only need to edit the main.cpp file.
7. Compile the project by following the instructions in Figure 12. The first compilation can take some time, but subsequent compilations will be much faster.

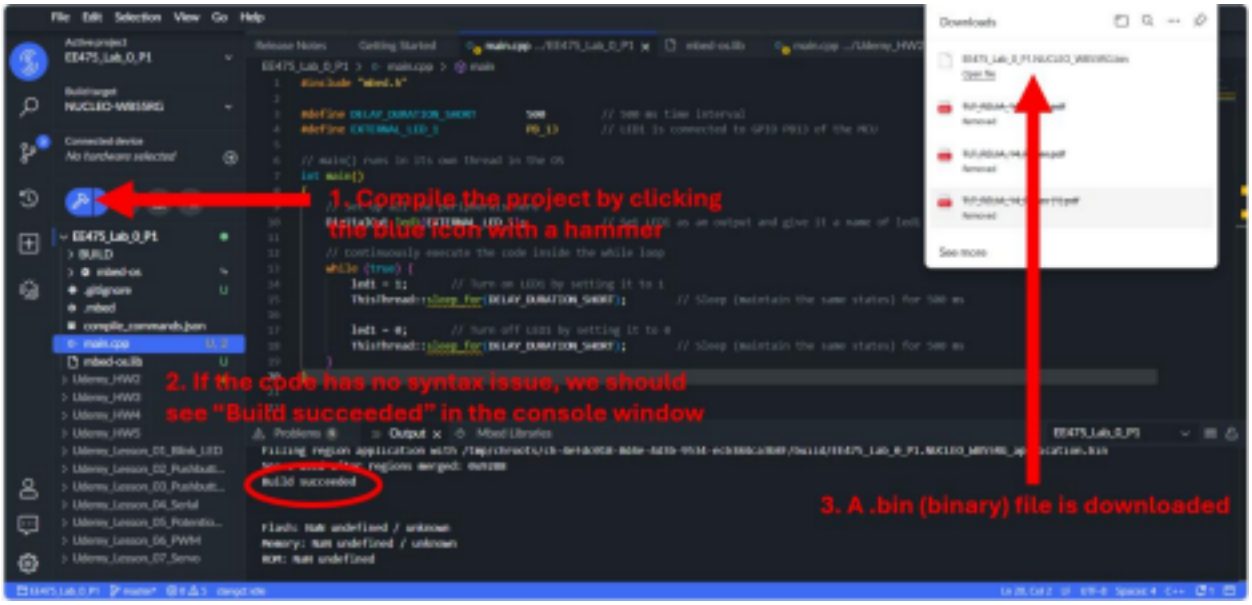

Figure 12

Assigned Task

Part 1

We will now create our first STM32 program. It is a simple program that blinks LED1 every second. Please take the following steps:

1. Create a new Mbed project in Keil Studio Cloud, as instructed in the Keil Studio Cloud section.
2. Copy and paste the following code into the project.

```
1. #include "mbed.h"
2.
3. #define DELAY_DURATION_SHORT 500 // 500 ms time interval
4. #define EXTERNAL_LED_1 PB_13 // LED1 is connected to GPIO PB13 of the MCU 5.
6. // main() runs in its own thread in the OS
7. int main()
8. {
9. // Set up all the peripherals here
10. DigitalOut led1(EXTERNAL_LED_1); // Set LED1 as an output and give it a name of led1 11.
12. // Continuously execute the code inside the while loop
13. while (true) {
14. led1 = 1; // Turn on LED1 by setting it to 1
15. ThisThread::sleep_for(DELAY_DURATION_SHORT); // Sleep for 500 ms 16.
17. led1 = 0; // Turn off LED1 by setting it to 0
18. ThisThread::sleep_for(DELAY_DURATION_SHORT); // Sleep for 500 ms 19. }
20. }
```

3. Compile the project by following the instructions in Figure 12.
4. Upload this binary file onto the STM32 remote lab by following Steps 4 – 6 in the STM32 Remote Laboratory section.
5. We should now see an LED on the top to the left of the servo blinking.

Questions (15 pts)

1. It is important to know where all the peripherals are connected to the STM32 IC. We can do this by referring to the Fritzing diagram. In Figure 13,
    a. What text is hidden under the gray boxes 1, 2, and 3?
    b. What text is hidden under the gray boxes 4, 5, and 6?

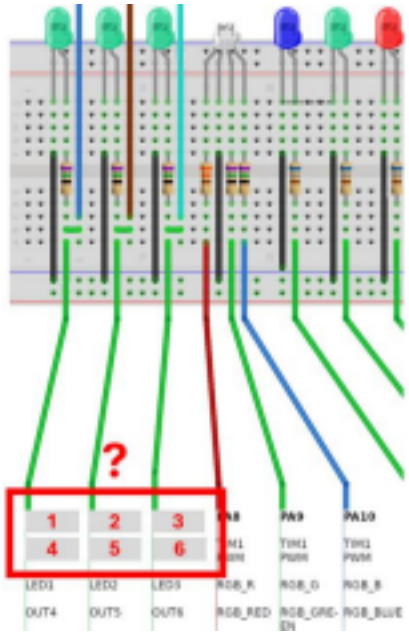

Figure 13

2. Attach a screenshot of the board while the LED is on.
3. If you wanted the LED to blink at a slower rate, such as stay 2 seconds turned on and then 2 seconds turned off, how would you go about it?

Part 2

We will now add pushbuttons into our project. A pushbutton is an input sensor that has different values, either 0 or 1, when it is pressed or released.

Mechanical pushbuttons are not ideal. When pressed or released, they often bounce and do not generate a clean 0->1 or 1->0 signal, also known as the chattering phenomenon. Instead, the signal usually looks like 0->1->0->1->1->0->0->1 or 1->0->1->0->0->1->1->0. An illustration of this phenomenon is shown in Figure 14. To overcome this, we can manually introduce a delay (usually between 50 – 100 ms) in our program after a button is pressed. In other words, we wait until the button stabilizes before we execute the next logic.

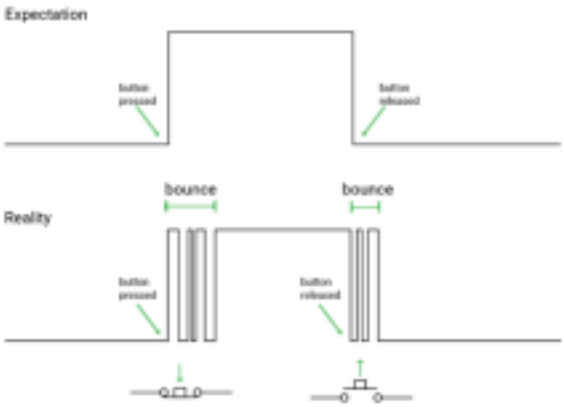

Figure 14

Two important lines of code for a pushbutton and their functionalities are shown below:

```
// Set up a button connected to GPIO EXTERNAL_Button_1 by
setting // its GPIO mode to PullDown and giving it a name
"button1"
// The lab also supports PullUp. Free feel to experiment with different
combinations.
DigitalIn button1(EXTERNAL_Button_1, PinMode::PullDown);

if (button1) { // Read the value of a button by calling its name directly
ThisThread::sleep_for(DELAY_DEBOUNCET); // Sleep for button debounce }
```

Create a program that does the following:

- When Button 1 is pressed (click the blue circle and hold), blink LED1 every 1 second (0.5 s off + 0.5 s on).
- When Button 1 is not pressed, turn LED1 off.

Note: Button 1 in the Fritzing diagram is represented by B1 on absLand.

Questions (20 pts)

1. Which GPIO pin did you use to read the state of Button 1?
2. When setting up peripherals, why did we use DigitalOut for an LED and DigitalIn for a pushbutton?
3. Is button debouncing logic necessary in this application? Why or why not? Try adding or removing it to see if you can observe any difference in behavior.
4. Attach a screenshot of your program in the Keil Studio Cloud.

Part 3

We will now replace the pushbutton in Part 2 with a toggle switch. A toggle switch is a simple electromechanical component that allows a microcontroller to digitally read its state—either 0 or 1—as a binary input for user interaction or mode selection.

Its program is very similar to a pushbutton introduced in Part 2, but because a switch does not suffer from the chattering phenomenon, we do not need to add any debouncing logic. We also do not need to set a PinMode for a switch, as it does not float if its circuit is properly connected.

Two important lines of code for a toggle switch and their functionalities are shown below:

```
// Set up a switch connected to GPIO EXTERNAL_Switch_1 by giving
it // a name "switch1"
DigitalIn switch1(EXTERNAL_Switch_1);
```

```
if (switch1) {} // Read the value of a switch by calling its name directly
```

Create a program that functions the same as Part 2, except with a toggle switch this time:

- When SW1 is set to 1, blink LED1 every 1 second (0.5 s off + 0.5 s on).
- When SW1 is set to 0, turn LED1 off.

Questions (15 pts)

1. Which GPIO pin did you use to read the states of SW1?
2. Attach a screenshot of the board while
    a. The SW1 is set to 1, and LED1 is on.
    b. The SW1 is set to 1, and LED1 is off.
    c. The SW1 is set to 0, and LED1 is off.
3. Attach a screenshot of your program in the Keil Studio Cloud.

Submission Instructions

Please compile your responses to the lab questions into a PDF report. Ensure that your report is well-organized and includes screenshots of the remote laboratory where necessary. Clearly label and number your responses according to the corresponding question numbers.