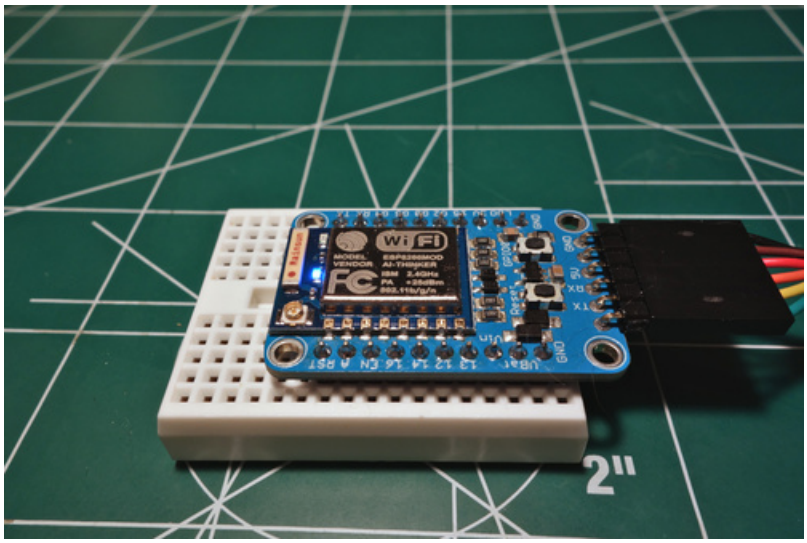
adafruit learning system

Building and Running MicroPython on the ESP8266

Created by Tony DiCola

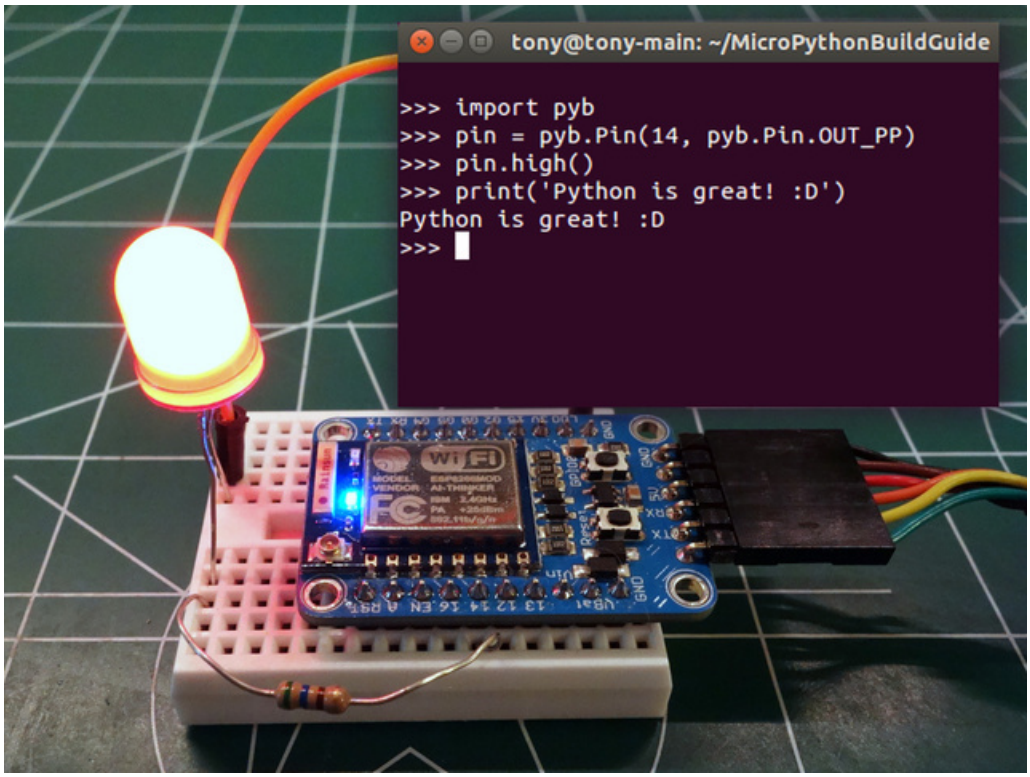


Last updated on 2017-10-18 07:54:00 PM UTC

Guide Contents

Overview

Since this guide was published you can now load MicroPython on the ESP8266 using pre-built firmware images that greatly simplify the process. See the updated loading MicroPython guide here:



MicroPython is an awesome little Python interpretor that can run on embedded platforms. Using the familiar Python programming language you can talk to hardware and control it, much like controlling hardware with an Arduino or other embedded board. The MicroPython board makes it easy to get started using MicroPython, but did you know you can use MicroPython on other platforms like the nifty ESP8266 WiFi module? Thanks to recent contributions to MicroPython you can turn an ESP8266 into a MicroPython device, and this guide will show you how to get started by compiling and installing MicroPython firmware for the ESP8266!

Before you get started one thing to note is that MicroPython's support for the ESP8266 is still in **very early stages of development and not all MicroPython board functionality is available**. Currently you can access GPIO pins, connect to a WiFi network, and talk to the internet using a low-level socket-like interface with MicroPython on the ESP8266. Access to I2C, SPI, or other parts of hardware are not yet supported--**consider MicroPython on the ESP8266 just for advanced users right now!**

Update August 2016: Since this guide was published MicroPython on the ESP8266 has evolved significantly! Support for the ESP8266 is now quite good and worth checking out even for casual users. See these later guides that explain what is MicroPython and how to more easily load firmware on the board:

- **MicroPython Basics: What is MicroPython?**
- **MicroPython Basics: How to load MicroPython on a Board**

However if you're interested in hacking on and even contributing to MicroPython for the ESP8266 this guide will show

you how to setup a toolchain for building the MicroPython ESP8266 firmware. You can even use the toolchain built from this guide to create other ESP8266 projects!

Also many thanks to the contributors of the MicroPython, ESP open SDK, and esptool projects. These excellent open source projects make possible this and other awesome MicroPython & ESP8266 projects--thanks!

Build Firmware

To use MicroPython on the ESP8266 you'll need a firmware file to load on the ESP8266. The best way to get the firmware is to build it yourself from its source code. This way you can get the latest version of MicroPython and even make changes to add features or extend MicroPython on the ESP8266. This page will show you how to setup a toolchain in a virtual machine that can compile MicroPython firmware.

However if you just want to try out MicroPython on the ESP8266 check out the easy install option below to download a pre-made firmware image.

Easy Install

Since his guide was originally published the MicroPython team has now started to provide pre-built firmware images of MicroPython for the ESP8266. Check out this brand new guide that explains how to download and flash the official MicroPython firmware images to an ESP8266. If you're a beginner or new to MicroPython you probably want to start there first!

Toolchain Setup

To build MicroPython firmware for the ESP8266 you'll need to first build the ESP open SDK toolchain that can compile code for the ESP8266's processor. You could manually compile and install this SDK on your computer, however it's much easier to use a small virtual machine running Linux to compile and use the toolchain. This way you can use the ESP open SDK from any computer regardless of it running Windows, Mac OSX, or even Linux, and you can keep the SDK's tools in an environment that's isolated and won't conflict with any other development tools on your machine.

Dependencies

To setup the virtual machine you'll need to install a few pieces of software:

- VirtualBox - This is open source virtualization software that is a free download.
- Vagrant - This is an open source wrapper around VirtualBox which makes it easy to create and run a virtual machine from the command line. Vagrant is also free to download.
- Git - Source control system used to download the configuration for this project. Git is also free and open source.

Once you have VirtualBox, Vagrant, and Git installed on your system then you can move on to provisioning the virtual machine.

Provision Virtual Machine

To create and provision the virtual machine you'll need to download a Vagrant configuration file. This file defines what operating system to install (Ubuntu 14.04) and some commands to prepare the operating system for building the ESP SDK.

Start by opening a command line terminal (in Windows make sure to open a 'Git Bash' command window as you'll need to use Git commands) and run the following command to clone the repository for this project and navigate inside it:

```
git clone https://github.com/adafruit/esp8266-micropython-vagrant.git
cd esp8266-micropython-vagrant
```

Now 'turn on' the virtual machine by running this command:

```
vagrant up
```

The first time the '**vagrant up**' command runs it will take a bit of time as it downloads the operating system image, but later '**vagrant up**' commands will be faster as the OS image is cached internally.

Once the virtual machine is running you should see the command finish with text like:

```
...
==> default: Installing esp-open-sdk and micropython source...
==> default: Cloning into 'esp-open-sdk'...
==> default: Submodule 'crosstool-NG' (https://github.com/jcmvbkbc/crosstool-NG) registered for path 'cro
==> default: Submodule 'esptool' (https://github.com/themadinventor/esptool) registered for path 'esptool
==> default: Submodule 'lx106-hal' (https://github.com/tommie/lx106-hal) registered for path 'lx106-hal'
==> default: Cloning into 'crosstool-NG'...
==> default: Submodule path 'crosstool-NG': checked out '7c6bc14de33e7a331ad4932bb505385ba0963eea'
==> default: Cloning into 'esptool'...
==> default: Submodule path 'esptool': checked out '12debb7bfe5e978060ff1919bf87d00ed52c4a72'
==> default: Cloning into 'lx106-hal'...
==> default: Submodule path 'lx106-hal': checked out 'ecdc98953f2fc5058a79168528387cd14b287636'
==> default: Cloning into 'micropython'...
==> default: Finished provisioning, now run 'vagrant ssh' to enter the virtual machine.
```

If you see an error go back and make sure you've installed both VirtualBox and Vagrant. Also make sure you're executing the command from inside the cloned repository's directory, there should be a file named **Vagrantfile** inside the directory you're running these commands from.

Once the virtual machine is running you can enter a Linux command terminal on it by executing the command:

```
vagrant ssh
```

After a moment you should be at an Ubuntu Linux command prompt that looks something like:

```
vagrant@vagrant-ubuntu-trusty-64:~$
```

Woo hoo! Your virtual machine is now running and ready to compile the ESP open SDK and MicroPython firmware.

Compile ESP Open SDK

Once inside the virtual machine you'll first need to compile the ESP open SDK. The SDK source code has already been downloaded in the **esp-open-sdk** subdirectory during the virtual machine provisioning. You just need to change to the directory and execute a command to make the project. Run the following commands:

```
cd ~/esp-open-sdk
make STANDALONE=y
```

Note that the compilation will take a bit of time. On my machine compilation took about 30 minutes, but on an older or slower machine it might take an hour or more. Luckily you only need to compile the ESP open SDK once and then can quickly build MicroPython using the compiled SDK tools.

If during the compilation you see it fail with an error like:

```
[ERROR]    collect2: error: ld terminated with signal 9 [Killed]
[ERROR]    make[4]: *** [cc1] Error 1
[ERROR]    make[3]: *** [all-gcc] Error 2
```

This means the virtual machine ran out of memory as it compiled the SDK tools. You can resolve this by increasing the amount of memory available to the SDK. Shut down the VM and edit the **Vagrantfile** to change this line:

```
# Bump the memory allocated to the VM up to 1 gigabyte as the compilation of
# the esp-open-sdk tools requires more memory to complete.
v.memory = 1024
```

Try increasing it to 1.5 or even 2 gigabytes and '**vagrant up**' the machine again to try the compilation again. I found only about 1 gigabyte of memory was needed to compile the toolchain (and this is the default value in the Vagrantfile configuration).

If you see the compilation fail with a different error then there might be a problem with the ESP open SDK. Try checking the github issues for it to see if there is a known issue with the error you received.

Once the compilation has finished you should see it end with text like:

```
...
Xtensa toolchain is built, to use it:

export PATH=/home/vagrant/esp-open-sdk/xtensa-lx106-elf/bin:$PATH

Espressif ESP8266 SDK is installed, its libraries and headers are merged with the toolchain
```

Now the ESP open SDK is compiled and you're almost ready to build MicroPython (or any other ESP8266 code you'd ever like to compile). First though you need to add the ESP open SDK tools to the virtual machine's path so MicroPython can find them. Run this command to update the .profile file that runs whenever you log into the virtual machine:

```
echo "PATH=$(pwd)/xtensa-lx106-elf/bin:\$PATH" >> ~/.profile
```

To make this updated path available log out and back in to the virtual machine by running:

```
exit
vagrant ssh
```

Once logged in again you should see your path environment variable has the ESP open SDK tools in it. You can check this by running the command:

```
echo $PATH
```

You should see a path value that looks something like this (notice the **esp-open-sdk** tools in the path):

```
/home/vagrant/esp-open-sdk/xtensa-lx106-elf/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
```

Excellent! Now you're all set to compile MicroPython in the next section. Remember you only need to perform the ESP open SDK compilation steps above once in the virtual machine.

Compile MicroPython Firmware

Next you can build the MicroPython firmware for the ESP8266. Make sure you've followed all the steps above and have a virtual machine running and the ESP open SDK compiled.

The MicroPython source code has already been downloaded to the **micropython** folder during the virtual machine provisioning. However you need to run a small git command to pull in external dependencies before you can build it. To install this dependency and then start the compilation execute the following commands inside the virtual machine:

```
cd ~/micropython
git submodule update --init
make -C mpy-cross
cd ~/micropython/esp8266
make axtls
make
```

The MicroPython library compilation should be quick and only take a few minutes at most. Once it completes you should see text like the following:

```
LINK build/firmware.elf
   text    data     bss     dec     hex filename
 304096    1332   53776  359204   57b24 build/firmware.elf
Create build/firmware-combined.bin
('flash   ', 52800)
('padding ', 12736)
('irom0text', 252672)
('total   ', 318208)
```

If you see an error, try searching the MicroPython issues to see if the error is a known issue.

Once finished the output will be the file **./build/firmware-combined.bin**. In the next section you'll walk through how to load the firmware on an ESP8266 board, however first you'll need to copy the firmware bin file out of the virtual machine. Luckily Vagrant has a special directory inside the virtual machine which can be used to copy files between the virtual machine and the host computer running Vagrant. Execute the following command to copy out the firmware bin file to this shared folder:

```
cp ./build/firmware-combined.bin /vagrant/
```

Now exit the virtual machine by running the following command:

```
exit
```

Then check that the **firmware-combined.bin** file is on your computer in the same directory that you ran '**vagrant up**' to start the VM. If you don't see the file then log back in to the virtual machine (using '**vagrant ssh**') and make sure the

command to copy the **firmware-combined.bin** file to the **/vagrant** folder inside the VM succeeded.

Once you have successfully compiled the **firmware-combined.bin** file and retrieved it from inside the VM you're done using the virtual machine. Run the following command to 'turn off' the virtual machine:

```
vagrant halt
```

If you'd like to start the VM again in the future just navigate to the same directory as the Vagrantfile and run the '**vagrant up**' command again. After a few moments the VM will be running and you can run '**vagrant ssh**' to enter a terminal on the virtual machine. All of the files and SDK tools that were compiled previously should be available again in the VM.

Continue on to learn how to flash the compiled ESP8266 MicroPython firmware to the hardware.

Flash Firmware

> Note this page describes how to flash firmware onto an ESP8266. You can flash either CircuitPython or MicroPython firmwares, just change the firmware file you're using. Note if you're flashing CircuitPython you can only use the CircuitPython API, see its documentation: http://circuitpython.readthedocs.io/en/latest/?

To flash the MicroPython firmware on the ESP8266 follow the steps below to use the excellent esptool.py script on any platform, or on Windows the GUI nodemcu-flasher tool. Before continuing make sure you have a **firmware-combined.bin** file that was compiled or downloaded using the steps on the previous page!

esptool.py Flasher

The esptool.py flasher is a simple Python script that can flash firmware to an ESP8266 board. This tool is great for running on a platform like Mac OSX, Linux, or Windows. If you're on Windows you might consider a GUI-tool like nodemcu-flasher though--see the steps further below for more details.

Dependencies

Before you install the esptool script you first must have the following software installed:

- Python 2.7 - You'll need the latest 2.7 version of Python to use the script.
- PySerial Library - On Windows grab the pyserial-2.7.win32.exe installer and run it to install the library. For Mac OSX & Linux, install pip and then run '**sudo pip install pyserial**' (without quotes) to install the library.
- Git - Again you'll need git installed to download the code for the esptool script.

Installation

To install the esptool script you just need to clone the source for it. Inside a terminal navigate to a directory you'd like to keep the source code and run the following commands to download the source and change into its directory:

```
git clone https://github.com/themadinventor/esptool.git
cd esptool
```

Now you can run the script (using the python interpretor) with the -h parameter to see usage details printed. Run the following:

```
python esptool.py -h
```

You should see usage information printed that starts with something like the following:

```
usage: esptool [-h] [--port PORT] [--baud BAUD]
           {load_ram,dump_mem,read_mem,write_mem,write_flash,run,image_info,make_image,elf2image,read_mac
               ...
...
```

That's all you need to do to install and run the esptool script. Next you'll learn how to flash the firmware using this tool.

Flash Firmware

To flash the firmware of an ESP8266 with the compiled MicroPython firmware first make sure you have the **firmware-combined.bin** file copied in to the directory you're working from (the esptool folder).

Also make sure an ESP8266 board is connected to your computer using a serial to USB cable. Find the serial port name for this device using either Device Manager on Windows or running a command like '**ls -l /dev/tty***' (without quotes) on Linux or Mac OSX.

When you're ready to flash the firmware, first put the ESP8266 into its programming mode. With the Huzzah ESP breakout do this by holding the GPIO0 button and pressing the reset button, then releasing reset and finally releasing GPIO0. With a bare ESP8266 breakout you'll want to follow instructions like these to manually pull the GPIO0 pin low and reset the board.

Now run the following command to use esptool.py to upload the firmware:

```
python esptool.py -p SERIAL_PORT_NAME --baud 460800 write_flash --flash_size=detect 0 firmware-combined.b
```

Where **SERIAL_PORT_NAME** is the name of the serial port the ESP8266 is connected to (like **COM4**, **/dev/ttyUSB0**, etc.) and firmware-combined.bin is the path to the firmware-combined.bin file that was compiled earlier (if you've copied it to the same directory as esptool.py then you're all set with the command above, otherwise put in the full path to the file).

Note if you receive an error that detect is not a valid flash_size parameter you might be using an older version of esptool.py. Make sure to download the latest code as shown above and try again.

After esptool.py flashes the chip you should see text like the following printed:

```
Connecting...
Erasing flash...
Writing at 0x0004d800... (100 %)

Leaving...
```

Congratulations, you've loaded the MicroPython firmware on the ESP8266! In the future if you'd like to load the firmware again you can repeat the process above to put the ESP8266 into programming mode and run esptool.py to upload code.

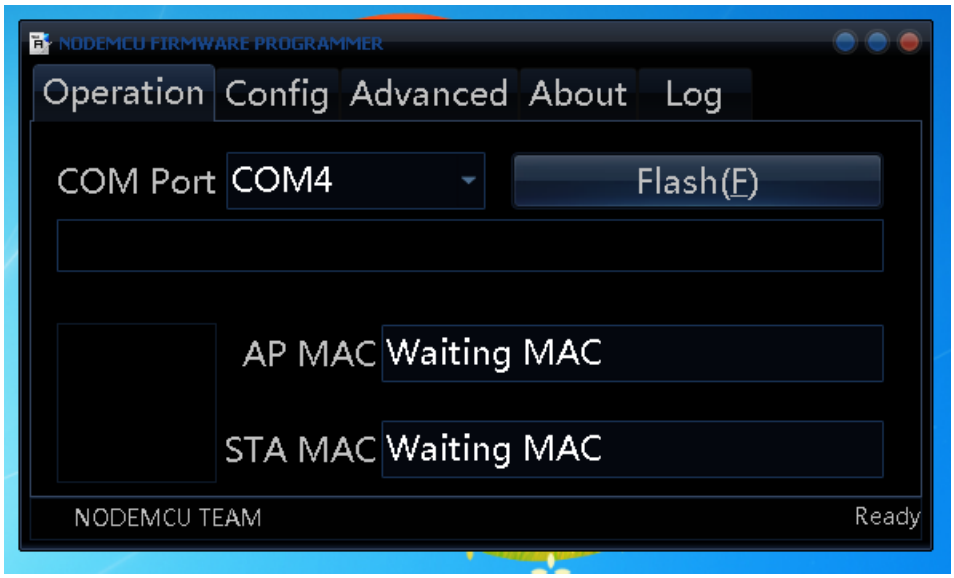
Continue on to learn how to connect to and use MicroPython on the ESP8266.

nodemcu-flasher Tool (Windows Only)

The nodemcu-flasher tool is a nice little GUI tool to flash firmware to the ESP8266. Follow the steps below to load the firmware-combined.bin file on to your ESP8266 board. Before you get started make sure the ESP8266 is connected to your computer using a serial to USB cable!

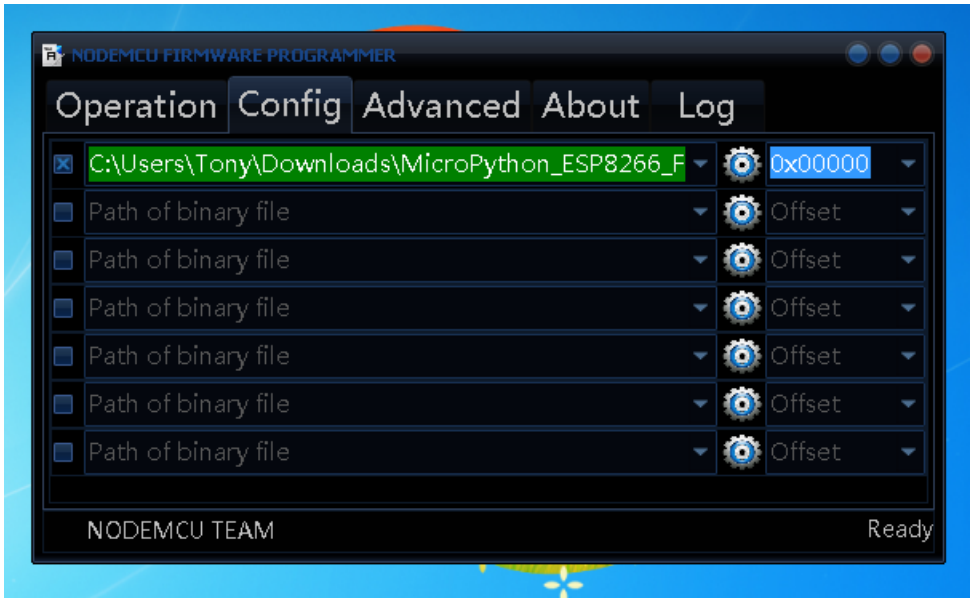
Installation

To install the tool download either the 32-bit binary or 64-bit binary from its Github home. Then run the program and you should see it detect the ESP8266 like follows:



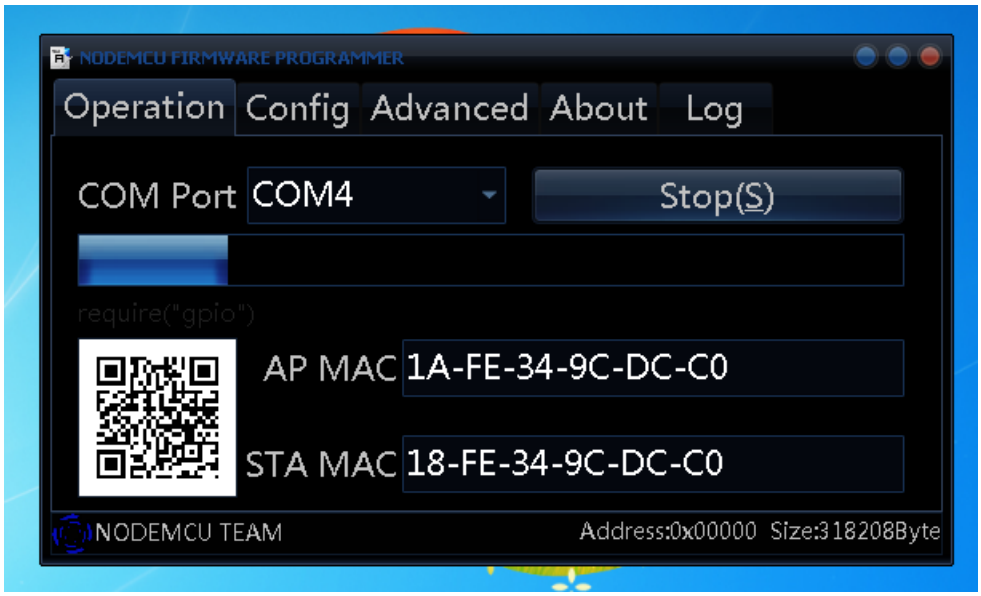
Flash Firmware

To flash the firmware click the **Config** tab at the top and change the first line from nodemcu firmware to our custom MicroPython **firmware-combined.bin** file. Click the gear icon next to the **0x00000** offset and navigate to the **firmware-combined.bin** file location. You should see a configuration like follows:

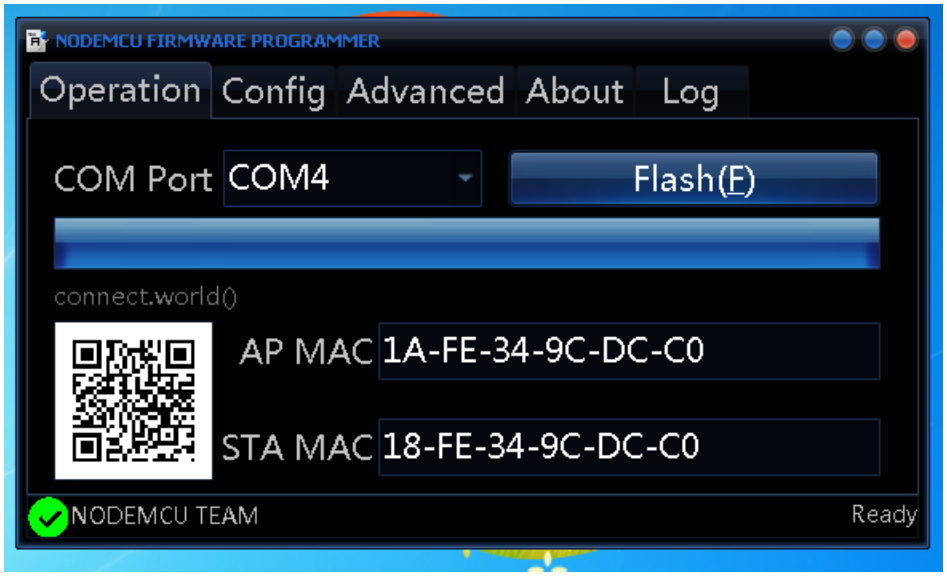


It's very important that your configuration looks like the above with the firmware loaded to offset 0x00000. If you have other firmware files or a different offset then the firmware won't work!

Now go back to the **Operation** tab at the top and verify the right COM port is selected for your ESP8266 board. When you're ready to flash the firmware, first put the ESP8266 into its programming mode. With the Huzzah ESP breakout do this by holding the GPIO0 button and pressing the reset button, then releasing reset and finally releasing GPIO0. With a bare ESP8266 breakout you'll want to follow instructions like these to manually pull the GPIO0 pin low and reset the board. Then click **Flash** to flash the firmware! You should see the progress bar move as the firmware is flashed:



Once the firmware is flashed you should see it complete like the following:



Congratulations, you've loaded the MicroPython firmware on the ESP8266! In the future if you'd like to load the firmware again you can repeat the process above to put the ESP8266 into programming mode and run nodemcu-flasher to upload code.

Continue on to learn how to connect to and use MicroPython on the ESP8266.

MicroPython Usage

NOTE: This page describes using MicroPython firmware, not CircuitPython firmware. If you're using CircuitPython be sure to check out the CIrcuitPython guides: https://learn.adafruit.com/category/circuitpython and API docs: http://circuitpython.readthedocs.io/en/latest/? The MicroPython APIs described below won't work on CircuitPython firmware!

NOTE: The details below are a little out of date as the project has evolved significantly since the guide was written. Check out the latest documentation on MicroPython ESP8266 here for the most up to date usage details: http://docs.micropython.org/en/latest/esp8266/index.html

To use MicroPython on the ESP8266 you'll need to connect to its serial port in a serial terminal at 115200 baud. You can use any serial terminal, like PuTTY on Windows or screen on Linux & Mac OSX.

Once connected you can start entering MicroPython code in a read-eval-print loop (REPL). For example here's the classic Hello World and counting to 10:

```
>>> print("Hello world!")
Hello world!
>>> for i in range(1, 11):
...     print("Number {0}".format(i))
...
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
>>>
```

Remember MicroPython isn't exactly the same as the Python you would use on the desktop. In particular most of the python standard library isn't available. Check out MicroPython's documentation for more information on what is supported by the language.

Also note the ESP8266 support for MicroPython is very beta and does not support all of MicroPython's libraries! As of the time of this writing there's only enough support to access GPIO pins and connect to WiFi using a very low level socket-like interface.

Below are some tips on what you can do with MicroPython on the ESP8266 right now.

GPIO Access

Access to the GPIO pins on the ESP8266 is provided through an interface similar to the pyb.Pin class. Here's a small example of blinking an LED connected to GPIO #14 ten times (make sure to connect a ~300+ ohm resistor in series with the LED):

```
>>> import pyb
>>> pin = pyb.Pin(14, pyb.Pin.OUT)  # Set pin 14 as an output.
>>> for i in range(10):
...     pin.value(0)     # Set pin low (or use pin.low())
...     pyb.delay(1000)  # Delay for 1000ms (1 second)
...     pin.value(1)     # Set pin high (or use pin.high())
...     pyb.delay(1000)
```

One thing to be very careful about is putting your code in an infinite loop (like by replacing the for loop with a while True loop). **If you use an infinite loop then you won't be able to use the REPL to enter code anymore!** This might completely block you from using MicroPython and necessitate reflashing the firmware, so be careful using loops!

You can also use GPIO pins as inputs, for example here's reading pin 14 as an input:

```
>>> import pyb
>>> pin = pyb.Pin(14, pyb.Pin.IN)
>>> pin.value()  # Read pin value, will show 0 when low (connected to ground).
0
>>> pin.value()  # Read pin value again, will show 1 when high (connected to 3.3V).
1
```

WiFi Access

Thanks to some excellent contributions the MicroPython ESP8266 firmware has basic support for using the ESP8266's WiFi radio. **Be aware the support is basic and only offers a low-level socket-like interface so you'll need to implement protocols yourself.** Here's a basic example to connect to a WiFi network and download a small test web page:

```
>>> import esp
>>>
>>> # Connect to a WiFi network.
>>> esp.connect('YOUR WIFI SSID NAME', 'YOUR WIFI SSID PASSWORD')
>>>
>>> # Define function to print data received from socket.
>>> def socket_printer(socket, data):
>>>     print(data)
>>>
>>> # Create a socket and setup the print function.
>>> soc = esp.socket()
>>> soc.onrecv(socket_printer)
>>>
>>> # Connect to adafruit.com at port 80.
>>> soc.connect(('207.58.139.247', 80))
>>>
>>> # Send a request for the wifi test page.
>>> soc.send('GET /testwifi/index.html HTTP/1.0\r\n\r\n')
37
>>> b'HTTP/1.1 200 OK\r\nDate: Tue, 12 May 2015 18:44:49 GMT\r\nServer: Apache\r\nAccess-Control-Allow-He
```

As you can see the esp module provides a simple socket-like interface to access internet services. You'll need to implement protocols like HTTP yourself to access web pages, etc.

If you're curious what other functions exist on the esp.socket class (or any other object in MicroPython) you can see them by using the dir function:

```
>>> dir(esp.socket)
['__del__', 'close', 'bind', 'listen', 'accept', 'connect', 'send', 'recv', 'sendto', 'recvfrom', 'oncon
```

Or check out the code for the module to see exactly how it works. MicroPython code is similar to writing a C extension to Python so it will help to be familiar with the process of extending Python. You'll also want to be aware of the key differences between MicroPython and normal desktop Python.

Running Code From A Script

You might have noticed all of the Python code so far has been entered by hand at the serial console. Can you save a Python script and have it run with MicroPython on the ESP8266? It turns out you can, but be aware support for saving code and running it is very limited and only allows for one file to be compiled in to the MicroPython firmware and run at boot. **MicroPython on the ESP8266 does not currently support running Python code off a SD card or other file system like other more mature MicroPython boards!**

To make a script run at boot you'll need to compile a new slightly modified version of MicroPython's firmware. Start up the VM for compiling MicroPython that you created in the previous steps (run the '**vagrant up**' command and then '**vagrant ssh**' command in the directory with the VM's Vagrantfile). Once connected navigate to the MicroPython ESP8266 scripts directory and open the **main.py** script in a text editor like nano:

```
cd ~/micropython/esp8266/scripts/
nano main.py
```

You should see a comment that says this script is run at boot:

```
# This script is run on boot
```

Now you can enter any Python script you'd like to have run as soon as the board boots. You can use the network, access GPIO, etc. For example here's code to blink an LED connected to pin 14 forever:

```
# This script is run on boot
import pyb
pin = pyb.Pin(14, pyb.Pin.OUT_PP)
while True:
    pin.value(1)
    pyb.delay(1000)
    pin.value(0)
    pyb.delay(1000)
```

Note if you use an infinite loop in the main script then the serial console REPL will not be accessible! MicroPython doesn't have threading or multiprocessing support so it can only do one thing at a time. If the main code never returns then the serial console REPL will never run! You'll need to reflash the ESP8266 with firmware that has an empty main.py to bring back access to the serial console.

Save the modified main.py and exit nano by typing Ctrl-O and Ctrl-X. Then rebuild the MicroPython ESP8266 firmware by executing:

```
cd ~/micropython/esp8266
make
```

Again the output file is the **./build/firmware-combined.bin** file. You can copy this to the **/vagrant** folder and back to your main machine, then flash to an ESP8266 board just like in the previous steps.

That's all there is to running a script on boot with MicroPython on the ESP8266!