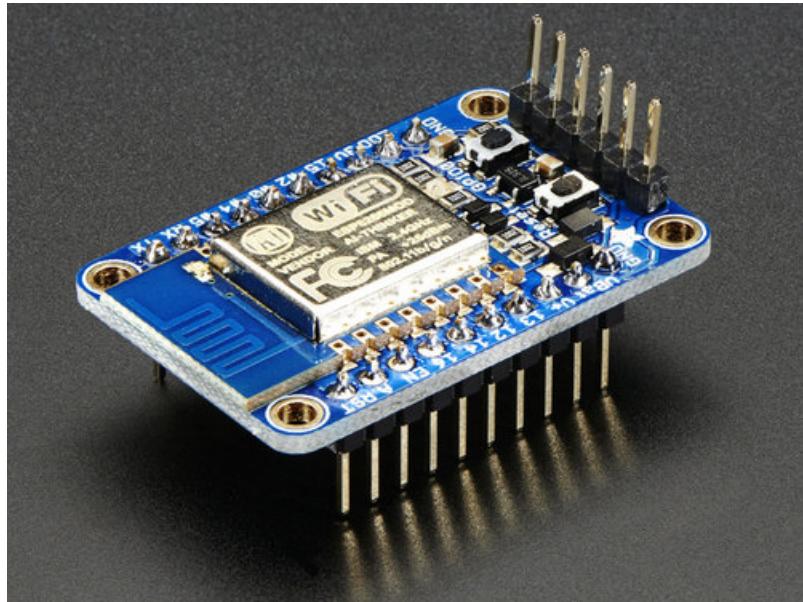


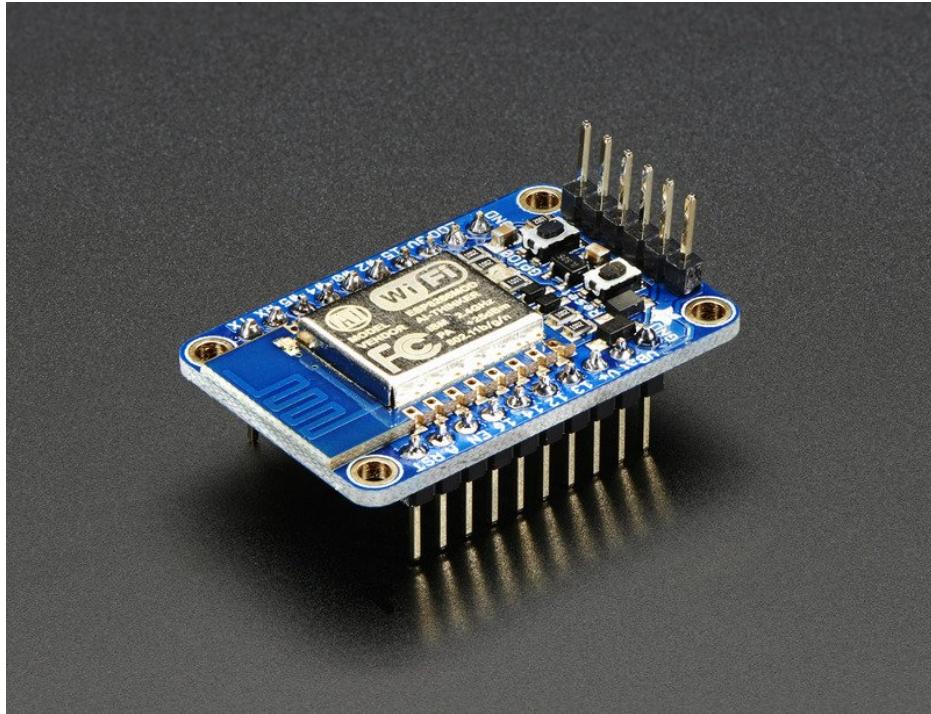
Adafruit HUZZAH ESP8266 breakout

Created by lady ada

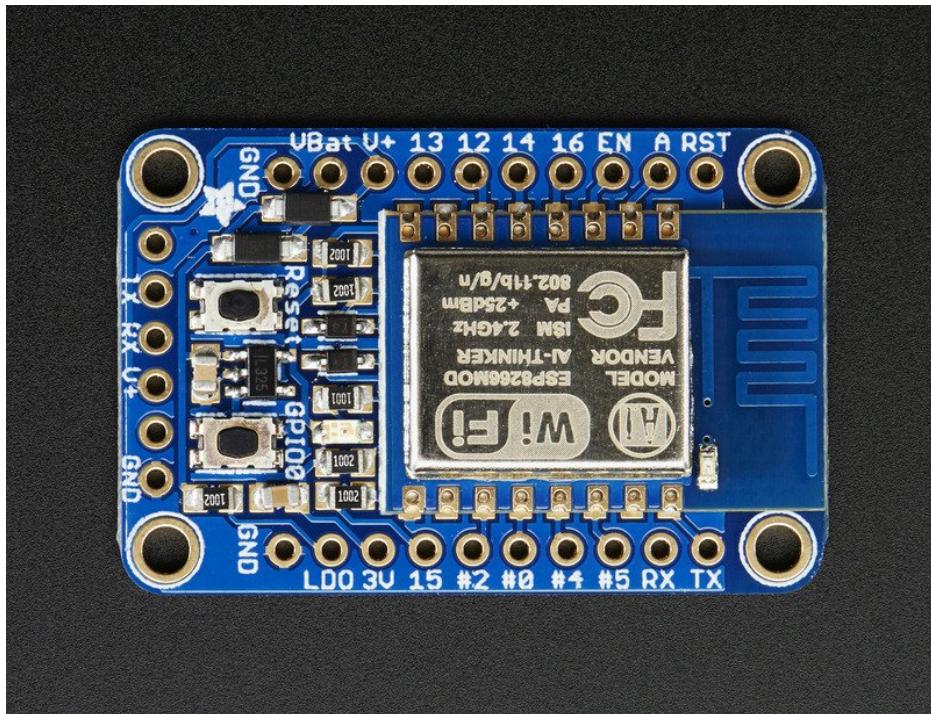


Last updated on 2019-10-21 03:46:45 PM UTC

Overview



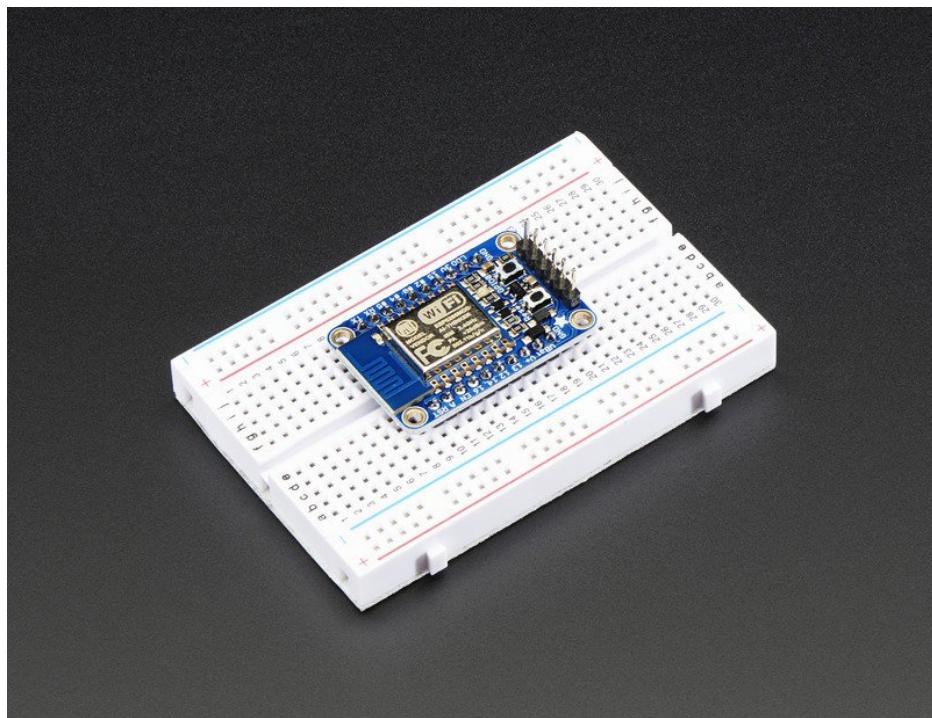
Add Internet to your next project with an adorable, bite-sized WiFi microcontroller, at a price you like! The ESP8266 processor from Espressif is an 80 MHz microcontroller with a full WiFi front-end (both as client and access point) and TCP/IP stack with DNS support as well. While this chip has been very popular, it's also been very difficult to use. Most of the low cost modules are not breadboard friendly, don't have an onboard 500mA 3.3V regulator or level shifting, and aren't CE or FCC emitter certified....**UNTIL NOW!**



The HUZZAH ESP8266 breakout is what we designed to make working with this chip super easy and a lot of fun. We took a certified module with an onboard antenna, and plenty of pins, and soldered it onto our designed breakout PCBs. We added in:

- Reset button
- User button that can also put the chip into bootloading mode,
- Red LED you can blink
- Level shifting on the UART and reset pin
- 3.3V out, 500mA regulator (you'll want to assume the ESP8266 can draw up to 250mA so budget accordingly)
- Two diode-protected power inputs (one for a USB cable, another for a battery)

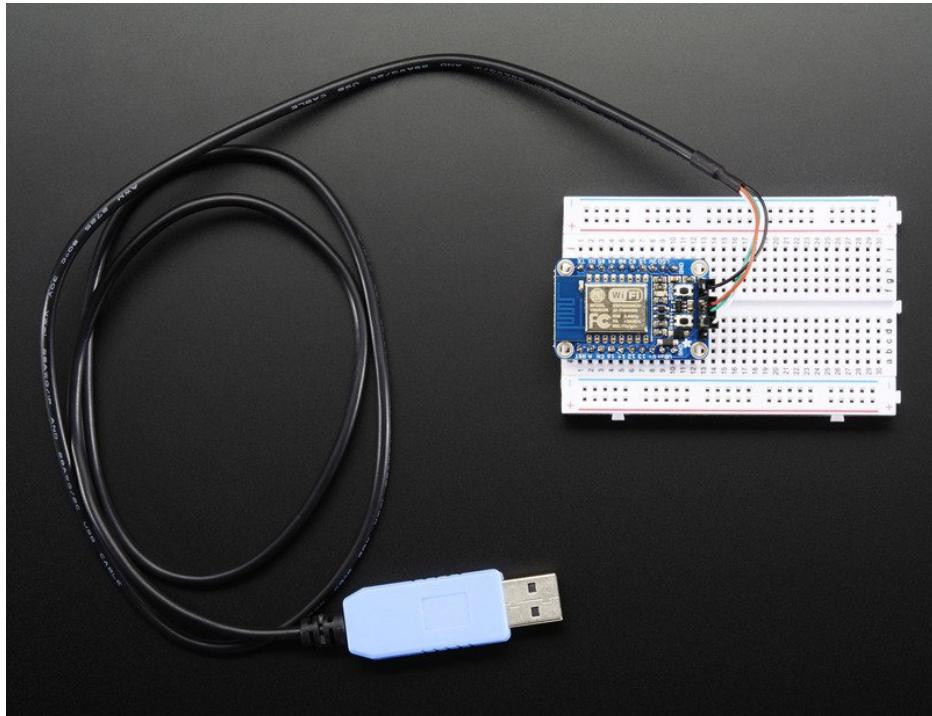
The module itself is FCC certified and contains the ESP8266 chip with 64 KiB of instruction RAM, 96 KiB of data RAM, and 4 MB of QIO FLASH (32 Megabits)



Two parallel, breadboard-friendly breakouts on either side give you access to:

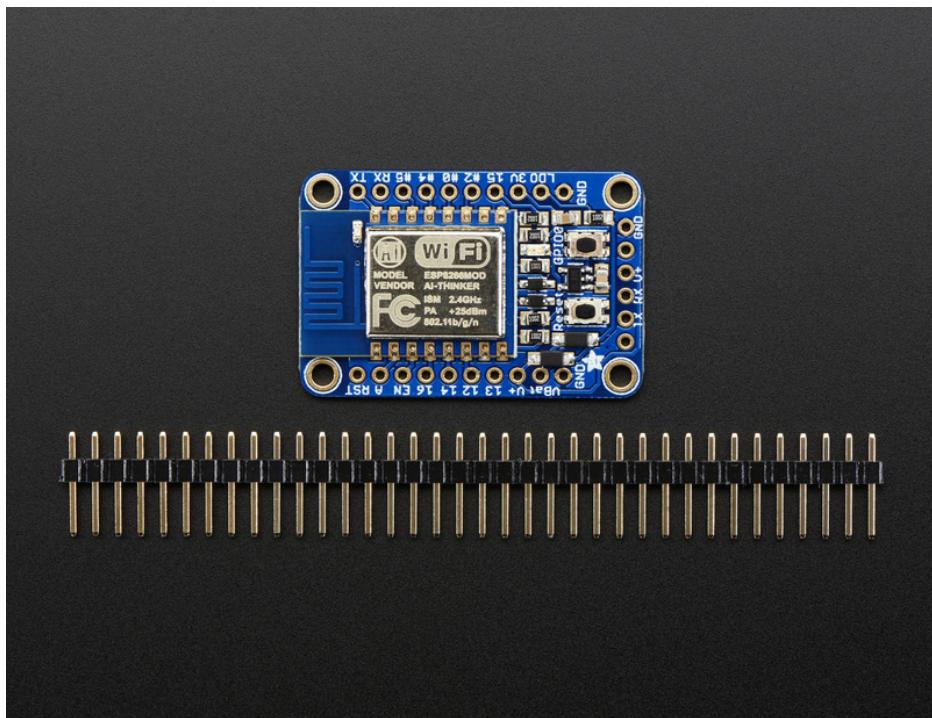
- 1 x Analog input (1.0V max)
- 9 x GPIO (3.3V logic), which can also be used for I2C or SPI
- 2 x UART pins
- 2 x 3-6V power inputs, reset, enable, LDO-disable, 3.3V output

One breakout at the end has an "FTDI" pinout so you can plug in an FTDI or console cable to upload software and read/write debugging information via the UART. When you're done with your coding, remove the cable, and this little module can be embedded into your project box.



Each module comes pre-loaded with NodeMCU's Lua interpreter (NodeMCU 0.9.5 build 20150318 / Lua 5.1.4 to be specific) (<https://adafru.it/inA>), you can run commands, and 'save' Lua programs directly to the module's Flash using a USB-Serial converter cable.

But, if you'd like, you can skip Lua and go direct to using the Arduino IDE. Once you download the ESP8266 core, you can treat it just like a microcontroller+WiFi board, no other processors needed (<https://adafruit.it/inB>)!



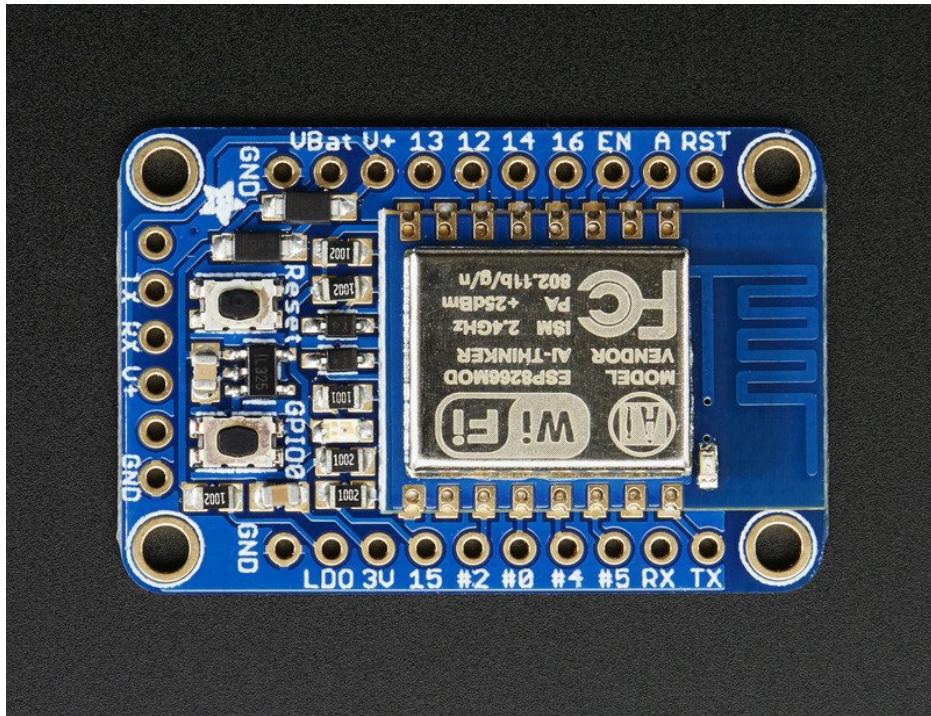
Each order comes with one assembled and tested HUZZAH ESP8266 breakout board, and a stick of 0.1" header that you can solder on and plug the breakout into a breadboard. A soldering iron and solder is required for that, and aren't

included. You'll also need a USB-serial cable such as a [USB console cable \(Windows only\)](https://adafru.it/dDd) (<https://adafru.it/dDd>), [FTDI Friend \(any OS\)](http://adafru.it/284) (<http://adafru.it/284>), or [FTDI cable \(any OS\)](http://adafru.it/70) (<http://adafru.it/70>) to upload software to the HUZZAH ESP8266!

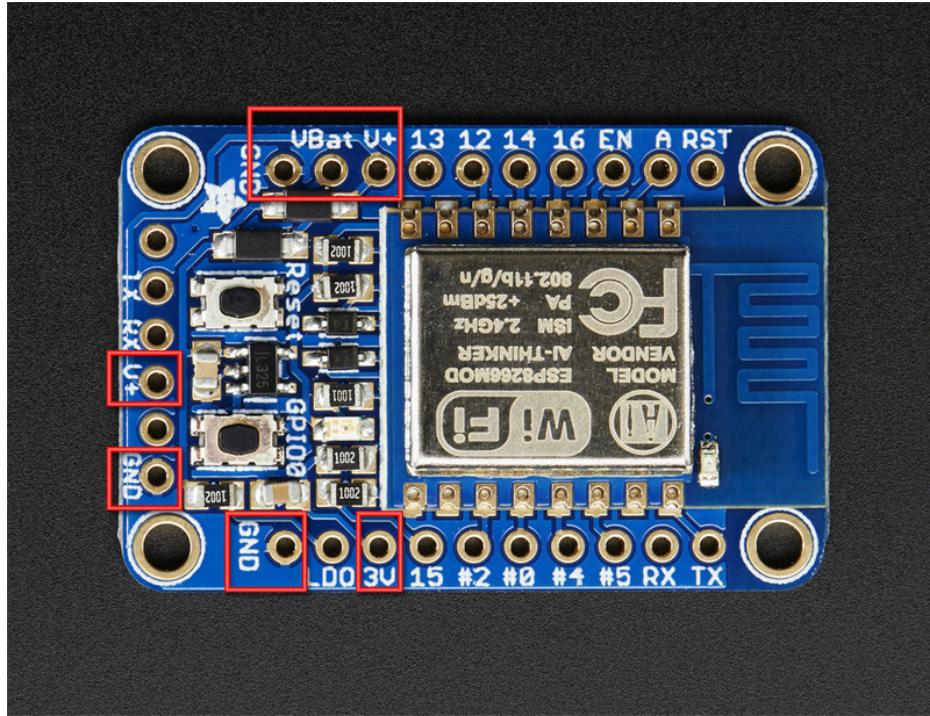
Don't forget to visit esp8266.com for the latest and greatest in ESP8266 news, software and gossip! (<https://adafru.it/f1F>)

Pinouts

- The ESP8266 runs on 3.3V power and logic, and unless otherwise specified, GPIO pins are not 5V safe! The analog pin is also 1.0V max!



This ESP8266 breakout has a ton of pins available, compared to the mini ESP-01 module. When programming the breakout in Lua or via the Arduino IDE, you can control these I/O pins to light up LEDs, read buttons, talk to sensors etc. There's also a bunch of pins for power and control.



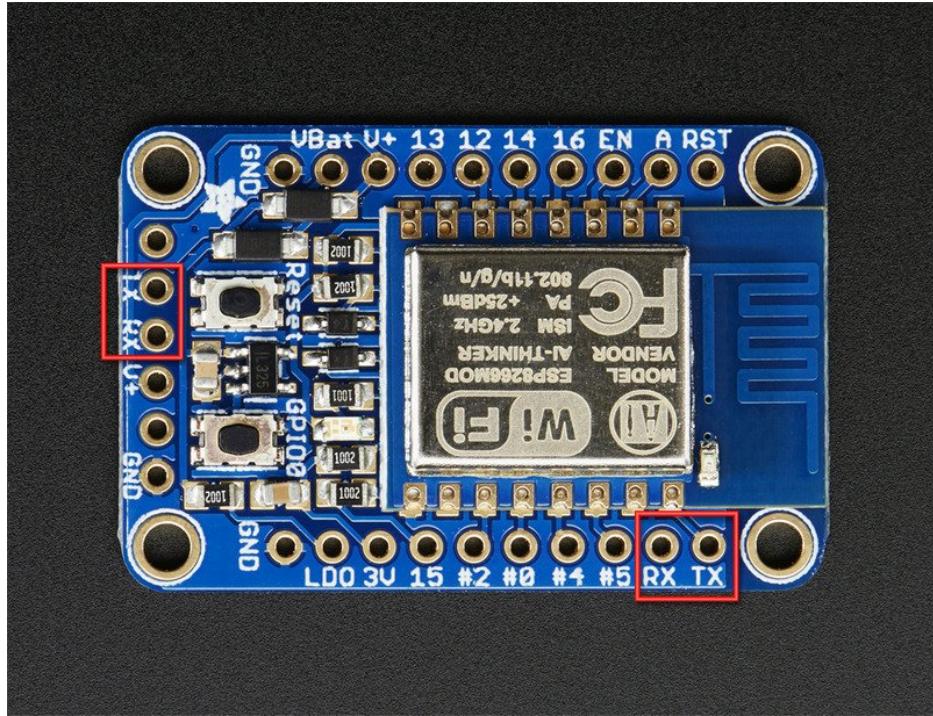
Power Pins

The ESP8266 requires 3.3V power voltage and peaks at 500mA or so of current for small periods of time. You'll want to assume the ESP8266 can draw up to 250mA so budget accordingly. To make it easier to power, we put a high-current-capable 3.3V voltage regulator on the board. It can take 3.4-6V in but you should stick to 4-6V since the ESP8288 has high current usage when WiFi is on.

There are two inputs for the regulator, **V+** and **VBat**. Both have Schottky diodes so you can connect both at different voltages and the regulator will simply power from the higher voltage. The **V+** pin is also on the FTDI/serial header at the bottom edge.

We recommend connecting your LiPoly or AA battery pack directly to **VBat** and then keeping **V+** unused for when an FTDI cable is attached. You can also use the extra **V+** pin for getting the 5V USB power from the FTDI connector for servos, NeoPixels, etc!

There's also a 3.3V output from the regulator available on the **3V** pin



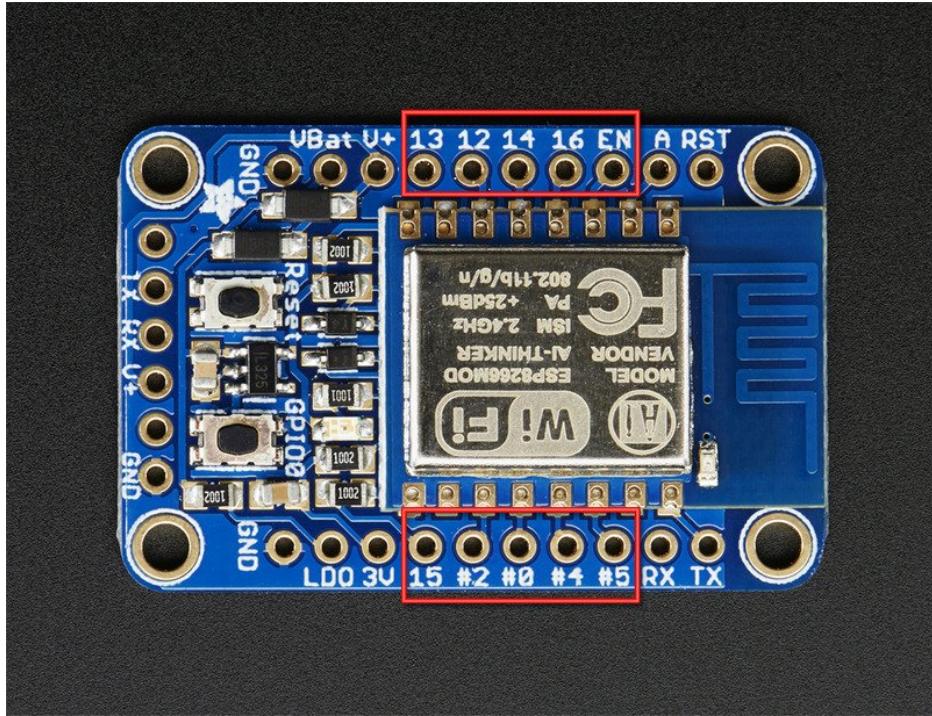
Serial pins

RX and **TX** are the serial control and bootloading pins, and are how you will spend most of your time communicating with the ESP module.

The **TX** pin is the output *from* the module and is 3.3V logic.

The **RX** pin is the input *into* the module and is 5V compliant (there is a level shifter on this pin)

The pins are available in two places, one set is on the right side breakout. The same pins are also at the bottom on the "FTDI/Serial" breakout



GPIO pins

This breakout has 9 GPIO: #0, #2, #4, #5, #12, #13, #14, #15, #16 all GPIO are 3.3V logic level in and out, and are **not 5V compatible**. Read the [full spec sheet](https://adafruit.it/f1E) (<https://adafruit.it/f1E>) to learn more about the GPIO pin limits, but be aware the maximum current drawn per pin is **12mA**.

These pins are general purpose and can be used for any sort of input or output. Most also have the ability to turn on an internal pullup. Many have *special*/functionality:

GPIO #0, which does not have an internal pullup, and is also connected to both a mini tactile switch and red LED. This pin is used by the ESP8266 to determine when to boot into the bootloader. If the pin is held low during power-up it will start bootloading! That said, you can always use it as an output, and blink the red LED.

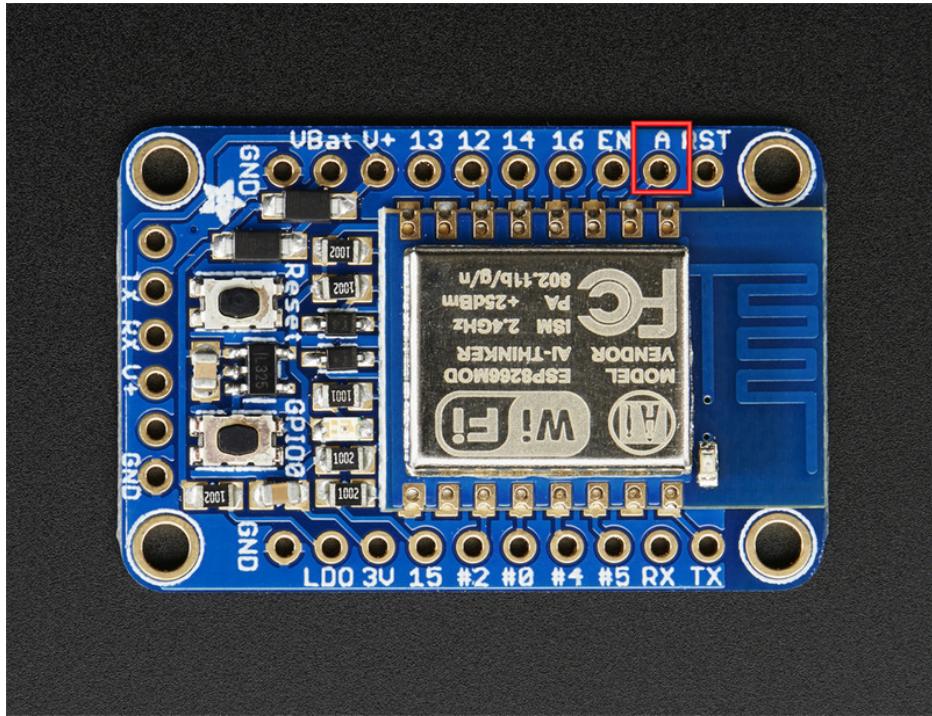
GPIO #2, is also used to detect boot-mode. It also is connected to the blue LED that is near the WiFi antenna. It has a pullup resistor connected to it, and you can use it as any output (like #0) and blink the blue LED.

GPIO #15, is also used to detect boot-mode. It has a pulldown resistor connected to it, make sure this pin isn't pulled high on startup. You can always just use it as an output

GPIO #16 can be used to wake up out of deep-sleep mode, you'll need to connect it to the RESET pin

GPIO #4 and **#5** are used for I2C SDA and SCL (respectively) by default on Arduino IDE. We recommend that if you are using Arduino, connect your I2C devices there!

Rev A of this board had GPIO #4 and #5 swapped (the modules changed pinouts on us) so if #4/#5 aren't working for you, try swapping!



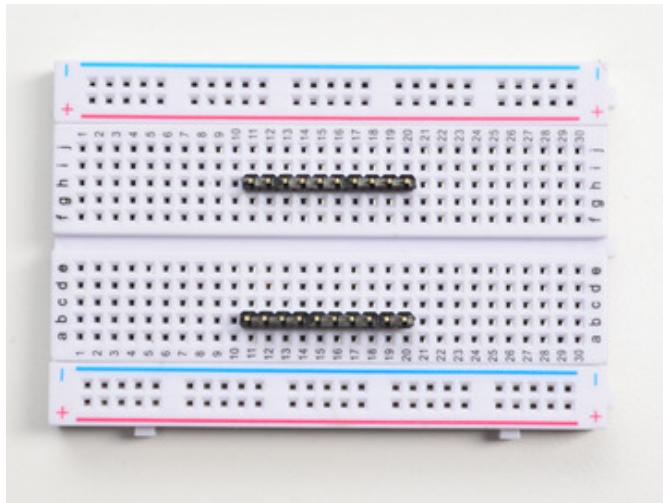
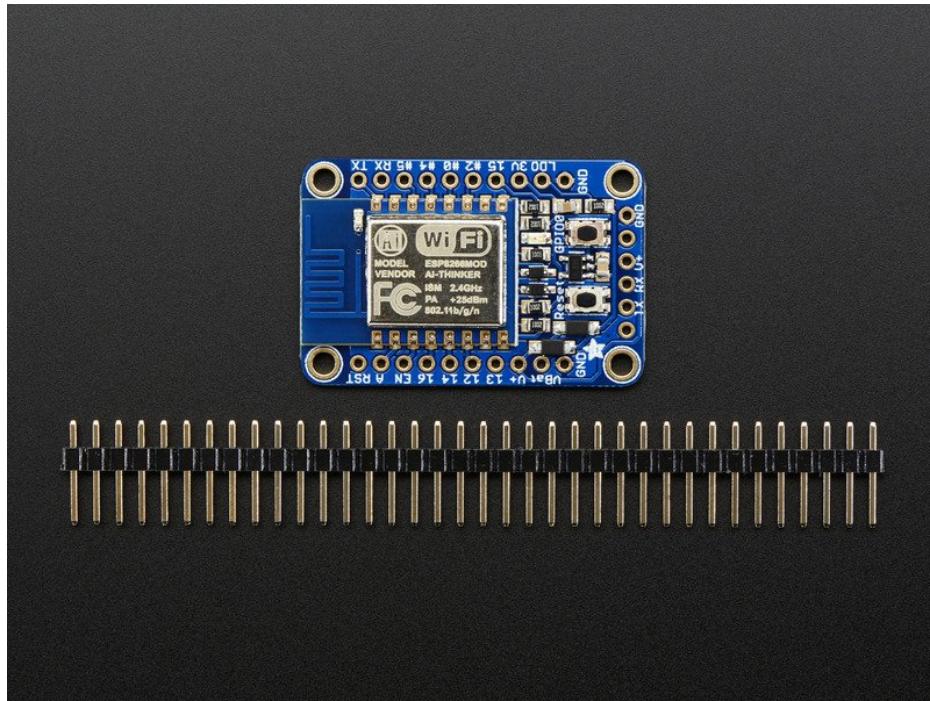
Analog Pins

There is also a single analog input pin called **A**. This pin has a $\sim 1.0V$ maximum voltage, so if you have an analog voltage you want to read that is higher, it will have to be divided down to 0 - 1.0V range

Other control pins

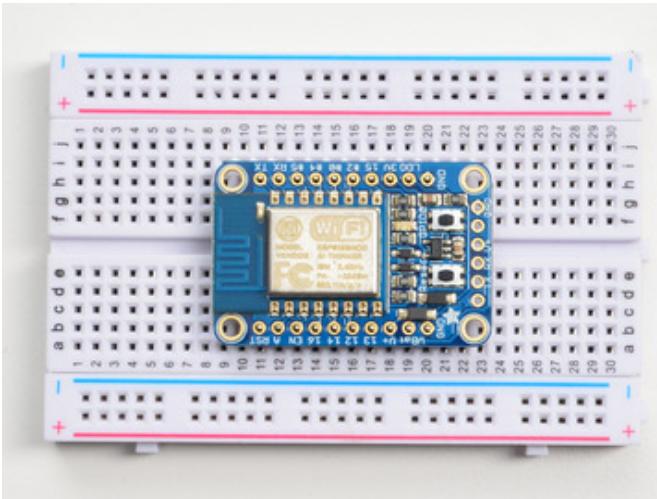
We have a few other pins for controlling the ESP8266

- **LDO** - this is the enable pin for the regulator. By default it's pulled high, when connected to ground it will turn off the 3.3V regulator and is an easy way to cut power off to the whole setup. There is a 10K pullup to whatever has a higher voltage, **V+** or **VBat**. (See the schematic if you're unclear)
- **RST** - this is the reset pin for the ESP8266, pulled high by default. When pulled down to ground momentarily it will reset the ESP8266 system. This pin is 5V compliant.
- **EN (CH_PD)** - This is the enable pin for the ESP8266, pulled high by default. When pulled down to ground momentarily it will reset the ESP8266 system. This pin is 3.3V logic only

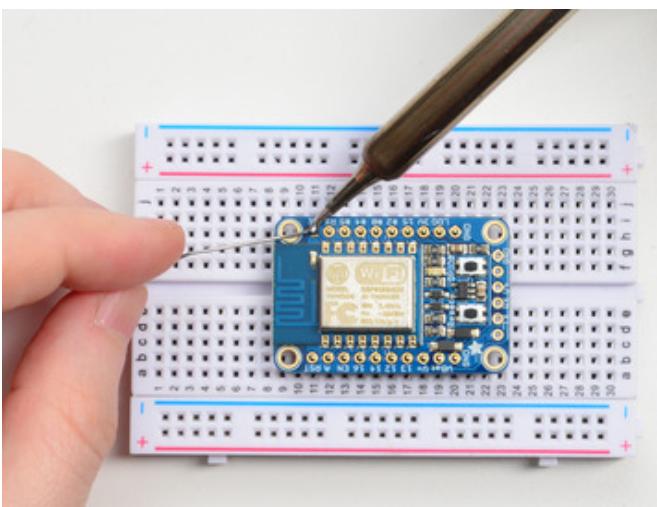


Prepare the header strip:

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



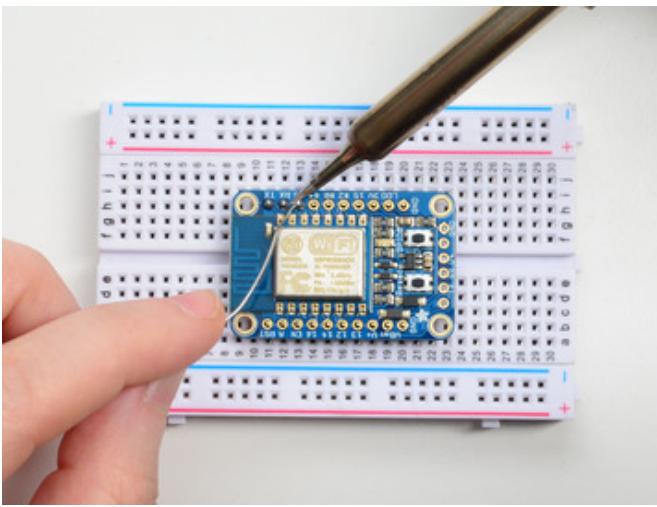
Add the breakout board:
Place the breakout board over the pins so that the short pins poke through the breakout pads

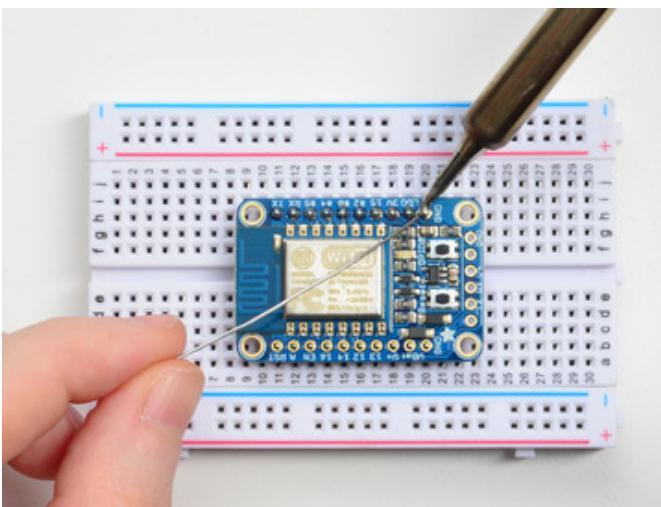
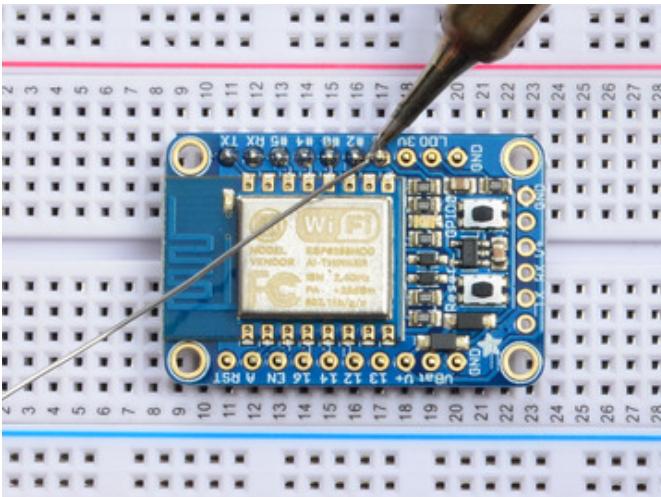


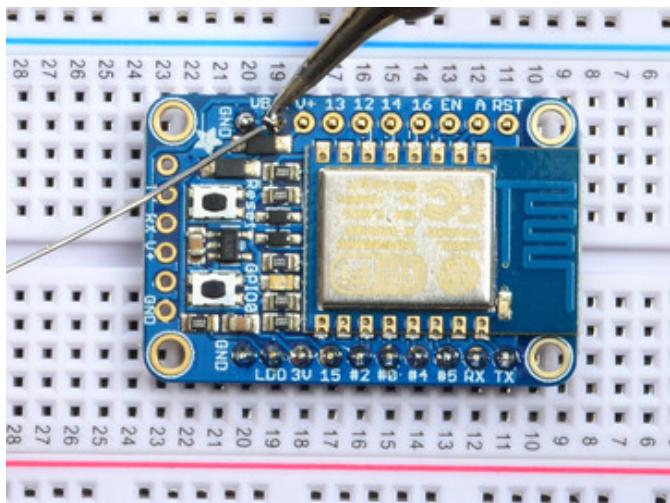
And Solder!
Be sure to solder all pins for reliable electrical contact.

Solder one side of the board at first

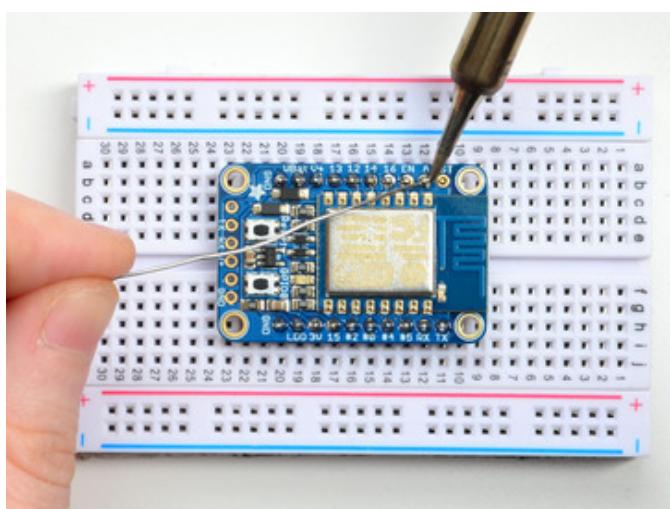
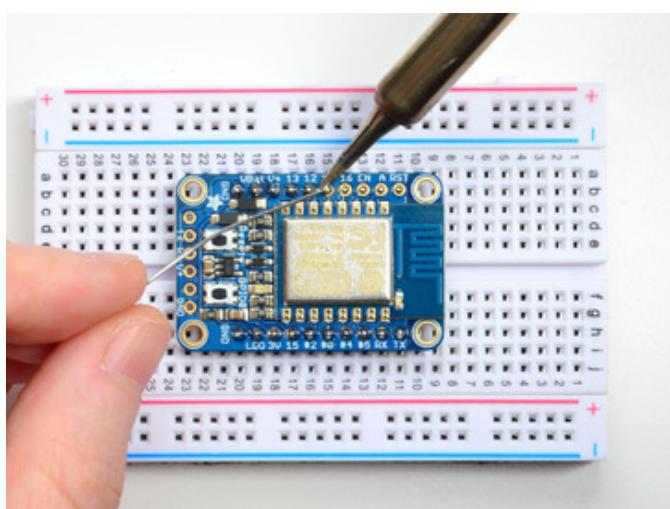
(For tips on soldering, be sure to check out our [Guide to Excellent Soldering](#) (<https://adafru.it/aTk>)).

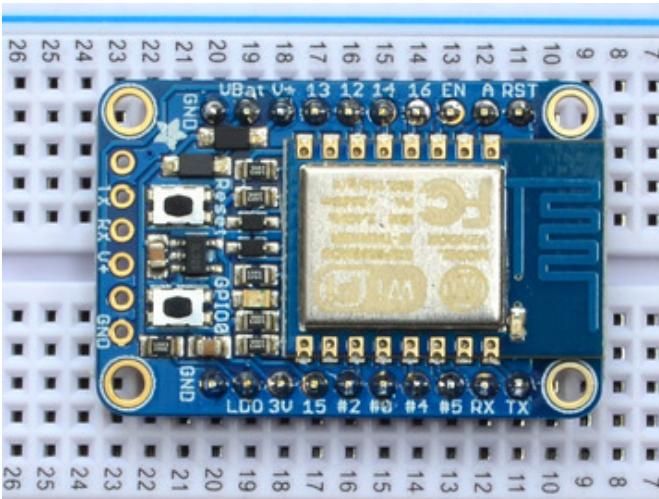




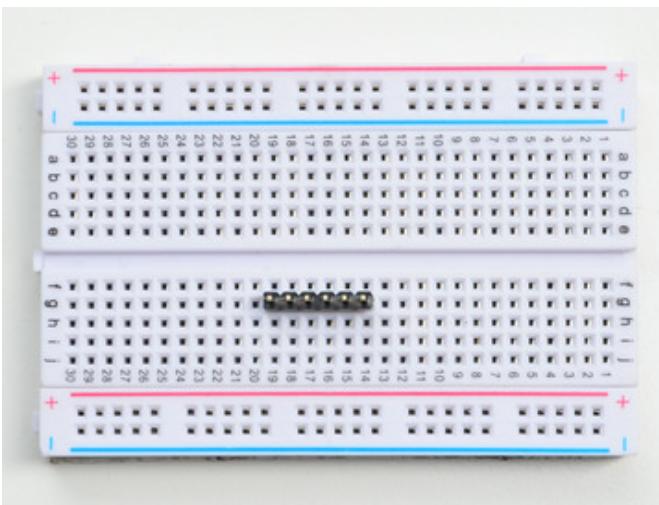


Flip the breadboard around to solder the other strip



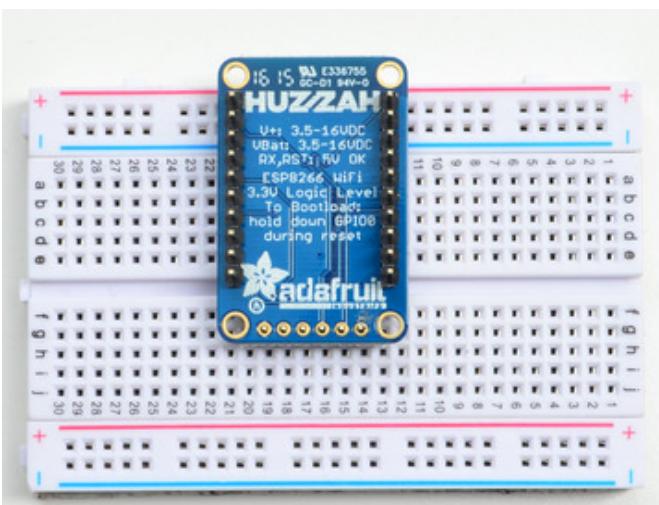


You're done! Check your solder joints visually and continue onto the next steps



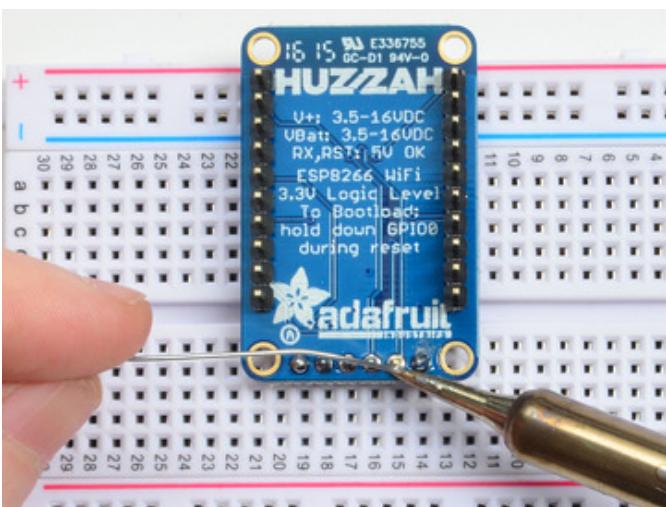
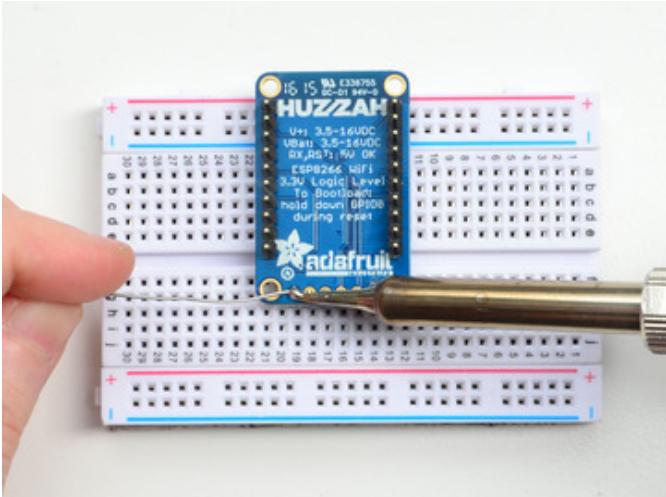
If you're planning on programming with an FTDI or console cable, it's handy to have 6 pins soldered on the end for plugging in.

Cut another 6-pin strip to length if necessary. Insert it into a breadboard - **long pins down**

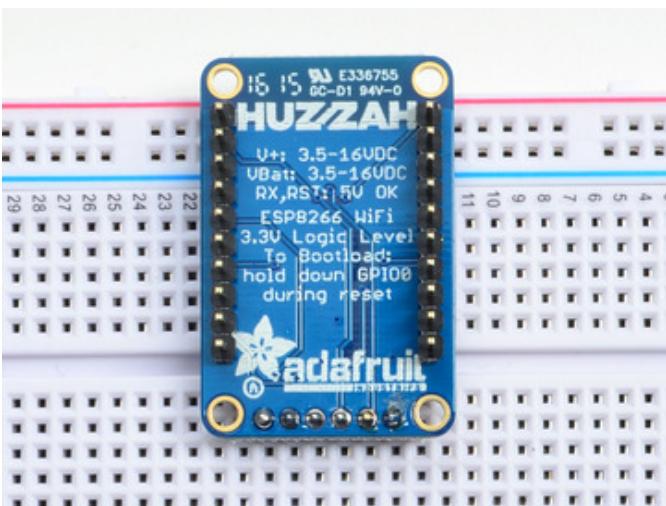


Place the breakout on the breadboard facing down

Solder all 6 pins!



You're done! Check your solder joints visually and continue onto the next steps



- We have found Lua is best for typing in short commands for testing things out, we suggest Arduino IDE for the best and most-tested programming interface!

Each HUZZAH ESP8266 breakout comes pre-programmed with NodeMCU's Lua interpreter. As of this writing, we ship with **NodeMCU 0.9.5 build 20150318** powered by **Lua 5.1.4** but it may be more recent

Lua is still a work in progress, so we strongly recommend visiting NodeMCU and [updating your Lua version to the very latest as they have the ability to make you the latest continuous build](https://adafru.it/wAq) (<https://adafru.it/wAq>). Then follow their guide on [how to update Lua!](https://adafru.it/wAr) (<https://adafru.it/wAr>)

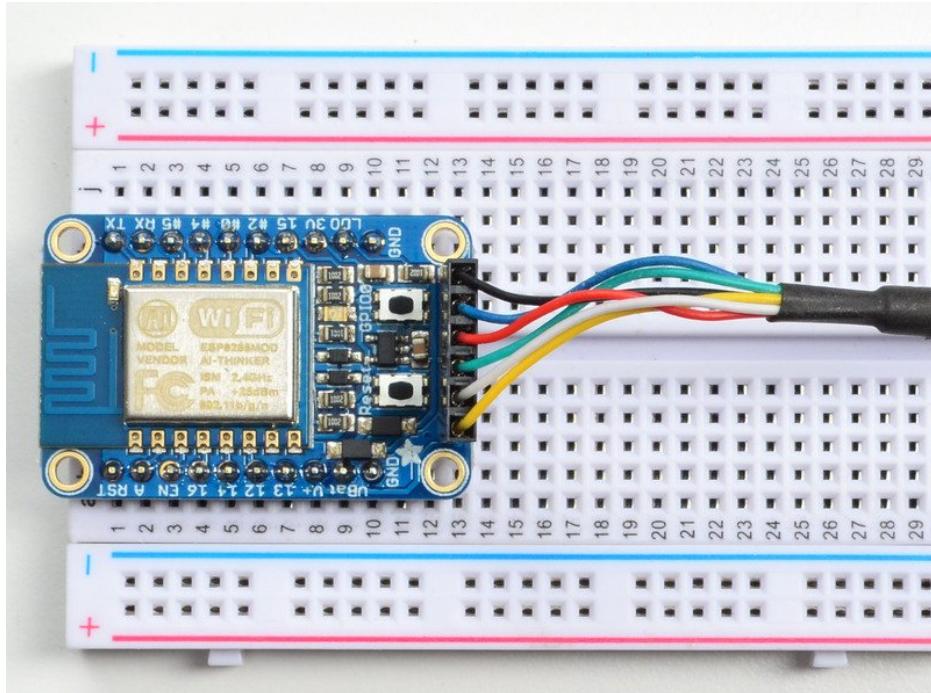
The Lua interpreter runs on the ESP8266 and you can type in commands and read out the results over serial. A serial console cable is perfect for this! [Use either an FTDI cable](https://adafru.it/dNN) (<https://adafru.it/dNN>) or [any console cable](http://adafru.it/954) (<http://adafru.it/954>), you can use either 3V or 5V logic and power as there is level shifting on the RX pin.

Don't forget to visit esp8266.com for the latest and greatest in ESP8266 news, software and gossip! ([https://adafru.it/f1F](http://adafru.it/f1F))

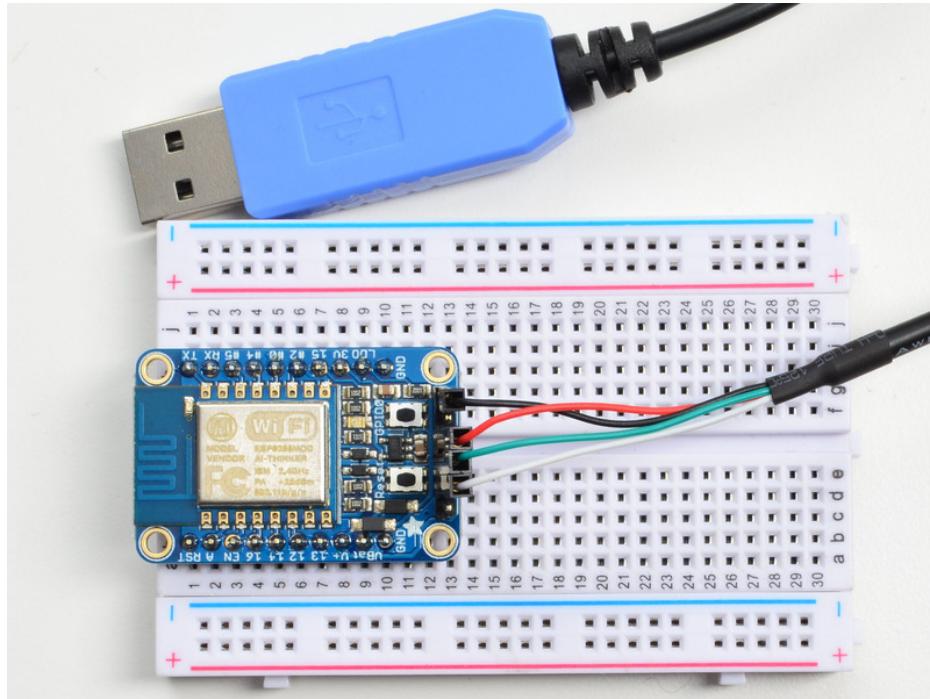
- The ESP8266 uses a lot of current, so if you're getting flakey behavior make sure you are plugging your console cable into either a motherboard USB port or a powered USB hub. Don't use the 'extra' USB port on your monitor or keyboard.

Connect USB-Serial cable

Connect either your console cable or FTDI cable. If using FTDI, make sure the black wire goes to the GND (ground) pin



If using a console cable, connect the black wire to ground, red wire to **V+**, white wire to **TX** and green wire to **RX**



You will see the red and blue onboard LED flicker when powered up, but they will not stay lit.

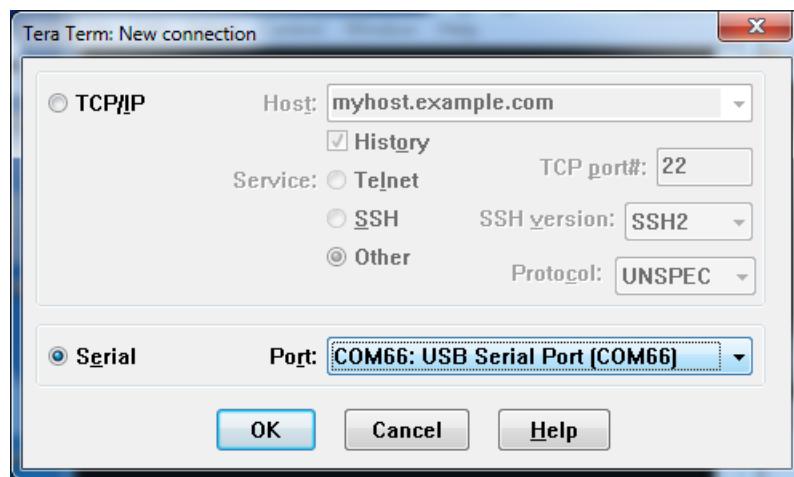
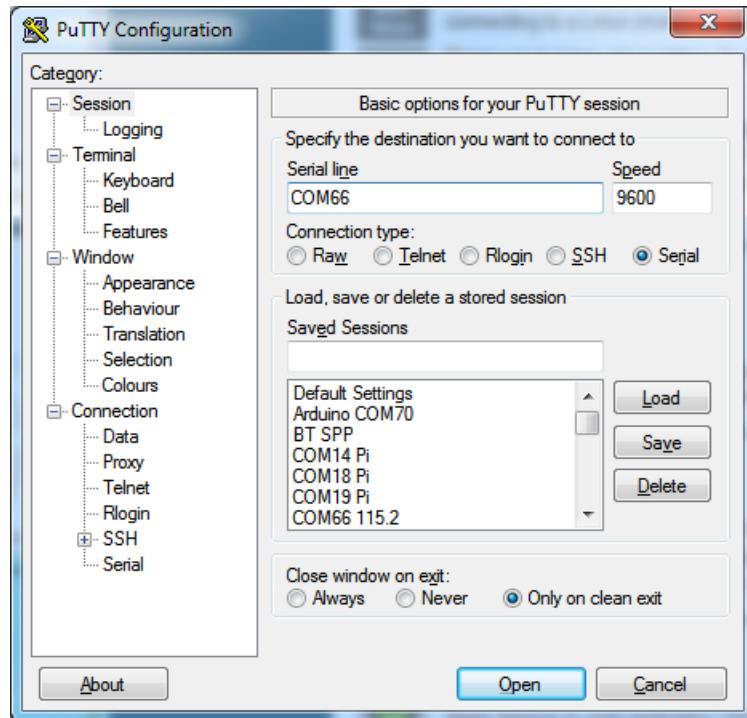
Open up serial console

Next up, on your computer, use a serial console program such as **CoolTerm** (Mac) or **Putty** (Windows) or **screen** (linux). Teraterm seems to dislike the initial 115.2kbps data stream from the ESP8266 so you can try it but you'll possibly need to reset the terminal software.

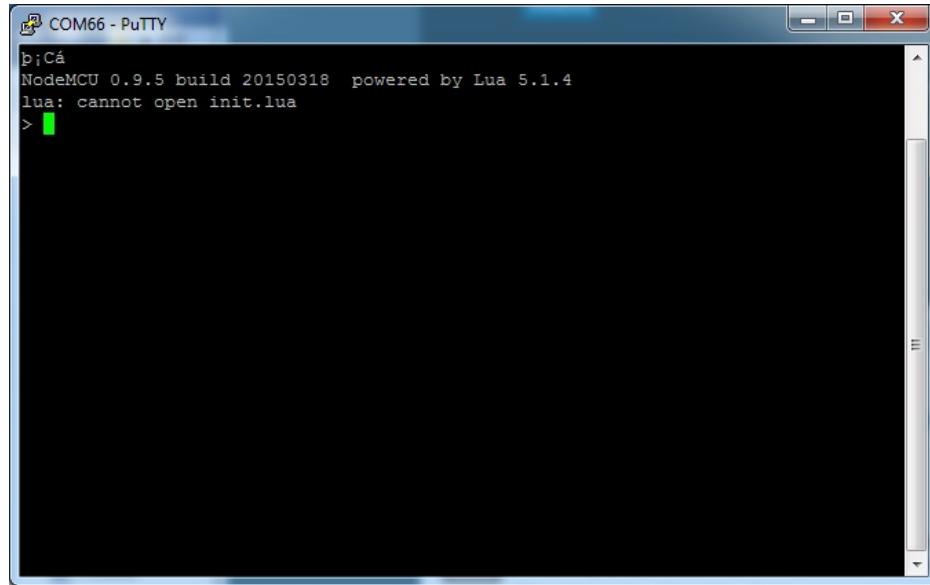
Connect up to the COM or Serial port used by your cable, at 9600 Baud

Make sure you have turned off any hardware handshake or flow control

Also make sure you have **line endings** set to CRLF "\r\n" You may also want to turn on inter-character delay if you are pasting in large chunks of code. Each terminal software is different in setting it up, check the manual for the software you're using!



Once the terminal software is connected, click the **Reset** button on the HUZZAH ESP board to reset it and have it print out the welcome message:

A screenshot of a PuTTY terminal window titled "COM66 - PuTTY". The window shows the NodeMCU startup message: "p;Cá", "NodeMCU 0.9.5 build 20150318 powered by Lua 5.1.4", and "lua: cannot open init.lua". A small green LED icon is visible in the bottom left corner of the terminal window.

If you don't get this message, first check that the red/blue leds flickered when you press the reset button. If they didn't, make sure the board is powered via **V+** or **Vbat**. If they do flicker, make sure you've got the right baud rate selected in the software (9600) and the RX/TX/GND pins connected right

Hello world!

Ok we can now turn on an LED. There is a red LED on each board, connected to**GPIO #0**

 NodeMCU's pinouts are not the same as the Arduino/gcc pinouts. We print the Arduino pinouts on the board so watch out!

 Rev A of this board has GPIO #4 and #5 swapped (the modules changed pinouts on us) so if #4/#5 aren't working for you, try swapping! We'll fix in the next PCB run

Pin Notes	PCB/Arduino	NodeMCU/Lua
No pullups!	0	3
	2	4
	3	9
	4	1
	5	2
	9	11
	10	12
	12	6
	13	7
	14	5
	15	8
	16	0

So to set the pin #0 LED on and off first make it an output by typing (not copy & paste)

```
gpio.mode(3, gpio.OUTPUT)
```

Turn the LED on by typing (not copy & paste)

```
gpio.write(3, gpio.LOW)
```

And off by typing (not copy & paste)

```
gpio.write(3, gpio.HIGH)
```

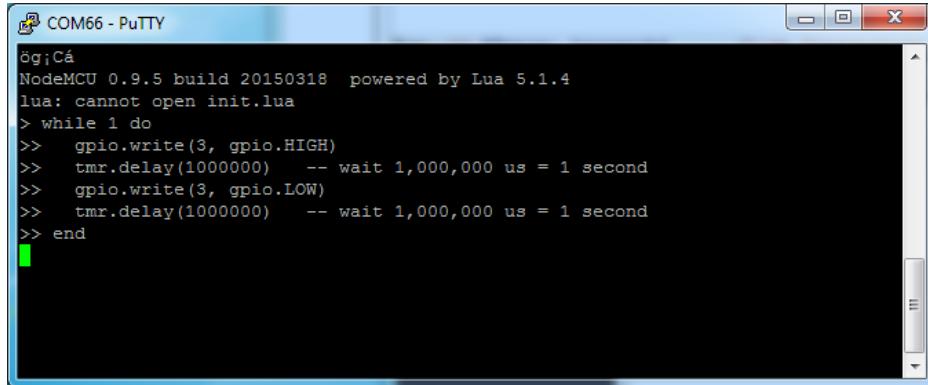
You can make this a little more automated by running a longer script.

For longer text, pasting can be difficult as the lua interpreter needs a little delay time between characters and also require CR-LF settings. For that reason you may want to paste each line and then hit return manually.

```

gpio.mode(3, gpio.OUTPUT)
while 1 do
    gpio.write(3, gpio.HIGH)
    tmr.delay(1000000)  -- wait 1,000,000 us = 1 second
    gpio.write(3, gpio.LOW)
    tmr.delay(1000000)  -- wait 1,000,000 us = 1 second
end

```



The LED will now be blinking on and off.

Note that since it's in a loop, it's not possible to get it to stop via the interpreter. To stop it, click the **Reset** button again!

This code halts the processor during the tmr.delay, a smarter way to blink an LED is to use the timer capability to set off the LED control ([code from here \(https://adafruit.it/wGA\)](https://adafruit.it/wGA))

```

-- Pin definition
local pin = 3
local status = gpio.LOW
local duration = 1000  -- 1 second duration for timer

-- Initialising pin
gpio.mode(pin, gpio.OUTPUT)
gpio.write(pin, status)

-- Create an interval
tmr.alarm(0, duration, 1, function ()
    if status == gpio.LOW then
        status = gpio.HIGH
    else
        status = gpio.LOW
    end

    gpio.write(pin, status)
end)

```

Scanning & Connecting to WiFi

We'll continue with a quick demo of scanning for WiFi and connecting.

Once you're back at the Lua prompt, set the ESP8266 into WiFi Client mode with

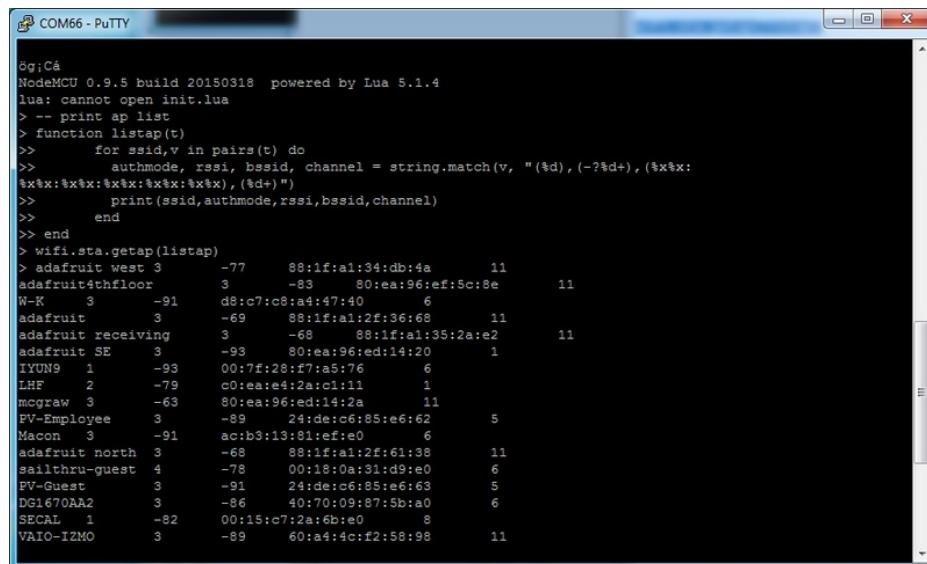
```
wifi.setmode(wifi.STATION)
```

Then you can run the scanner and have it print out the available AP's

```
-- print ap list
function listap(t)
    for k,v in pairs(t) do
        print(k.." : "..v)
    end
end
wifi.sta.getap(listap)
```

or for more detail...

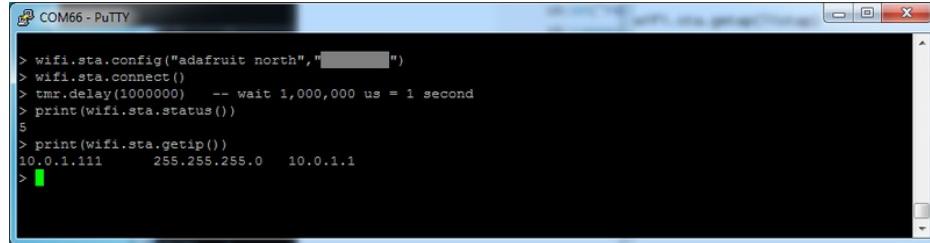
```
-- print ap list
function listap(t)
    for ssid,v in pairs(t) do
        authmode, rssi, bssid, channel =
            string.match(v, "(%d),(-?%d+),(%x%x:%x%x:%x%x:%x%x:%x%x),(%d+)")
        print(ssid,authmode,rssi,bssid,channel)
    end
end
wifi.sta.getap(listap)
```



```
8g;Cá
NodeMCU 0.9.5 build 20150318  powered by Lua 5.1.4
lua: cannot open init.lua
> -- print ap list
> function listap(t)
>>     for ssid,v in pairs(t) do
>>         authmode, rssi, bssid, channel = string.match(v, "(%d),(-?%d+),(%x%x:%x%x:%x%x:%x%x:%x%x),(%d+)")
>>         print(ssid,authmode,rssi,bssid,channel)
>>     end
>> end
> wifi.sta.getap(listap)
> adafruit west 3      -77      88:1f:a1:34:db:4a      11
adafruit4thfloor      3       -83      80:ea:96:ef:5c:8e      11
W-K      3       -91      d8:c7:c8:a4:47:40      6
adafruit      3       -69      88:1f:a1:2f:36:68      11
adafruit receiving  3       -68      88:1f:a1:35:2a:e2      11
adafruit SE      3       -93      80:ea:96:ed:14:20      1
IYUN9      1       -93      00:7f:28:f7:a5:76      6
LHF      2       -79      c0:ea:e4:2a:c1:11      1
mcgraw      3       -63      80:ea:96:ed:14:2a      11
PV-Employee      3       -89      24:de:c6:85:e6:62      5
Macon      3       -91      ac:b3:13:81:ef:e0      6
adafruit north      3       -68      88:1f:a1:2f:61:38      11
sailthru-guest      4       -78      00:18:0a:31:d9:e0      6
PV-Guest      3       -91      24:de:c6:85:e6:63      5
DG1670AA2      3       -86      40:70:09:87:5b:a0      6
SECAL      1       -82      00:15:c7:2a:6b:e0      8
VAIO-IZMO      3       -89      60:a4:4c:f2:58:98      11
```

We can connect to the access point with **wifi.sta.config** and **wifi.sta.connect** - it will take a second or two to complete the connection, you can query the module to ask the status with **wifi.sta.status()** - when you get a 5 it means the connection is completed and DHCP successful

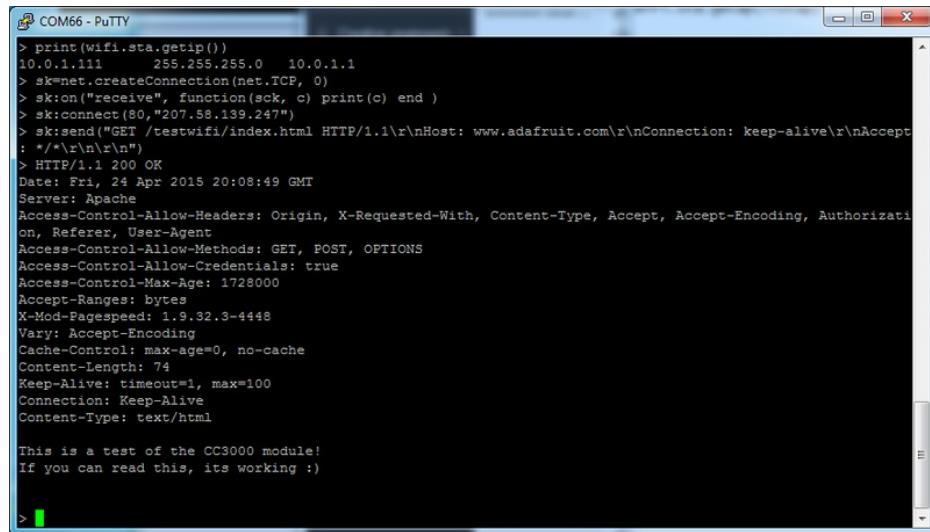
```
wifi.sta.config("accesspointname","yourpassword")
wifi.sta.connect()
tmr.delay(1000000) -- wait 1,000,000 us = 1 second
print(wifi.sta.status())
print(wifi.sta.getip())
```



WebClient example

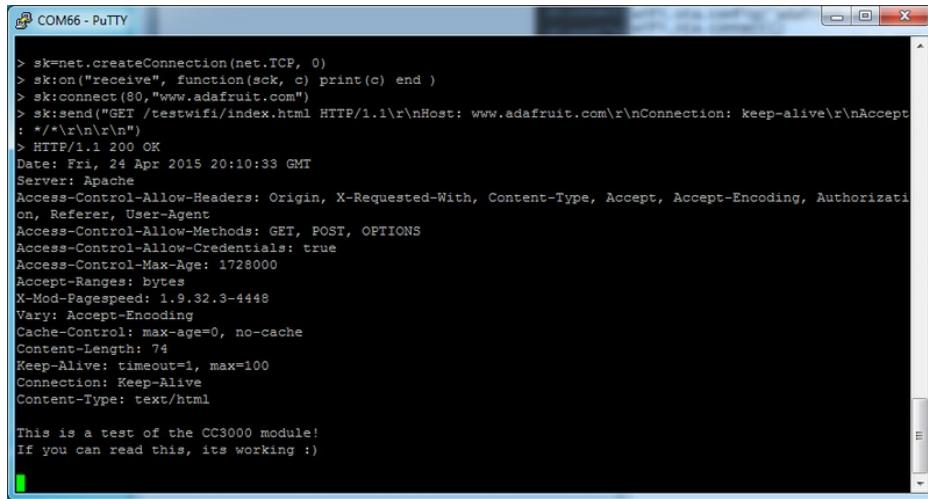
Once you've got the IP address you can connect to adafruit, for example, and read a webpage and print it out:

```
sk=net.createConnection(net.TCP, 0)
sk:on("receive", function(sck, c) print(c) end )
sk:connect(80, "104.236.193.178")
sk:send("GET /testwifi/index.html HTTP/1.1\r\nHost: wifitest.adafruit.com\r\nConnection: keep-alive\r\nAccept: */*\r\n\r\n")
```



You can also have the module do DNS for you, just give it the hostname instead of IP address:

```
sk=net.createConnection(net.TCP, 0)
sk:on("receive", function(sck, c) print(c) end )
sk:connect(80, "wifitest.adafruit.com")
sk:send("GET /testwifi/index.html HTTP/1.1\r\nHost: wifitest.adafruit.com\r\nConnection: keep-alive\r\nAccept: */*\r\n\r\n")
```



```
COM66 - PuTTY

> sk=net.createConnection(net.TCP, 0)
> sk:on("receive", function(sck, c) print(c) end )
> sk:connect(80, "www.adafruit.com")
> sk:send("GET /testwifi/index.html HTTP/1.1\r\nHost: www.adafruit.com\r\nConnection: keep-alive\r\nAccept: */*\r\n\r\n")
> HTTP/1.1 200 OK
Date: Fri, 24 Apr 2015 20:10:33 GMT
Server: Apache
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept, Accept-Encoding, Authorization, Referer, User-Agent
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Allow-Credentials: true
Access-Control-Max-Age: 1728000
Accept-Ranges: bytes
X-Mod-Pagespeed: 1.9.32.3-4448
Vary: Accept-Encoding
Cache-Control: max-age=0, no-cache
Content-Length: 74
Keep-Alive: timeout=1, max=100
Connection: Keep-Alive
Content-Type: text/html

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

This is just a light overview of testing out your HUZZAH ESP breakout! For much more, check out NodeMCU's documentation page <https://nodemcu.readthedocs.io/> (<https://adafru.it/wGB>) for the details on what functions are available to you, as well as <http://www.lua.org> (<https://adafru.it/f1N>) to learn more about the Lua scripting language

While the HUZZAH ESP8266 breakout comes pre-programmed with NodeMCU's Lua interpreter, you don't have to use it! Instead, you can use the Arduino IDE which may be more familiar. **This will write directly to the firmware, erasing the NodeMCU firmware, so if you want to go back to Lua, use the flasher to re-install it (<https://adafru.it/f1O>)**

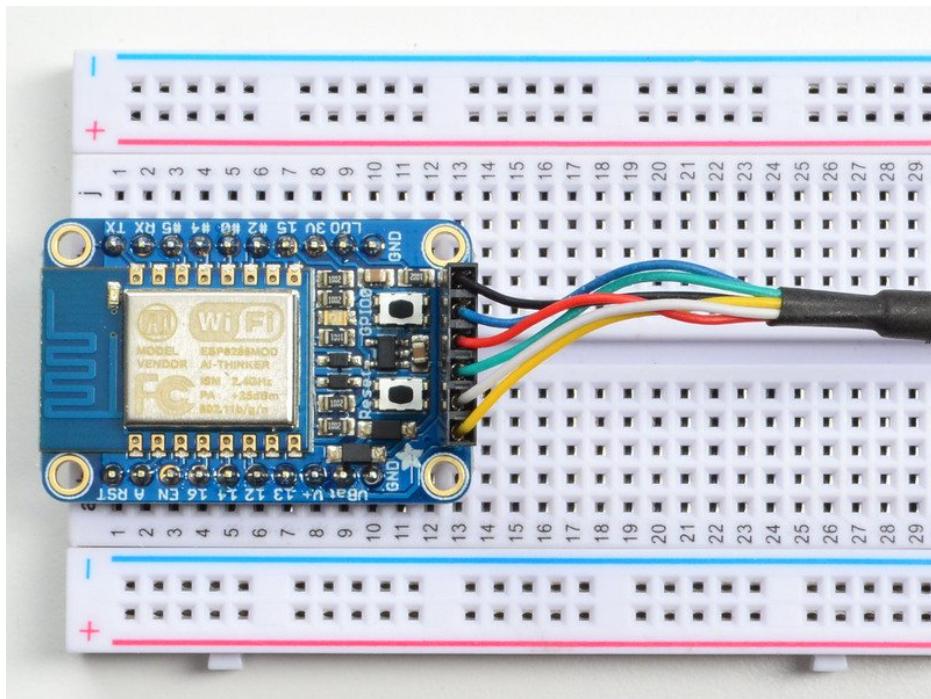
In order to upload code to the ESP8266 and use the serial console, you will need a USB to serial converter! **Use either an FTDI cable (<https://adafru.it/dNN>) or any console cable (<http://adafru.it/954>), you can use either 3V or 5V logic and power as there is level shifting on the RX pin.**

Don't forget to visit esp8266.com for the latest and greatest in ESP8266 news, software and gossip! (<https://adafru.it/f1F>)

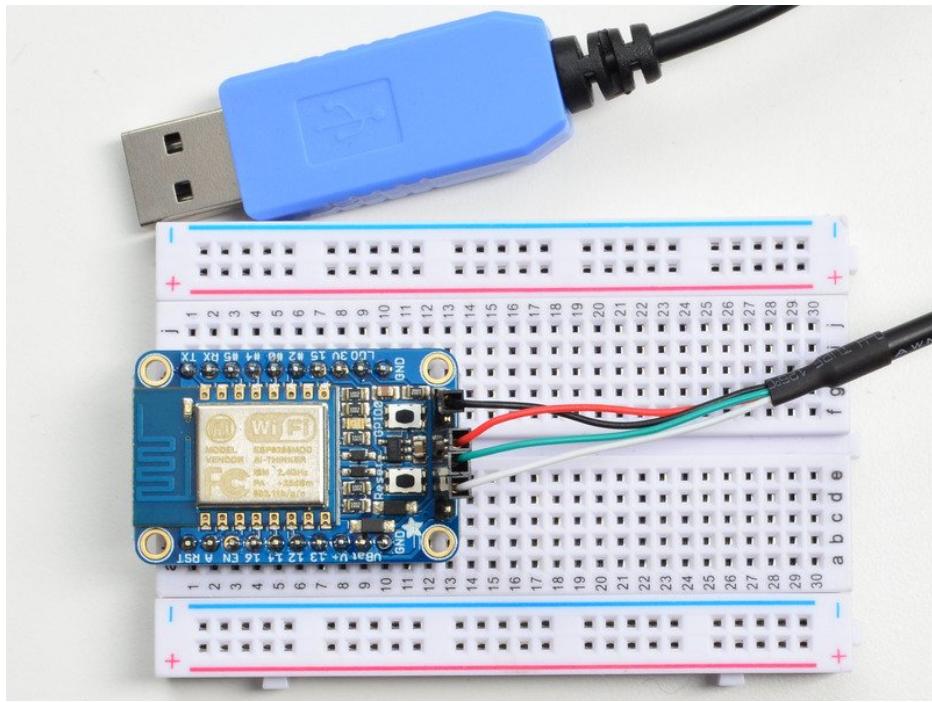
 The ESP8266 uses a lot of current, so if you're getting flakey behavior make sure you are plugging your console cable into either a motherboard USB port or a powered USB hub. Don't use the 'extra' USB port on your monitor or keyboard.

Connect USB-Serial cable

Connect either your console cable or FTDI cable. If using FTDI, make sure the black wire goes to the GND (ground) pin



If using a console cable, connect the black wire to ground, red wire to **V+**, white wire to **TX** and green wire to **RX**



You will see the red and blue onboard LED flicker when powered up, but they will not stay lit.

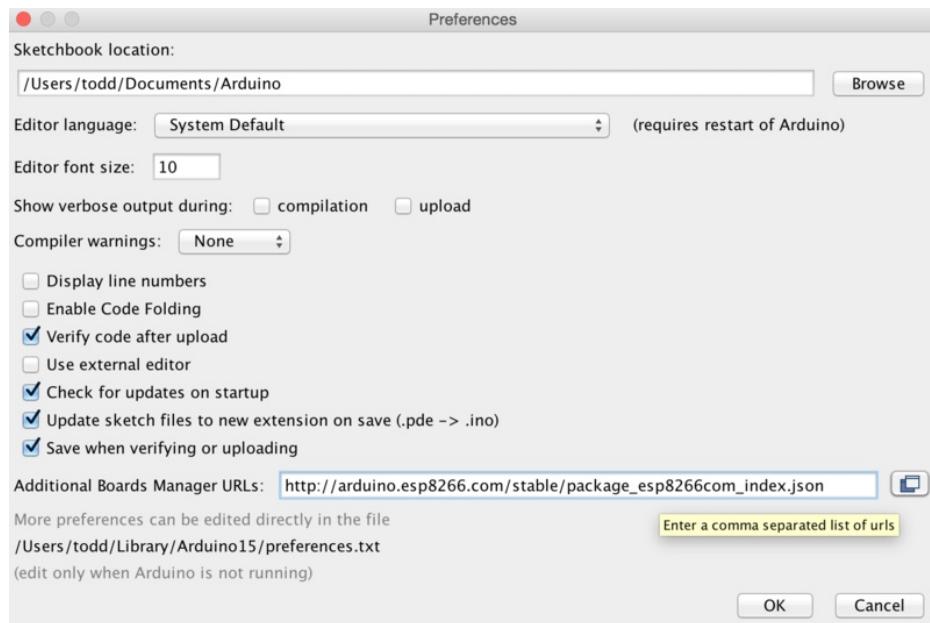
Install the Arduino IDE 1.6.4 or greater

[Download Arduino IDE from Arduino.cc \(1.6.4 or greater\)](http://arduino.cc) - don't use 1.6.2! You can use your existing IDE if you have already installed it (<https://adafru.it/f1P>)

You can also try downloading the ready-to-go package from the [ESP8266-Arduino project](https://adafru.it/eSH) (<https://adafru.it/eSH>), if the proxy is giving you problems

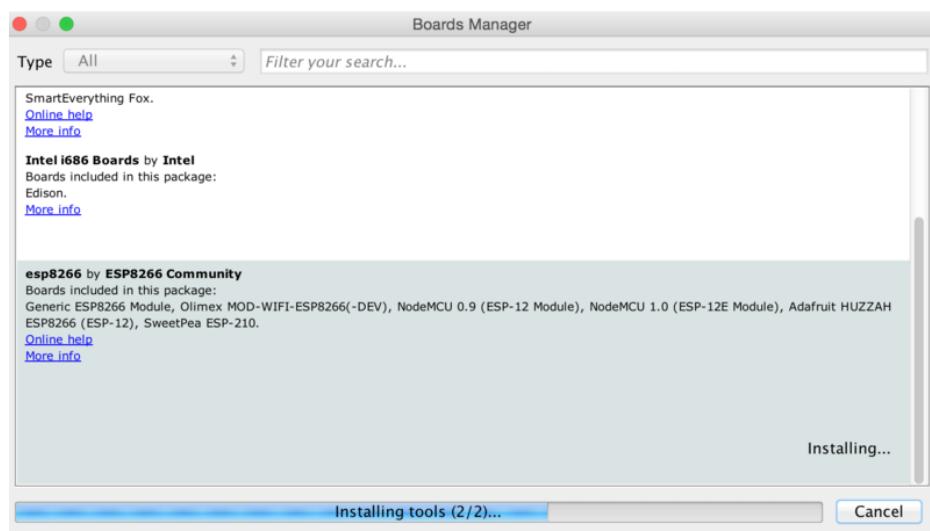
Install the ESP8266 Board Package

Enter http://arduino.esp8266.com/stable/package_esp8266com_index.json into *Additional Board Manager URLs* field in the Arduino v1.6.4+ preferences.



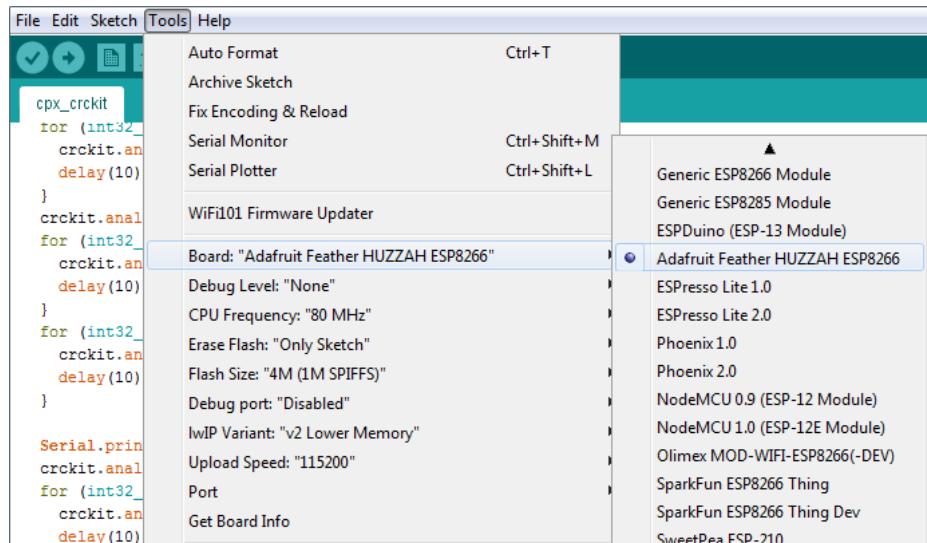
Visit our guide for how to add new boards to the Arduino 1.6.4+ IDE for more info about adding third party boards (<https://adafru.it/f7X>).

Next, use the **Board manager** to install the ESP8266 package.

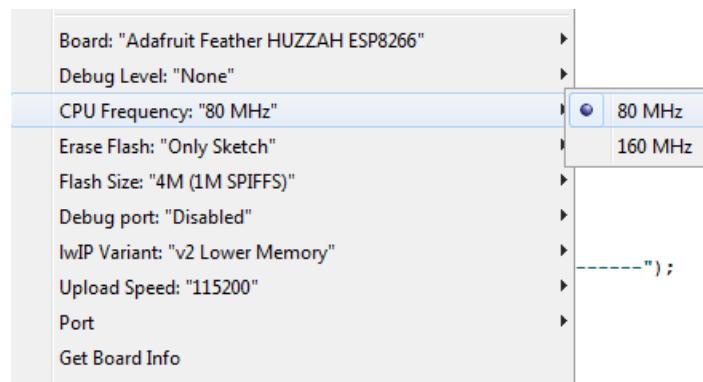


Setup ESP8266 Support

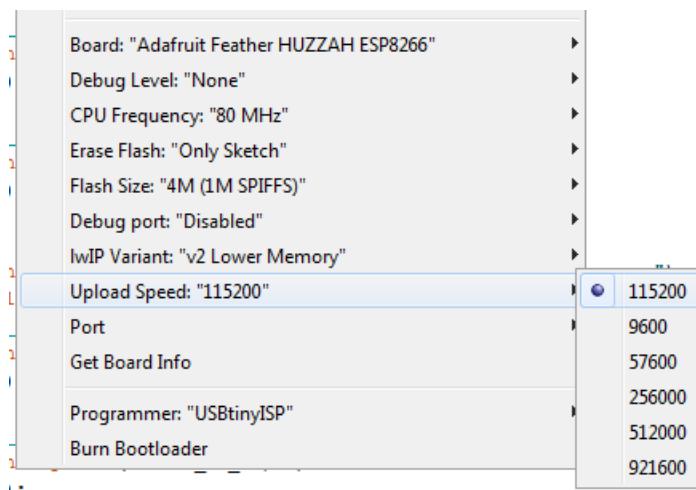
When you've restarted, select **Adafruit Feather HUZZAH ESP8266** from the Tools->Board dropdown. Note that even though this is a Huzzah breakout, its the same 'definition' file as the Feather so just use that!



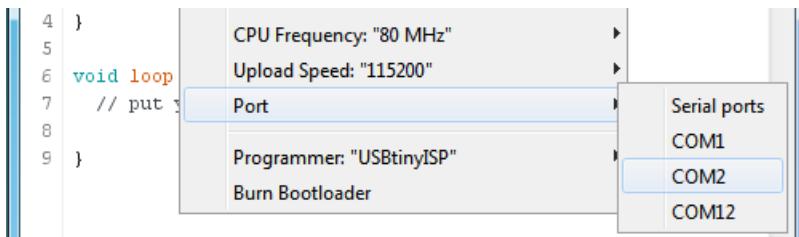
80 MHz as the CPU frequency (you can try 160 MHz overclock later)



115200 baud upload speed is a good place to start - later on you can try higher speeds but 115200 is a good safe place to start. You can move down to lower speeds if you are having upload issues, or speed it up if your setup is stable, for faster uploads!



The matching COM/serial port for your FTDI or USB-Serial cable



You don't have to set **Programmer** because it will always use the serial port, just ignore that menu item!

Blink Test

We'll begin with the simple blink test

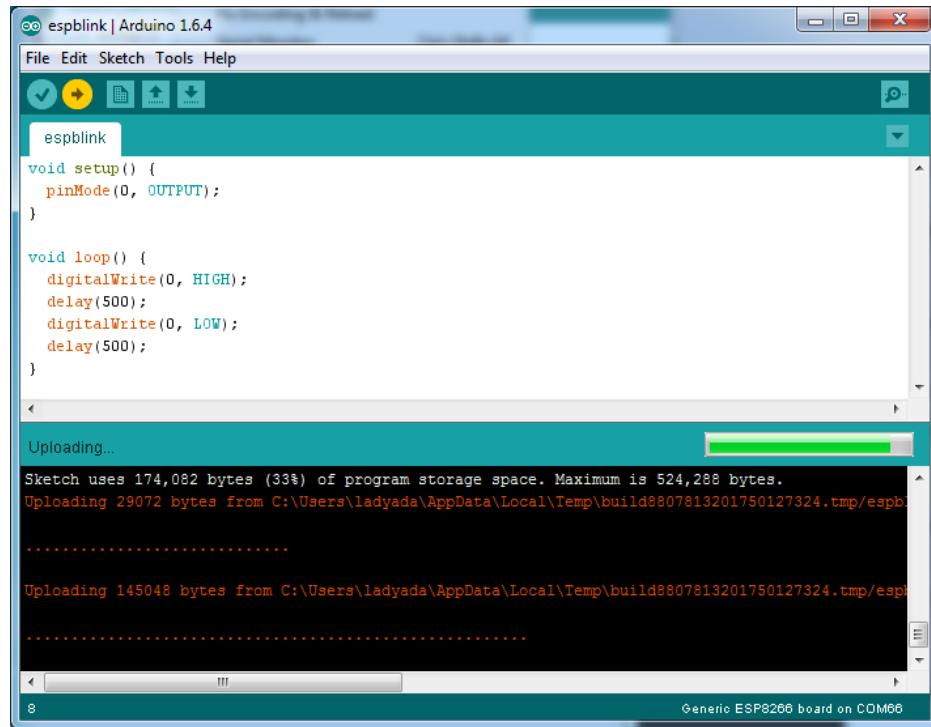
Enter this into the sketch window (and save since you'll have to)

```
void setup() {  
  pinMode(0, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(0, HIGH);  
  delay(500);  
  digitalWrite(0, LOW);  
  delay(500);  
}
```

Now you'll need to put the board into bootload mode. You'll have to do this before each upload. There is no timeout for bootload mode, so you don't have to rush!

1. Hold down the **GPIO0** button, the red LED will be lit
2. While holding down **GPIO0**, click the **RESET** button
3. Release **RESET**, then release **GPIO0**
4. When you release the RESET button, the red LED will be lit dimly, this means it's ready to bootload

Once the ESP board is in bootload mode, upload the sketch via the IDE



If the upload is successful, it should end with this message:

Hard resetting via RTS pin...

Once you see that, press the **RESET** button and the sketch will then run.

Connecting via WiFi

OK once you've got the LED blinking, let's go straight to the fun part, connecting to a webserver. Create a new sketch with this code:

```

/*
 * Simple HTTP get webclient test
 */

#include <ESP8266WiFi.h>

const char* ssid      = "yourssid";
const char* password = "yourpassword";

const char* host = "wifitest.adafruit.com";

void setup() {
  Serial.begin(115200);
  delay(100);

  // We start by connecting to a WiFi network

  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

```

```

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
Serial.print("Netmask: ");
Serial.println(WiFi.subnetMask());
Serial.print("Gateway: ");
Serial.println(WiFi.gatewayIP());
}

int value = 0;

void loop() {
  delay(5000);
  ++value;

  Serial.print("connecting to ");
  Serial.println(host);

  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

  // We now create a URI for the request
  String url = "/testwifi/index.html";
  Serial.print("Requesting URL: ");
  Serial.println(url);

  // This will send the request to the server
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
    "Host: " + host + "\r\n" +
    "Connection: close\r\n\r\n");
  delay(500);

  // Read all the lines of the reply from server and print them to Serial
  while(client.available()){
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

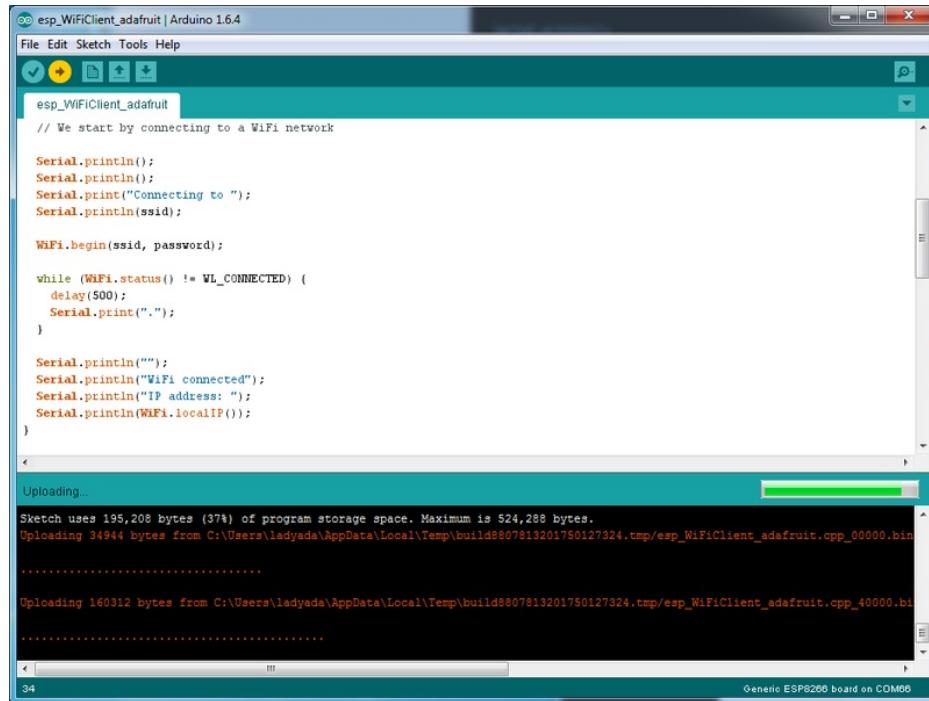
  Serial.println();
  Serial.println("closing connection");
}

```

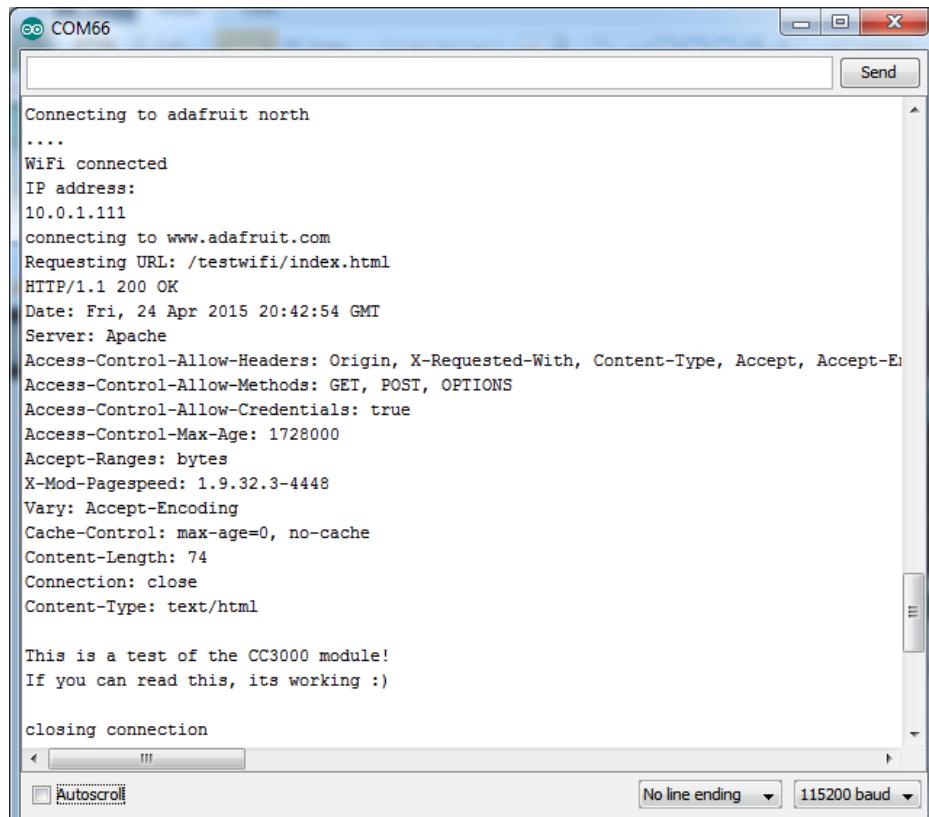
Dont forget to update

```
const char* ssid    = "yourssid";
const char* password = "yourpassword";
```

to your access point and password, then upload the same way: get into bootload mode, then upload code via IDE



Open up the IDE serial console at 115200 baud to see the connection and webpage printout!



That's it, pretty easy!

This page was just to get you started and test out your module. For more information, check out the [ESP8266 port](#) [github repository \(https://adafru.it/eSH\)](#) for much more up-to-date documentation!

Other Options

- You can load MicroPython onto your ESP8226 and write code in Python! It's an official MicroPython core and is fairly well supported. We even have tons of projects and getting started guides! (<https://adafru.it/xEp>)
- You can also try using `embedXcode` (<https://adafru.it/f5J>) which has a template for the ESP8266 with Xcode
- `esp-open-sdk` (<https://adafru.it/f5K>) is a toolchain that will let you program the ESP8266 processor directly (more info at the `esp8266.com` wiki (<https://adafru.it/f5L>))

Downloads

Datasheets

- [ESP8266 specification sheet](https://adafru.it/f1E) (<https://adafru.it/f1E>)
- [ESP8266 Datasheet v4.3](https://adafru.it/rdl) (<https://adafru.it/rdl>)
- [SPX3819 3.3V linear regulator on board for rev A](https://adafru.it/f4z) (<https://adafru.it/f4z>)
- [AP2112K-3.3 linear regulator on board for rev C and greater](https://adafru.it/o4a) (<https://adafru.it/o4a>)
- [FCC test report](https://adafru.it/f1S) (<https://adafru.it/f1S>) for the module used on this breakout
- [CE test report for the module used on this breakout](https://adafru.it/f1U) (<https://adafru.it/f1U>)

CAD Files

- [Fritzing object in the Adafruit Fritzing Library](https://adafru.it/aP3) (<https://adafru.it/aP3>)
- [EagleCAD PCB files on GitHub](https://adafru.it/rdm) (<https://adafru.it/rdm>)

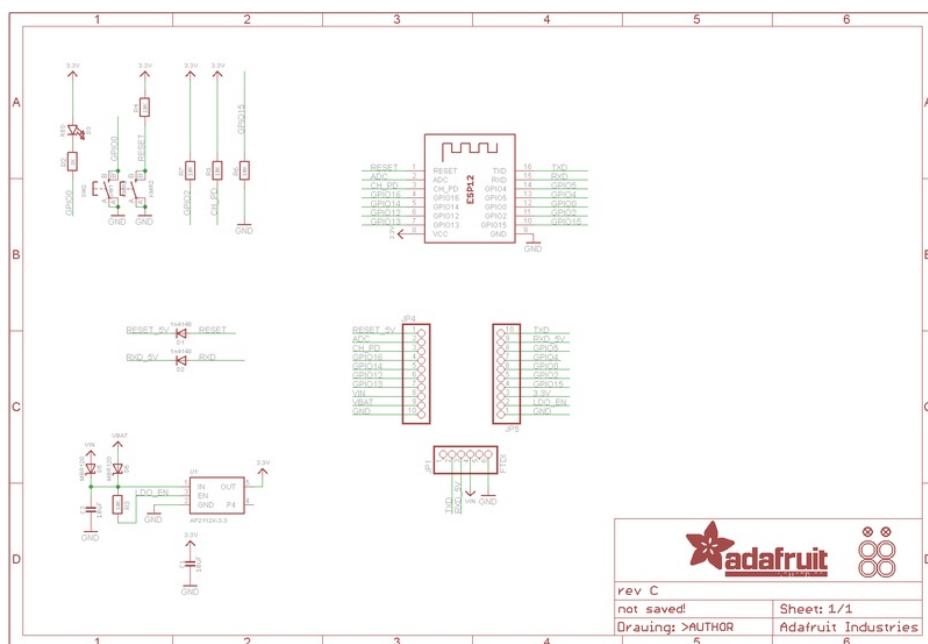
More Information!

- Huuuuge amount of information on <http://www.esp8266.com/> (<https://adafru.it/f1F>) community forum!
- [NodeMCU \(Lua for ESP8266\) webpage](https://adafru.it/f1G) (<https://adafru.it/f1G>) with examples and documentation on the Lua framework
- [Arduino IDE support for ESP8266](https://adafru.it/eSH) (<https://adafru.it/eSH>)
- [NodeMCU PyFlasher - a cross platform ESP flashing tool](https://adafru.it/Bg4) (<https://adafru.it/Bg4>)

Don't forget to visit [esp8266.com](http://www.esp8266.com/) for the latest and greatest in ESP8266 news, software and gossip! (<https://adafru.it/f1F>)

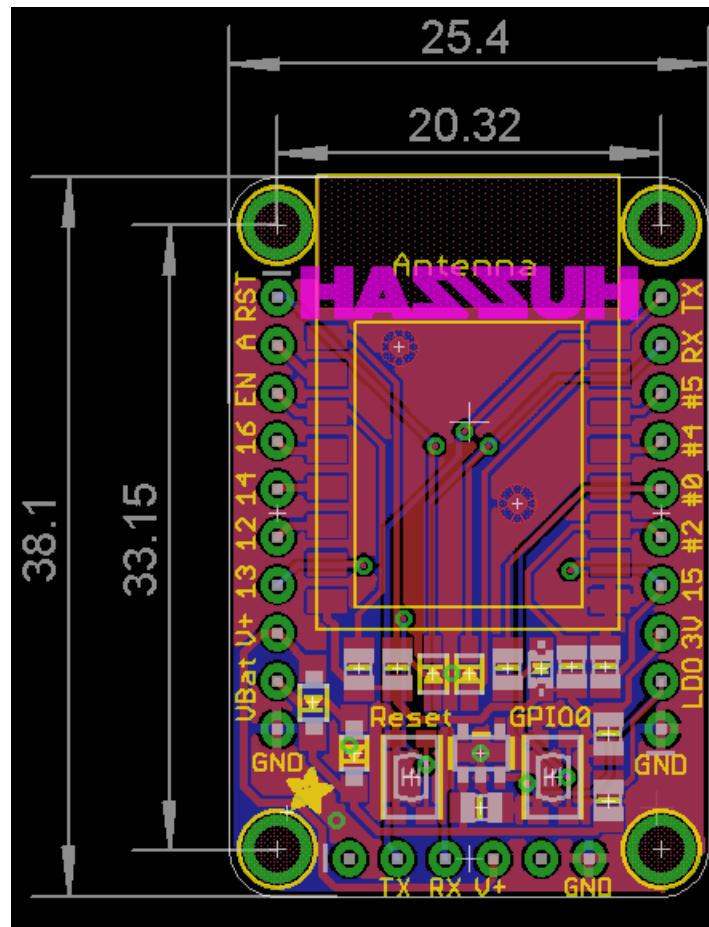
Schematic

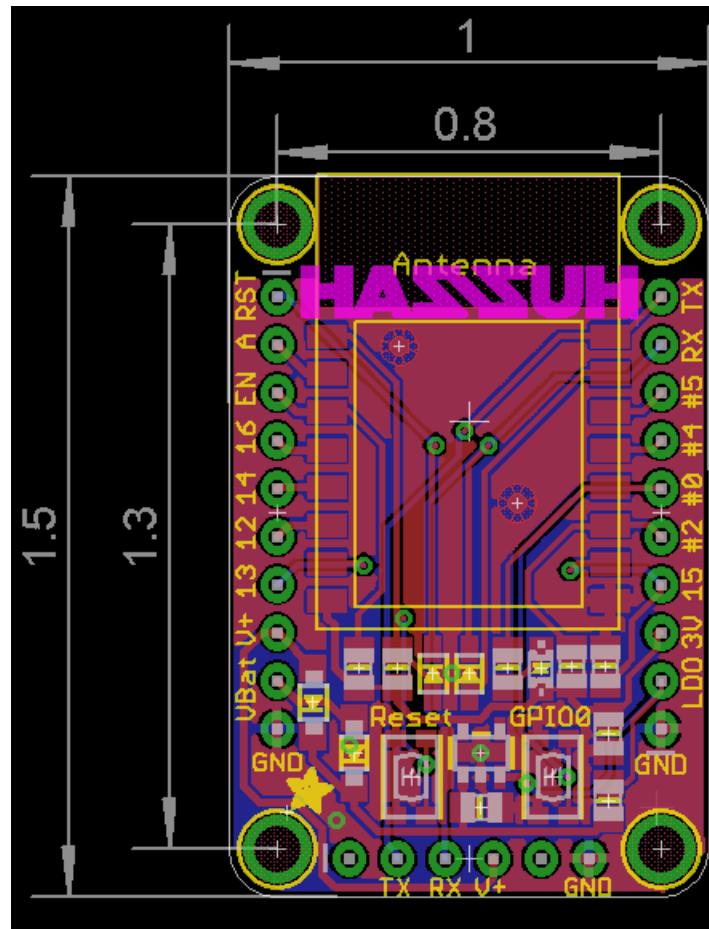
LDO regulator changed from SPX3819 to AP2112 in rev C, now 6V max (but lower dropout) and 600mA out max



Fabrication print

Dimensions in mm & inches





ESP8266 F.A.Q.

□ When I connect stuff to some of the pins, the Huzzah stops working. Whats up with that?

The ESP8266 uses some of the pins as 'boot mode' pins so on boot they must be set to certain values:

- **CH_PD (EN)** should be always pulled high (it will disable the entire module if low)
- **RST** should be always pulled high (it will disable the entire module if low)
- **GPIO 0** sets whether the bootloader is active, it must be pulled HIGH during power up/reset for the user program to run. If it's pulled LOW, it will activate the bootloader. The built-in red LED on #0 pulls it up
- **GPIO 2** must be pulled high on power up/reset.
- **GPIO 15** must be pulled low on power up/reset.

□ My Huzzah board keeps crashing and resetting, whats up with that?

The most common reason for crashes is power failure. Make sure you're powering the Huzzah with a good ~5V power supply, and if you're using a USB-Serial cable, that it's plugged into the mainboard of your computer or through a **powered** hub!

□ I can't seem to find the Serial port on my computer for the Feather HUZZAH?

Don't forget to install the [CP2104 VCP drivers](#) for your computer, they are required!

□ I still can't seem to find the Serial port on my computer for the Feather Huzzah!

Many cheap electronics come with charge-only USB cables, which cause headaches later. Make sure you are using a proper data/sync USB cable. If you find a cable that is charge-only (not data/sync also) throw it out so you don't have the same problem again.

So, I'm getting a 'no such file' error compiling for ESP8266 on my Mac

If your error message looks like this:

```
fork/exec /Users/xxxxxx/Library/Arduino15/packages/esp8266/tools/xtensa-lx106-elf-gcc/1.20.0-26-gb404fb9-2/bin/xtensa-lx106-elf-g++: no such file or directory
Error compiling.
```

To fix this problem, do this:

1. Open the Boards Manager in the Arduino IDE
2. Uninstall the ESP8266 support
3. go to your ~Library folder (in the Finder, select "Go:Go to folder:, and enter ~Library). Find the folder Arduino15.
4. In the Arduino15 folder, go into packages, and delete the folder **esp8266**
5. Go back to the Arduino IDE, and install ESP8266 board support.
6. Now go back to the Finder, and check that you have the xtensa-lx106-elf-g++ file in the path *Arduino15/packages/esp8266/tools/xtensa-lx106-elf-gcc/1.20.0-26-gb404fb9-2/bin/xtensa-lx106-elf-g++*
7. That's it!

□ Whenever I start or reset the ESP8226 there's a bunch of "gibberish" on the Serial console

This is the ROM debug messages, it's transmitted at 74880 baud so you rarely see it in proper 'ascii output' - instead usually it gets corrupted into a bunch of strange characters.

□ I'm having difficulties uploading to the HUZZAH with the Arduino IDE

Make sure you're using a good quality USB/Serial cable. Install the official drivers for that cable too! We've also noticed that PL2303-based cables don't work on Macs for some reason. FTDI or CP210x based chipsets work best

□ I tried that, but I'm still having difficulties uploading with the Arduino IDE

Sometimes, it helps to switch the board type to "Generic ESP8266 Module". Set the Reset Method to "nodemcu"

See [this forum post](#)

□ I'm stuck in bootloader mode and can't upload

You say your led is stuck on dim and you get an error trying to upload? And you're sure your serial cable is connected and working correctly? Well, here's a potential fix: Connect the GPIO0 pin to GND through a 220 ohm resistor. Leave it connected while you upload. You may have to try it a couple of times, but it should eventually upload and get the HUZZAH unstuck from bootload mode! You can then remove the resistor connection, and your HUZZAH will be happy ever after!

(Note: you may also have to tie RST and EN (CH_PD) together to get this to work. Remove the connection once you have the module programmed).

Thanks to [forum user misslevania for the tip!](#)

□ I can't get Lua to respond to my commands

Make sure your terminal software is sending correct line endings! The default PuTTY settings may be wrong when trying to talk to Lua on an ESP8266. Lua expects CRLF "\r\n" line endings, and apparently PuTTY defaults to just LF "\n"!

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

[Adafruit:](#)

[2471](#)