EE186 Lab 1:
Hello, World!

1 Introduction

Throughout this series of labs, you will use the STM Nucleo-L4R5ZI-P development board to apply embedded systems concepts you've learned in classes. The first lab serves as the "Hello, World!" equivalent for embedded systems. Unlike traditional programming environments where output can be printed to a console, embedded systems might not necessarily have standard I/O interfaces. Instead, the simplest way to verify that your code is running correctly is by interacting with physical components—such as reading input from switches or controlling LEDs. This lab will guide you through setting up and programming the STM32L4R5 microcontroller (MCU) on the Nucleo board. You will learn how to configure the development environment, flash code onto the microcontroller, and program LEDs on the board.

Throughout this and future labs, you will frequently reference the following documents:

- STM32 Nucelo-144 User Manual (UM2179): Datasheet for the Nucleo Board. Useful for finding hardware on the board (e.g. push buttons, LEDs) or pinouts.
- STM32L4R5xx Datasheet (STM32L4R5xx): Provides an overview of the microcontroller's features (e.g. electrical characteristics, pinouts).
- STM32L4 Reference Manual (RM0432): Contains detailed technical information about this L4 series of microcontrollers.

In this lab, we will provide the specific section numbers from the manual to help you locate the necessary information. However, in future labs, **you will be responsible for finding this information on your own.**

2 Flashing & Debugging Code

You will use the STM32Cube Integrated Development Environment (IDE) to flash and debug code on your STM32 microcontroller. Begin by installing STM32CubeIDE and creating a new project:

- Go to `File` → `New` → `STM32 Project`.
- In the Board Selector, choose **NUCLEO-L4R5ZI-P** (see Figure 1).
- Click `Next`, name your project (e.g., `lab1`), click Targeted Project Type as `Empty`, and click `Finish` (see Figure 2).
- When asked to initialize peripherals in default mode, click `No` (Figure 3).

Once the IDE generates your project, open `Core/Src/main.c` and replace its contents with the following code:

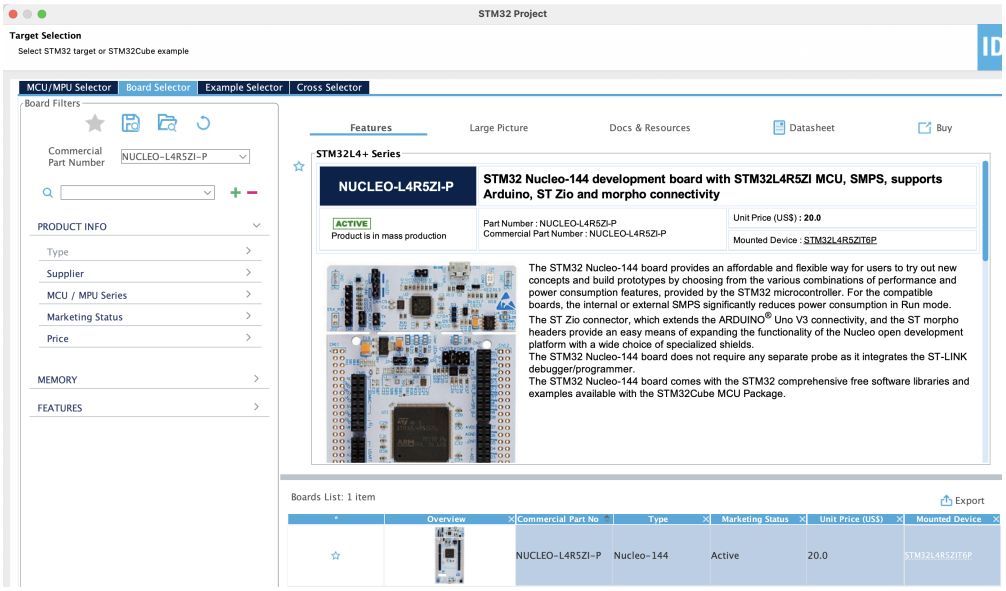

Figure 1: Board Selector during project setup.

```c
#include <stdint.h>

int a = 1;
int b = 2;
int c;

int main(void)
{
    int d;
    c = a + b;
    d = c;

    while (1);
}
```

Once your code is written,

1. Click `Run` → `Debug As` → `STM32 MCU C/C++ Application`.
2. The debugger will start. Step through the code line-by-line using the step controls in the toolbar.
3. Use the `Expressions` tab to watch variables `a`, `b`, `c`, and `d`:
   - Observe their values before and after assignment.
   - Notice what happens when the program enters the while loop.

Deliverables

- (4 Points) When you click `Run > Debug As > STM32 MCU C/C++ Application`, STM32CubeIDE compiles your code and flashes it to the microcontroller via the ST-Link debugger. Describe what happens during the flashing process. What files are generated when you build the project? What part of memory is written to on the MCU? What enables STM32CubeIDE to communicate with your board? What tool or protocol is used to transfer the compiled binary to the microcontroller? An answer to each question is worth one point.

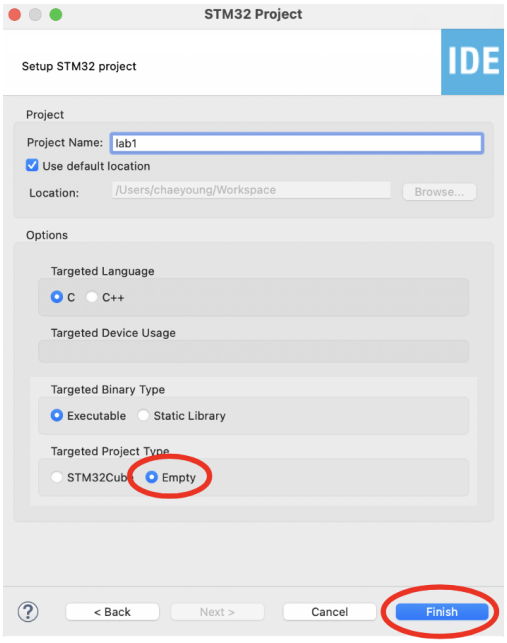

Figure 2: Initialize project as empty.

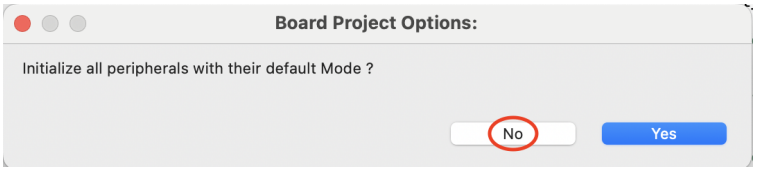

Figure 3: Decline to initialize peripherals with default mode.

- (2 Points) Screenshot of STM32CubeIDE debugger with the `Expressions` window open, showing the values of `a`, `b`, `c`, and `d` as the program executes.

3 Blinking LEDs

In this part of the lab, you will learn to blink the LEDs on the provided Nucleo-L4R5ZI-P board. Let's first understand your board's LEDs.

1. What user LEDs are available on the NUCLEO-L4R5ZI-P board?
2. What are the pin names (e.g., `PA5`, `PB13`, etc.) associated with each LED?
3. Are these pins configured as GPIOs? What does that mean?
4. Should they be configured as input or output? Why?

We will toggle these LEDs using software delays without using timers or interrupts.

The user LEDs are controlled by the General-Purpose Input/Output (GPIO) peripherals. Let's review the concepts of GPIO hardware. The GPIO peripherals in the STM32L4R5 are connected to the processor through the AHB2 bus. Using it, the processor can read or write to the peripheral as if it were a memory location. In other words, we can treat GPIO peripherals as a Memory-Mapped Input/Ouput (MMIO). For example, if GPIO Port A has an address of 0x10001234, you can access it as follows:

```
uint32_t* gpio_a = (uint32_t*) 0x10001234;
*gpio_a = 1; // write
int x = *gpio_a; // read
```

By default, GPIO peripherals are disabled to save power. Before using them, we need to enable their associated clock using the Reset and Clock Control (RCC), specifically the RCC AHB2 register, which controls the peripherals. First, identify which GPIO pins are connected to the three LEDs. Locate the register address for the RCC AHB2 peripheral clock control, and enable the clock for the relevant GPIO port by setting the appropriate bit. When finding the clock register address, look for the base address and the offset address.

After enabling the clock, use the debugger to inspect the RCC AHB2 register and verify that the correct bit has been set. Take a screenshot of this and include it in your lab report.

Now, let's configure the GPIO pins. Each LED is connected to a specific GPIO pin, which must be configured properly before use. For each pin, configure the following settings: mode, output type, output speed, and pull-up/pull-down. Refer to Section 8.4 of RM0432. You can find the base register address in page 97 in the same reference manual.

After configuring the GPIO pins, you will implement a sequence that turns the LEDs on and off in the following pattern:

1. Turn on the green LED.
2. Wait using a software delay. A simple delay can be implemented using a decrementing counter in a while loop.
3. Turn off the green LED and turn on the blue LED. Turn off the blue LED and turn on the red LED.
4. Repeat this pattern from Step 1.

Deliverables

- (2 Points) Answer questions 1-4 about LEDs in one paragraph.
- (3 Points) Take a screenshot of the debugger showing the RCC register values after enabling the GPIO clock.
- (5 Points) Submit a short video demonstrating the blinking LEDs along with your source code.

4 Blinking LED in Assembly

Now that you have implemented LED blinking in C, you will write the equivalent program in Assembly. This exercise will deepen your understanding of how the processor interacts with peripherals at the instruction level.

For example, in C, enabling the GPIO clock might look like this:

```
uint32_t *rcc_ahb2enr = (uint32_t *)0x10001234;
*rcc_ahb2enr = 0x00001000;
```

The equivalent Assembly code is:

```
.equ RCC_AHB2ENR_ADDR, 0x10001234
movw r0, #:lower16:RCC_AHB2ENR_ADDR
movt r0, #:upper16:RCC_AHB2ENR_ADDR
ldr r1, [r0]
mov r1, #0x00001000
str r1, [r0]
```

Using this example as a reference, write an Assembly program that enables the GPIO clock, configures the appropriate GPIO pin, and **turns the blue LED on once**. In other words, unlike in the last section, you only need to set up for the blue LED and turn it on once without using a software delay.
It is possible to insert Assembly code into a C file using inline Assembly.

```
int main (void) {
    __asm__ volatile (
        "LDR R0, =0x10001234\n\t" // Load address to R0
        "LDR R1, [R0]\n\t" // Load value of R0 to R1
    );

    while (1);
}
```

You can compile the code and flash it on the board for validation.

Deliverables

(5 Points) Write Assembly code that turns the blue LED on once. Submit this Assembly code as part of a complete C file (.c) using inline assembly.