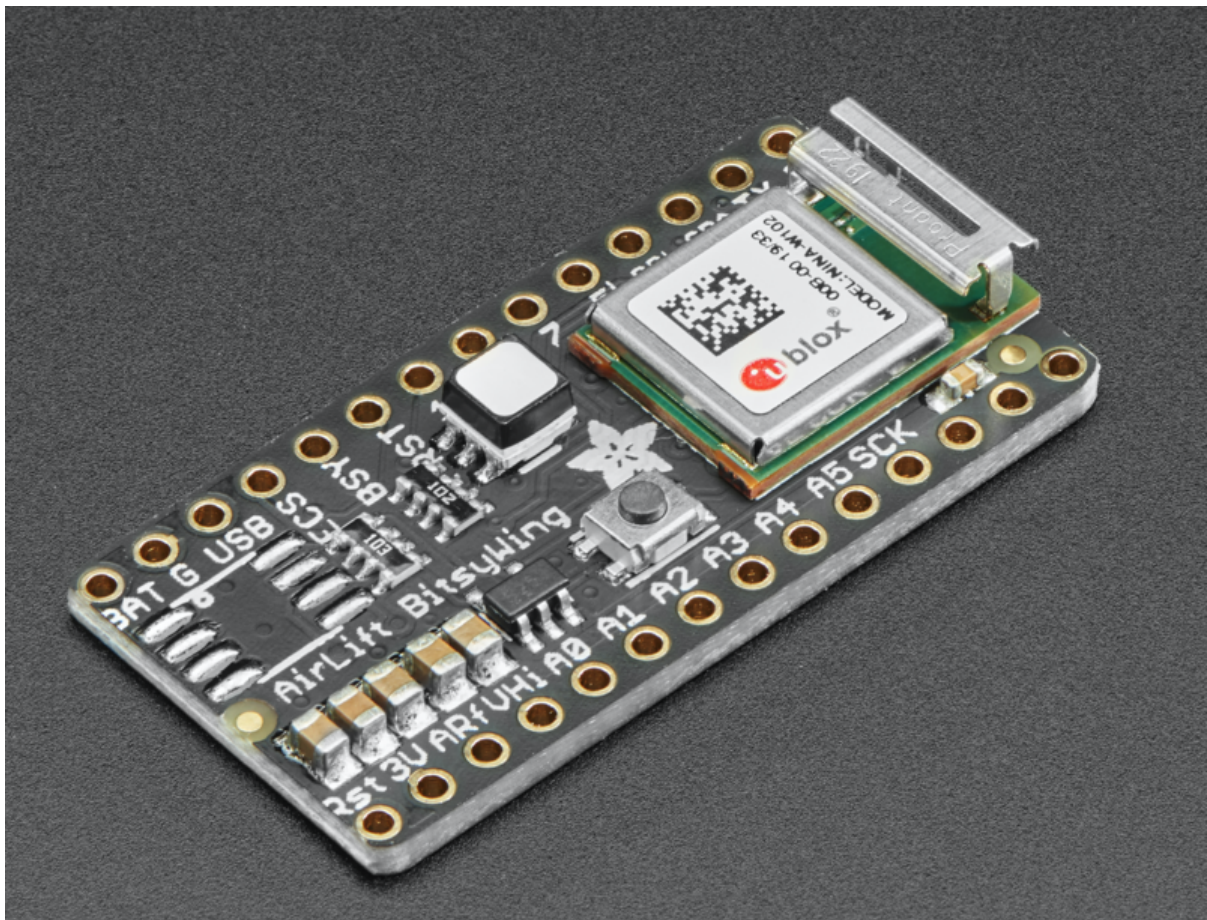




Adafruit Airlift Bitsy Add-On - ESP32 WiFi Co-Processor

Created by Bryan Siepert



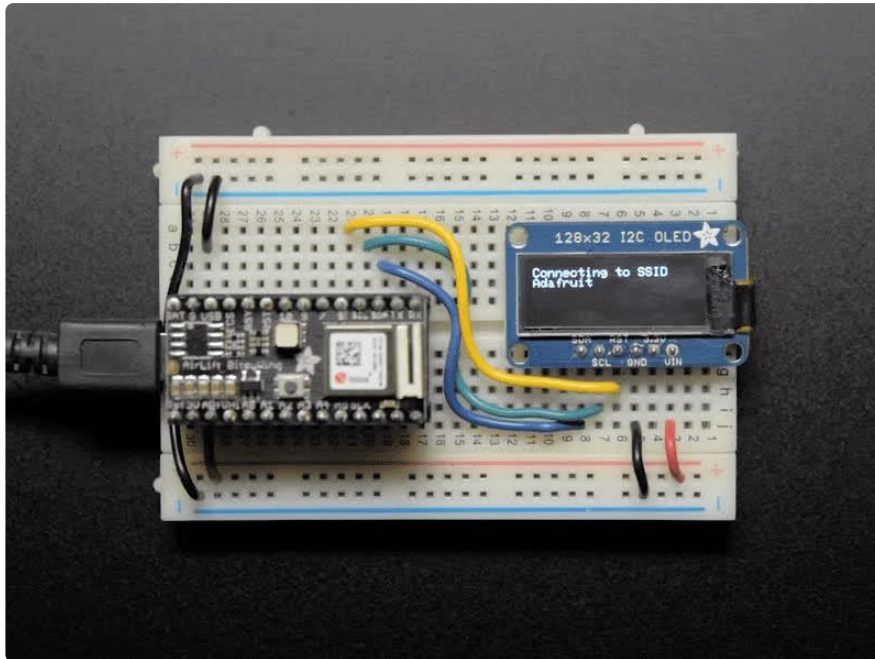
<https://learn.adafruit.com/adafruit-airlift-bitsy-add-on-esp32-wifi-co-processor>

Last updated on 2023-08-29 04:12:50 PM EDT

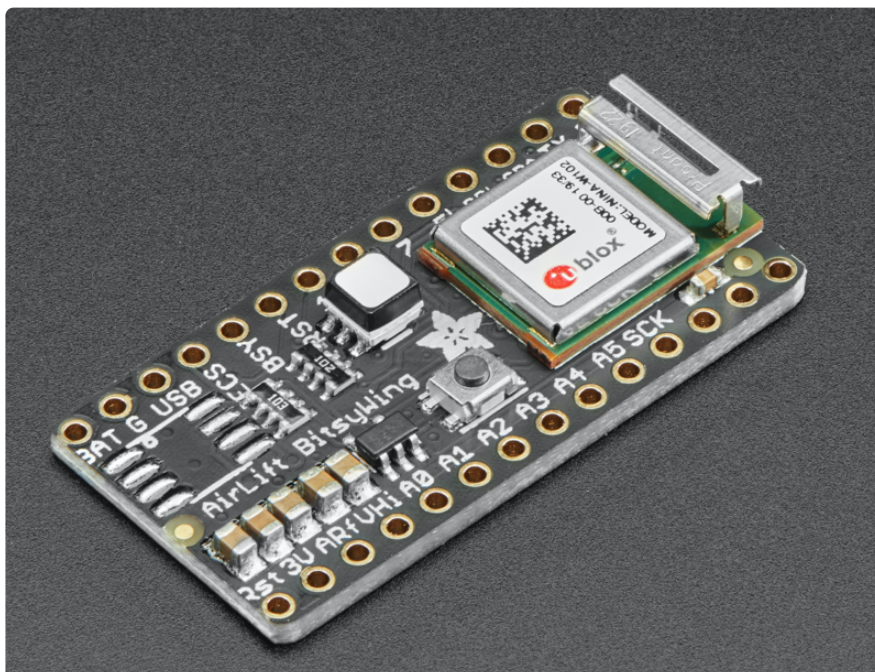
Table of Contents

Overview	3
Pinouts	5
<ul style="list-style-type: none">• Power Pins• SPI and Control Pins• RGB LED	
Assembly	9
CircuitPython WiFi	10
<ul style="list-style-type: none">• CircuitPython Microcontroller Pinout• CircuitPython Setup• CircuitPython Usage	
Internet Connect!	12
<ul style="list-style-type: none">• What's a secrets file?• Connect to WiFi• Requests• HTTP GET with Requests• HTTP POST with Requests• Advanced Requests Usage• WiFi Manager	
OLD - CircuitPython WiFi	27
<ul style="list-style-type: none">• CircuitPython Microcontroller Pinout• CircuitPython Installation of ESP32SPI Library• CircuitPython Usage	
CircuitPython BLE	29
<ul style="list-style-type: none">• CircuitPython BLE UART Example• Update the AirLift Firmware• Install CircuitPython Libraries• Install the Adafruit Bluefruit LE Connect App• BLE Example• Talk to the AirLift via the Bluefruit LE Connect App	
Arduino WiFi	35
<ul style="list-style-type: none">• Library Install• First Test• WiFi Connection Test• Secure Connection Example• JSON Parsing Demo• Adapting Other Examples	
Downloads	44
<ul style="list-style-type: none">• Files• Schematic• Fab Print	

Overview

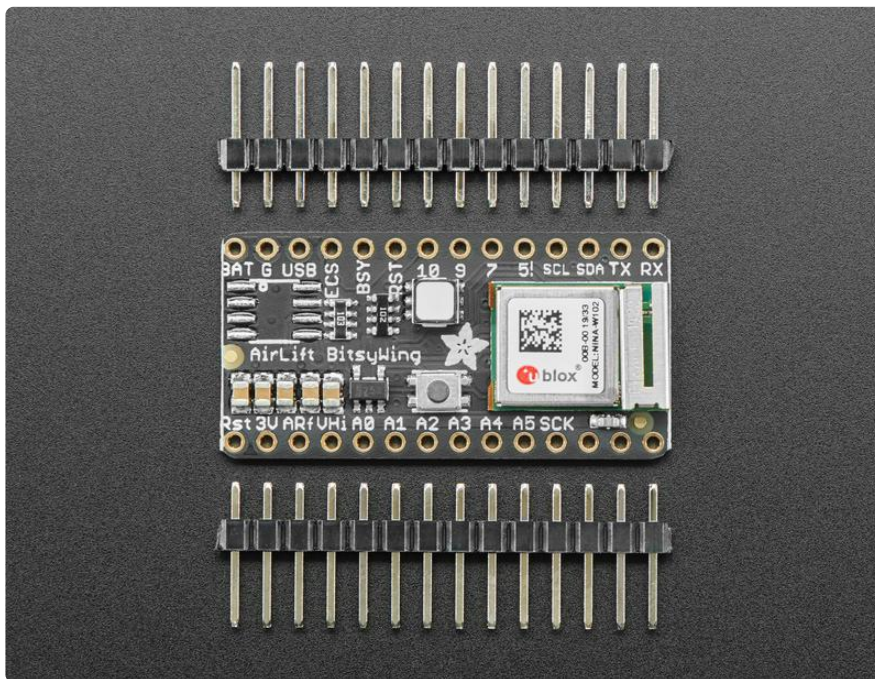


Give your ItsyBitsy project a lift with the Adafruit AirLift Bitsy Add-On! This sweet add-on for the ItsyBitsy lets you use the powerful ESP32 as a WiFi or BLE co-processor. You probably have your favorite ItsyBitsy ([like the ItsyBitsy M4](#) ()) that comes with its own set of awesome peripherals and lots of libraries. But it doesn't have WiFi built in! So let's 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).



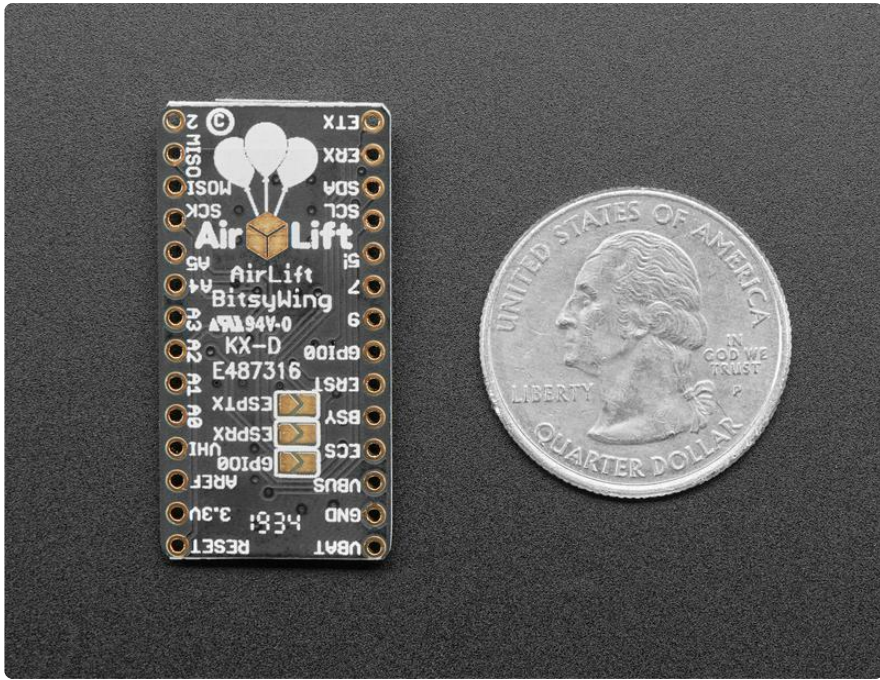
Having WiFi managed by a separate chip means your code is simpler, you don't have to cache socket data, or compile in & debug an SSL library. Instead the Airlift Add-On allows you to send basic but powerful socket-based commands over 8MHz SPI for high speed data transfer. You can use the 3V 32u4, M0, or M4 ItsyBitsy in Arduino although the '32u4 will not be able to do very complex tasks or buffer a lot of data because they do not have a lot of RAM. The add-on also works great with CircuitPython, though 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.

The ESP32 also supports BLE (Bluetooth Low Energy), though not simultaneously with WiFi. Many of our CircuitPython builds include native support for ESP32 BLE. Right now, we only support the ESP32 acting as a BLE peripheral, but that's sufficient to communicate with the ESP32 from a phone, tablet, host computer, or another BLE-capable board. You use a few control pins and the RX and TX pins to talk to the ESP32 when it's in BLE mode.

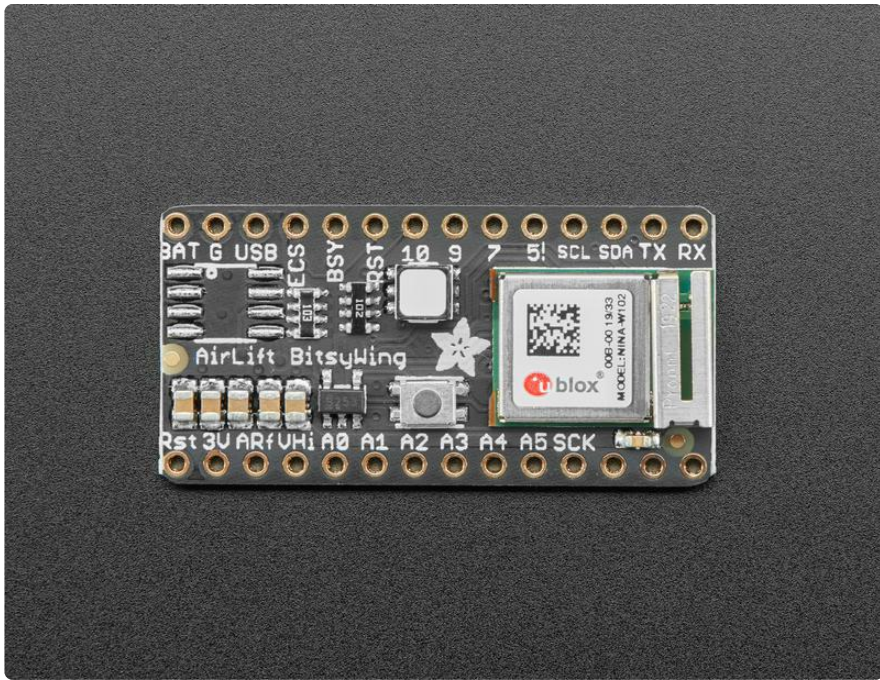


We placed an ESP32 module on a BitsyWing with a tri-state chip for MOSI so you can share the SPI bus. The add on comes fully assembled and tested, pre-programmed with ESP32 SPI WiFi co-processor firmware that [you can use in CircuitPython to use this into WiFi co-processor over SPI + 2 pins \(\)](#). We also toss in some header so you can solder it in though you'll need stacking headers for your ItsyBitsy to put the add-on on top.

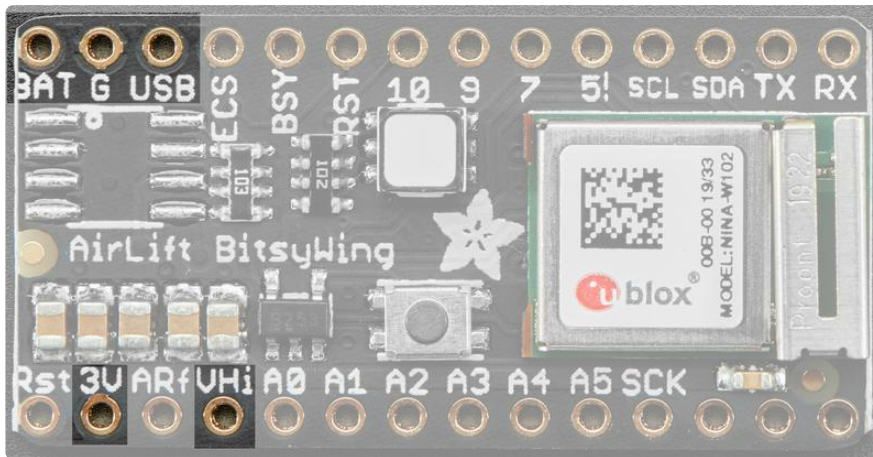
[The firmware on board is a slight variant of the Arduino WiFinINA core, which works great! \(\)](#)



Pinouts



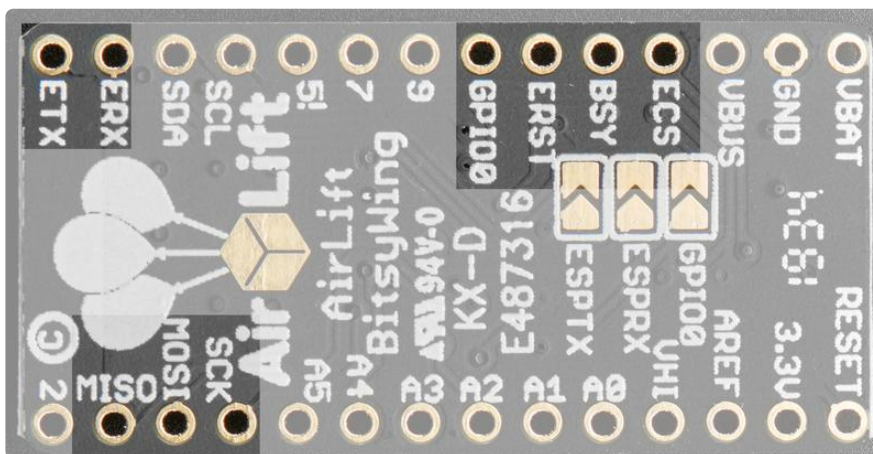
Power Pins



- GND - Common power/logic ground.
- 3.3V - Power supply for the ublox WiFi module. You'll need up to 250mA to power the module. ItsyBitsy's tend to have a 500mA regulator so they should be OK, just make sure you're aware of your power budget

SPI and Control Pins

To keep transfers speedy, we use SPI not UART Serial. UART 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, logic level is 3.3V only
- MISO - SPI Data from the AirLift to the microcontroller, logic level is 3.3V only. 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, logic level is 3.3V only.
- ESPCS - SPI Chip Select from the microcontroller to start sending commands to the AirLift, and to choose BLE mode on reset, logic level is 3.3V only

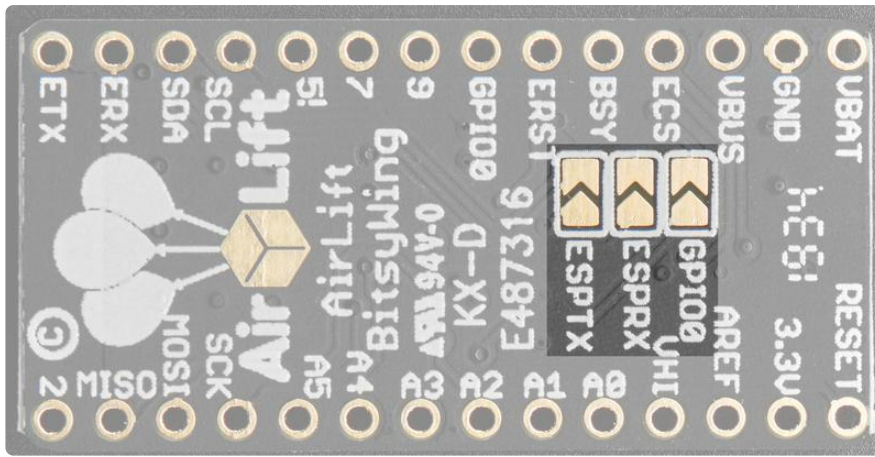
Required Control Pins:

- 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. 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. Logic level is 3.3V

Warning! The initial batch has the silk screen labels for ESPBUSY and ESPRST swapped! The order should be ECS/RST/BSY not ECS/BSY/RST. The schematic, code, and fritzing object have been fixed. The silk will be corrected for the next run

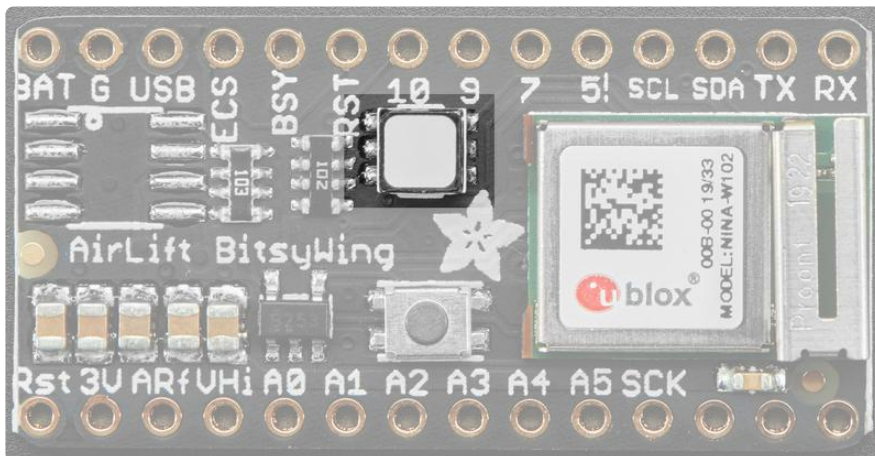
Optional Control Pins:

- ESPGPIO0 - this is the ESP32 GPIO0 pin, which is used to put the WiFi module it into bootloading mode if you want to update the firmware. 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 in WiFi mode, but you'll need to connect it for BLE mode. You'll need to solder the pad on the bottom of the Bitsy Add-on to connect it.
- ESPRX & ESPTX - Serial data in and Serial data out, used for bootloading new firmware, and for communication when in BLE mode. Leave disconnected if not using BLE or when not uploading new WiFi firmware to the AirLift (which is a rare occurrence). You'll need to solder the two pads on the bottom of the Bitsy Add-on to use these pins. Soldering the pads connects ESPTX to RX and ESPTX to RX, respectively.

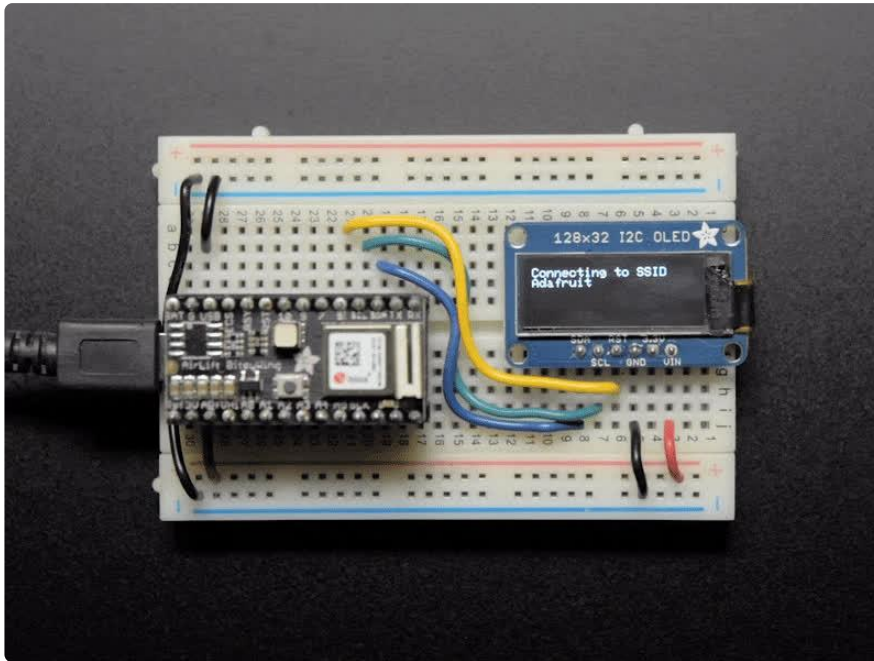


RGB LED

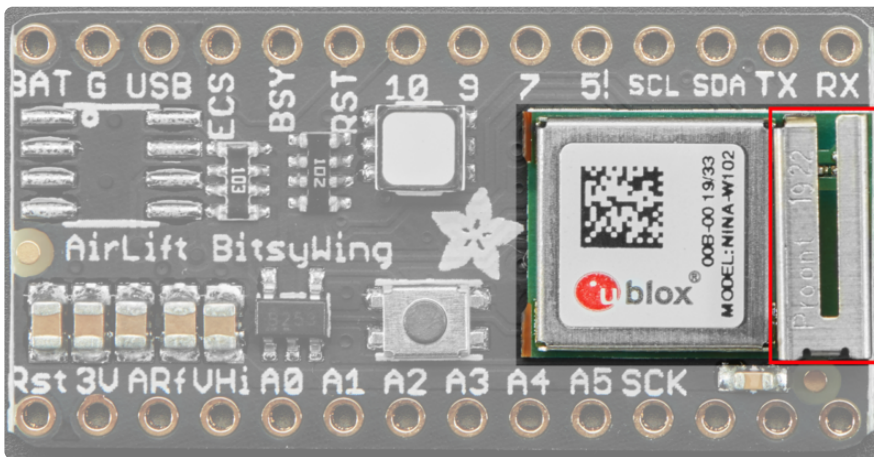
There is a small RGB LED to the left of the ESP32. These RGB LEDs are available in the Arduino and CircuitPython libraries if you'd like to PWM them for a visual alert. They're connected to the ESP32's pins 26 (Red), 25 (Green), and 27 (Blue).

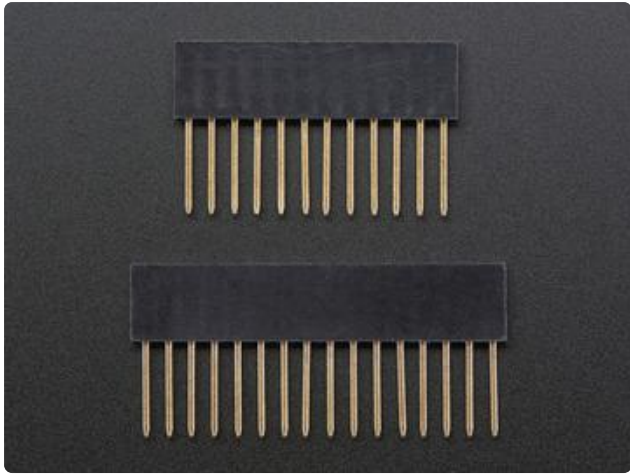


Assembly



The antenna on the new, smaller ESP32 module is very delicate! Avoid touching it and grab the add-on by the long sides to avoid smushing the antenna





To stack the Airlift Add-on on top of the ItsyBitsy, you'll need stacking headers for the ItsyBitsy. For now you'll need to cut down a set of [Feather Stacking Headers](#) () to fit the ItsyBitsy however we will have headers with the correctly length soon!

CircuitPython WiFi

It's easy to use the Adafruit AirLift breakout with CircuitPython and the [Adafruit CircuitPython ESP32SPI](#) () module. This module allows you to easily add WiFi to your project.

The ESP32SPI library requires a microcontroller with ~128KB of RAM or more. The SAMD21 will not work.

CircuitPython Microcontroller Pinout

Since all CircuitPython-running ItsyBitsies 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_reset = DigitalInOut(board.D12)
esp32_ready = DigitalInOut(board.D11)
```

Then, include the following code to use the pin:

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

CircuitPython Setup

First make sure you are running the [latest version of Adafruit CircuitPython](#) () for your board.

Next you'll need to install the necessary libraries to use the hardware. Thankfully, we can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the entire lib folder and the code.py file to your CIRCUITPY drive.

Your CIRCUITPY/lib folder should contain the following folders and files:

- /adafruit_bus_device
- /adafruit_esp32spi
- adafruit_requests.mpy



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_reset = DigitalInOut(board.D12)
esp32_ready = DigitalInOut(board.D11)

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 console \(\)](#) to see the output. It should look something like the following:

```
code.py output:
ESP32 SPI hardware test
ESP32 found and in idle mode
Firmware vers. bytearray(b'1.3.0\x00')
MAC addr: ['0xbd', '0xb0', '0xe', '0x33', '0x4f', '0xc4']
Get scan
  Adafruit          RSSI: -50
  Adafruit          RSSI: -57
  ESP_88EF6C        RSSI: -61
  consulatewireless RSSI: -70
  Adafruit          RSSI: -71
  Consulate Guest   RSSI: -71
  consulatewireless RSSI: -72
  Consulate Guest   RSSI: -73
  consulatewireless RSSI: -74
  ndm-studiopro2-hotspot RSSI: -74
Done!

Press any key to enter the REPL. Use CTRL-D to reload.
```

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. Note that access to enterprise level secured WiFi networks is not currently supported, only WiFi networks that require SSID and password.

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 it's 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 \(\)](http://worldtimeapi.org/timezones) 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

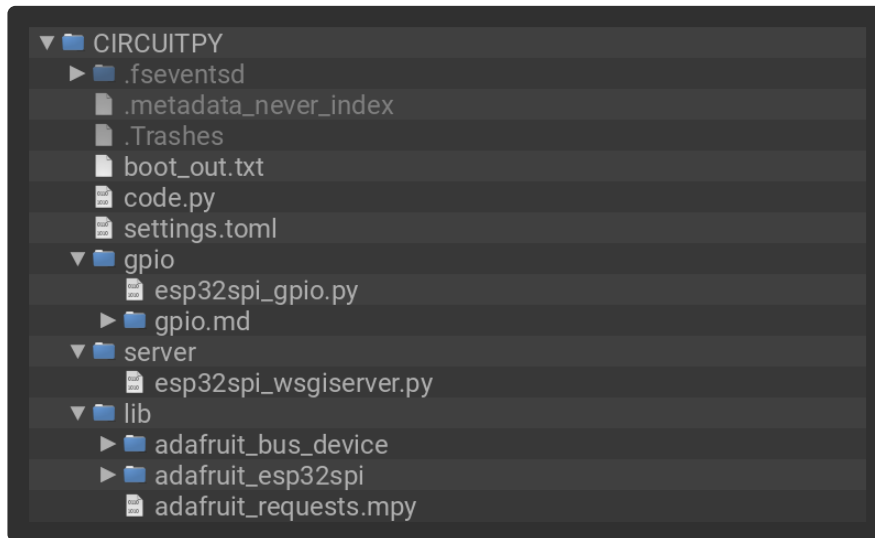
OK now you have your secrets setup - you can connect to the Internet.

To do this, you need to first install a few libraries, into the lib folder on your CIRCUITPY Y drive. Then you need to update code.py with the example script.

Thankfully, we can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the directory examples/ and then

click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

Your CIRCUITPY drive should now look similar to the following image:



```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

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

# 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

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 AirLift Shield:
# esp32_cs = DigitalInOut(board.D10)
# esp32_ready = DigitalInOut(board.D7)
# esp32_reset = DigitalInOut(board.D5)

# If you have an AirLift Featherwing or ItsyBitsy AirLift:
# esp32_cs = DigitalInOut(board.D13)
# esp32_ready = DigitalInOut(board.D11)
# esp32_reset = DigitalInOut(board.D12)

# If you have an externally connected ESP32:
# NOTE: You may need to change the pins to reflect your wiring
```



```

# 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(secrets["ssid"], secrets["password"])
    except OSError 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"))

# 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.

Don't forget you'll also need to create the secrets.py file as seen above, with your WiFi ssid and password.

In a serial console, you should see something like the following. For more information about connecting with a serial console, view the guide [Connecting to the Serial Console \(\)](#).

```

COM61 - PuTTY
ESP32 SPI webclient test
ESP32 found and in idle mode
Firmware vers. bytearray(b'1.2.2\x00')
MAC addr: ['0x1', '0x5c', '0xd', '0x33', '0x4f', '0xc4']
MicroPython-d45f8a          RSSI: -44
adafruit_tw                 RSSI: -63
FIOS-QOGLB                  RSSI: -63
adafruit                     RSSI: -71
AP819                       RSSI: -73
FIOS-K57GI                   RSSI: -74
AP819                       RSSI: -77
linksys_SES_2868            RSSI: -79
linksys_SES_2868            RSSI: -79
FIOS-K57GI                   RSSI: -83
Connecting to AP...
Connected to adafruit       RSSI: -65
My IP address is 10.0.1.54
IP lookup adafruit.com: 104.20.38.240
Ping google.com: 30 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
-----
This is a test of the CC3000 module!
If you can read this, its working :)
-----
Fetching json from http://api.coindesk.com/v1/bpi/currentprice/USD.json
-----
{'time': {'updated': 'Feb 27, 2019 03:11:00 UTC', 'updatedISO': '2019-02-27T03:11:00+00:00', 'updateduk': 'Feb 27, 2019 at 03:11 GMT'}, 'disclaimer': 'This data was produced from the CoinDesk Bitcoin Price Index (USD). Non-USD currency data converted using hourly conversion rate from openexchangerates.org', 'bpi': {'USD': {'code': 'USD', 'description': 'United States Dollar', 'rate_float': 3832.74, 'rate': '3,832.7417'}}}
-----
Done!

```

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)

```

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...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["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"))
```

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 \(\)](#) - 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 \(\)](#) library for web interfacing named [Adafruit_CircuitPython_on_Requests \(\)](#). This library allows you to send HTTP/1.1 requests without "crafting" them and provides helpful methods for parsing the response from the server.

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your CIRCUITPY drive. Then you need to update code.py with the example script.

Thankfully, we can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the directory examples/ and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

Your CIRCUITPY drive should now look similar to the following image:



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

# 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

# Add a secrets.py to your filesystem that has a dictionary called secrets with
"ssid" and
# "password" keys with your WiFi credentials. DO NOT share that file or commit it
into Git or other
# source control.
# pylint: disable=no-name-in-module,wrong-import-order
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)

# If you have an AirLift Featherwing or ItsyBitsy AirLift:
# 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)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["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
socket.set_interface(esp)
requests.set_socket(socket, esp)

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

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 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)
```

HTTP GET with Requests

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

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`.

Having requested data from the server, we'd now like to see 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

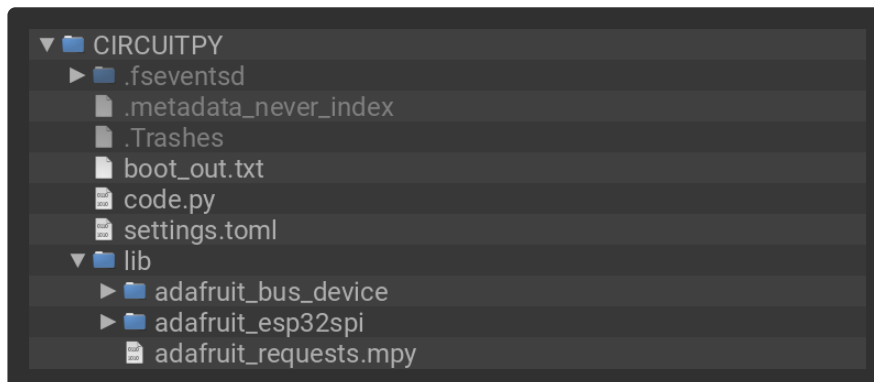
Want to send custom HTTP headers, parse the response as raw bytes, or handle a response's http status code in your CircuitPython code?

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

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your CIRCUITPY drive. Then you need to update code.py with the example script.

Thankfully, we can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the directory examples/ and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

Your CIRCUITPY drive should now look similar to the following image:



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

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

# Add a secrets.py to your filesystem that has a dictionary called secrets with
"ssid" and
# "password" keys with your WiFi credentials. DO NOT share that file or commit it
into Git or other
```



```

# source control.
# pylint: disable=no-name-in-module,wrong-import-order
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)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["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
socket.set_interface(esp)
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)

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

```

WiFi Manager

That simplest example works but it's 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:

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your CIRCUITPY drive. Then you need to update code.py with the example script.

Thankfully, we can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the directory examples/ and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

Your CIRCUITPY drive should now look similar to the following image:



```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

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 OSError as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        continue
    response = None
    time.sleep(15)

```

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:

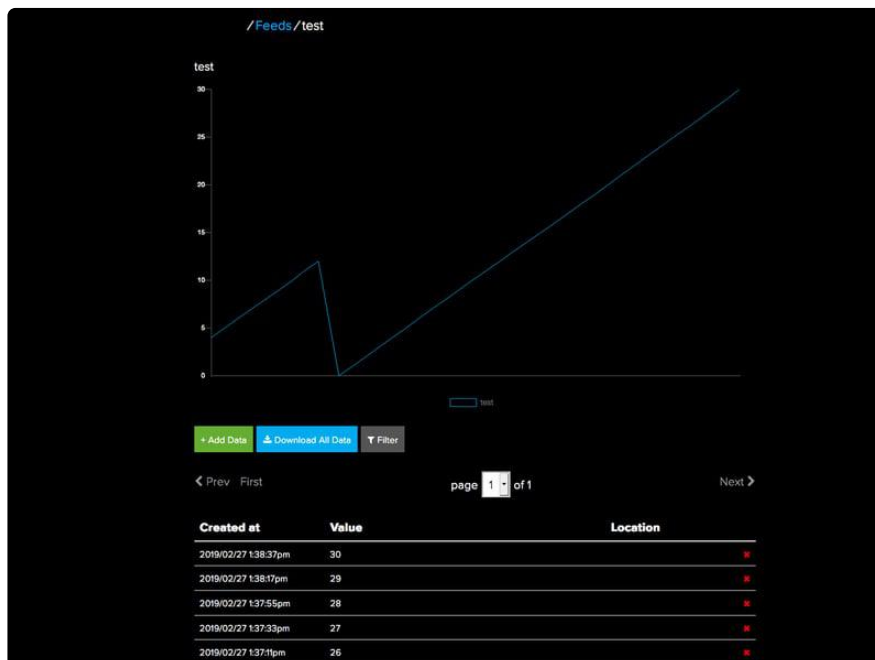
```
# This file is where you keep secret settings, passwords, and tokens!  
# If you put them in the code you risk committing that info or sharing it  
  
secrets = {  
    'ssid' : '_your_ssid_',  
    'password' : '_your_wifi_password_',  
    'timezone' : "America/Los_Angeles", # http://worldtimeapi.org/timezones  
    'aio_username' : '_your_aio_username_',  
    'aio_key' : '_your_aio_key_',  
}
```

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**.](#) ()

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!



OLD - CircuitPython WiFi

It's easy to use the Adafruit AirLift breakout with CircuitPython and the [Adafruit CircuitPython ESP32SPI \(\)](#) module. This module allows you to easily add WiFi to your project.

You need plenty of RAM for CircuitPython/requests/json, so the ItsyBitsy M0 will not work! Please use an Itsy with at least 100KB of RAM, like the ItsyBitsy M4!

CircuitPython Microcontroller Pinout

Since all CircuitPython-running ItsyBitsies 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_reset = DigitalInOut(board.D12)
esp32_ready = DigitalInOut(board.D11)
```

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 \(\)](#) library on your CircuitPython board.

First make sure you are running the [latest version of Adafruit CircuitPython \(\)](#) 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 \(\)](#). Our CircuitPython starter guide has [a great page on how to install the library bundle \(\)](#).

You can manually install the necessary libraries from the bundle:

- adafruit_esp32spi
- adafruit_bus_device

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

Next [make sure you are set up to connect to the serial console \(\)](#)

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_reset = DigitalInOut(board.D12)
esp32_ready = DigitalInOut(board.D11)

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 console \(\)](#) to see the output. It should look something like the following:

```
code.py output:
ESP32 SPI hardware test
ESP32 found and in idle mode
Firmware vers. bytearray(b'1.3.0\x00')
MAC addr: ['0xbd', '0xb0', '0xe', '0x33', '0x4f', '0xc4']
Get scan
  Adafruit          RSSI: -50
  Adafruit          RSSI: -57
  ESP_88EF6C       RSSI: -61
  consulatewireless RSSI: -70
  Adafruit          RSSI: -71
  Consulate Guest  RSSI: -71
  consulatewireless RSSI: -72
  Consulate Guest  RSSI: -73
  consulatewireless RSSI: -74
  ndm-studiompro2-hotspot RSSI: -74
Done!

Press any key to enter the REPL. Use CTRL-D to reload. █
```

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

CircuitPython BLE

CircuitPython BLE UART Example

It's easy to use Adafruit AirLift ESP32 co-processor boards for Bluetooth Low Energy (BLE) with CircuitPython. When you reset the ESP32, you can put it in WiFi mode (the default), or in BLE mode; you cannot use both modes simultaneously.

Here's a simple example of using BLE to connect CircuitPython with the Bluefruit Connect app. Use CircuitPython 6.0.0 or later.

Note: Don't confuse the ESP32 with the ESP32-S2, which is a different module with a similar name. The ESP32-S2 does not support BLE.

Currently, AirLift BLE support is not currently available on boards with Espressif chips. If the Espressif board provides `_bleio`, it is for native BLE support (e.g. ESP32-S3), not AirLift.

Currently the AirLift support for CircuitPython only provides BLE peripheral support. BLE central is under development. So you cannot connect to BLE devices like Heart Rate monitors, etc., but you can act as a BLE peripheral yourself.

Adafruit Airlift Bitsy ESP32 Add-On Wiring

If you have an Adafruit Airlift Bitsy ESP32 Add-On, you will need to solder three jumpers closed on the bottom side of the board to enable BLE. The rest of the ESP32 pins you need are already jumpered to certain ItsyBitsy pins.

Update the AirLift Firmware

You will need to update the AirLift's firmware to at least version 1.7.1. Previous versions of the AirLift firmware do not support BLE.

Follow the instructions in the guide below, and come back to this page when you've upgraded the AirLift's firmware:

Upgrade External ESP32 AirLift
Firmware

Ensure the AirLift firmware is version 1.7.1 or higher for BLE to work.

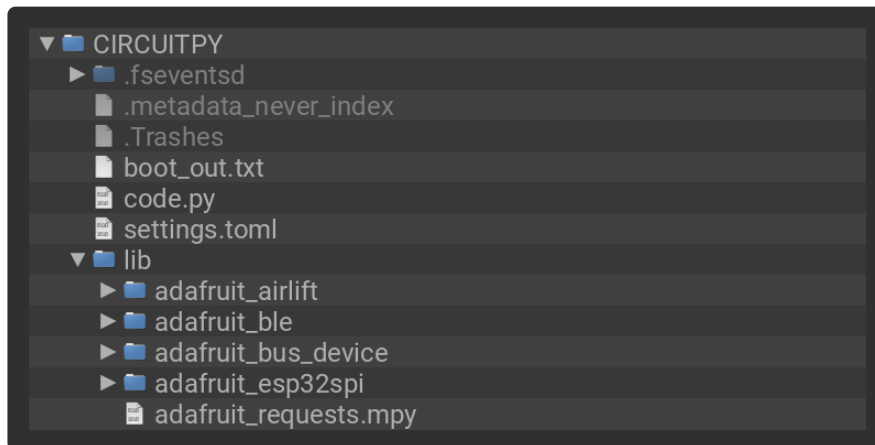
Install CircuitPython Libraries

First make sure you are running the [latest version of Adafruit CircuitPython \(\)](#) for your board.

Next you'll need to install the necessary libraries to use the hardware. Thankfully, we can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the entire lib folder and the code.py file to your CIRCUITPYTHON drive.

Your CIRCUITPY/lib folder should contain the following folders and files:

- /adafruit_airlift
- /adafruit_ble
- /adafruit_bus_device
- /adafruit_esp32spi
- adafruit_requests.mpy



Install the Adafruit Bluefruit LE Connect App

The Adafruit Bluefruit LE Connect iOS and Android apps allow you to connect to BLE peripherals that provide a over-the-air "UART" service. Follow the instructions in the [Bluefruit LE Connect Guide \(\)](#) to download and install the app on your phone or tablet.

BLE Example

TAKE NOTE: Adjust the program as needed to suit the AirLift board you have. Comment and uncomment lines 19-55 below as necessary.

```
# SPDX-FileCopyrightText: 2020 Dan Halbert, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

# pylint: disable=unused-import
import board
import busio
from digitalio import DigitalInOut
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_airlift.esp32 import ESP32

# If you are using a Metro M4 Airlift Lite, PyPortal,
# or MatrixPortal, you can use the default pin settings.
```

```

# Leave this DEFAULT line uncommented.
# If you are using a board with pre-defined ESP32 Pins:
esp32 = ESP32()

# If you are using a Metro M7 **OR**
# if you are using CircuitPython 6.0.0 or earlier,
# on PyPortal and PyPortal Titano only, use the pin settings
# below. Comment out the DEFAULT line above and uncomment
# the line below. For CircuitPython 6.1.0, the pin names
# have changed for these boards, and the DEFAULT line
# above is correct.
# esp32 = ESP32(tx=board.TX, rx=board.RX)

# If you are using an AirLift FeatherWing or AirLift Bitsy Add-On,
# use the pin settings below. Comment out the DEFAULT line above
# and uncomment the lines below.
# If you are using an AirLift Breakout, check that these
# choices match the wiring to your microcontroller board,
# or change them as appropriate.
# esp32 = ESP32(
#     reset=board.D12,
#     gpio0=board.D10,
#     busy=board.D11,
#     chip_select=board.D13,
#     tx=board.TX,
#     rx=board.RX,
# )

# If you are using an AirLift Shield,
# use the pin settings below. Comment out the DEFAULT line above
# and uncomment the lines below.
# esp32 = ESP32(
#     reset=board.D5,
#     gpio0=board.D6,
#     busy=board.D7,
#     chip_select=board.D10,
#     tx=board.TX,
#     rx=board.RX,
# )

adapter = esp32.start_bluetooth()

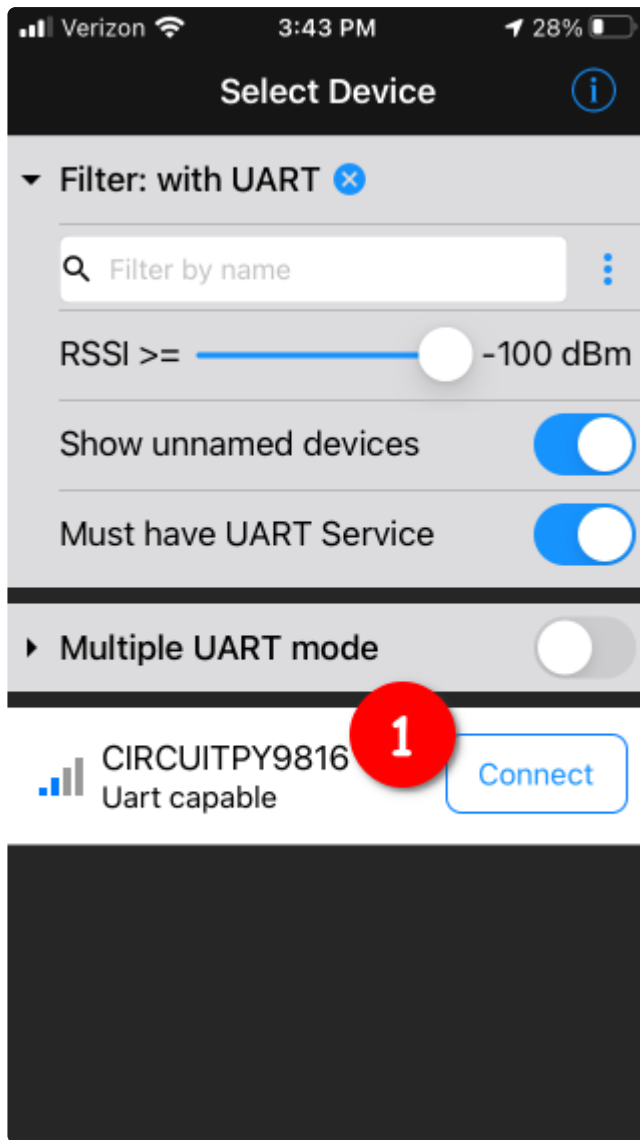
ble = BLERadio(adapter)
uart = UARTService()
advertisement = ProvideServicesAdvertisement(uart)

while True:
    ble.start_advertising(advertisement)
    print("waiting to connect")
    while not ble.connected:
        pass
    print("connected: trying to read input")
    while ble.connected:
        # Returns b'' if nothing was read.
        one_byte = uart.read(1)
        if one_byte:
            print(one_byte)
            uart.write(one_byte)

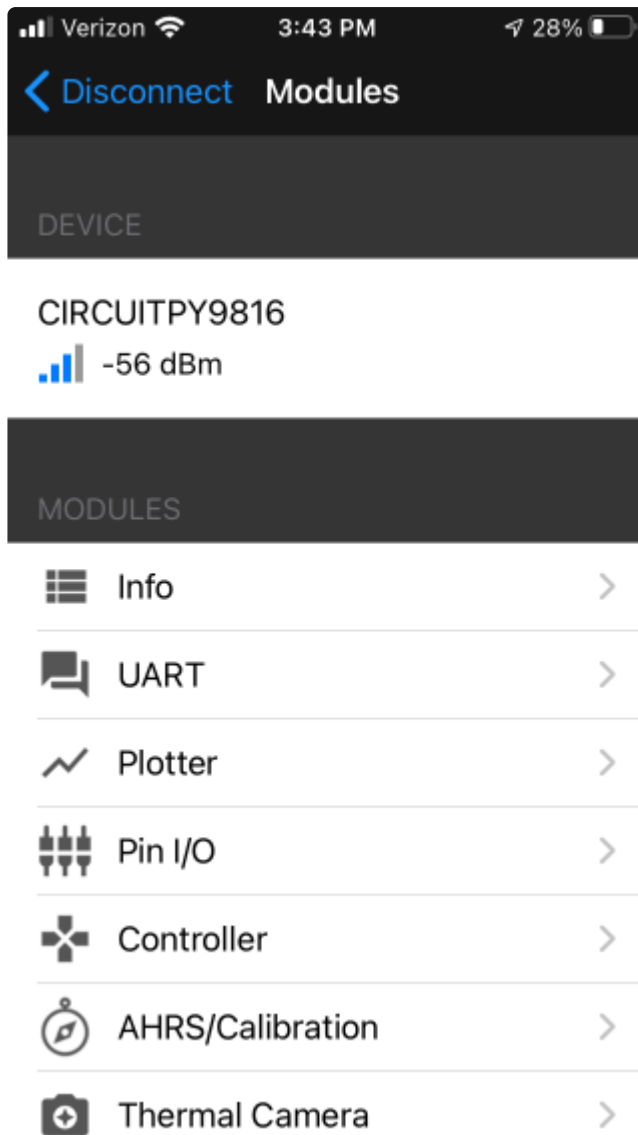
```

Talk to the AirLift via the Bluefruit LE Connect App

Start the Bluefruit LE Connect App on your phone or tablet. You should see a CIRCUITPY device available to connect to. Tap the Connect button (1):



You'll then see a list of Bluefruit Connect functions ("modules"). Choose the UART module (2):



On the UART module page, you can type a string and press Send (3). You'll see that string entered, and then see it echoed back (echoing is in gray).



Arduino WiFi

You can use the AirLift with Arduino. Unlike CircuitPython, it work work with most of the Arduino compatible ItsyBitsies, even the 3V ItsyBitsy 32u4. However, if you want to use libraries like ArduinoJSON or add sensors and SD card, you'll really want an ATSAM21 (Cortex M0) or ATSAM51 (Cortex M4), both of which have plenty or RAM

Library Install

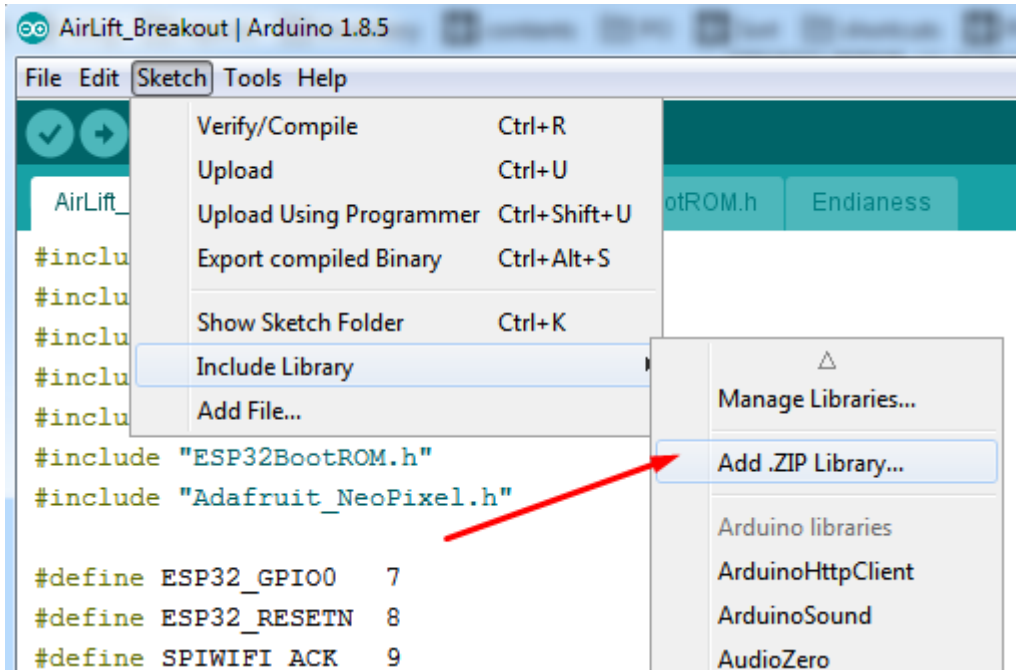
We're using a variant of the Arduino WiFiNINA library, which is amazing and written by the Arduino team! The official WiFi101 library won't work because it doesn't support the ability to change the pins.

So! We made a fork that you can install.

Click here to download the library:

Download Adafruit's version of WiFiNINA

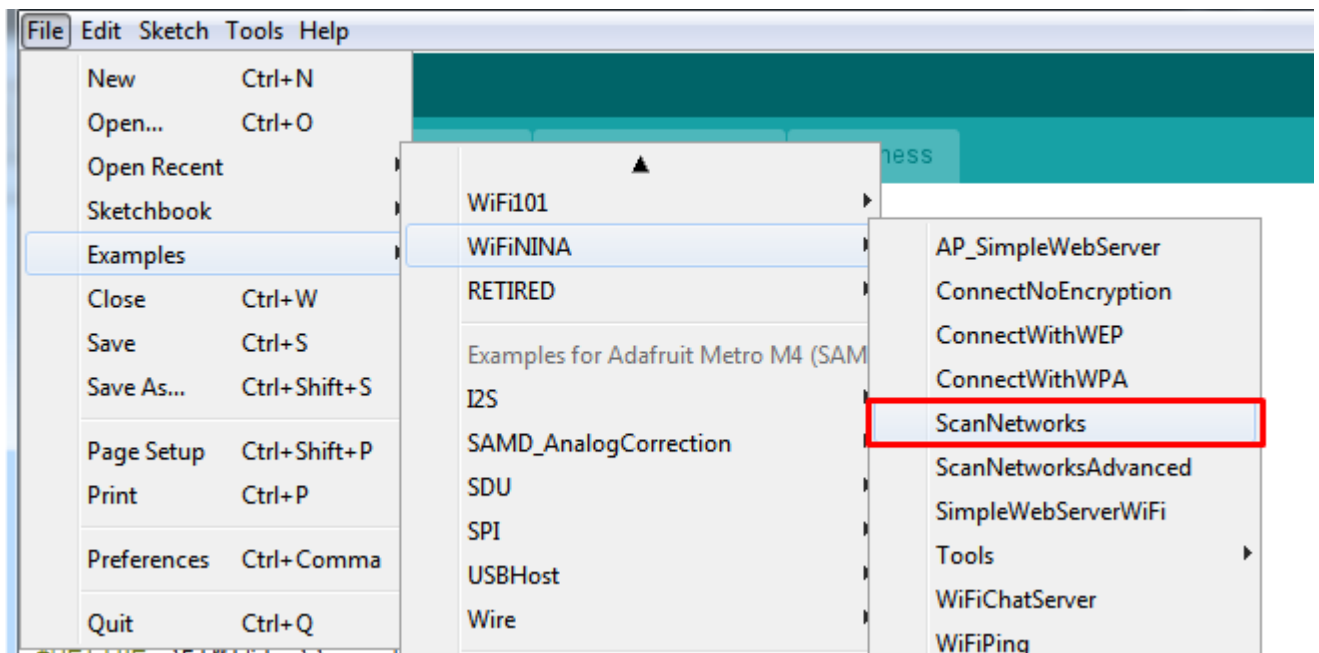
Within the Arduino IDE, select Install library from ZIP...



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



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
  
```

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
```

()

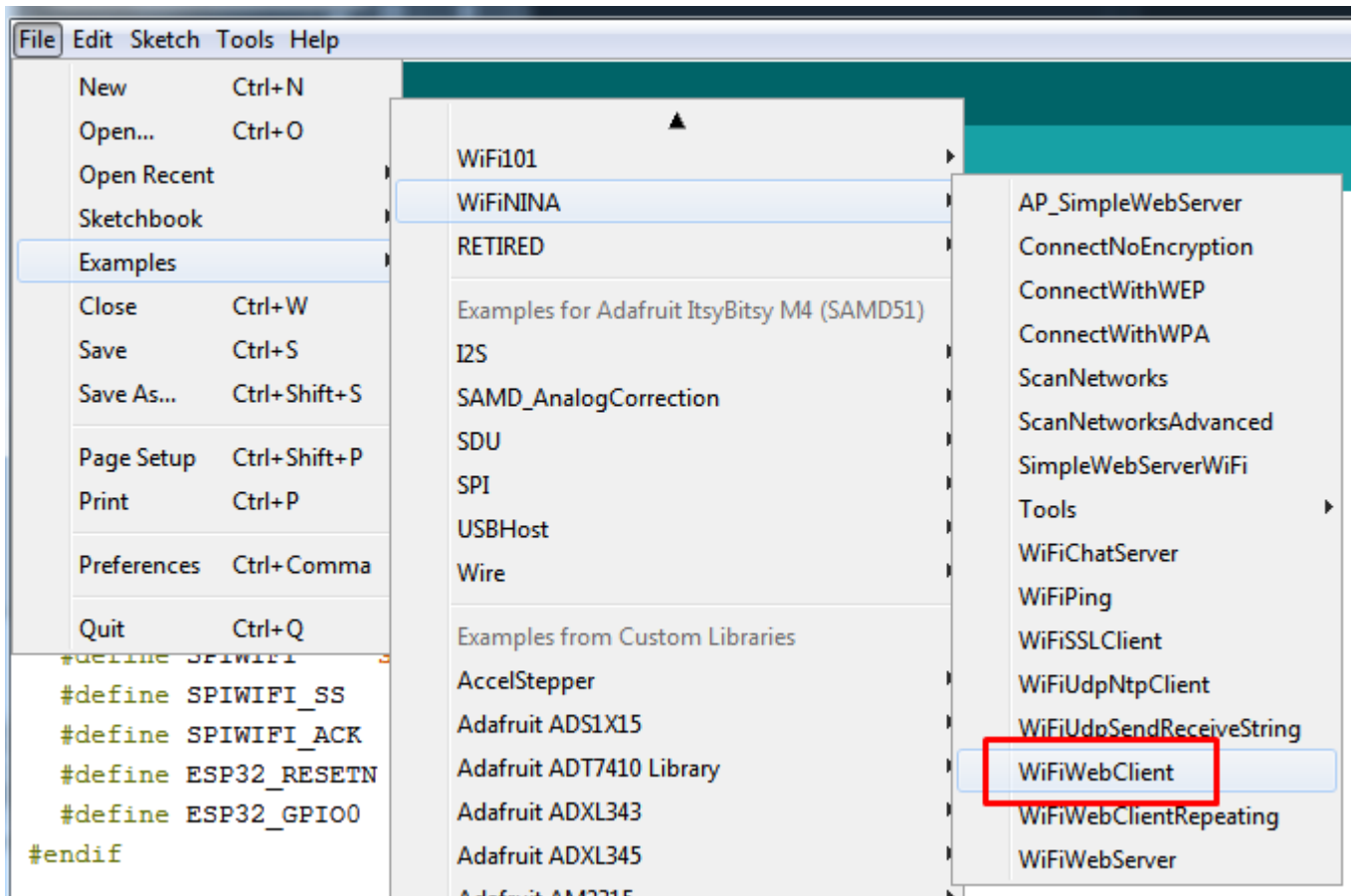
If you don't even get the MAC address printed out, check your wiring.

If you get the MAC address but cannot scan any networks, check your power supply. You need a solid 3.3V into Vin in order for the ESP32 not to brown out.

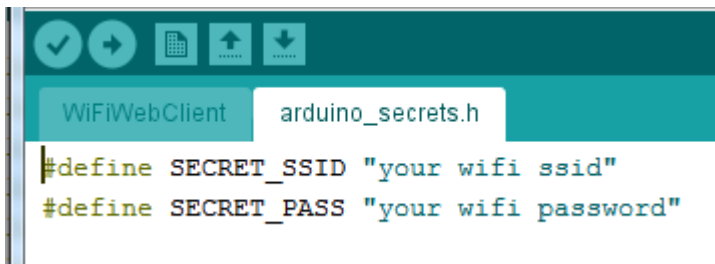
WiFi Connection Test

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

Open up the WiFiWebClient example



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



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

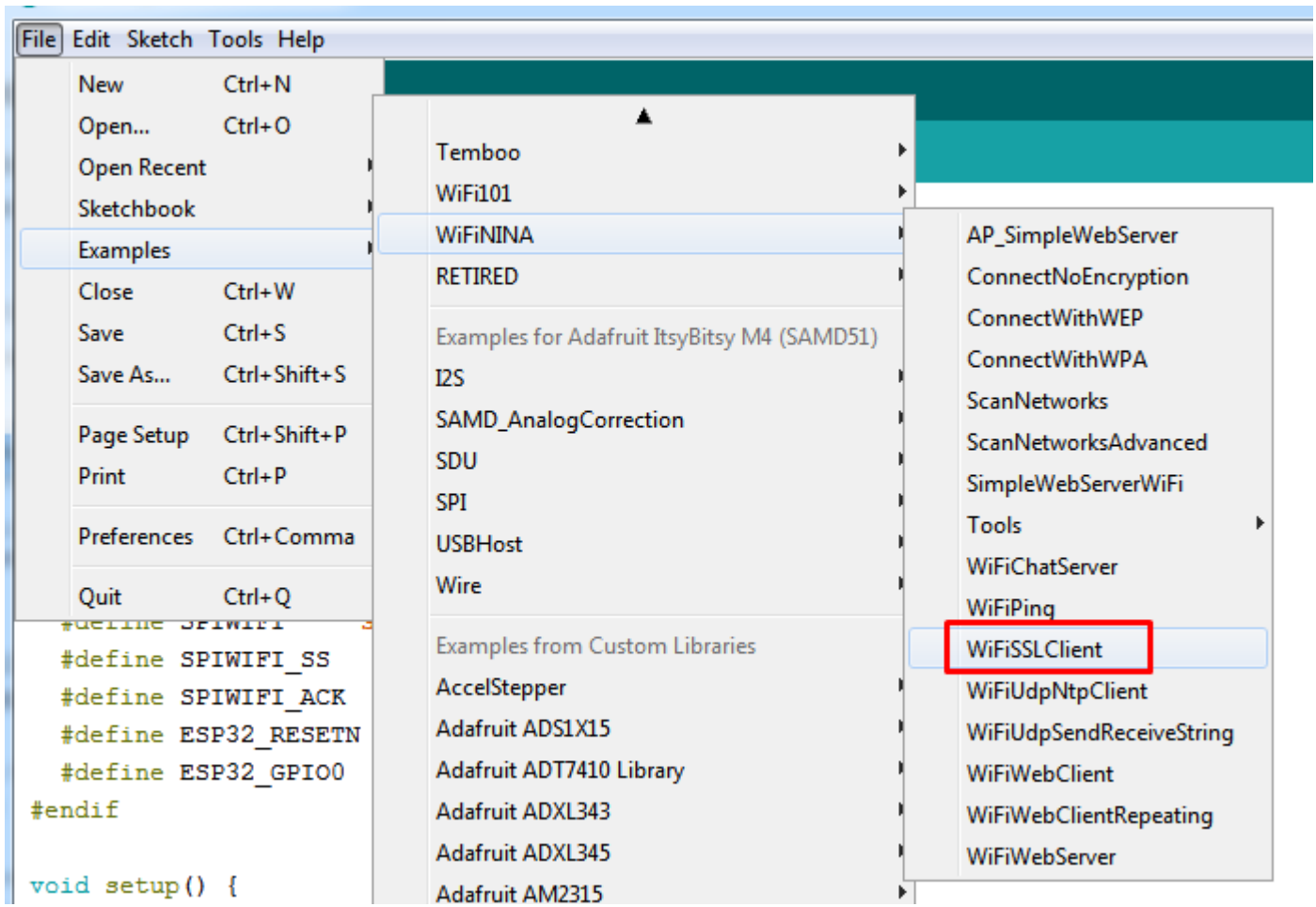
Starting connection to server...
connected to server
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Wed, 10 Apr 2019 20:55:51 GMT
Content-Type: text/html
Content-Length: 73
Last-Modified: Thu, 16 Feb 2017 17:42:29 GMT
Connection: close
ETag: "58a5e485-49"
Accept-Ranges: bytes

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

()

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:



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

```
COM161 (Adafruit ItsyBitsy M4 (SAMD51))

Attempting to connect to SSID: Adafruit
Connected to wifi
SSID: Adafruit
IP Address: 10.0.1.179
signal strength (RSSI):-52 dBm

Starting connection to server...
connected to server
HTTP/1.1 200 OK
cache-control: must-revalidate, max-age=600
content-disposition: attachment; filename=json.json
content-type: application/json;charset=utf-8
expires: Wed, 10 Apr 2019 21:17:24 GMT
last-modified: Wed, 10 Apr 2019 21:07:24 GMT
strict-transport-security: max-age=631138519
timing-Allow-Origin: *
x-connection-hash: ab527136393fa0f3bb7779f53c657fae
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-response-time: 12
x-xss-protection: 1; mode=block; report=https://twitter.com/i/xss_report
Content-Length: 197
Accept-Ranges: bytes
Date: Wed, 10 Apr 2019 21:07:24 GMT
Via: 1.1 varnish
Age: 0
Connection: close
X-Served-By: cache-bwi5023-BWI
X-Cache: MISS
X-Timer: S1554930445.534696,VS0,VE25
Vary: Accept-Encoding

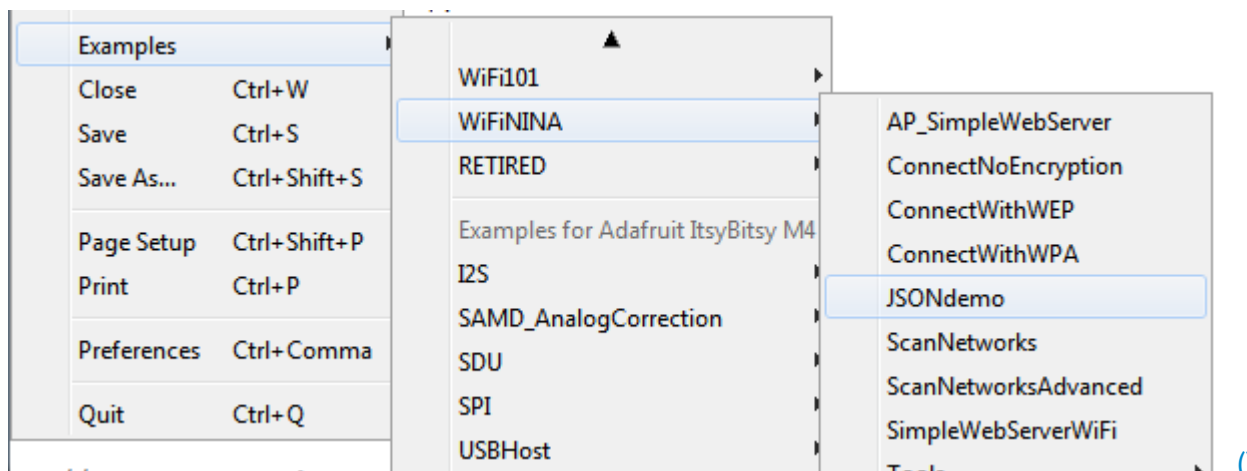
[{"following":false,"id":"20731304","screen_name":"adafruit","name":"adafruit industries"}]
disconnecting from server.
Read 959 bytes
```

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 \(\)](#) 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 \(\)](#).

Then load the example JSONdemo



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
```

Adapting Other Examples

Once you've got it connecting to the Internet you can check out the other examples. Most of the sketches included with the Adafruit fork of the WiFiNINA library above will

have the pin mapping done for the ItsyBitsies. For other examples the only change you'll want to make is at the top of the sketches, add:

```
#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_GPI00  -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 Bitsy Add-on and set **#define ESP32_GPI00** to the correct pin for your microcontroller. For the ItsyBitsies it will be D10/10

And then before you check the `status()` of the module, call the function `WiFi.setPins(SPIWIFI_SS, SPIWIFI_ACK, ESP32_RESETN, ESP32_GPI00, &SPIWIFI);`

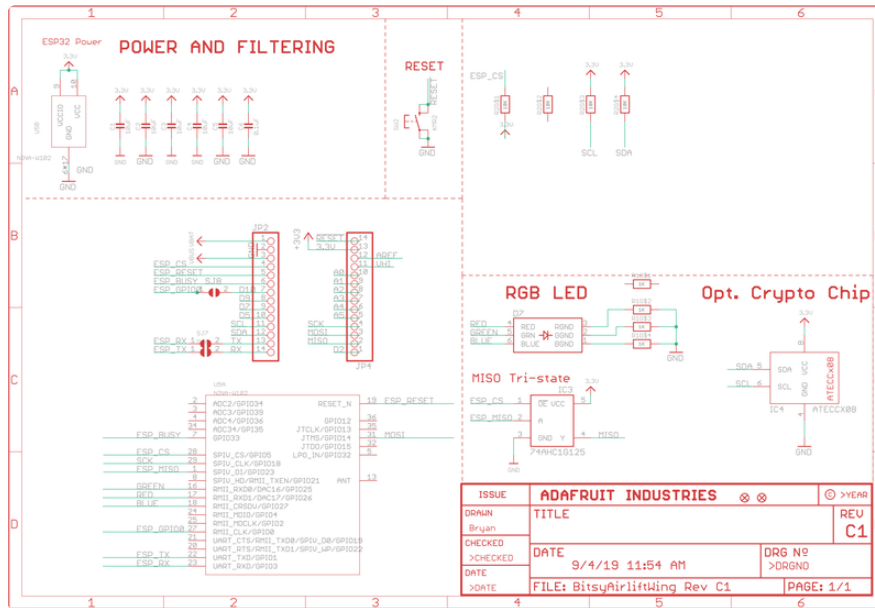
```
// check for the WiFi module:
WiFi.setPins(SPIWIFI_SS, SPIWIFI_ACK, ESP32_RESETN, ESP32_GPI00, &SPIWIFI);
while (WiFi.status() == WL_NO_MODULE) {
  Serial.println("Communication with WiFi module failed!");
  // don't continue
  delay(1000);
}
```

Downloads

Files

- [NINA-W102 ESP32 Module Datasheet \(\)](#)
- [EagleCAD files on GitHub \(\)](#)
- [Fritzing object in Adafruit Fritzing Library \(\)](#)

Schematic



Fab Print

