**Internet of Things - Intelligent and Connected Systems**

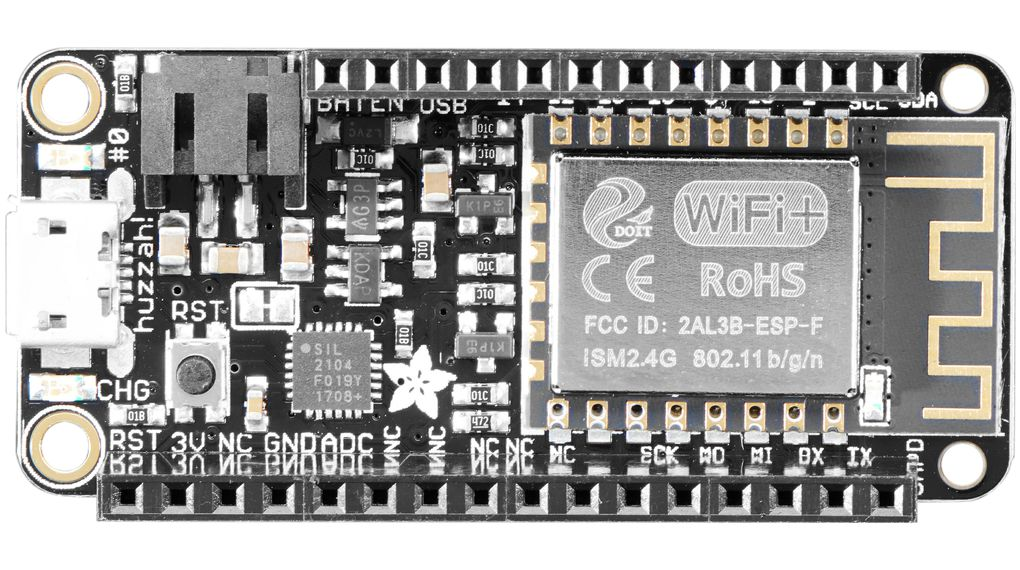
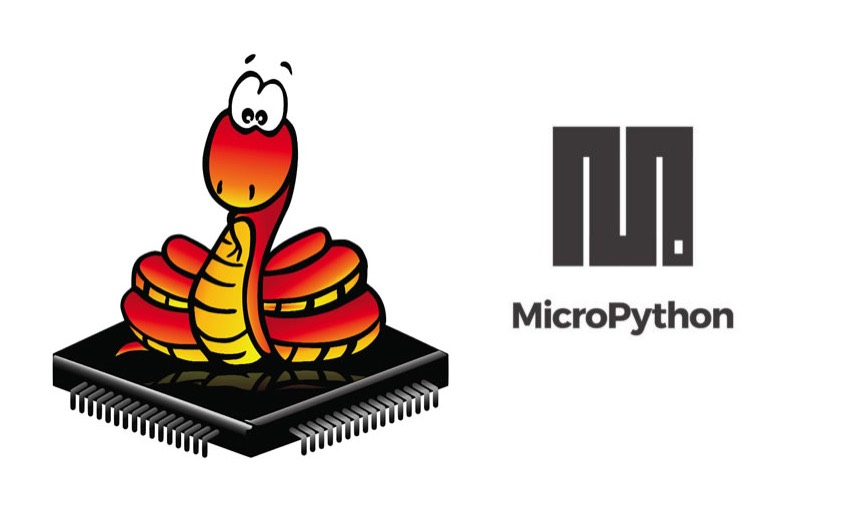
**EECS E4764**

**Columbia University**

**Prof. Xiaofan Jiang**

**Fall 2022**

**Lab 1 - Environment Setup and Feather HUZZAH Introduction**

****

# Part 1: Setting Up Environment

Before we start programming our microcontroller, we need to set up the environment on our computer. We will first install Python and some Python packages, then flash the microcontroller with MicroPython firmware.

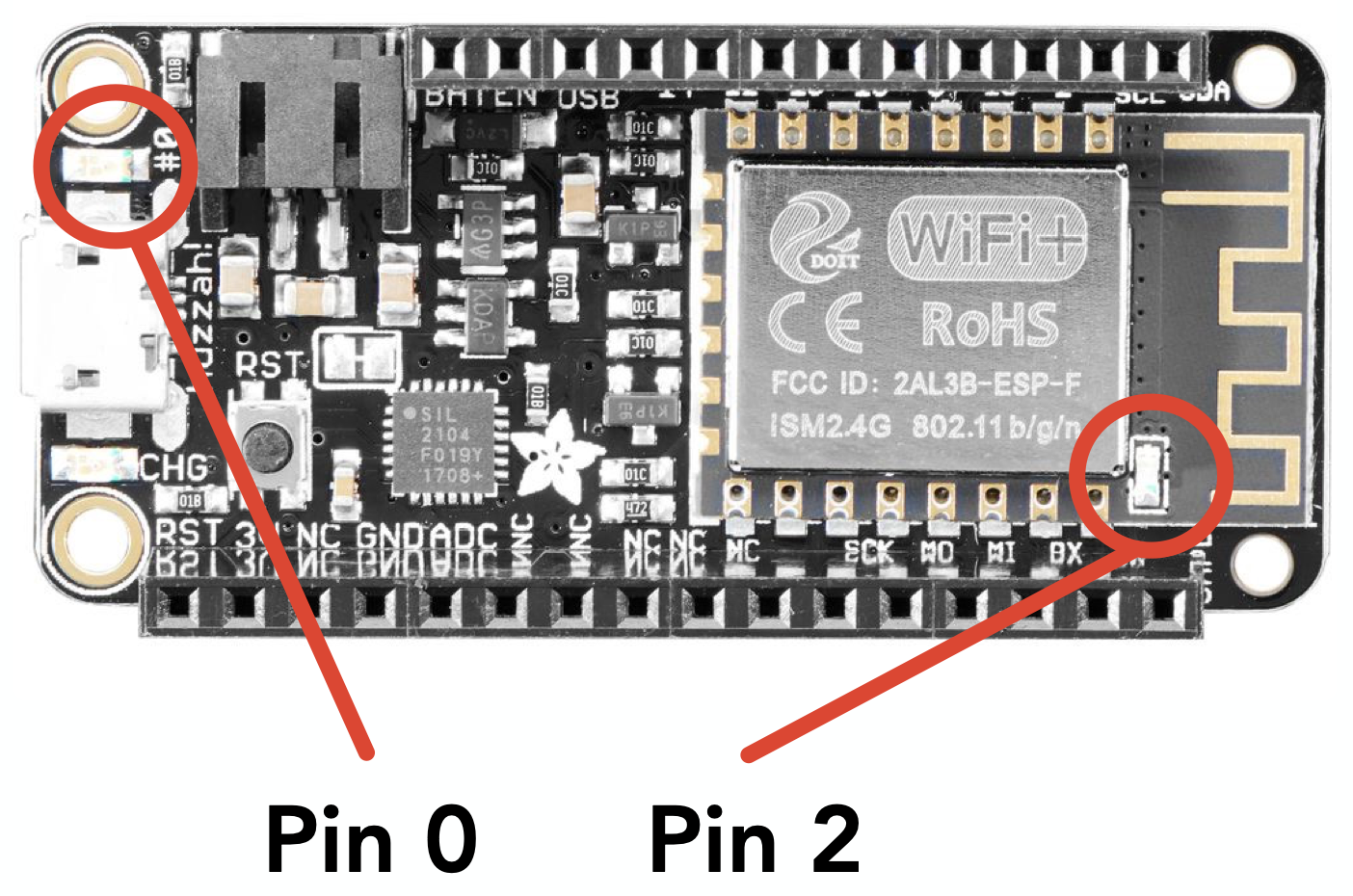
## Task

Follow the instructions on the document “Setting Up Environment” to complete the set-up for your specific OS.

# Part 2: Blink LEDs

Now that you have the environment set up to be able to code and run programs on your HUZZAH, we will now write our first few programs. These exercises hopefully can get you familiar with the Feather HUZZAH ESP8266 environment, which will form the basis of the smartwatch.

In this part, we will write programs to interact with the two built-in LEDs on Feather HUZZAH, circled below.

****

## 

## Tutorial

Before we start, let’s first learn how to upload code onto our ESP8266 board. We have tried typing code to the Python interpreter in “repr” mode during our environment setup, but what if we want our code to be persistently stored in our microcontroller and automatically executed when the board is powered on?

Turns out our board contains an internal flash that can store .py code files we upload from our computer. When the microcontroller boots up, it first finds, and if it exists, executes “boot.py”, then “main.py”. “boot.py” is usually used for low-level initial set-ups, like Wi-Fi configuration, etc, whereas “main.py” is the main entry point.

To save code in our microcontroller, we simply create an “main.py” and upload it onto the microcontroller using *mpfshell*, then click the “RST” button to reboot the microcontroller for it to start running.

Give the following code a try. Save it as main.py and use mpfshell to upload it onto the ESP8266. Remember if you are on macOS or Linux, you have to remove “/dev/” from the <PORT\_NAME>. If you are on Windows, your <PORT\_NAME> is COM\*.

> mpfshell -nc "open <PORT\_NAME>; mput main.py"

***main.py***

| *from* machine *import* Pin  *import* utime  builtin\_led = Pin(0, Pin.OUT)  antenna\_led = Pin(2, Pin.OUT)  builtin\_led.value(1) *# Built in LED - 1 is off*  antenna\_led.value(0) *# Antenna LED - 0 is on*  *while True*:  builtin\_led.value(*not* builtin\_led.value())  antenna\_led.value(*not* antenna\_led.value())  utime.sleep(1) |
| --- |

If you don’t like the name main.py, you can name your code lab1\_check1.py, and simply put one line in main.py, then upload both files.

import lab1\_check1

This line will upload all \*.py in your current folder

> mpfshell -nc "open <PORT\_NAME>; mput .\*\.py"

You can read more on the usage of mpfshell on

<https://github.com/wendlers/mpfshell#shell-usage>

## **Task 1**

Write a program to blink one of the built-in LEDs (see picture above for pin number) to encode “SOS'' in morse code. In morse code, each letter is represented by a series of dots (short tones) and dashes (long tones). You should replicate this on your LED. For dots, the LED should blink at around twice the speed of dashes to represent the string.

| **Checkpoint 1**  Continuously blink a morse code message through the HUZZAH using an LED |
| --- |

## 

## 

## 

## 

## 

## Task 2

Write a program to blink two separate built-in LEDs simultaneously at two different rates (e.g. blink LED 1 at 100 ms, LED 2 at 500 ms). The stipulation is that you are not allowed to use any interrupts or timers, and that the blinking rates should be discernible to the human eye.

| **Checkpoint 2**  Simultaneously blink two separate LEDs at different rates without using interrupts or timers |
| --- |

# Group Sign-up Sheet:

# https://docs.google.com/spreadsheets/d/1Yzf6O64La-aEe6N5dM1aiW6xOHqLUZt6RBmYAqhNAKo/edit#gid=0

# Lab 1 Checkpoints:

1. Continuously blink a morse code message through the HUZZAH using an LED.

2. Simultaneously blink two separate LEDs at different rates without using interrupts or timers.

# Submission:

Submit the file for each checkpoint onto courseworks → Assignments → Lab1

The submitted files should be named like

labx\_{member1\_uni}\_{member2\_uni}\_{member3\_uni}\_check{X}.py

There should be 2 files for this lab.

e.g.

* lab1\_kh3119\_sm4928\_rl3154\_check1.py
* lab1\_kh3119\_sm4928\_rl3154\_check2.py