Adding a WiFi Co-Processor to CircuitPython

Created by lady ada

https://learn.adafruit.com/adding-a-wifi-co-processor-to-circuitpython-esp8266-esp32

Last updated on 2023-08-29 03:57:22 PM EDT
Table of Contents

Overview 3
- Yallah! Let's go!
- Use a chip with integrated WiFi such as the ESP8266 or ESP32
- Use a separate chip with WiFi and use it just for wireless data
- Which way for CircuitPython?

ESP8266 or ESP32? 5
- What should I use?
- Get your Hardware

Firmware Files 7
- ESP32 Only SPI Firmware
- AT Command Firmware

Program with esptool 9
- Program over USB-to-Serial
- Burning the AT Command Firmware with esptool
- Programming ESP32 SPI Firmware with esptool
- Checking AT Command or SPI Debug Output

Program ESP8266 via CircuitPython 11
- Program Directly - ESP8266 Module

Program Particle Argon 14
- Program Directly - Built-in ESP32 Module on Particle ARGON

AT Setup & Test 17
- ESP Wiring
- Test wiring & WiFi with AP Scan

AT: Webclient Demo 21
Overview

This guide is obsolete. See our guides for AirLift for current WiFi co-processor hardware and software: https://learn.adafruit.com/search?q=airlift

We love Blinka & CircuitPython, and want to share it with the world! But how can we get CircuitPython talking to everyone?

This guide will show you how to connect your CircuitPython board to the Internet, by using an ESP8266 or ESP32 as the 'Wireless modem' - we'll even show you how to upload the required AT command firmware to the chip, all from your CircuitPython board.

Yallah! Let's go!

The Internet offers wonders beyond belief, but first we have to connect to it. There's two ways we can do that.

Use a chip with integrated WiFi such as the ESP8266 or ESP32

The all-in-one chips are powerful and low cost, but have some drawbacks. The ESP8266 has very few GPIO pins, and they are limited in functionality - for example only one ADC and it's not easy to use. A RTOS also takes up a lot of processor time, and makes real-time control difficult for some things like NeoPixels. The ESP32 solves a lot of these issues but doesn't have the native USB we require for a good CircuitPython experience.

Adafruit HUZZAH32 – ESP32 Feather Board
Aww yeah, it's the Feather you have been waiting for! The HUZZAH32 is our ESP32-based Feather, made with the official WROOM32 module. We packed everything you love...
https://www.adafruit.com/product/3405
Use a separate chip with WiFi and use it just for wireless data

For example, the Feather M0 WINC1500 has a SAMD21 chipset to do pin twiddling, I2C sensing, and display driving, while the WINC1500 provides just the WiFi part. When the M0 chip wants to send or receive data, it packetizes the commands over SPI and tells the WINC what to do. Works pretty well in Arduino!

Which way for CircuitPython?

A mix of the two! We don't support CircuitPython on the ESP8266/ESP32 because of the missing USB, and the WINC1500 is a complex beast that costs more than a bare ESP module. So we go right down the middle, by using an ESP8266 or ESP32 as a 'Wireless Co-processor'. This gets us:

- A main chip like a SAMD21 or SAMD51 with USB, peripherals, pins, and timers
- Wireless handling by a separate chip, so we don't have to run the WiFi stack natively - saves a lot of FLASH/RAM space
• Re-programmable co-processor (the WINC1500 is completely closed, we cannot change the firmware)

ESP8266 or ESP32?

This guide is obsolete. See our guides for AirLift for current WiFi co-processor hardware and software: https://learn.adafruit.com/search?q=airlift

Right now, there are two chips you can use and two communications methods.

Supported Transports:

• UART AT Commands - Just like those old modems you can sent AT control commands to firmware written by Espressif to connect to servers and transfer data. Does it work? Pretty much! Does it work well? Not really. But sometimes you don't have a choice and sometimes its OK if its flakey and inconsistent cause your code can keep re-trying till it works. You can run it on ESP32 or ESP8266/ESP8285. You can use as little as 3 pins (RX, TX and RTS)
• SPI Packetized Commands - This is a way superior method, that was developed by Arduino for use as an Arduino co-processor. Since the code is open source it can be fixed, modded, improved and enhanced. SPI packets are also way easier and faster to handle than UART AT commands. You can only use this with ESP32. It requires at least 5 pins (MOSI, MISO, SCK, CS, BUSY)

Supported Chipsets:

• ESP8266/ESP8285 - This is lower cost and can only use the UART AT command set code. Its less expensive, somewhat more common than the ESP32 but is (in our opinion) flakey and with slow connections and inconsistent SSL support. We don't recommend it.
• ESP32 - This is a little more expensive but can use both UART AT commands and SPI commands. SSL support is much better, and when used with SPI its all very snappy and a good experience.

What should I use?

If you can choose - use ESP32 with SPI Commands - it's the best experience by far.
If you are using a Particle Argon, you are forced to use ESP32 with AT Commands because SPI pins were not connected inside (if you're a customer, you could ask Particle if they'll fix this on their next revision).

If you are for some reason totally out of luck, you can use ESP8266 or ESP8285 with AT commands, but it can be frustrating.

We wrote this guide up during our experimentations but have decide to NO LONGER maintain the ESP + AT Commands firmware or code. It works-ish, and some friendly community members will be hacking on it, but we offer zero support for it!

Get your Hardware

The first step is to get an ESP8266, ESP32 or a Particle Argon with it built-in ESP32 module. We recommend an ESP32 breakout or Feather.

It is not recommended that the AT Firmware be used with other ESP32 boards since they can support the SPI interface for the Adafruit_CircuitPython_ESP32SPI library

Adafruit HUZZAH ESP8266 Breakout
Add Internet to your next project with an adorable, bite-sized WiFi microcontroller, at a price you like! The ESP8266 processor from Espressif is an 80 MHz microcontroller with a full...
https://www.adafruit.com/product/2471

Adafruit HUZZAH32 – ESP32 Feather Board
Aww yeah, it’s the Feather you have been waiting for! The HUZZAH32 is our ESP32-based Feather, made with the official WROOM32 module. We packed everything you love...
https://www.adafruit.com/product/3405
Using the Particle Argon requires that you replace the Particle.IO firmware with the Adafruit Bootlader and Circuitpython. This requires special tools and is for experienced users. See [Installing CircuitPython on a Particle Argon ()](https://www.adafruit.com/product/3993)

**Particle Argon Kit - nRF52840 with BLE and WiFi**
Discontinued - you can grab the Adafruit ESP32 Feather V2 - 8MB Flash + 2 MB PSRAM - STEMMA QT instead!...
https://www.adafruit.com/product/3993

---

### Firmware Files

This guide is obsolete. See our guides for AirLift for current WiFi co-processor hardware and software: [https://learn.adafruit.com/search?q=airlift](https://learn.adafruit.com/search?q=airlift)

The ESP almost certainly does not have the right firmware on it. You will have to program it. There are two ways of doing that:

1. Program it using a USB-to-serial adapter or built-in USB to bootloader in the AT firmware
2. Program it directly over UART using our CircuitPython programming tool

If you can do #1, you might as well, its often easier

### ESP32 Only SPI Firmware

This is built from our fork of the Arduino nina-fw repo (). It can be built by following the instructions in the repo (not for people who are uncomfortable with installing toolchains and messing with path variables)

**Version 1.3.0**

Adds 'enterprise' SSID connection support

**NINA_W102-1.3.0.zip**
This firmware has the following connection definitions:

- SPI CS on GPIO 5
- SPI SCK on GPIO 18
- SPI MOSI on GPIO 14
- SPI MISO on GPIO 23
- BUSY/READY on GPIO 33

The controller may also connect to GPIO 0 and RESET for controlling the ESP32

AT Command Firmware

All these firmwares will use the UART at a default leisurely 115200 baud

Please remember, the AT command firmware is not recommended unless you absolutely have to use it! SPI mode is waaaay better and more complete!

Here’s the firmware for classic ESP8266. Download and unzip, there will be a folder called esp8266/

For ESP8285 modules with a 26 MHz crystal, try this:

If you’re using a Particle Argon, use this version we built that uses the main UART for AT communication, and defaults to 115200 baud
Program with esptool

Program over USB-to-Serial

Connect to your ESP chip with a USB serial converter, perhaps it's even built-in depending on the board.

Burning the AT Command Firmware with esptool

This section assumes you know how to use 'esptool' to upload firmware to your ESP! If you're not sure, check [https://github.com/espressif/esptool](https://github.com/espressif/esptool) and look for tutorials.

(Download and unzip the ESP8266 AT Firmware zip file. There will be a folder called esp8266/ containing 3 files.

Put the board into bootload mode (if you have to) by pulling GPIO 0 down to ground while resetting. Use the `esptool` command line to upload the firmware, for example (your serial port name may vary).

```
esptool.py --chip esp8266 -p /dev/ttyUSB0 write_flash 0 esp8266/AT_firmware_1.6.2.0.bin 0x3fc000 esp8266/esp_init_data_default_v05.bin 0x3fe000 esp8266/blank.bin
```

or

```
esptool --chip esp8266 -p COM6 write_flash 0 esp8266/AT_firmware_1.6.2.0.bin 0x3fc000 esp8266/esp_init_data_default_v05.bin 0x3fe000 esp8266/blank.bin
```
Programming ESP32 SPI Firmware with esptool

The SPI Firmware is much simpler, we use the combine.py file to turn it into one ~1MB file. You can program this into your ESP32 with:

```
esptool.py --port /dev/ttyS6 --baud 115200 write_flash 0 NINA_W102.bin
```

or

```
esptool.py --port COM5 --baud 115200 write_flash 0 NINA_W102.bin
```

Change the name of the binary file to match, and the serial port of your USB-serial adapter or Huzzah32 UART connection.

Checking AT Command or SPI Debug Output

Once you upload the firmware, use the same serial connection you used for uploading to see that the firmware ‘took’ and some status/debug information:

You can connect with a terminal over the USB-Serial connection. Use 115200 baud.
Hit the reset button to see the debug output.

If you have the AT command firmware installed, when you type, characters will be echo'd. You can try typing `AT` to get an AT OK reply. Make sure your terminal software sends new-lines and carriage returns when you hit return or send data. They are required.

Program ESP8266 via CircuitPython

Program Directly - ESP8266 Module

You don't have to wire up a separate USB serial converter if you don't like or can't get access to the pins. Instead, you can use miniesptool() which will run on a CircuitPlayground Express board.

You will need at least 1MB of disk space on the Express board, so clear off some space for the firmware!

Likewise, your ESP8266 or ESP8285 module must have at least 1MB of Flash - old 512KB ESP-01's wont work!

The library to support the upload is adafruit_miniesptool and it is include in the latest Bundle() or you can download adafruit_miniesptool from github() and install it on your Express board in the root directory of you CIRCUITPY drive.

Also grab the firmware binaries esp8266/ folder from the ESP8266 Firmware Zip file in the Firmware files page, and copy it to the root directory of your CIRCUITPY drive.

Note: in the screenshot below, adafruit_miniesptool.py in in the root directory -- If you are using it from the bundle and placed in /lib then you will not see it.
We'll be using a Feather M4 to demonstrate, M0 boards may not have enough memory! Wire up the ESP8266 module as shown:

- VBat from ESP8266 to VBat on Feather M4 (if your board doesn't have a Battery power pin, skip this)
- Vin from ESP8266 to VUSB on Feather M4 - use whatever 5V (no higher!) power source you have
- GND to GND
- ESP8266 Reset to D5 (you can use any pin)
- ESP8266 GPIO #0 to D6 (you can use any pin)
- ESP8266 RX to UART TX
- ESP8266 TX to UART RX

Note the UART pins are 'cross wired' - RX to TX

Save the following esp uploading example as `miniesptool.esp8266program.py`
import time
import board
import busio
from digitalio import DigitalInOut
import adafruit_miniesptool

print("ESP8266 mini prog")
uart = busio.UART(board.TX, board.RX, baudrate=115200, timeout=1)
resetpin = DigitalInOut(board.D5)
gpio0pin = DigitalInOut(board.D6)
# On ESP8266 we will 'sync' to the baudrate in initialization
esptool = adafruit_miniesptool.miniesptool(uart, gpio0pin, resetpin, flashsize=1024 * 1024, baudrate=256000)

esptool.debug = False
esptool.sync()

print("Synced")
print(esptool.chip_name)
print("MAC ADDR: ", [hex(i) for i in esptool.mac_addr])
esptool.flash_file("esp8266/AT_firmware_1.6.2.0.bin", 0x0)
# 0x3FC000 esp init_data_default_v05.bin
esptool.flash_file("esp8266/esp_init_data_default_v05.bin", 0x3FC000)
# 0x3FE000 blank.bin
esptool.flash_file("esp8266/blank.bin", 0x3FE000)
esptool.reset()
time.sleep(0.5)

Change the GPIO0 and Reset pin assignment if necessary.

Open up the serial connection and connect to the REPL then type:

import miniesptool_esp8266program

to start the uploading process, it will take a few minutes.
When complete, you may delete the esp8266/ folder to free up space.

Program Particle Argon

Program Directly - Built-in ESP32 Module on Particle ARGON

Some boards like the Particle ARGON have an ESP32 wired directly in place, so you can only program it via CircuitPython.

Download the [ESP32 for particle_argon firmware](https://example.com) bundle above, unzip it, and place the esp32 folder in the root directory of your CircuitPython drive.

```
CIRCUITPY > esp32
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootloader</td>
<td>12/16/2018</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>customized_partitions</td>
<td>12/16/2018</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>at_customize.bin</td>
<td>12/16/2018</td>
<td>BIN File</td>
<td>881K</td>
</tr>
<tr>
<td>esptool.bin</td>
<td>12/16/2018</td>
<td>BIN File</td>
<td>8K</td>
</tr>
<tr>
<td>ota_data_init.bin</td>
<td>12/16/2018</td>
<td>BIN File</td>
<td>3K</td>
</tr>
<tr>
<td>partitions_at.bin</td>
<td>12/16/2018</td>
<td>BIN File</td>
<td>1K</td>
</tr>
<tr>
<td>phy_init_data.bin</td>
<td>12/16/2018</td>
<td>BIN File</td>
<td></td>
</tr>
</tbody>
</table>

Note that unlike the ESP8266 firmware, this is a bundle of files. The ESP8266 firmware is 'combined' but we wouldn't be able to fit the combined ESP32 firmware on a 2MB flash disk so we kept the files separate.

Save the following as `miniesptool_esp32argon.py` on your board. Open the serial terminal, and connect to the REPL.
import time  
import board  
import busio  

from digitalio import DigitalInOut, Direction  # pylint: disable=unused-import  
import adafruit_miniesptool  

print("ESP32 mini prog")  

# With a Metro or Feather M4  
# uart = busio.UART(TX, RX, baudrate=115200, timeout=1)  
# resetpin = DigitalInOut(board.D5)  
# gpio0pin = DigitalInOut(board.D6)  

# With a Particle Argon, we need to also turn off flow control  
uart = busio.UART(board.ESP_RX, board.ESP_TX, baudrate=115200, timeout=1)  
resetpin = DigitalInOut(board.ESP_WIFI_EN)  
gpio0pin = DigitalInOut(board.ESP_BOOT_MODE)  
esp_cts = DigitalInOut(board.ESP_CTS)  
esp_cts.direction = Direction.OUTPUT  
esp_cts.value = False  
esptool = adafruit_miniesptool.miniesptool(  
    uart, gpio0pin, resetpin, flashsize=4 * 1024 * 1024  
)  
esptool.debug = False  
esptool.sync()  
print("Synced")  
print("Found: ", esptool.chip_name)  
if esptool.chip_name != "ESP32":  
    raise RuntimeError("This example is for ESP32 only")  
esptool.baudrate = 912600  
print("MAC ADDR: ", [hex(i) for i in esptool.mac_addr])  

# 0x10000 ota_data_initial.bin  
esptool.flash_file(  
    "esp32/ota_data_initial.bin", 0x10000, "84d04c9d6cc8ef35bf825d51a5277699"  
)  

# 0x10000 bootloader/bootloader.bin  
esptool.flash_file(  
    "esp32/bootloader/bootloader.bin", 0x1000, "195dae16eda6ab703a45928182b9863"  
)  

# 0x20000 at_customize.bin  
esptool.flash_file(  
    "esp32/at_customize.bin", 0x20000, "9053055e077ba0c90cd70691b9d8c3d5"  
)  

# 0x24000 customized_partitions/server_cert.bin  
esptool.flash_file(  
    "esp32/customized_partitions/server_cert.bin",  
    0x24000,  
    "766fa1e87aabb9ab78ff4023f6f624d",  
)  

# 0x26000 customized_partitions/server_key.bin  
esptool.flash_file(  
    "esp32/customized_partitions/server_key.bin",  
    0x26000,  
    "05da790776c3d5160f26bf870592459",  
)  

# 0x28000 customized_partitions/server_ca.bin
esptool.flash_file(
    "esp32/customized_partitions/server_ca.bin",
    0x28000,
    "e0169f36f9cb09c6705343792d353c0a",
)

# 0x2a000 customized_partitions/client_cert.bin
esptool.flash_file(
    "esp32/customized_partitions/client_cert.bin",
    0x2A000,
    "428ed3bae5d58b721b8254cbeb8004ff",
)

# 0x2c000 customized_partitions/client_key.bin
esptool.flash_file(
    "esp32/customized_partitions/client_key.bin",
    0x2C000,
    "136f563811930a5d3bf04c946f430ced",
)

# 0x2e000 customized_partitions/client_ca.bin
esptool.flash_file(
    "esp32/customized_partitions/client_ca.bin",
    0x2E000,
    "25ab638695819daae67bcd8a4bfc5626",
)

# 0xf000 phy_init_data.bin
esptool.flash_file(
    "esp32/phy_init_data.bin", 0xF000, "bc9854aa3687ca73e25d213d20113b23"
)

# 0x100000 esp-at.bin
esptool.flash_file("esp32/esp-at.bin", 0x100000, "ae256e4ab546354cd8dfa241e1056996")

# 0x80000 partitions_at.bin
esptool.flash_file(
    "esp32/partitions_at.bin", 0x8000, "d3d1508993d61aedf17280140fc22a6b"
)
esptool.reset()
time.sleep(0.5)

From the REPL, type the following then press enter:

```python
import miniesptool_esp32argon
```

This'll start the uploading process, it'll take a few minutes.

ESP32 programming is similar but more advanced, and the files are bigger. We recommend adding md5 sums for each file, you can calculate these with command line tools on your computer. That way, each file is checksummed after writing to avoid mis-writes!
If you aren't using the exact same firmware we are, all the MD5's will need re-calculating! We don't calculate them in-chip since CircuitPython doesn't have built-in MD5 and we wanted to keep the code simple.

AT Setup & Test

The AT Command firmware is no longer actively supported - we moved to the ESP+SPI solution for a better experience and more security. However, this code does work and there may be some community improvements.

OK once that is done, you can finally use your freshly-programmed module to connect to the 'net. You can remove `miniesptool` and the firmware files from your Express' CIRCUITPY drive..

Download and install the [ESP AT control library](#) onto your CircuitPython board.
Since there are many steps, it’s best to run full examples by saving them to code.py on your board rather than interacting with the REPL. That said, you really want to have a serial connection open to see what’s going on.

**ESP Wiring**

We’ll be using a Feather M4 to demonstrate. Wire up the ESP8266 module as shown:

- VBat from ESP8266 to VBat on Feather M4 (if your board doesn't have a Battery power pin, skip this)
- Vin from ESP8266 to VUSB on Feather M4 - use whatever 5V (no higher!) power source you have
- GND to GND
- ESP8266 Reset to D5 (you can use any pin)
- ESP8266 RX to UART TX
- ESP8266 TX to UART RX
- ESP8266 GPIO13 to D6 (you can use any pin) (NOT shown on fritzing diagram -- yet..)

Note the UART pins are ‘cross wired’ - RX to TX. We don't use GPIO 0 but if you want you can keep it connected so that you can go back and upload firmware if you need.
Test wiring & WiFi with AP Scan

Create a secrets.py file containing settings for your local WiFi router. See the example template below. Change the values to match the name of your wireless network and its password. Copy this file to your CIRCUITPY drive.

There are several examples in the GitHub repo for the library, available here.

It is suggested you start with the esp_atcontrol_simpletest.py program. Save the example below to your board as code.py or save it by another name and execute it manually at the REPL.

```python
# secrets.py
secrets = {
    "ssid": "MY SSID",
    "password": "MY PASSWORD"
}

# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
import time
import board
import busio
from digitalio import DigitalInOut
from digitalio import Direction
from adafruit_espatcontrol import adafruit_espatcontrol

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# Debug Level
# Change the Debug Flag if you have issues with AT commands
dflag = False

if board.board_id == "challenger_rp2040_wifi":
    RX = board.ESP_RX
    TX = board.ESP_TX
    resetpin = DigitalInOut(board.WIFI_RESET)
    rtspin = False
    uart = busio.UART(TX, RX, baudrate=11520, receiver_buffer_size=2048)
    esp_boot = DigitalInOut(board.WIFI_MODE)
    esp_boot.direction = Direction.OUTPUT
    esp_boot.value = True
else:
    RX = board.ESP_TX
    TX = board.ESP_RX
    resetpin = DigitalInOut(board.ESP_WIFI_EN)
    rtspin = DigitalInOut(board.ESP_CTS)
    uart = busio.UART(TX, RX, timeout=0.1)
    esp_boot = DigitalInOut(board.ESP_BOOT_MODE)
    esp_boot.direction = Direction.OUTPUT
    esp_boot.value = True
print("ESP AT commands")
# For Boards that do not have an rtspin like challenger_rp2040_wifi set rtspin to False.
esp = adafruit_espatcontrol.ESP_ATcontrol( 
    uart, 115200, reset_pin=resetpin, rts_pin=rtspin, debug=debugflag 
)
print("Resetting ESP module")
esp.hard_reset()

first_pass = True
while True:
    try:
        if first_pass:
            # Some ESP do not return OK on AP Scan.
            # See https://github.com/adafruit/Adafruit_CircuitPython_ESP_ATcontrol/ 
            # Comment out the next 3 lines if you get a No OK response to AT+CWLAP 
            print("Scanning for AP's")
            for ap in esp.scan_APs():
                print(ap)
            print("Checking connection...")
            secrets dictionary must contain 'ssid' and 'password' at a minimum
            print("Connecting...")  
            esp.connect(secrets)
            print("IP address ", esp.local_ip)
            first_pass = False
        print("Pinging 8.8.8.8...", end="")
        print(esp.ping("8.8.8.8"))
        time.sleep(10)
    except (ValueError, RuntimeError, adafruit_espatcontrol.OKError) as e:
        print("Failed to get data, retrying
        ...
        Resetting ESP module")
        esp.hard_reset()
        continue

If you did not set the SSID/password in your secrets.py file you will get an error on connection

```python
File "code.py", line 36, in <module>
File "adafruit_espatcontrol.py", line 282, in join_AP
File "adafruit_espatcontrol.py", line 333, in at_response
RuntimeError: No OK response to AT+CWLAP="my ssid","the password"```
Update those lines and try again! On success the connection will be reported and the program will Ping IP address (8.8.8.8) every 30 seconds

AT: Webclient Demo

Now that you're connected, you will probably want to get data from the Internet! Here's a demo showing how you can get data from the web via a URL.

The ESP supports SSL, so if you want to use secure connections, start the URL with https://.

Update your SSID and password and try it out!

```python
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
import busio
from digitalio import DigitalInOut
from digitalio import Direction
import adafruit_requests as requests
import adafruit_espatcontrol.adafruit_espatcontrol_socket as socket
from adafruit_espatcontrol import adafruit_espatcontrol

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# Change the Debug Flag if you have issues with AT commands
debugflag = False

# How long between queries
TIME_BETWEEN_QUERY = 60  # in seconds

if board.board_id == "challenger_rp2040_wifi":
    RX = board.ESP_RX
    TX = board.ESP_TX
    resetpin = DigitalInOut(board.WIFI_RESET)
    rtspin = False
    uart = busio.UART(TX, RX, baudrate=11520, receiver_buffer_size=2048)
    esp_boot = DigitalInOut(board.WIFI_MODE)
    esp_boot.direction = Direction.OUTPUT
    esp_boot.value = True
    status_light = None
else:
    RX = board.ESP_TX
    TX = board.ESP_RX
    resetpin = DigitalInOut(board.ESP_WIFI_EN)
    rtspin = DigitalInOut(board.ESP_CTS)
    uart = busio.UART(TX, RX, timeout=0.1)
    esp_boot = DigitalInOut(board.ESP_BOOT_MODE)
    esp_boot.direction = Direction.OUTPUT
    esp_boot.value = True
    status_light = None
```
print("ESP AT commands")
esp = adafruit_espatcontrol.ESP_ATcontrol(
    uart, 115200, reset_pin=resetpin, rts_pin=rtspin, debug=debugflag)

URL = "http://wifitest.adafruit.com/testwifi/index.html"
print("ESP AT GET URL", URL)

print("Resetting ESP module")
esp.hard_reset()

requests.set_socket(socket, esp)

while True:
    try:
        print("Checking connection...")
        while not esp.is_connected:
            print("Connecting...")
            esp.connect(secrets)
        # great, lets get the data
        print("Retrieving URL..., end="")
        r = requests.get(URL)
        print("Status:", r.status_code)
        print("Content type:", r.headers["content-type"])
        print("Content size:", r.headers["content-length"])
        print("Encoding:", r.encoding)
        print("Text:", r.text)
        print("Sleeping for: {0} Seconds".format(TIME_BETWEEN_QUERY))
        time.sleep(TIME_BETWEEN_QUERY)
    except (ValueError, RuntimeError, adafruit_espatcontrol.OKError) as e:
        print("Failed to get data, retrying\n" e)
        continue

While we work on the library, you may notice it takes a lot of retries to get going. Did we mention parsing AT commands isn't the most fun way to communicate with the Internet? However, it will eventually work and usually once it starts working, it will keep up.

Notice that we add loops and try: except: blocks to catch failures and automatically retry our commands, this is essential for any network project due to the inconsistencies/reliability issues you will experience. That's true even for Ethernet!