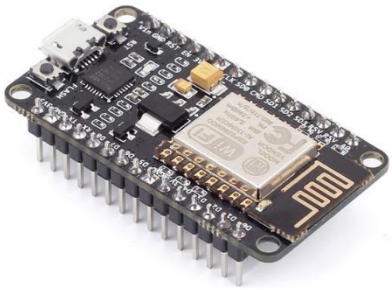
EECE 4263: Embedded Systems Name: _____
Due: 12/17/23, 5:00pm

Lab01

Introduction to the NodeMCU ESP-12E Development Board



Overview:

As part of this course, you'll be using the popular **NodeMCU ESP-12E** development board manufactured by Amica. The ESP-12E is an open source, single-board, ESP8266-based microcontroller development platform containing various I/O pins and built-in WiFi. Because of its size, power requirements, and built-in wireless connectivity, it is ideally suited for Internet of Things (IoT) applications. This lab will acquaint you with the NodeMCU and its programming environment and give you more practice using GitHub and GitHub Classroom.

Background:

There are at least three versions of the NodeMCU Devkit in the wild, and lack of standardization among manufacturers has created a rather confusing set of features and nomenclatures. A good introduction to the various types of NodeMCU development boards can be found at **frightanic.com/iot/comparison-of-esp8266-nodemcu-development-boards/**. Use this link and the fact that we're using the **Amica NodeMCU 1.0/V2** board to answer the following questions. (*NB:* **en.wikipedia.org/wiki/NodeMCU** will also prove useful. (You will need to do some searching on your own to answer all these questions!)

1. What CPU does the NodeMCU 1.0/V2 use? _____
2. What is its clock frequency? _____
3. How much RAM memory does it have? _____
4. How much Flash storage does it have? _____
5. Looking at the NodeMCU pin diagram, how many RS232 lines are there? _____
6. What is the resolution of the on-board Analog to Digital Converter? _____

All questions relate *specifically* to the **NodeMCU 1.0/V2**.

7. How many GPIO pins are there? Identify which ones are ***not*** usable by you and explain why.

8. The NodeMCU board uses a CP2102 chip from Silicon Labs. Why? Be specific.

Prelab – Cloning the Remote Lab01 Assignment Repository

There is a pre-lab assignment that should have been completed. (If not, you will need to have me add you to the GitHub Classroom at before you can get the start repo.

You should have received an email from your instructor containing a GitHub link associated with this lab assignment. The link takes you to **GitHub Classroom** where you'll see an "Accept this assignment" link. Click to accept the Lab01 assignment and GitHub will create a remote repository containing files you need for this lab. Once the repo has been created, a link to it will be shown (e.g. `https://github.com/EECE4263_22-23B/Lab01-YourGitHubUserName`). Click that link to go to the cloud `lab01-YourGitHubUserName` repo.

A few things to note. First, you have full read/write access to this repo but the owner is an organization named **EECE4263_22-23B**, not you. Future assignment repos will also be owned by **EECE4263-S2023** and will be named similarly (i.e. `labXX-YourGitHubUserName`). Second, assignment repos are **private**, and only you and your instructor have access to them, so do not share the repo with other students. Third, this repo is remote so **you'll need to create a local repo** to edit-, stage-, and commit changes (explained below). Of course, periodically you should stage, commit, and push your local changes up to the remote repo on GitHub for safekeeping!

> Once you've completed a lab, the remote repo becomes part of your submission. Therefore, **do not rename or delete the remote assignment repository**, as it will become **ungradable**.

The process of creating a local copy of a remote repo is called "**cloning**." You can do this using whatever means you are comfortable. If you have not used GitHub before (or even if you have), consider one of these options.

If you are on a lab computer, the old GitShell should still be available and can be used as it has been in the past. You must use a USB key to save your files to make sure things don't get lost. If you are using your own computer, the files could be on your hard drive.

It is recommended to build a directory tree that looks like this:

- EECE4263
  - Lab01
  - Lab02
  - …

GitHub Desktop (Alternative #1)

From the makers of GitHub, this is a Gui interface that is designed for GitHub.

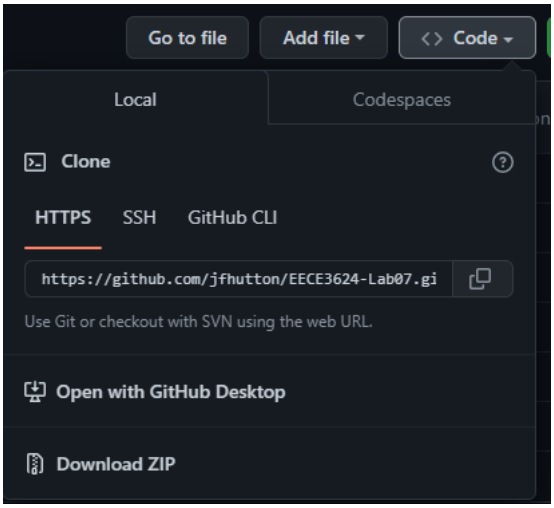
Install from here: https://desktop.github.com/

Once you have it installed and linked to your GitHub account, you can do:

- `File->Clone repository`
- Set/select the cloud repo and give it the local repo target location
- Download the repo

When you want to update your cloud repo from the local, simply select `Push origin` on your GUI and you will enter a message and update the files.

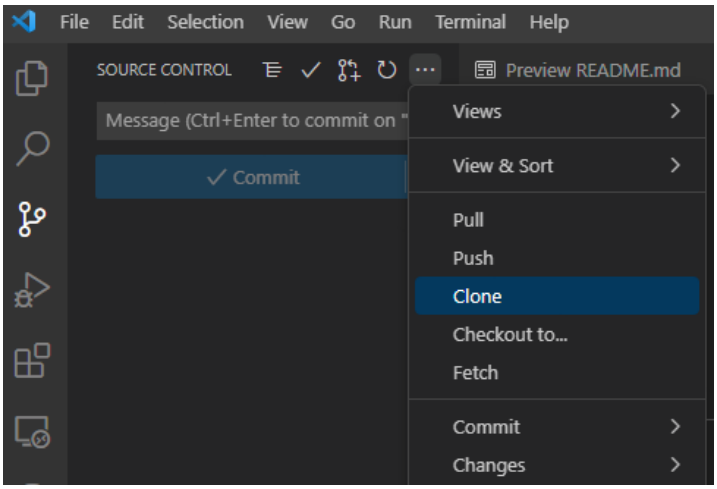
A shortcut from the GitHub repo is to simple select the code pulldown and select `Open with GitHub Desktop`.



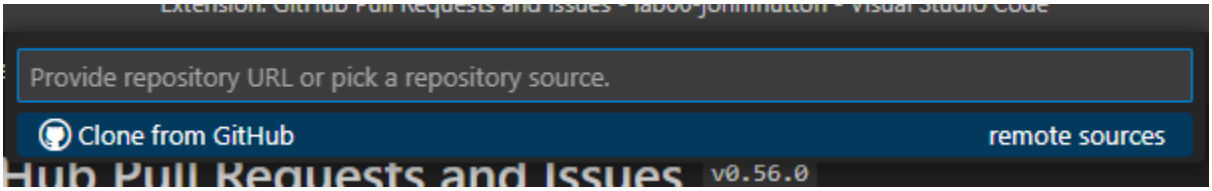
VS Code Integrated Git (Alternative #2)

If you plan to use Visual Studio Code + Platform IO instead of the Arduino + ESP8266 environment, you may want to use the VS Code integrated Git tools.

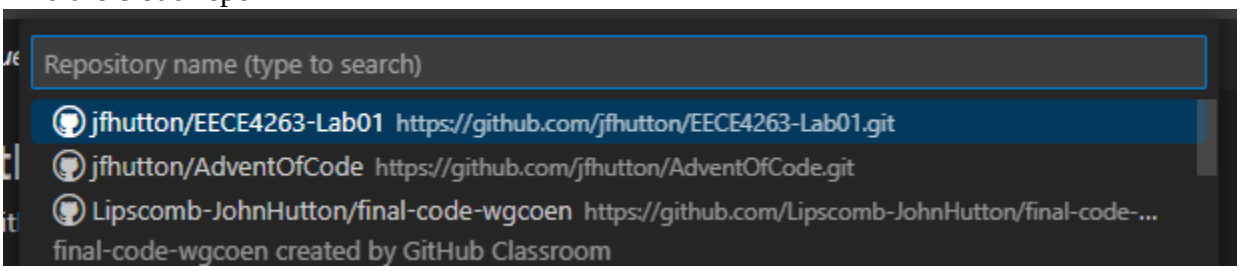
- Select `Source Control` from the left toolbar in Code. 
- From the ···, select `Clone`



- Select `Clone from GitHub`



- Find the cloud repo



- Open the directory for the local repo and `Select as Repository Destination`.

To commit your files after changes, you can use the options in this same side window to do the commit.

Any other git interface (Alternative #3+)

If you use another method, then the GitHub Code pulldown will have the HTTPS or other links you will need to clone the repo in the tool of your choice.

Part of this first lab is **making sure this works** as we will use this for the entire course to check in code.

Now you're ready to begin the lab.

Lab Procedure:

Originally, ESP8266-based NodeMCU boards were programmed in LUA, an embedded scripting language. Later, board definitions were created for the Arduino IDE environment allowing NodeMCUs to be programmed using the Arduino IDE. You'll be using the Arduino IDE.

The lab computers are already set up for programming NodeMCUs via the Arduino IDE. It is also possible to use VS Code + Platform IO to write, compile, and upload code to our NodeMCU devices.

Arduino IDE (Option #1)

If you need to install on your own computer, the IDE can be found here: https://www.arduino.cc/en/software

You'll need to add the proper board definition via the Arduino Boards Manager. This is easy to do and is fully explained in the "**Installing with Boards Manager**" section of the README.md file found at https://github.com/esp8266/Arduino.

VS Code + Platform IO (Option #2)

VS Code + Platform IO may be required for a future version of this class as it has support for inline debugging (if we upgrade from ESP8266 to ESP32 MCUs). Today, it works like the Arduino environment. You can install VS code from here: https://code.visualstudio.com/download. Once installed, you must add the extension PlatformIO IDE.

This link has some breakdown of the steps: https://www.electronicshub.org/programming-esp8266-using-vs-code-and-platformio/. (Note: I do NOT think the python is required, but I have not fully tested this as I do have python on my test system for other reasons. I would try without it 1st.)

Double Blink Programming

Connect the NodeMCU board to your computer via the micro USB cable. Open the Arduino IDE, then open the Tools menu. Select **NodeMCU 1.0 (ESP-12E Module)** as the Board type. (Or, for VS Code, follow the "Create a new project" from the link above.) The default upload speed is 115200. This is quite reliable but will take about 15-20 seconds to transfer even small programs to the NodeMCU. I've had very good luck using an upload speed of 921600, and the download only takes a few seconds. However, **YMMV**…

In ArduinoIDE, `Choose File -> Examples -> 01.Basics -> Blink` to open a simple program to blink the NodeMCU's on-board LED.

In VS Code, chose `Project Examples`, find `arduino-blink`, and `Import`. In the `platformio.ini` file, make life easier by deleting all environments accept `env:nodemcuv2`.

Immediately save this file into your `EECE4263\Lab01` directory.

In Arduino, using the `Tools -> Port` menu, select the proper COM port, then click the Upload icon.

In VS Code, select the PlatformIO:Upload from the bottom option or from the left side menu

options.

After the code compiles and uploads you should see a red or blue on-board LED blinking about once per second.

The ESP-12E module on the NodeMCU board contains a second LED (also red or blue). Use the `ESP-12E_Schematic.png` file found in your lab repo to determine which GPIO pin this LED is attached to. Modify the Blink program to **make the two LEDs alternately flash** as follows: The first LED should come on for 40mS then go off for 160mS. Following that, the second LED should come on for 40mS and go off for 160mS. Repeat this pattern indefinitely. **Show your working program to your instructor**.

Once your program is running correctly, update the comments section at the top of the program to indicate that you modified this code on a certain date, and save the file, then use Git Shell to **stage** any changed files, **commit** them (with a descriptive message!) and **push** them to the remote repo. Next, answer these questions:

Determine which **GPIO ports** (e.g., GPIO12, GPIO14) the first and second LEDs are connected to, and which **digital pins** those GPIO ports correspond to (e.g., D6, D5). Also, determine whether the LEDs are active high or active low devices. Write your answers below.

	GPIO Port #	**Digital Pin #**	**Active High or Low?**
LED1			
LED2			

Regarding the active state of the two LEDs (High or Low), **check that the code's original comments agree with your findings**. If they don't, update the comments and push your changes to your remote assignment repo.

MAC Address

Lastly, determine the media access control (MAC) address of your NodeMCU board. Every wired or wireless Ethernet device has a globally unique MAC address. Before your NodeMCU can join the campus network, permission must be granted by the campus IT department, and that requires the device's MAC address. (*NB:* Your NodeMCU may already have its MAC address labeled on the underside of the board. Even so, **you must still run the code and verify that the MAC address label is correct**.)

To determine/verify the MAC address, use the Arduino IDE to open the `MAC.ino` sketch (located in the MAC folder of your lab repo). For VS Code, cut/paste this into your main.cpp. Next open the Serial Monitor (Arduino: `Tools -> Serial Monitor`, VS Code: `PlatformIO:Serial Monitor` on bottom icons). You may need to set the monitor baud rate to 115200. Compile and run the program with the serial monitor and **record the MAC address of your ESP8266 in the space below**. (*NB:* MAC addresses consist of six 2-digit hexadecimal numbers separated by colons, e.g. `4e:fa:7d:23:7b:8c`). Also, indicate if your NodeMCU was already labeled with a MAC address and if it was correct.

MAC Address = ____ : ____ : ____ : ____ : ____ : ____
Previously labeled (place "X")? Yes____ No____
If previously labeled, was it correct (place "X")? Yes____ No____

Lab Report

This is a **one-week lab**. Because of its introductory nature, **no formal lab report is required**. Instead, make sure your remote GitHub Classroom Lab01 repository is checked in.

Option #1: your .ino files
Option #2: your .cpp files

(I will try to use the correct version to test your code if needed)

Also, you need a PDF version of this filled-out sheet submitted on canvas.

This lab must be completed before the start of next week's lab. I will only look at and grade submissions posted before the deadline!