

EE3220: System-On-Chip Design

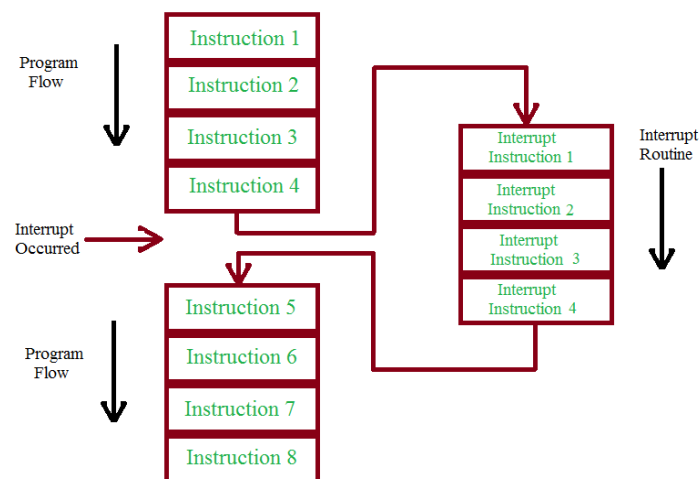
Lab 2: Interrupt Programming

Objectives

At the end of this lab, students should be able to:

- Get familiar with interrupt programming on Mbed
- Configure and program the timer and serial interrupts for the ARM Cortex M4 Microcontroller in the Nucleo- F401RE board using the Mbed OS 6.
- Get more familiar with serial communication and GPIOs with the Mbed platform
- Use Keil Studio Cloud to debug ARM Mbed projects

System Overview:



In this lab, students will learn using interrupt on the Keil Studio Cloud using the Mbed OS 6 and the STM32 Nucleo-F401RE board. We will learn how to configure and handle interrupts on the Mbed platform. Nucleo-F401RE board can handle various interrupt sources. It allows priority to be set with multiple interrupts. In the first part of the lab, an LED will be blinked continuously until a timer interrupt occurs. The LED blinking will be interrupted when a timer timeout occurs. The second part will use both the timer and the serial interrupts. The LED will blink twice when the timer interrupt occurs at the timeout of the timer. On the other hand, the serial communication interrupt is set when a keyboard is pressed. the serial USB UART interrupts the processor to receive the digit typed on the keyboard. After the interrupt, the LED is blinked 'n' times if a digit 'n' is received. For example if 2 is received the LED is blinked 2 times, the LED is blinked 5 times if digit 5 is received, and soon. Finally, the compiled programs will be load and run on the board.

Overview of Interrupt in Mbed OS 6

Interrupt is a common and important feature of processors (microcontrollers/microprocessors) which is a better alternative to polling. It allows the processor to respond to external and internal events very quickly. An interrupt is a concept designed to allow a processor defer its current code execution to execute another code (instructions) written in the interrupt service routine. When an interrupt occurs due to an external or an internal event (e.g., the timer), the CPU stops executing the current instruction and jumps to execute the code called the interrupt service routine (ISR). After executing the ISR, the CPU returns to continue with its previous code execution from where it was before the interrupt. Interrupt occurs after the occurrence of an event

(stimulus) known as the interrupt source. For example, an interrupt can occur when a button is pressed, timer overflows, ADC conversion completes, and many more events. Before these events start to cause an interrupt, the programmer is required to enable them as interrupts, write the ISR, and completes the interrupt settings.

Interrupt is useful in the sense that it allows the processor to continue running the main programming while a time consuming task (such as ADC) is ongoing instead of waiting until the task completes without doing something else (polling). When the task completes, the processor is informed by sending a signal to interrupt the processor. Furthermore, interrupt allows a processor to attend an urgent or time critical event promptly, and allows priorities be set among the occurring events and tasks. When a processor is idle, it can go to sleep to reduce energy consumption. The processor is interrupted whenever there is an available task to do.

Warnings:

- No blocking code in ISR: avoid any call to wait, infinite while loop or blocking calls in general.
- No printf, malloc or new in ISR: avoid any call to bulky library functions. In particular, certain library functions (such as printf, malloc and new) are non re-entrant, and their behavior could be corrupted when called from an ISR.
- For `printfs` from interrupt context, use [Event](#) instead

In Mbed OS 6, the **InterruptIn** Class, **Ticker**, and **Timeout** handle the digital input interrupts, repeated timer interrupt and onetime timer interrupt respectively. Serial interrupt can be achieved using the Serial (deprecated in Mbed OS 6), BufferedSerial and the UnbufferedSerial classes.

Use the **Ticker** and **Timeout** interfaces to set up an interrupt to call a function after a specified delay. The difference between Ticker and Timeout only that Ticker keeps repeating after every time interval while Timeout interrupt occurs only once. To repeat a Timeout interrupt, you need to create (attach it again). Use the Ticker interface to set up a recurring interrupt; it calls a function repeatedly and at a specified rate. You can create any number of interrupt objects to allow multiple interrupts at the same time.

Table I below shows the public member functions for the Ticker and Timeout classes.

TABLE I: TICKER AND TIMEOUT PUBLIC MEMBER FUNCTIONS

Functions	Description
void attach (F &&func, float t)	Attach a function to be called by the Ticker, specifying the interval in seconds.
void attach (Callback< void()> func, std::chrono::microseconds t)	Attach a function to be called by the Ticker specifying the interval in microseconds.
void attach_us (Callback < void()> func, us_timestamp_t t)	Attach a function to be called by the Ticker, specifying the interval in microseconds.
void detach ()	Detach the function.

Use the InterruptIn interface to trigger an event when a digital input pin changes. You can trigger interrupts on the rising edge (change from 0 to 1) or falling edge (change from 1 to 0) of signals. Table below shows the InterruptIn functions you can use to setup the InterruptIn interrupts. We won't test InterruptIn in this exercise due to remote access nature of the lab.

TABLE II: INTERRUPTIN PUBLIC MEMBER FUNCTIONS

SN	InterruptIn Functions	Description
1.	InterruptIn (PinName pin)	Create an InterruptIn connected to the specified pin
2.	InterruptIn (PinName pin, PinMode mode)	Create an InterruptIn connected to the specified pin, and the pin configured to the specified mode (Pull up, pull down etc)
3.	int read ()	Read the state of the input pin (0 or 1)
4.	void rise (Callback < void()> func)	Attach a function to call when a rising edge occurs on the input
5.	void fall (Callback < void()> func)	Attach a function to call when a falling edge occurs on the input
6.	void mode (PinMode pull)	Set the input pin mode
7.	void enable_irq ()	Enable IRQ
8.	void disable_irq ()	Disable IRQ

ARM Mbed and Keil Studio Cloud Brief

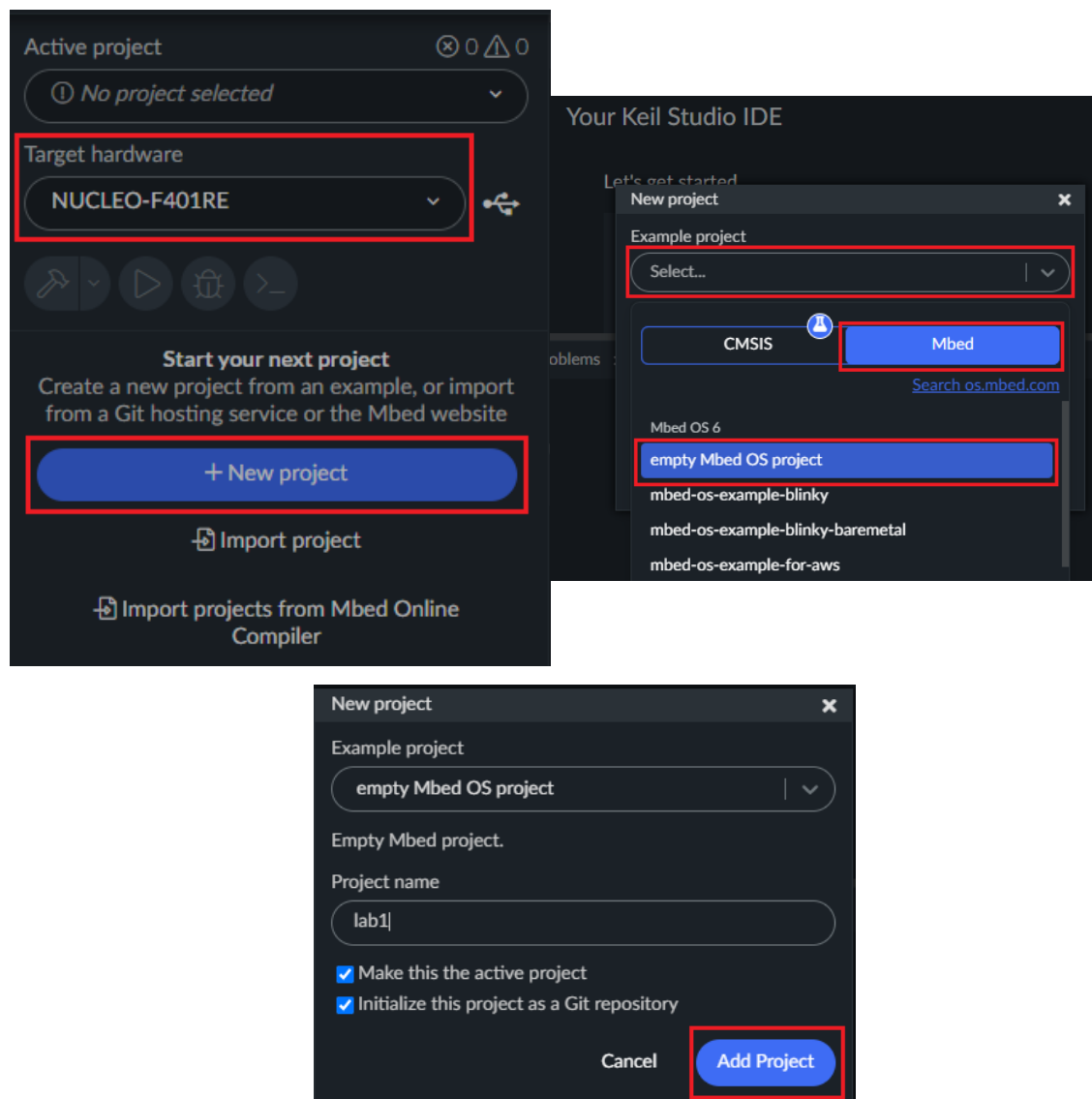
Arm® Mbed is a free online ARM compiler that can be used over an Internet link. It is an IDE platform and operating system based on 32-bit ARM Cortex-M microcontrollers. It is collaboratively developed by ARM® and its technical partners. Mbed is currently supported by over 60 partners and a community of 200,000 developers. It also provides the operating system, cloud services, tools, and developer ecosystem to make the creation and deployment of IoT solutions possible. Using Mbed we can write a program in C/C++, then compile the program, and upload the executable code to the target ARM processor. The advantage of using Mbed is that it is easy to learn and use and is supported by very large number of library functions. One of the major features of the Arm® Mbed™ systems is its web-based development environment. Just plug the device into computer using a USB cable, which will appear on your computer as a USB memory stick. Write and compile your software code using the Arm® Mbed™ Online Compiler, download the compiled code into the device, and press the onboard reset button to run! Mbed platform will be replaced in the near future by the Keil Studio Cloud platform <> also developed by the ARM. The new platform allows for direct debugging and adds many features to the Mbed. It gives you an option to program with the Mbed OS or using the CMSIS.

Part 1: The Interrupting a LED blinking

In this part, we are going to continuously blink an LED in the main program and also setup a Ticker and a Timeout timer interrupts. With the Ticker timer interrupt, the LED blinking is interrupted repeatedly. Using the interrupt, the LED is made to blink for some period and stop blinking for the same period. The pattern repeats forever. On the other hand, the LED blinking is interrupted once using the Timeout timer interrupt. When the Timeout interrupt occurs the LED is made to stop blinking forever unless the program is reset or the interrupt is attached/reset again.

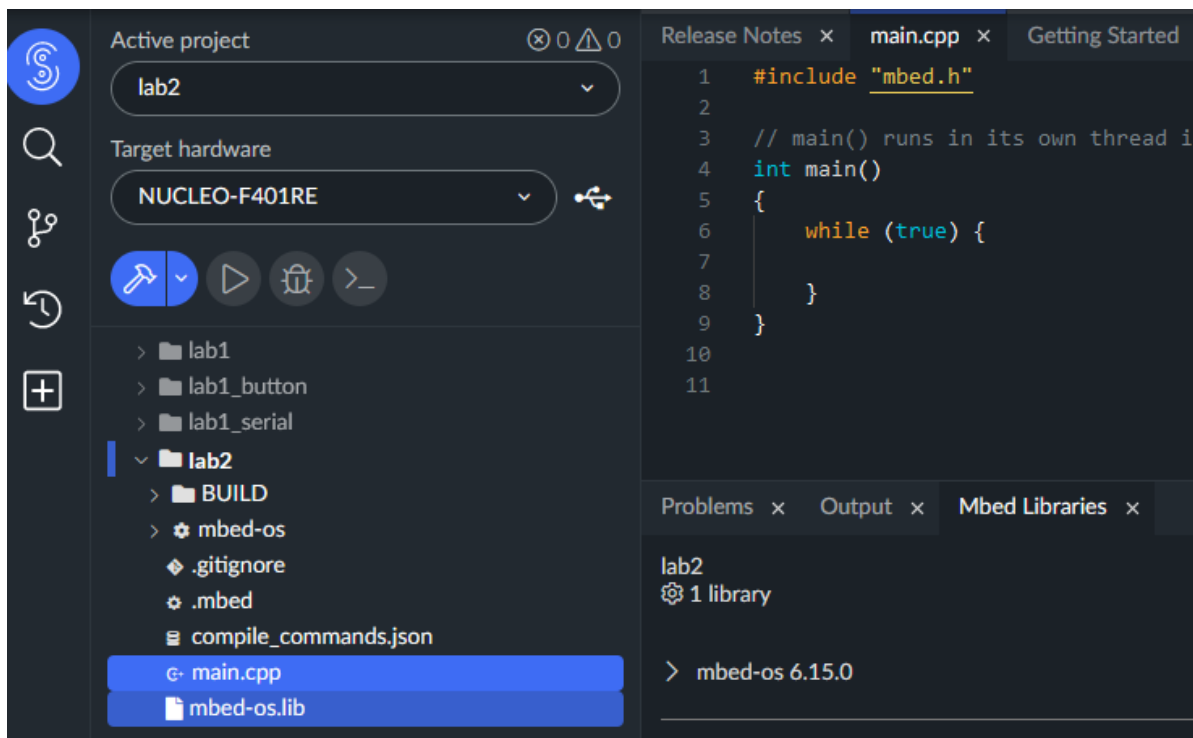
1. New Mbed project creation in Keil Studio Cloud

- Login to your registered ARM or Mbed account on the Keil Studio Cloud page <https://studio.keil.arm.com/auth/login/>
- Select **NUCLEO-F401RE** as the Target hardware.
- Click **+New project** tab or **File -> New project** to create a new project.
- Select Mbed and empty Mbed OS project as the example Mbed OS 6 project.
- Name the new project as **lab2** and click **Add Project** button to create the project.



2. Writing a C/C++ program to blink an LED

- Open the main.cpp file in the lab2 project. Ensure to set Lab2 as the Active Project by right clicking the project and setting it as the Active project.



- Study and then copy and paste the code provided below:

```
#include "mbed.h"

#define BLINKING_RATE    1000ms // Blinking rate in milliseconds

Ticker timer;              // define ticker timer object
DigitalOut led(LED1);

static volatile int blink_state = 1;
static volatile int intr_occurred = 0;

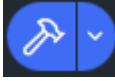
void isr_timer()
{
    blink_state = !blink_state;
    intr_occurred = 1 ;
}

int main()
{
    timer.attach(&isr_timer, 8000ms); // attach timer interrupt isr
    led = 0;

    while (true) {
        if(blink_state){
            led = !led;
            ThisThread::sleep_for(BLINKING_RATE);
        }
        if(intr_occurred){
            printf("Timer interrupt has occurred ! \r\n");
            intr_occurred = 0;
        }
    }
}
```

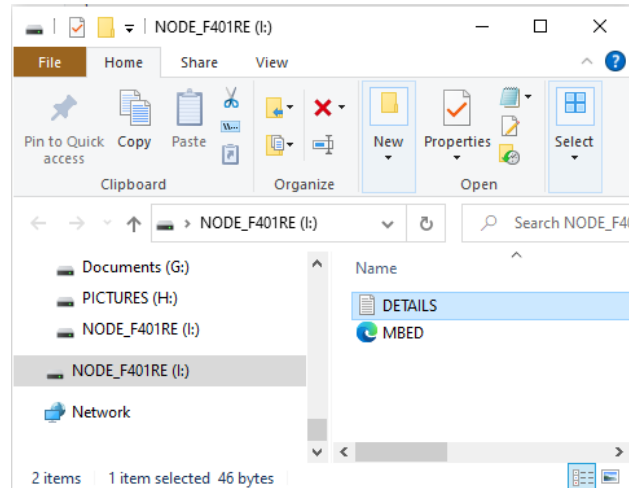
```
}  
}
```

3. Building/Compiling the project

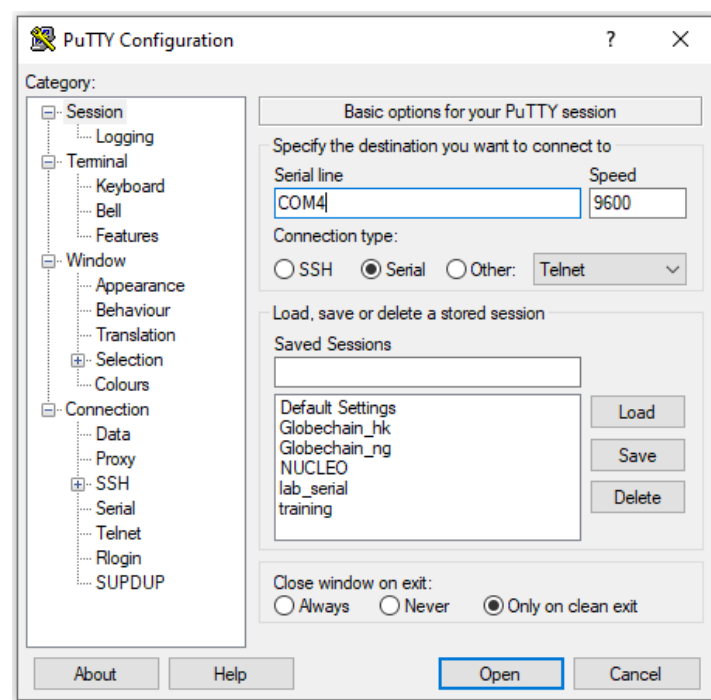
- Click the *hammer icon*  in the Explorer Window to build the project.
- Once the build completes, save the generated **lab2.NUCLEO_F401RE.bin** file in a folder.

4. Running/Programming the program on the board

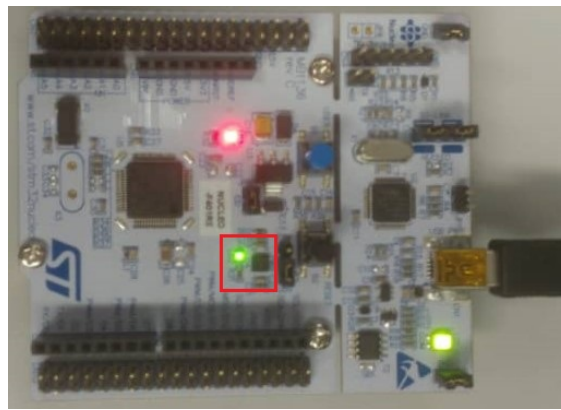
- Connect the board to the computer using the USB cable, the board folder will open.



- Open the **Device Manager -> Ports** and notice the **STLINK Virtual COM PORT** number.
- Open **PuTTY** and set the serial line and the speed (9600 baud rate).
- Click Open



- Copy the **lab2.NUCLEO_F401RE.bin** to the board's folder.
- Now the project runs on the board once the copy finishes
- Observe the blinking of the LED (LD2) on the board



Check Point 1:

Now try using the **Timeout** class instead of the Ticker by replacing the **Ticker** keyword with **Timeout** keyword in the code. What happens now?

[Note: Take enough pictures or videos as evidence for this video. You need to submit the Putty screens and other evidences in the lab report.]

Part 2: Handling Multiple Interrupts

In this part, we handle multiple interrupts i.e. the timer and the serial interrupts. The LED will blink twice when the Timeout timer interrupt occurs at the end of the specified timeout period. On the other hand, the serial communication interrupt is set when a keyboard is pressed. the serial USB UART interrupts the processor to receive the digit typed on the keyboard. After the interrupt, the LED is blinked 'n' times if a digit 'n' is received. For example if 2 is received the LED is blinked 2 times, the LED is blinked 5 times if digit 5 is received, and soon.

- Create a new project and name it **Lab2_b**
- Open the main.cpp file of the project, copy and paste the code provided below

```

/*****
MULTIPLE INTERRUPT PROGRAM
*****/
#include "mbed.h"

#define BLINKING_RATE    1000ms // Blinking rate in milliseconds
#define BLINKING_RATE_2  600ms // Blinking rate in milliseconds

static UnbufferedSerial serial_port(USBTX, USBRX);
static volatile int serial_rx;
static volatile int number_to_blink;
static volatile int timer_timeout;
Timeout timer; // define ticker timer

char msg1[] = "Serial Interrupt: LED will blink ";
char msg2[] = " times!\r\n";

void isr_timer()
{
    timer_timeout = 1;
}

```

```

void isr_serial()
{
    char c;
    if (serial_port.read(&c, 1)) {
        serial_rx = 1;
        number_to_blink = (int)c - (int)48 ;
        serial_port.write(msg1, sizeof(msg1));
        serial_port.write(&c, 1);
        serial_port.write(msg2, sizeof(msg2));
    }
}

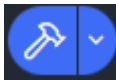
int main(void)
{
    char timer_blink;
    static DigitalOut led(LED1);
    // Set desired properties (9600-8-N-1).
    serial_port.baud(9600);
    serial_port.format(
        /* bits */ 8,
        /* parity */ SerialBase::None,
        /* stop bit */ 1
    );

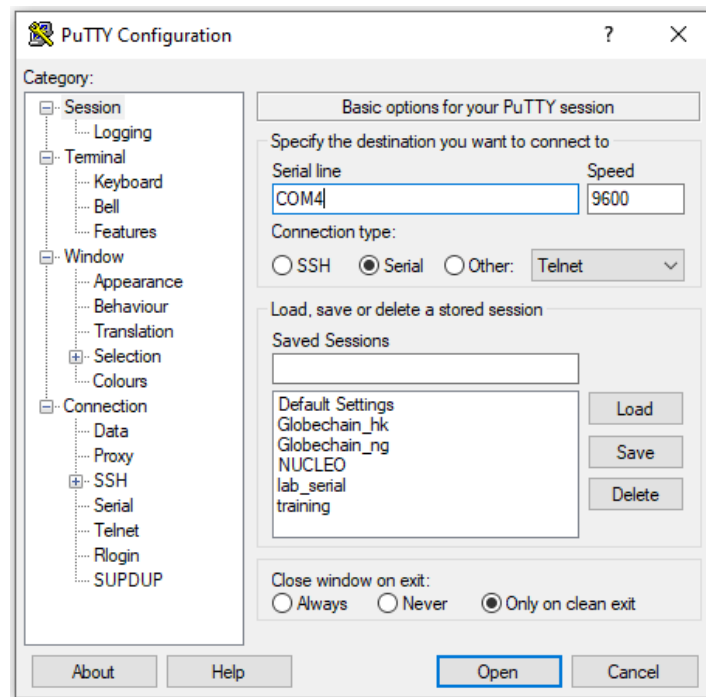
    serial_port.attach(&isr_serial, SerialBase::RxIrq);
    timer.attach(&isr_timer, 8000ms);

    serial_rx = 0;
    number_to_blink = 0;
    timer_blink = 4;

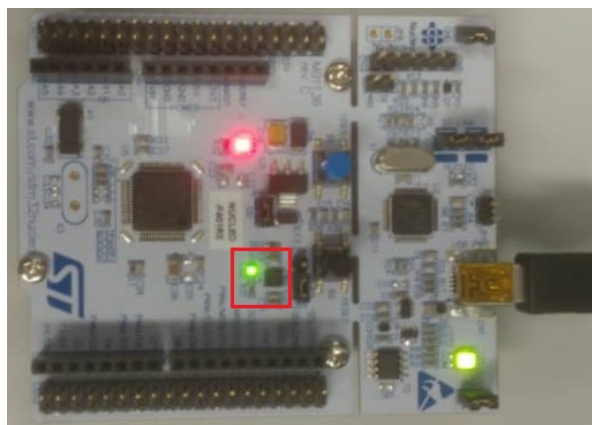
    while (true) {
        if(serial_rx == 1){
            while(number_to_blink >0){
                led = 1;
                ThisThread::sleep_for(BLINKING_RATE);
                led = 0;
                ThisThread::sleep_for(BLINKING_RATE);
                number_to_blink -= 1;
            }
            serial_rx = 0;
        }else if(timer_timeout){
            printf("Timer timeout occured ! \r\n");
            timer_timeout = 0;
            while(timer_blink){
                led = !led;
                ThisThread::sleep_for(BLINKING_RATE);
                timer_blink -= 1;
            }
        }else {
            led = 1;
        }
    }
}

```


- Click the *hammer icon*  in the Explorer Window to build the project.
- Once the build completes, save the generated **lab2.NUCLEO_F401RE.bin** file to your download folder.
- Open the **Device Manager ->Ports** and notice the **STLINK Virtual COM PORT** number.
- Open **Putty** and set the serial line and the speed (9600 baud rate).
- Click Open



- Copy the **lab2_bin.NUCLEO_F401RE.bin** to the board's folder.
- Wait for about 10 seconds to observe the timer Timeout interrupt.
- press a number on the keyboard and observe the blinking of the LED (LD2) on the board.
- Press another number and observe the LED blinking again.



Check Point 2:

Take enough pictures or videos as evidence for achieving this part as the checkpoint. You need to submit the Putty screens and other evidences in the lab report.

Bonus:

Set a breakpoint and try to debug the **lab2_b** project. Take pictures/videos for this task as an evidence.