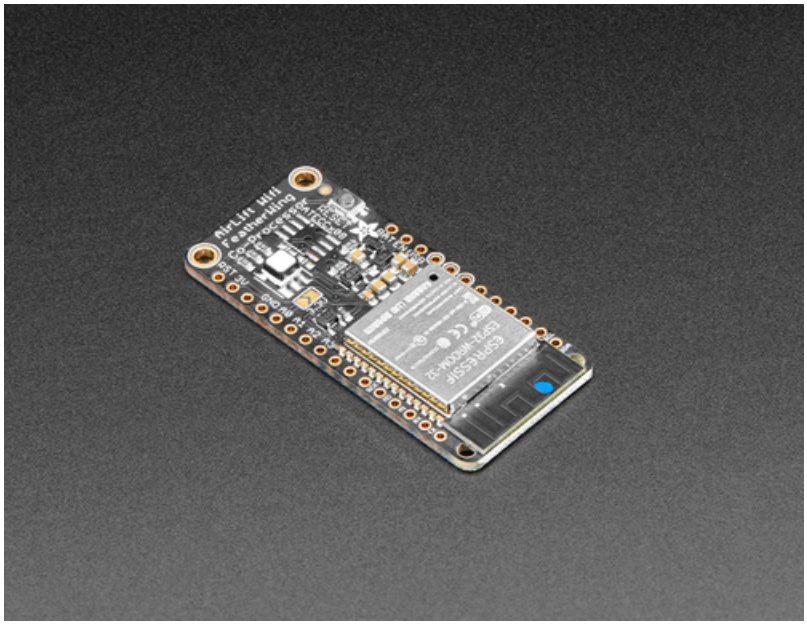
adafruit learning system

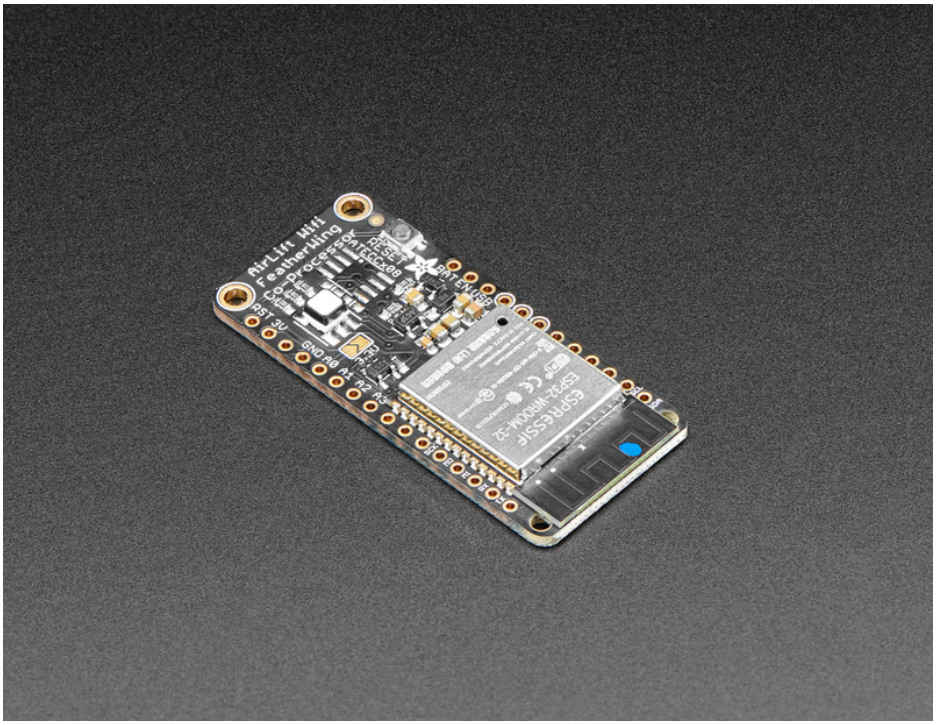
Adafruit AirLift FeatherWing - ESP32 WiFi Co-Processor

Created by Brent Rubell



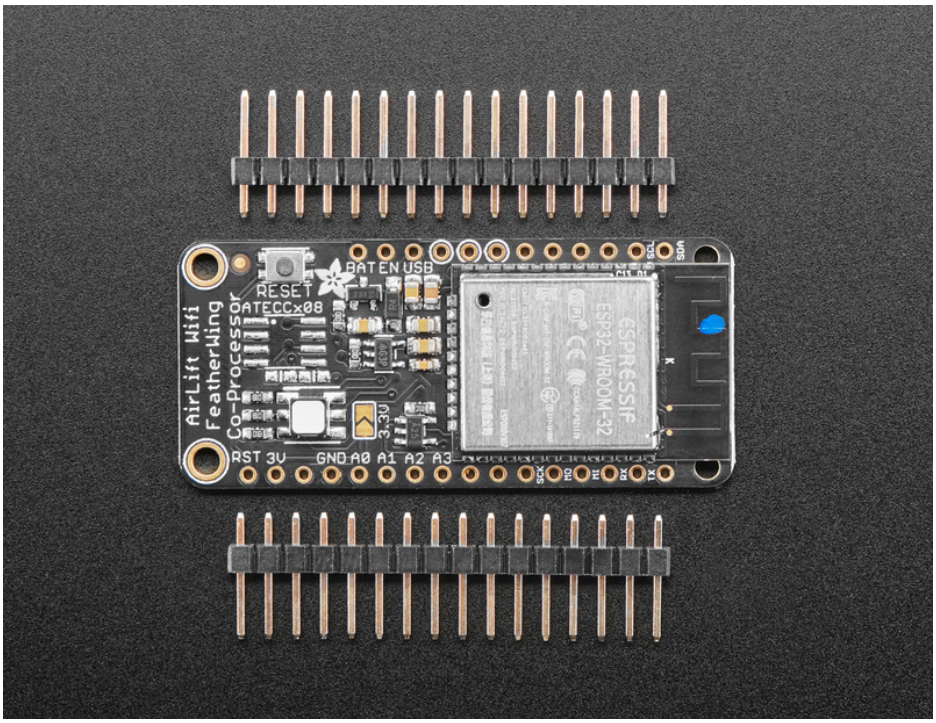
Last updated on 2019-07-29 04:56:09 PM UTC

Overview



Give your Feather project a *lift* with the Adafruit AirLift FeatherWing - a FeatherWing that lets you use the powerful ESP32 as a WiFi co-processor. You probably have your favorite Feather (like the Feather M4 (https://adafru.it/Cmy)) that comes with its own set of awesome peripherals and lots of libraries. But it doesn't have WiFi built in! So lets give that chip a best friend, the ESP32. This chip can handle all the heavy lifting of connecting to a WiFi network and transferring data from a site, even if its using the latest TLS/SSL encryption (it has root certificates pre-burned in).

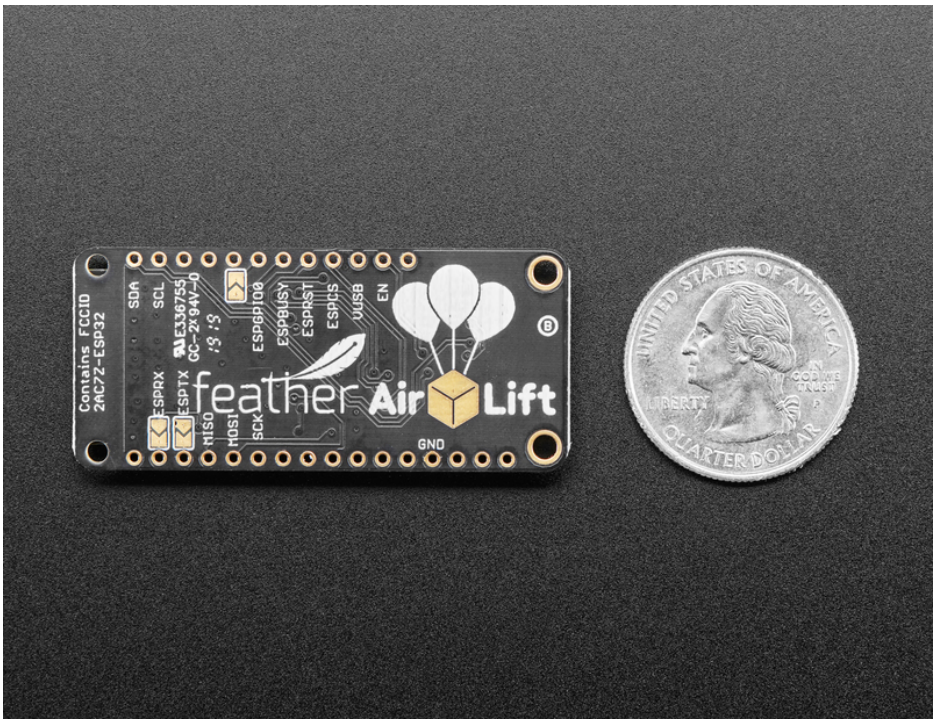
Having WiFi managed by a separate chip means your code is simpler, you don't have to cache socket data, or compile in & debug an SSL library. Send basic but powerful socket-based commands over 8MHz SPI for high speed data transfer. You can use 3V or 5V Arduino, any chip from the ATmega328 or up, although the '328 will not be able to do very complex tasks or buffer a lot of data. It also works great with CircuitPython, a SAMD51/Cortex M4 minimum required since we need a bunch of RAM. All you need is an SPI bus and 2 control pins plus a power supply that can provide up to 250mA during WiFi usage.



We placed an ESP32 module on a FeatherWing with a separate 3.3V regulator, and a tri-state chip for MOSI so you can share the SPI bus with other 'Wing. Comes fully assembled and tested, pre-programmed with ESP32 SPI WiFi co-processor firmware that you can use in CircuitPython to use this into WiFi co-processsor over SPI + 2 pins (https://adafru.it/Evl). We also toss in some header so you can solder it in and plug into a doubler, but you can also pick up a set of stacking headers to stack above/below your Feather.

We've tested this with all our Feathers and it should work just fine with them except the ESP8266 & ESP32 Feathers (cause they already have WiFi!). **For use in Arduino**, the '328 and '32u4 you can do basic connectivity and data transfer but they do not have a lot of RAM so we don't recommend them - use the M0, M4 or similar, for best results! **For CircuitPython use**, a Feather M4 or nRF52840 works best - the M0 series does not have enough RAM in CircuitPython.

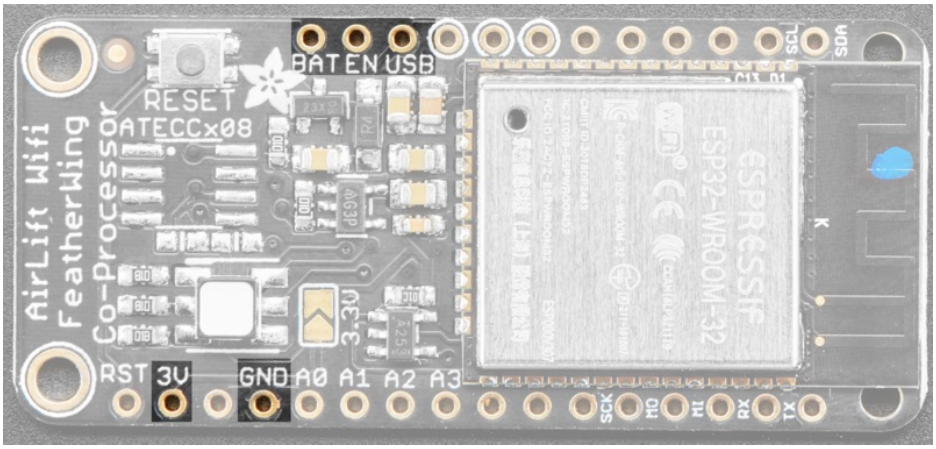
The firmware on board is a slight variant of the Arduino WiFiNINA core, which works great! (https://adafru.it/E7O)


Contains FCCID
2AC7Z-ESP32
SDA
SCL
E336755
GC-2794V-0
ESPBPIO0
ESPBUSY
ESPRST
ESPCS
VUSB
EN
ESPRX
ESPTX
MISO
MOSI
SCK
feather
Air
Lift
GND
UNITED STATES OF AMERICA
LIBERTY
QUARTER DOLLAR

Pinouts

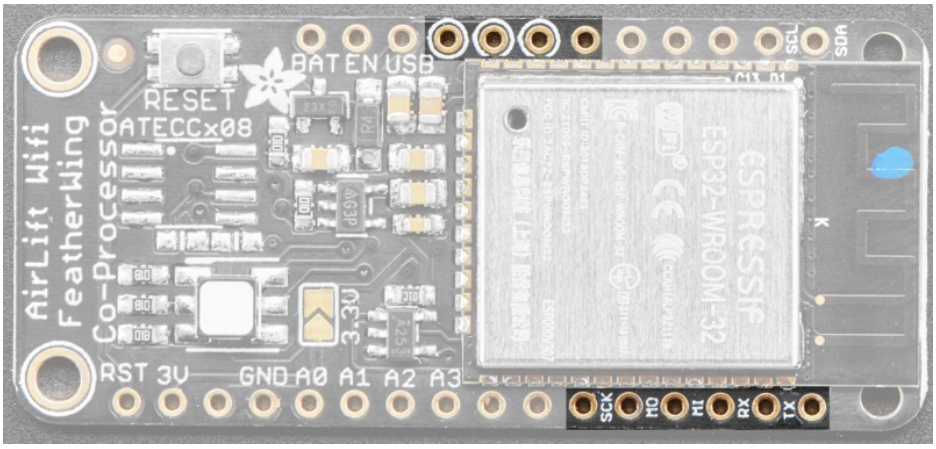
Power Pins

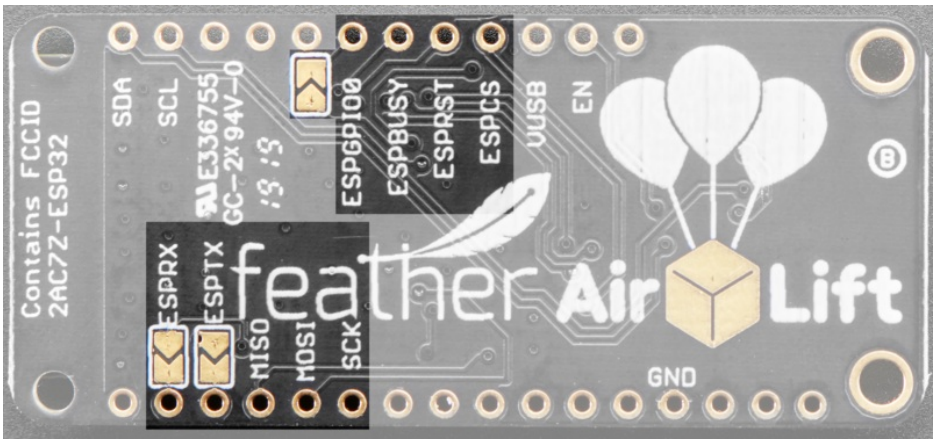
- **GND** - Common power/logic ground.
- **BAT** - Positive voltage from JST on Feather for an *optional* LiPo battery.
- **USB** - Positive voltage to/from the Micro USB jack if connected.
- **EN** - 3.3V regulator's enable pin. It's pulled up, so connect to ground to disable the 3.3V regulator
- **3V** - this is the output from the 3.3V regulator. The regulator can supply 500mA peak but half of that is drawn by the ESP32, and it's a fairly power-hungry chip. So if you need a ton of power for stuff like LEDs, motors, etc. Use the **USB** or **BAT** pins, and an additional regulator



SPI and Control Pins

To keep transfers speedy, we use SPI *not UART Serial.* Serial is too slow and hard to synchronize. This uses more pins but the experience is much better!





Classic SPI Pins:

- **SCK** - SPI Clock from your microcontroller, level shifted so can be 3-5V logic
- **MISO** - SPI Data *from* the AirLift *to* the microcontroller, this is 3.3V logic out, can be read by 3-5V logic. This is tri-stated when not selected, so you can share the SPI bus with other devices.
- **MOSI**- SPI Data *to* the AirLift *from* the microcontroller, level shifted so can be 3-5V logic
- **ESPCS** - SPI Chip Select from the microcontroller to start sending commands to the AirLift, level shifted so can be 3-5V logic

Required Control Pins:

- **ESPBUSY** - this pin is an input from the AirLift, it will let us know when its ready for more commands to be sent. This is 3.3V logic out, can be read by 3-5V logic. This pin *must* be connected.
- **ESPRST**- this pin is an output to the AirLift. Set low to put the AirLift into reset. You should use this pin, even though you might be able to run for a short while without it, it's essential to 'kick' the chip if it ever gets into a locked up state. Level shifted so can be 3-5V logic

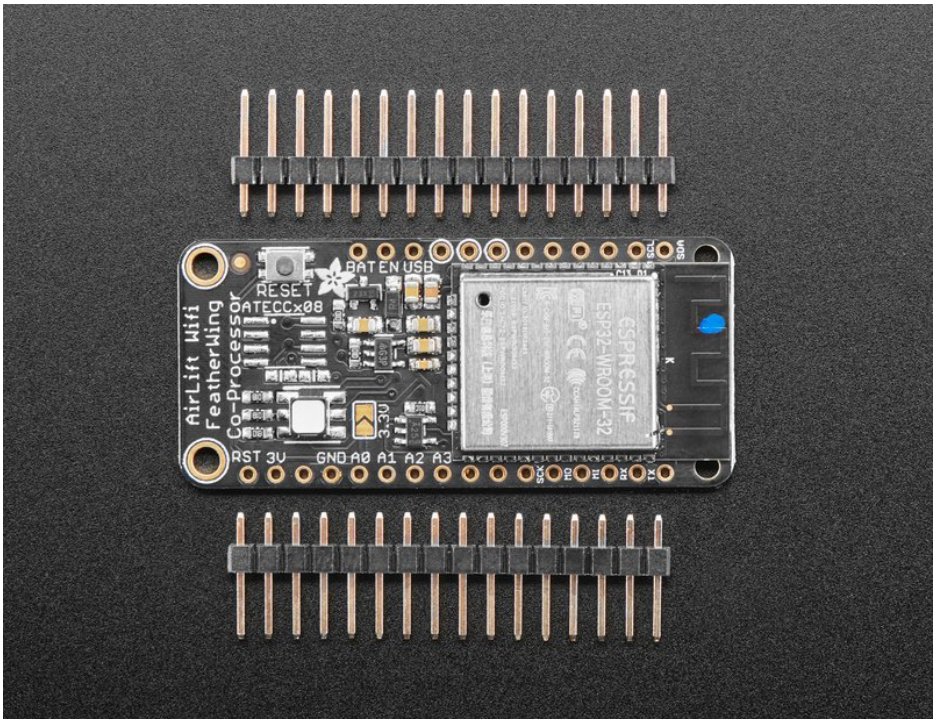
Optional Control Pins:

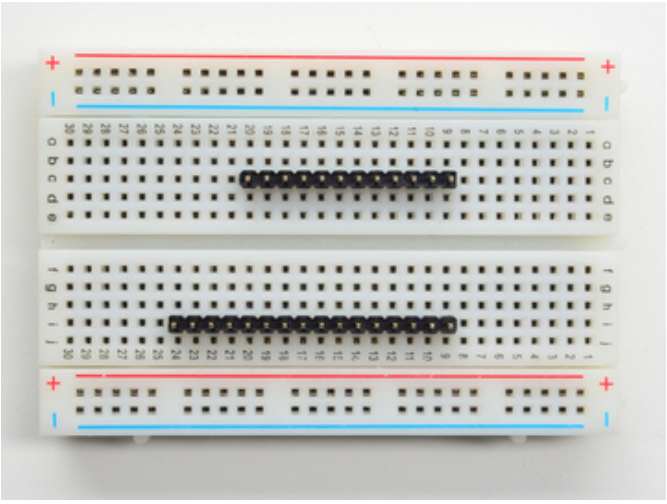
- **ESPGPIO0** - this is the ESP32 **GPIO0** pin, which is used to put it into bootloading mode. It is also used if you like when the ESP32 is acting as a server, to let you know data is ready for reading. It's not required, you'll need to solder the pad on the bottom of the FeatherWing to connect it.
- **ESPRX** & **ESPTX** - Serial data in and Serial data out, used for bootloading new firmware only. Leave disconnected when not uploading new WiFi firmware to the AirLift (which is a rare occurance). You'll need to solder the two pads on the bottom of the FeatherWing to use these pins.

RGB LED

There is a small RGB LED to the left of the ESP32. These RGB LEDs are available in the Arduino and CircuitPython libraries if you'd like to PWM them for a visual alert. They're connected to the ESP32's pins 26 (**Red**), 25 (**Green**), and 27 (**Blue**).

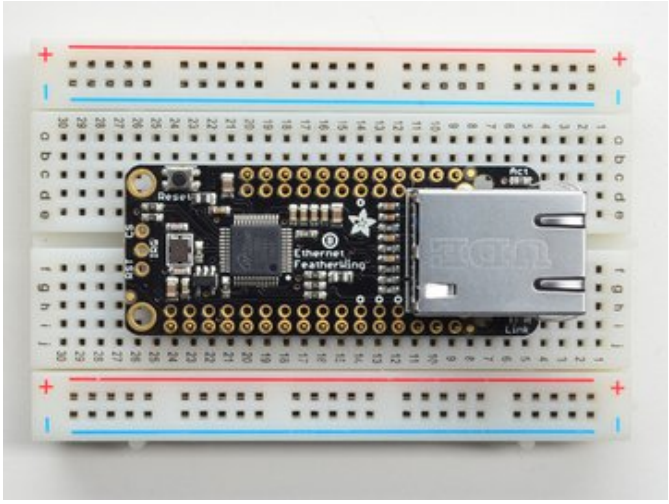
Assembly





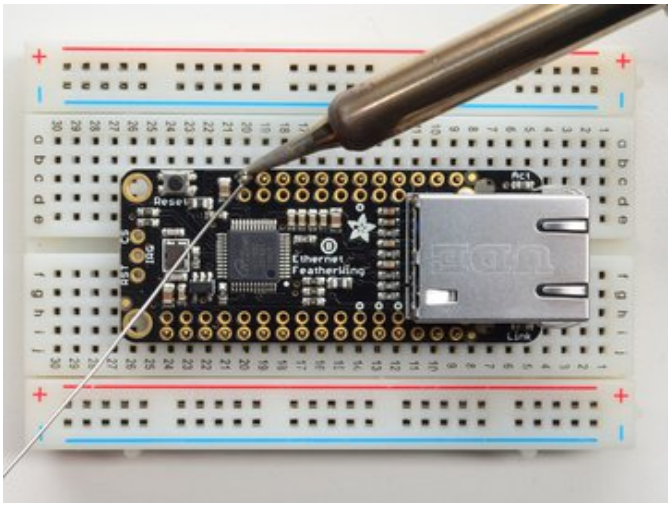
Prepare the header strip:

Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**



Add the FeatherWing:

Place the FeatherWing over the pins so that the short pins poke through the two rows of breakout pads

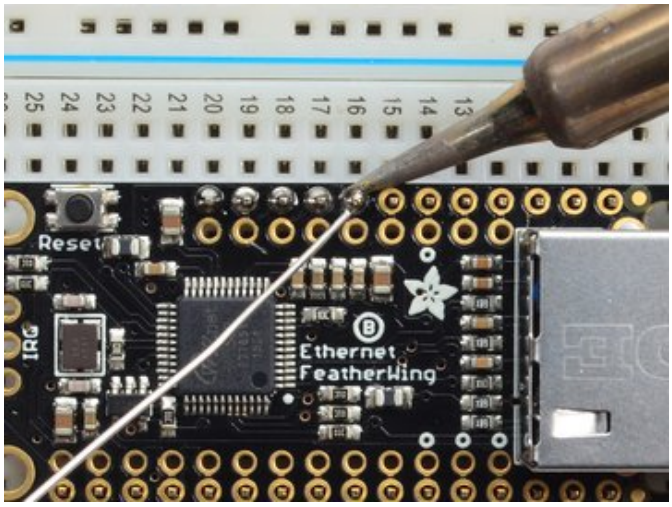


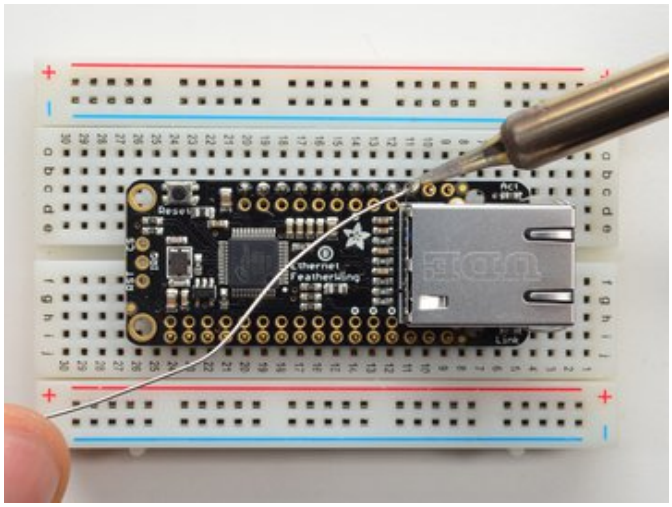
And Solder!

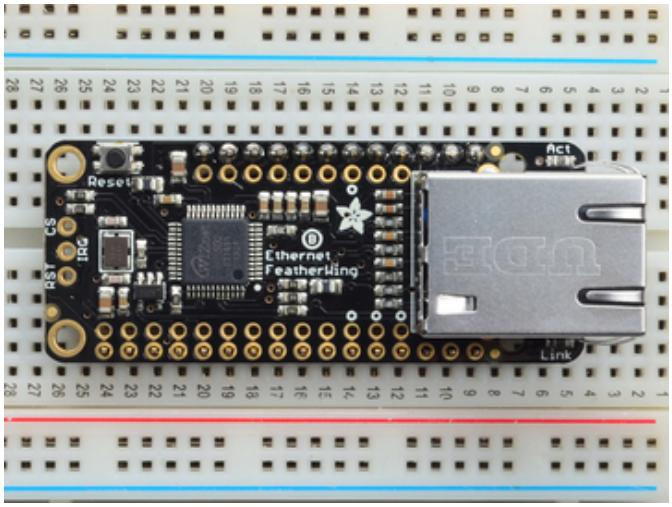
Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out our[Guide to Excellent Soldering](https://adafru.it/aTk)).

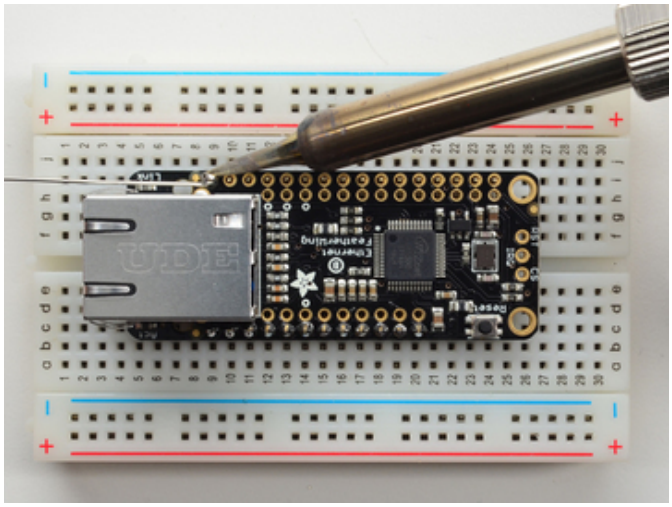
Start by soldering the first row of headers

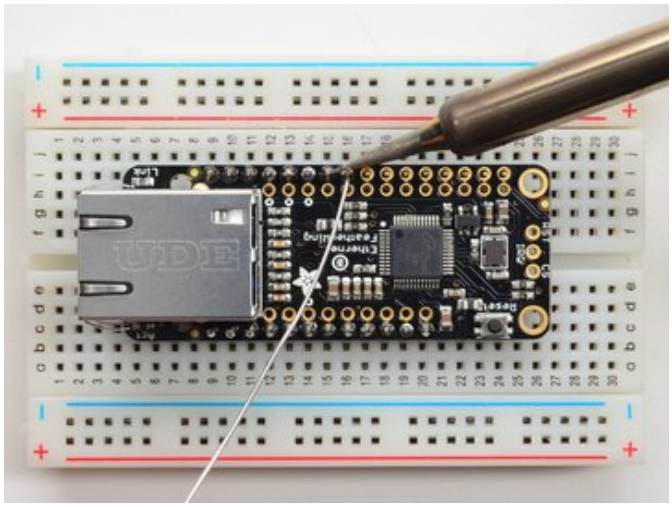


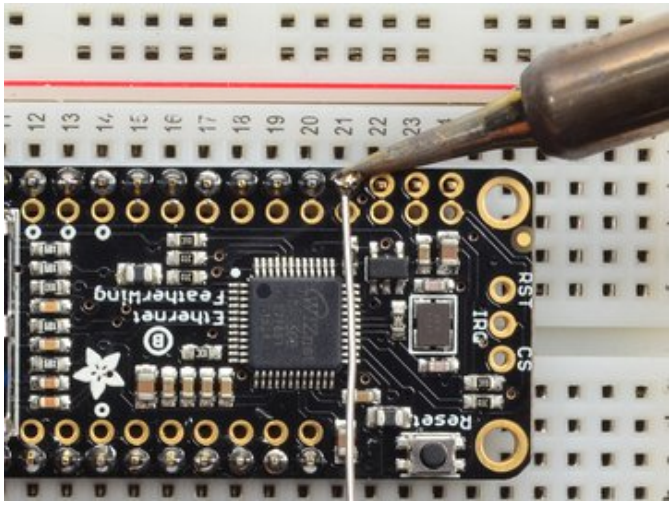


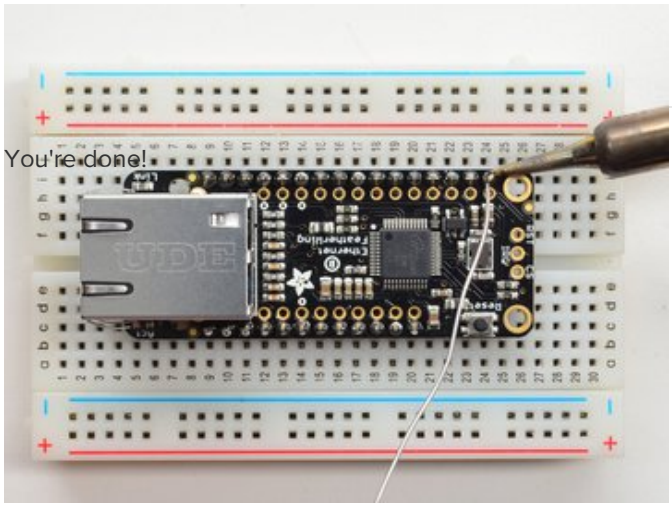


Now flip around and solder the other row completely






Ethernet FeatherWing
RST
IRQ
CS
Reset



You're done!

CircuitPython

It's easy to use the Adafruit AirLift breakout with CircuitPython and the Adafruit CircuitPython ESP32SPI (https://adafru.it/DWV) module. This module allows you to easily add WiFi to your project.

The ESP32SPI library requires an M4 or better microcontroller! The M0 will not work.

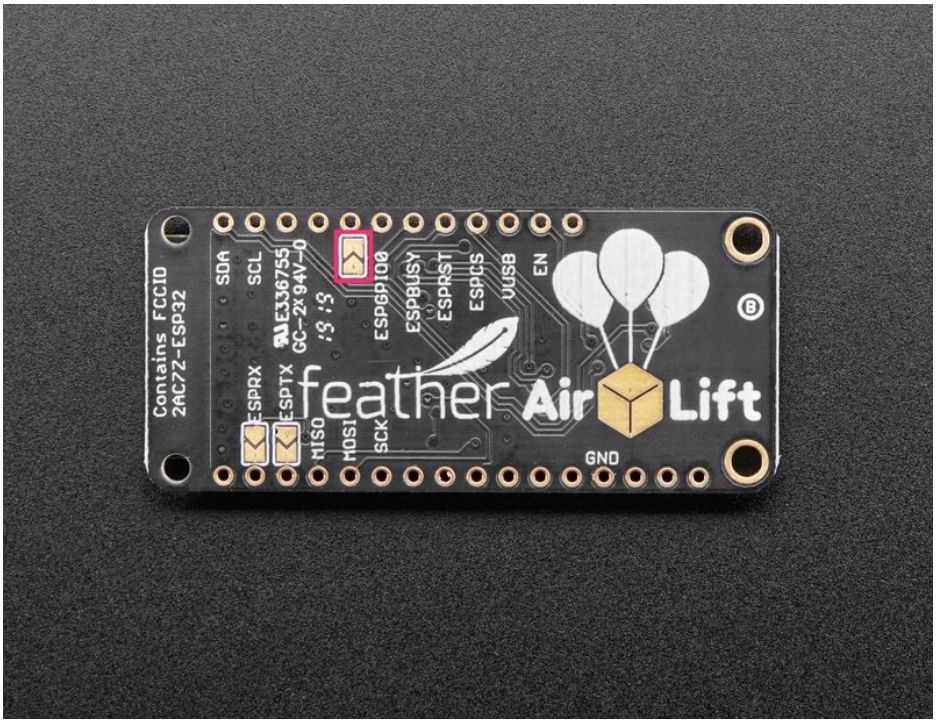
CircuitPython Microcontroller Pinout

Since all CircuitPython-running Feathers follow the same pinout, you do not need to change any of the pins listed below.

To use the ESP32's pins, copy the following lines into your code:

```
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)
```

If you wish to use the ESP32's **GPIO0** pin - solder the jumper on the back of the FeatherWing, highlighted in red.



Then, include the following code to use the pin:

```
esp32_gpio0 = DigitalInOut(board.D10)
```

CircuitPython Installation of ESP32SPI Library

You'll need to install the Adafruit CircuitPython ESP32SPI (https://adafru.it/DWV) library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/uap). Our CircuitPython starter guide has a great page on how to install the library bundle (https://adafru.it/ABU).

You can manually install the necessary libraries from the bundle:

- **adafruit_esp32spi.mpy**
- **adafruit_requests.mpy**
- **adafruit_bus_device**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_esp32spi.mpy,** and **adafruit_bus_device** files and folders copied over.

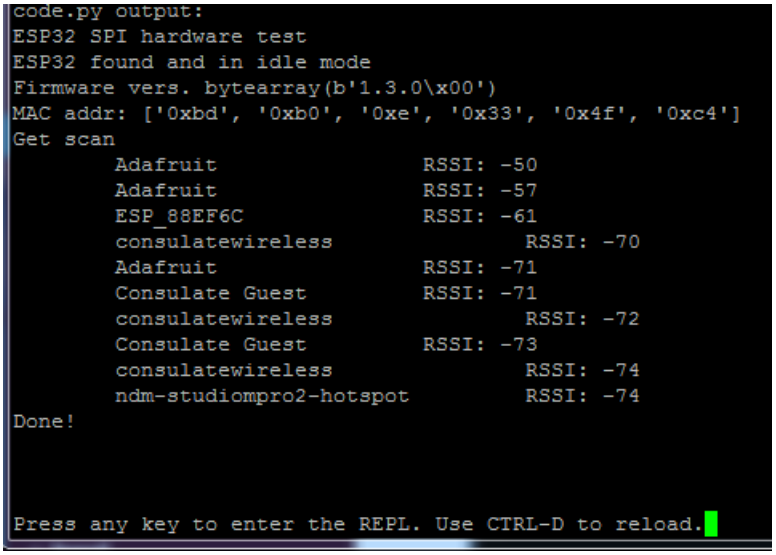
Next connect to the board's serial REPL (https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.

CircuitPython Usage

Copy the following code to your **code.py** file on your microcontroller:

```
import board
import busio
from digitalio import DigitalInOut

from adafruit_esp32spi import adafruit_esp32spi

print("ESP32 SPI hardware test")

esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", [hex(i) for i in esp.MAC_address])

for ap in esp.scan_networks():
    print("\t%s\t\tRSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))

print("Done!")
```

Connect to the serial monitor to see the output. It should look something like the following:



Make sure you see the same output! If you don't, check your wiring. Note that we've changed the pinout in the code example above to reflect the CircuitPython Microcontroller Pinout at the top of this page.

Once you've succeeded, continue onto the next page!

If you can read the Firmware and MAC address but fails on scanning SSIDs, check your power supply, you may be running out of juice to the ESP32 and it's resetting

Internet Connect!

Once you have CircuitPython setup and libraries installed we can get your board connected to the Internet.

To get connected, you will need to start by creating a *secrets file*.

What's a secrets file?

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a `secrets.py` file, that is in your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

Your **secrets.py** file should look like this:

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'github_token' : 'fawfj23rakjnfawiefa',
    'hackaday_token' : 'h4xx0rs3kret',
    }
```

Inside is a python dictionary named secrets with a line for each entry. Each entry has an entry name (say `'ssid'`) and then a colon to separate it from the entry key `'home ssid'` and finally a comma ,

At a minimum you'll need the `ssid` and `password` for your local WiFi setup. As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing github or the hackaday API. Other non-secret data like your timezone can also go here, just cause its called secrets doesn't mean you can't have general customization data in there!

For the correct time zone string, look at http://worldtimeapi.org/timezones (https://adafru.it/EcP) and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your **secrets.py** - keep that out of GitHub, Discord or other project-sharing sites.

Connect to WiFi

OK now you have your secrets setup - you can connect to the Internet using the ESP32SPI and the Requests modules.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU) for both express and non-express boards.

Remember for non-express boards like the, you'll need to manually install the necessary libraries from the bundle:

- adafruit_bus_device
- adafruit_esp32_spi
- adafruit_requests
- neopixel

Before continuing make sure your board's lib folder or root filesystem has the above files copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.

Into your `lib` folder. Once that's done, load up the following example using Mu or your favorite editor:

```
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"


# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

requests.set_socket(socket, esp)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", [hex(i) for i in esp.MAC_address])

for ap in esp.scan_networks():
    print("\t%s\t\tRSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
    except RuntimeError as e:
        print("could not connect to AP, retrying: ",e)
        continue
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
print("IP lookup adafruit.com: %s" % esp.pretty_ip(esp.get_host_by_name("adafruit.com")))
print("Ping google.com: %d ms" % esp.ping("google.com"))
```

```
print("Ping google.com: %d ms" % esp.ping("google.com"))

#esp._debug = True
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print('-'*40)
print(r.text)
print('-'*40)
r.close()

print()
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print('-'*40)
print(r.json())
print('-'*40)
r.close()

print("Done!")
```

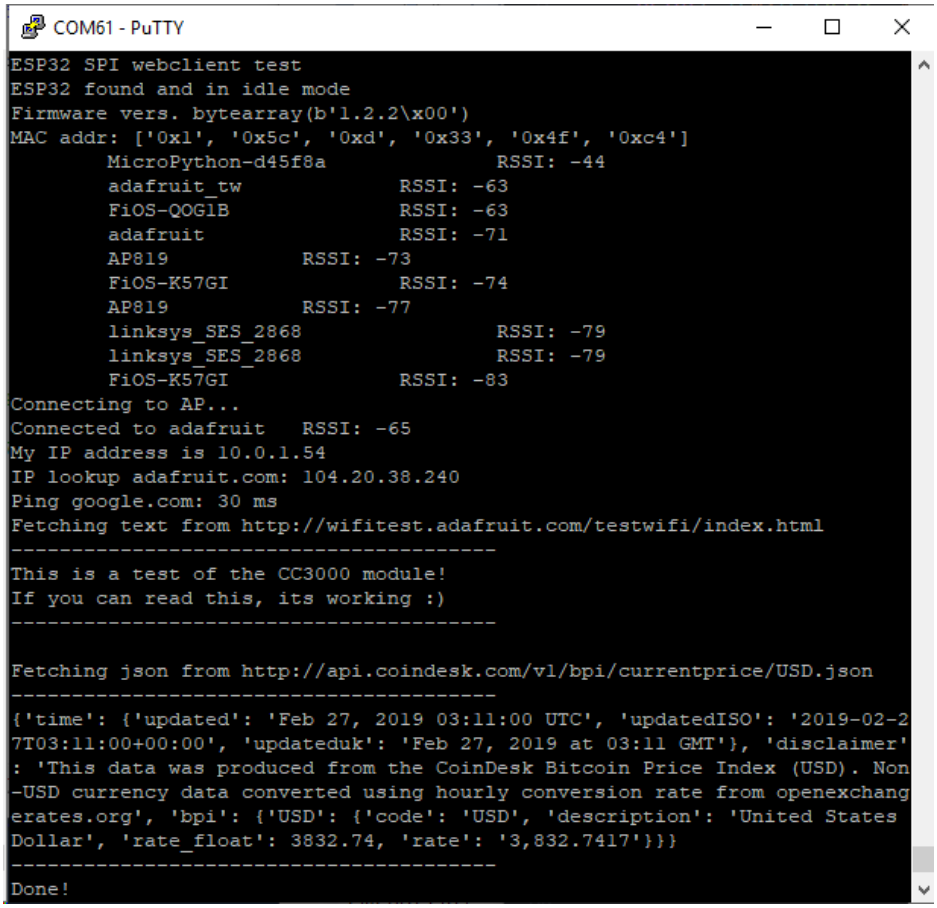
And save it to your board, with the name `code.py`.

This first connection example doesn't use a secrets file - you'll hand-enter your SSID/password to verify connectivity first!

Then go down to this line

`esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')`

and change `MY_SSID_NAME` and `MY_SSID_PASSWORD` to your access point name and password, keeping them within the '' quotes. (This example doesn't use the secrets' file, but its also very stand-alone so if other things seem to not work you can always re-load this. You should get something like the following:



In order, the example code...

Initializes the ESP32 over SPI using the SPI port and 3 control pins:

```
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
```

To use the AirLift FeatherWing's pins, replace the following lines into your code:

```
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)
```

Tells our `requests` library the type of socket we're using (socket type varies by connectivity type - we'll be using the `adafruit_esp32spi_socket` for this example). We'll also set the interface to an `esp` object. This is a little bit of a hack, but it lets us use `requests` like CPython does.

```
requests.set_socket(socket, esp)
```

Verifies an ESP32 is found, checks the firmware and MAC address

```
if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", [hex(i) for i in esp.MAC_address])
```

Performs a scan of all access points it can see and prints out the name and signal strength:

```
for ap in esp.scan_networks():
    print("\t%s\t\tRSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))
```

Connects to the AP we've defined here, then prints out the local IP address, attempts to do a domain name lookup and ping google.com to check network connectivity (note sometimes the ping fails or takes a while, this isn't a big deal)

```
    print("Connecting to AP...")
esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
print("IP lookup adafruit.com: %s" % esp.pretty_ip(esp.get_host_by_name("adafruit.com")))
print("Ping google.com: %d ms" % esp.ping("google.com"))
```

OK now we're getting to the really interesting part. With a SAMD51 or other large-RAM (well, over 32 KB) device, we can do a lot of neat tricks. Like for example we can implement an interface a lot like requests (https://adafru.it/E9o) - which makes getting data *really really easy*

To read in all the text from a web URL call `requests.get` - you can pass in `https` URLs for SSL connectivity

```
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print('-'*40)
print(r.text)
print('-'*40)
r.close()
```

Or, if the data is in structured JSON, you can get the json pre-parsed into a Python dictionary that can be easily queried or traversed. (Again, only for nRF52840, M4 and other high-RAM boards)

```
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print('-'*40)
print(r.json())
print('-'*40)
r.close()
```

Requests

We've written a requests-like (https://adafru.it/FpT) library for web interfacing named Adafruit_CircuitPython_Requests (https://adafru.it/FpW). This library allows you to send HTTP/1.1 requests without "crafting" them and provides helpful methods for parsing the response from the server.

Here's an example of using Requests to perform GET and POST requests to a server.

```
# adafruit_requests usage with an esp32spi_socket
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
    except RuntimeError as e:
        print("could not connect to AP, retrying: ",e)
        continue
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_GET_URL = "http://httpbin.org/get"
JSON_POST_URL = "http://httpbin.org/post"

print("Fetching text from %s"%TEXT_URL)
response = requests.get(TEXT_URL)
print('-'*40)

print("Text Response: ", response.text)
print('-'*40)
response.close()

print("Fetching JSON data from %s"%JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print('-'*40)

print("JSON Response: ", response.json())
print('-'*40)
response.close()
```

```
response.close()

data = '31F'
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print('-'*40)

json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:", json_resp['data'])
print('-'*40)
response.close()

json_data = {"Date" : "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print('-'*40)

json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:", json_resp['json'])
print('-'*40)
response.close()
```

The code first sets up the ESP32SPI interface. Then, it initializes a `request` object using an ESP32 `socket` and the `esp` object.

```
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# If you have an externally connected ESP32:
esp32_cs = DigitalInOut(board.D9)
esp32_ready = DigitalInOut(board.D10)
esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
    except RuntimeError as e:
        print("could not connect to AP, retrying: ",e)
        continue
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)
```

Make sure to set the ESP32 pinout to match your AirLift breakout's connection:

```
esp32_cs = DigitalInOut(board.D9)
esp32_ready = DigitalInOut(board.D10)
esp32_reset = DigitalInOut(board.D5)
```

HTTP GET with Requests

The code makes a HTTP GET request to Adafruit's WiFi testing website - http://wifitest.adafruit.com/testwifi/index.html (https://adafru.it/FpZ).

To do this, we'll pass the URL into `requests.get()`. We're also going to save the response *from* the server into a variable named `response`.

While we requested data from the server, we'd what the server responded with. Since we already saved the server's `response`, we can read it back. Luckily for us, **requests automatically decodes the server's response into human-readable text,** you can read it back by calling `response.text`.

Lastly, we'll perform a bit of cleanup by calling `response.close()`. This closes, deletes, and collect's the response's data.

```
    print("Fetching text from %s"%TEXT_URL)
response = requests.get(TEXT_URL)
print('-'*40)

print("Text Response: ", response.text)
print('-'*40)
response.close()
```

While some servers respond with text, some respond with json-formatted data consisting of attribute–value pairs.

CircuitPython_Requests can convert a JSON-formatted response from a server into a CPython `dict.` object.

We can also fetch and parse **json** data. We'll send a HTTP get to a url we know returns a json-formatted response (instead of text data).

Then, the code calls `response.json()` to convert the response to a CPython `dict`.

```
    print("Fetching JSON data from %s"%JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print('-'*40)

print("JSON Response: ", response.json())
print('-'*40)
response.close()
```

HTTP POST with Requests

Requests can also **POST data to a server by calling the `requests.post` method, passing it a `data` value.**

```
data = '31F'
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print('-'*40)

json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:", json_resp['data'])
print('-'*40)
response.close()
```

You can also post json-formatted data to a server by passing `json` data into the `requests.post` method.

```
json_data = {"Date" : "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print('-'*40)

json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:", json_resp['json'])
print('-'*40)
response.close()
```

Advanced Requests Usage

Want to send custom HTTP headers, parse the response as raw bytes, or handle a response's http status code in your CircuitPython code?

We've written an example to show advanced usage of the requests module below.

```
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
    except RuntimeError as e:
        print("could not connect to AP, retrying: ",e)
        continue
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)

JSON_GET_URL = "http://httpbin.org/get"

# Define a custom header as a dict.
headers = {"user-agent" : "blinka/1.0.0"}

print("Fetching JSON data from %s..."%JSON_GET_URL)
response = requests.get(JSON_GET_URL, headers=headers)
print('-'*60)

json_data = response.json()
headers = json_data['headers']
print("Response's Custom User-Agent Header: {0}".format(headers['User-Agent']))
print('-'*60)

# Read Response's HTTP status code
print("Response HTTP Status Code: ", response.status_code)
print('-'*60)

# Read Response, as raw bytes instead of pretty text
print("Raw Response: ", response.content)

# Close, delete and collect the response data
response.close()
```

WiFi Manager

That simpletest example works but its a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. It will wrap the connection/status/requests loop for you - reconnecting if WiFi drops, resetting the ESP32 if it gets into a bad state, etc.

Here's a more advanced example that shows the WiFi manager and also how to POST data with some extra headers:

```
import time
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

print("ESP32 SPI webclient test")

# 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

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2) # Uncomment for Most Boards
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
# Uncomment below for an externally defined RGB LED
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

counter = 0

while True:
    try:
        print("Posting data...", end='')
        data = counter
        feed = 'test'
        payload = {'value':data}
        response = wifi.post(
```

```
            "https://io.adafruit.com/api/v2/"+secrets['aio_username']+"/feeds/"+feed+"/data",
            json=payload,
            headers={"X-AIO-KEY":secrets['aio_key']})
        print(response.json())
        response.close()
        counter = counter + 1
        print("OK")
    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        continue
    response = None
    time.sleep(15)
```

Next, **set up an Adafruit IO feed named `test`**

- If you do not know how to set up a feed, follow this page and come back when you've set up a feed named `test` . (https://adafru.it/f5k)

You'll note here we use a secrets.py file to manage our SSID info. The wifimanager is given the ESP32 object, secrets and a neopixel for status indication.

Note, you'll need to add a some additional information to your secrets file so that the code can query the Adafruit IO API:

- `aio_username`
- `aio_key`

You can go to your adafruit.io View AIO Key link to get those two values and add them to the secrets file, which will now look something like this:

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid' : '_your_ssid_',
    'password' : '_your_wifi_password_',
    'timezone' : "America/Los_Angeles", # http://worldtimeapi.org/timezones
    'aio_username' : '_your_aio_username_',
    'aio_key' : '_your_aio_key_',
    }
```

We can then have a simple loop for posting data to Adafruit IO without having to deal with connecting or initializing the hardware!

Take a look at your **test** feed on Adafruit.io and you'll see the value increase each time the CircuitPython board posts data to it!

Arduino

You can use the AirLift with Arduino. Unlike CircuitPython, it work work with just about any Arduino chip, even a classic Arduino UNO. However, if you want to use libraries like ArduinoJSON or add sensors and SD card, you'll really want an ATSAMD21 (Cortex M0) or ATSAMD51 (Cortex M4), both of which have *plenty* or RAM

Arduino Microcontroller Pin Definitions

Because each Feather uses a different processor, you'll need to include the following pin definitions to your code depending on which board you are using:

Feather M0, M4, 32u4, or NRF52840

```
    #define SPIWIFI       SPI  // The SPI port
#define SPIWIFI_SS    13   // Chip select pin
#define ESP32_RESETN  12   // Reset pin
#define SPIWIFI_ACK   11   // a.k.a BUSY or READY pin
#define ESP32_GPIO0   10
```

Feather 328P

```
#define SPIWIFI       SPI  // The SPI port
#define SPIWIFI_SS     4   // Chip select pin
#define ESP32_RESETN   3   // Reset pin
#define SPIWIFI_ACK    2   // a.k.a BUSY or READY pin
#define ESP32_GPIO0   -1
```

Feather NRF52832

```
#define SPIWIFI       SPI  // The SPI port
#define SPIWIFI_SS    16  // Chip select pin
#define ESP32_RESETN  15  // Reset pin
#define SPIWIFI_ACK    7  // a.k.a BUSY or READY pin
#define ESP32_GPIO0   -1
```

Note: These pin definitions leave the the ESP32's **GPIO0** pin undefined (-1). If you wish to use this pin - **solder the pad on the bottom of the FeatherWing** and **set** `#define ESP32_GPIO0` to the correct pin for your microcontroller.

Library Install

We're using a variant of the Arduino WiFiNINA library, which is amazing and written by the Arduino team! **The official WiFi101 library won't work because it doesn't support the ability to change the pins**.

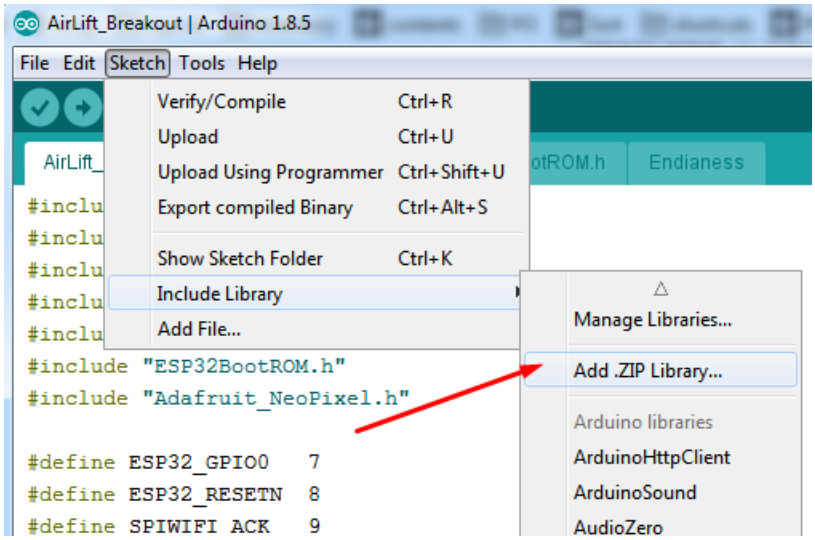
So! We made a fork that you can install.

Click here to download the library:

https://adafru.it/Evm

https://adafru.it/Evm

Within the Arduino IDE, select **Install library from ZIP...**

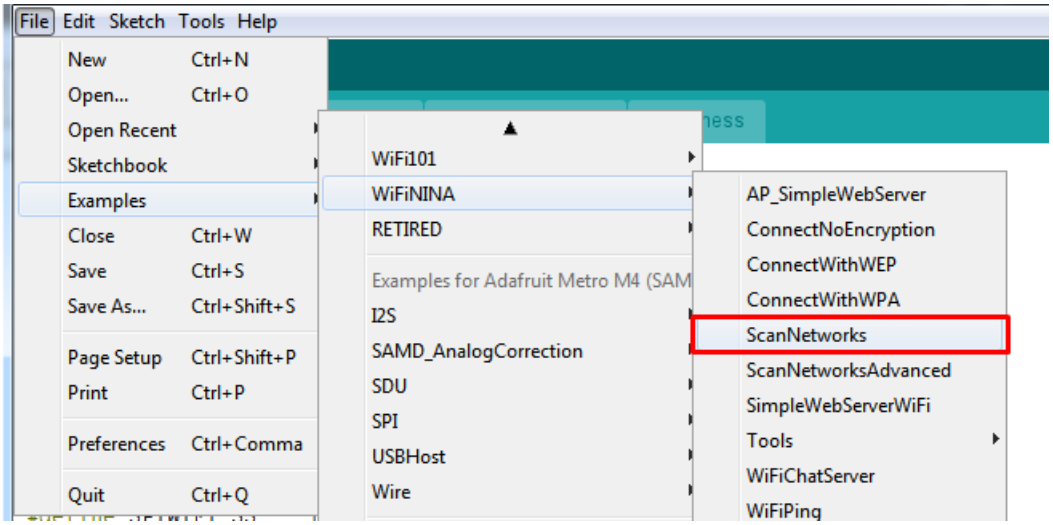


And select the zip you just downloaded.

First Test

OK now you have it wired and library installed, time to test it out!

Lets start by scanning the local networks. Load up the **ScanNetworks** example



(https://adafru.it/EVu)

At the top you'll see a section where the GPIO pins are defined

```
// Configure the pins used for the ESP32 connection
#define SPIWIFI     SPI   // The SPI port
#define SPIWIFI_SS    10  // Chip select pin
#define SPIWIFI_ACK   7   // a.k.a BUSY or READY pin
#define ESP32_RESETN  5   // Reset pin
#define ESP32_GPIO0   -1  // Not connected
```

(https://adafru.it/EVv)

If you don't see this, you may have the wrong WiFiNINA library installed. Uninstall it and re-install the Adafruit one as above.

Compile and upload to your board wired up to the AirLift

```
WiFi Scanning test
MAC: C4:4F:33:0E:B0:BD
Scanning available networks...
** Scan Networks **
number of available networks:10
0) Adafruit     Signal: -56 dBm Encryption: WPA2
1) Consulate Guest      Signal: -59 dBm Encryption: WPA2
2) consulatewireless    Signal: -60 dBm Encryption: WPA2
3) Adafruit     Signal: -66 dBm Encryption: WPA2
4) consulatewireless    Signal: -67 dBm Encryption: WPA2
5) Consulate Guest      Signal: -69 dBm Encryption: WPA2
6) Adafruit     Signal: -69 dBm Encryption: WPA2
7) Consulate Guest      Signal: -71 dBm Encryption: WPA2
8) consulatewireless    Signal: -72 dBm Encryption: WPA2
9) ESP_88EF6C   Signal: -75 dBm Encryption: None
```

(https://adafru.it/EVw)

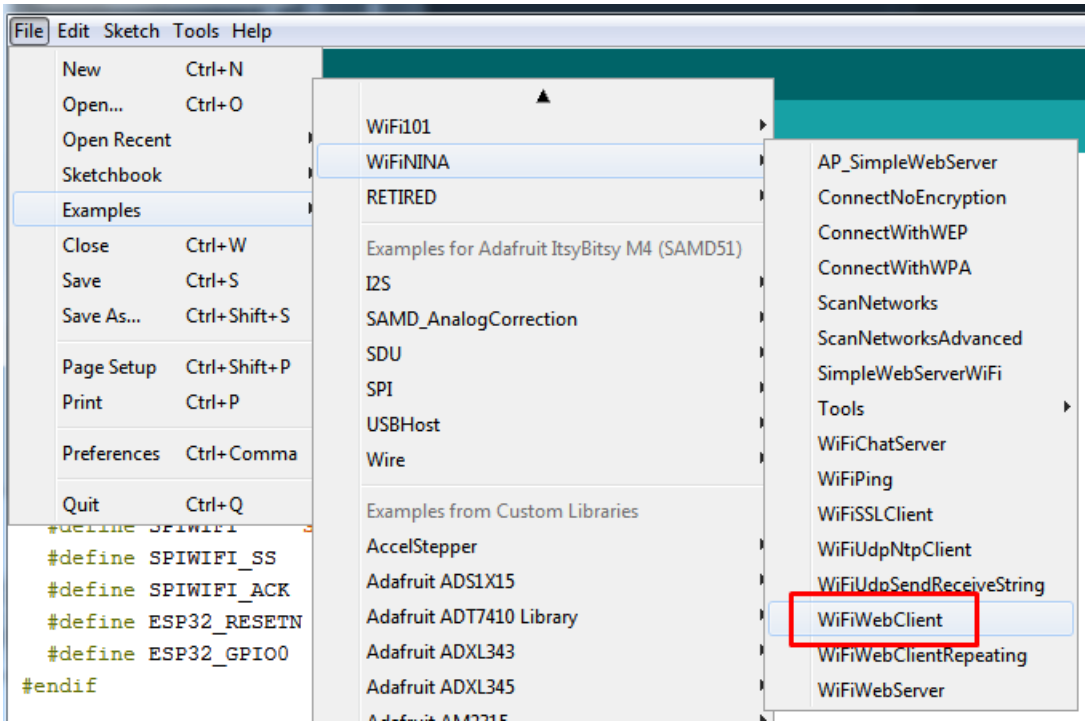
If you don't even get the MAC address printed out, check your wiring.

If you get the MAC address but cannot scan any networks, check your power supply. You need a solid 3-5VDC into **Vin** in order for the ESP32 not to brown out.

WiFi Connection Test

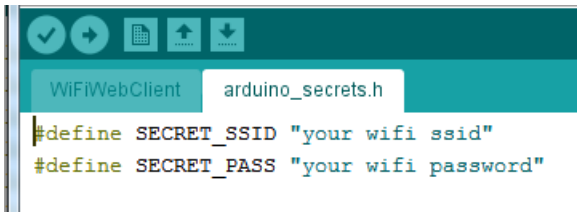
Now that you have your wiring checked, time to connect to the Internet!

Open up the **WiFiWebClient** example



(https://adafru.it/EVx)

Open up the secondary tab, **arduino_secrets.h**. This is where you will store private data like the SSID/password to your network.



(https://adafru.it/EVy)

You must change these string values before updating to your board!

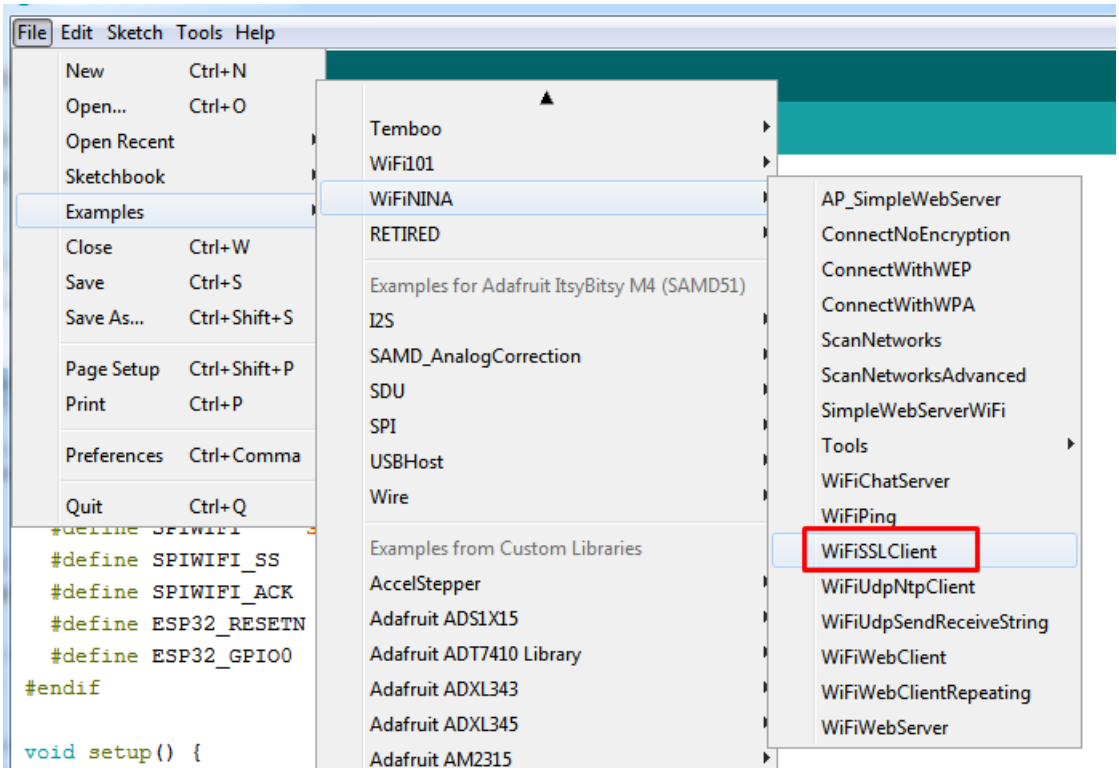
After you've set it correctly, upload and check the serial monitor. You should see the following. If not, go back, check wiring, power and your SSID/password

```
Found firmware 1.3.0
Attempting to connect to SSID: Adafruit
Connected to wifi
SSID: Adafruit
IP Address: 10.0.1.179
signal strength (RSSI):-44 dBm

Starting connection to server...
connected to server
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Wed, 10 Apr 2019 20:55:51 GMT
Content-Type: text/html
Content-Length: 73
Last-Modified: Thu, 16 Feb 2017 17:42:29 GMT
Connection: close
ETag: "58a5e485-49"
Accept-Ranges: bytes

This is a test of the CC3000 module!
If you can read this, its working :)
disconnecting from server.
```

(https://adafru.it/EVz)

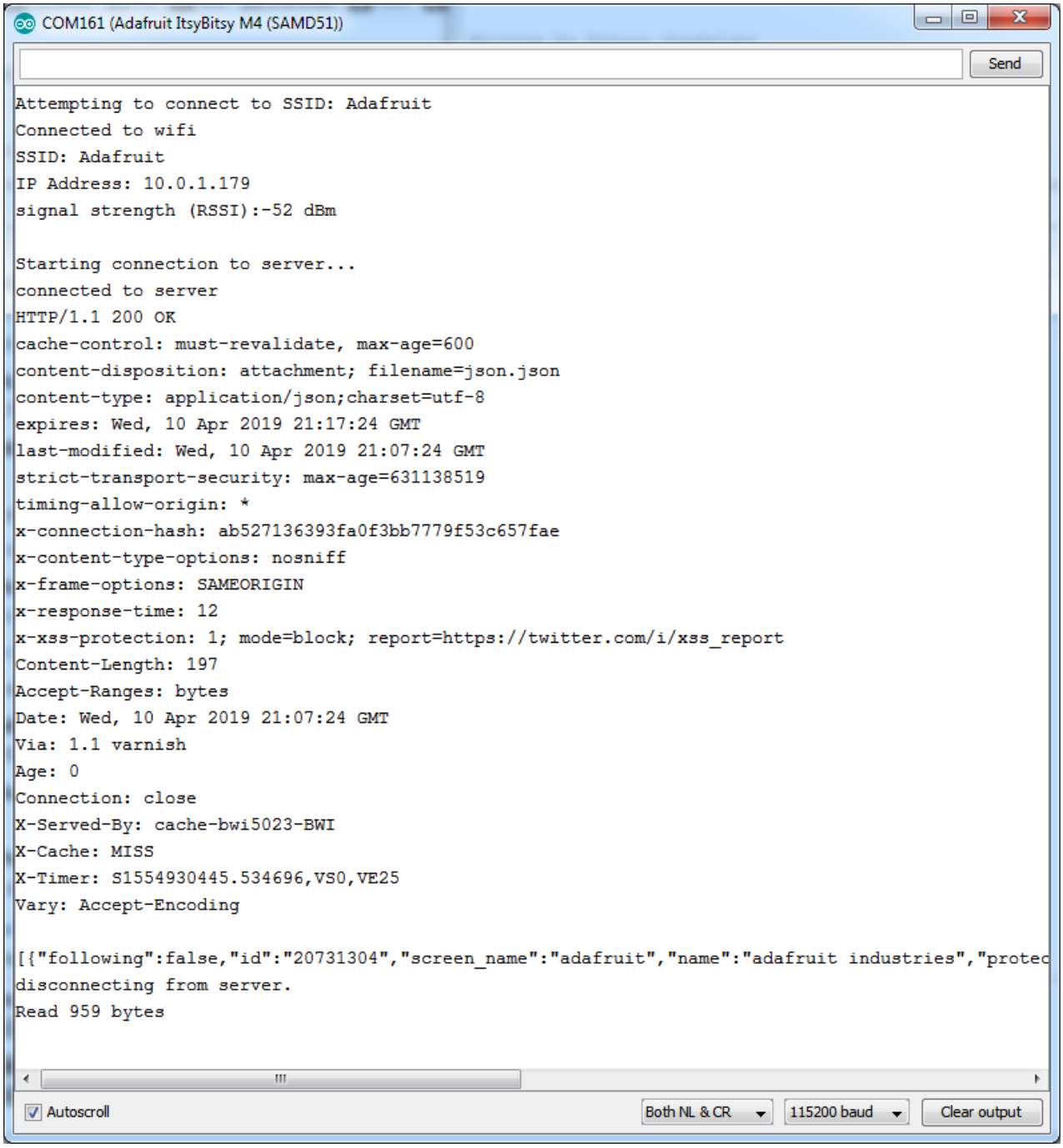
Secure Connection Example

Many servers today do not allow non-SSL connectivity. Lucky for you the ESP32 has a great TLS/SSL stack so you can have that all taken care of for you. Here's an example of a secure WiFi connection:



(https://adafru.it/EVA)

Note we use `WiFiSSLClient client;` instead of `WiFiClient client;` to require an SSL connection!



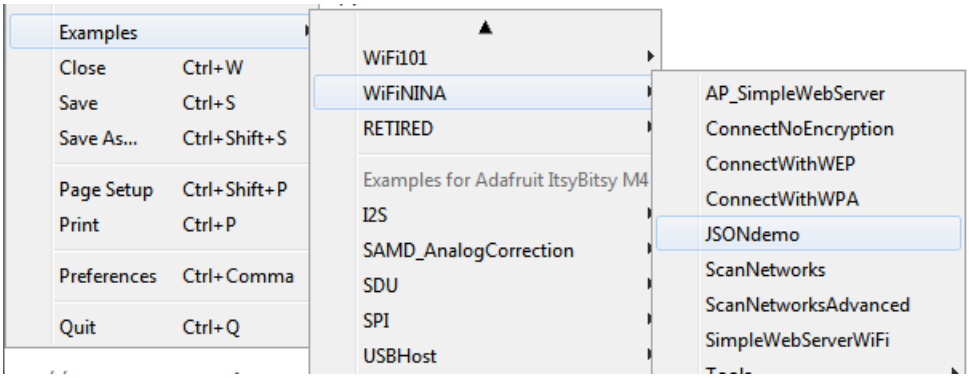
(https://adafru.it/EVB)

JSON Parsing Demo

This example is a little more advanced - many sites will have API's that give you JSON data. We'll use ArduinoJSON (https://adafru.it/Evn) to convert that to a format we can use and then display that data on the serial port (which can then be re-directed to a display of some sort)

First up, use the Library manager to install ArduinoJSON (https://adafru.it/Evo).

Then load the example **JSONdemo**



(https://adafru.it/EVC)

By default it will connect to to the Twitter banner image API, parse the username and followers and display them.

```
Attempting to connect to SSID: Adafruit
Connected to wifi
SSID: Adafruit
IP Address: 10.0.1.179
signal strength (RSSI):-51 dBm

Starting connection to server...
connected to server
Response:
Twitter username: adafruit
Twitter followers: 159265
```

(https://adafru.it/EVD)

Adapting Other Examples

Once you've got it connecting to the Internet you can check out the other examples. The only change you'll want to make is at the **top** of the sketches, add:

```
 // Configure the pins used for the ESP32 connection
#if defined(ADAFRUIT_FEATHER_M4_EXPRESS) || \
 defined(ADAFRUIT_FEATHER_M0_EXPRESS) || \
 defined(ARDUINO_AVR_FEATHER32U4) || \
 defined(ARDUINO_NRF52840_FEATHER)
  #define SPIWIFI       SPI  // The SPI port
  #define SPIWIFI_SS    13   // Chip select pin
  #define ESP32_RESETN  12   // Reset pin
  #define SPIWIFI_ACK   11   // a.k.a BUSY or READY pin
  #define ESP32_GPIO0   10

#elif defined(ARDUINO_AVR_FEATHER328P)
  #define SPIWIFI       SPI  // The SPI port
  #define SPIWIFI_SS     4   // Chip select pin
  #define ESP32_RESETN   3   // Reset pin
  #define SPIWIFI_ACK    2   // a.k.a BUSY or READY pin
  #define ESP32_GPIO0   -1

#elif defined(ARDUINO_NRF52832_FEATHER )
  #define SPIWIFI       SPI  // The SPI port
  #define SPIWIFI_SS    16  // Chip select pin
  #define ESP32_RESETN  15  // Reset pin
  #define SPIWIFI_ACK    7  // a.k.a BUSY or READY pin
  #define ESP32_GPIO0   -1

#elif defined(TEENSYDUINO)
  #define SPIWIFI       SPI  // The SPI port
  #define SPIWIFI_SS     5   // Chip select pin
  #define ESP32_RESETN   6   // Reset pin
  #define SPIWIFI_ACK    9   // a.k.a BUSY or READY pin
  #define ESP32_GPIO0   -1
#endif
```

And then **before** you check the `status()` of the module, call the function `WiFi.setPins(SPIWIFI_SS, SPIWIFI_ACK, ESP32_RESETN, ESP32_GPIO0, &SPIWIFI);`

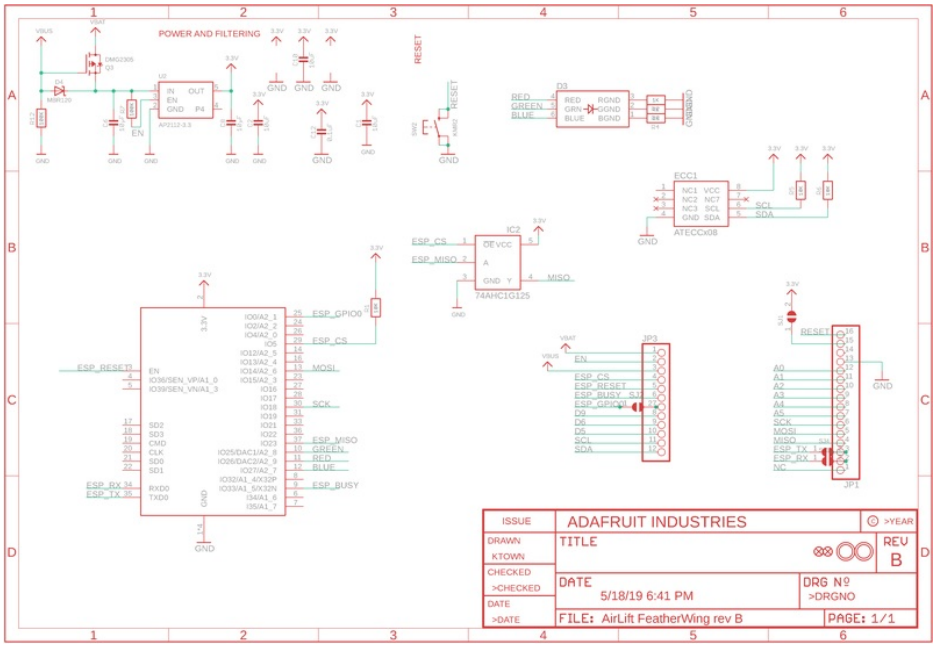
```
// check for the WiFi module:
  WiFi.setPins(SPIWIFI_SS, SPIWIFI_ACK, ESP32_RESETN, ESP32_GPIO0, &SPIWIFI);
  while (WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    // don't continue
    delay(1000);
  }
```

Downloads

Files

- ESP32 WROOM32 Datasheet (https://adafru.it/EVE)
- EagleCAD files on GitHub (https://adafru.it/EVF)
- Fritzing object in Adafruit Fritzing Library (https://adafru.it/EVG)
- 3D Models on GitHub (https://adafru.it/FcS)

Schematic



Fab Print