ESP8266 WiFi Module Quick Start Guide

Introduction

Your ESP8266 is an impressive, low cost WiFi module suitable for adding WiFi functionality to an existing microcontroller project via a UART serial connection. The module can even be reprogrammed to act as a standalone WiFi connected device–just add power!

The feature list is impressive and includes:

- 802.11 b/g/n protocol
- Wi-Fi Direct (P2P), soft-AP
- Integrated TCP/IP protocol stack

This guide is designed to help you get started with your new WiFi module so let's start!

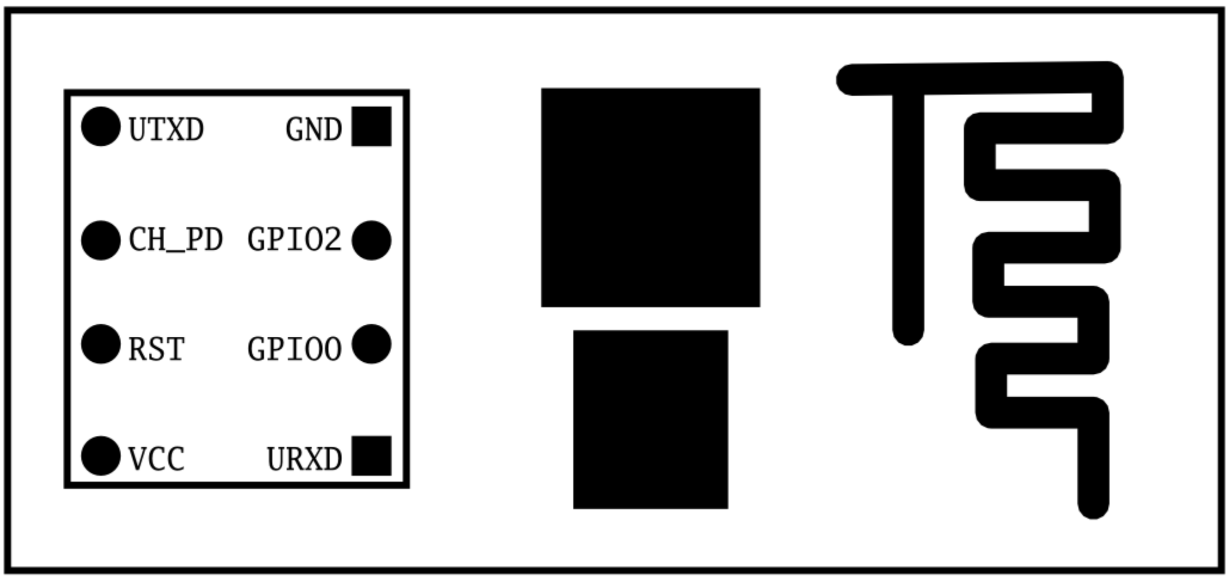
Hardware Connections

The hardware connections required to connect to the ESP8266 module are fairly straight-forward but there are a couple of important items to note related to power:

- **The ESP8266 requires 3.3V power–do not power it with 5 volts!**
- **The ESP8266 needs to communicate via serial at 3.3V and does not have 5V tolerant inputs, so you need level conversion to communicate with a 5V microcontroller like most Arduinos use.**

However, if you're adventurous and have no fear you can possibly get away with ignoring the second requirement. But nobody takes any responsibility for what happens if you do. :)

Here are the connections available on the ESP8266 WiFi module:



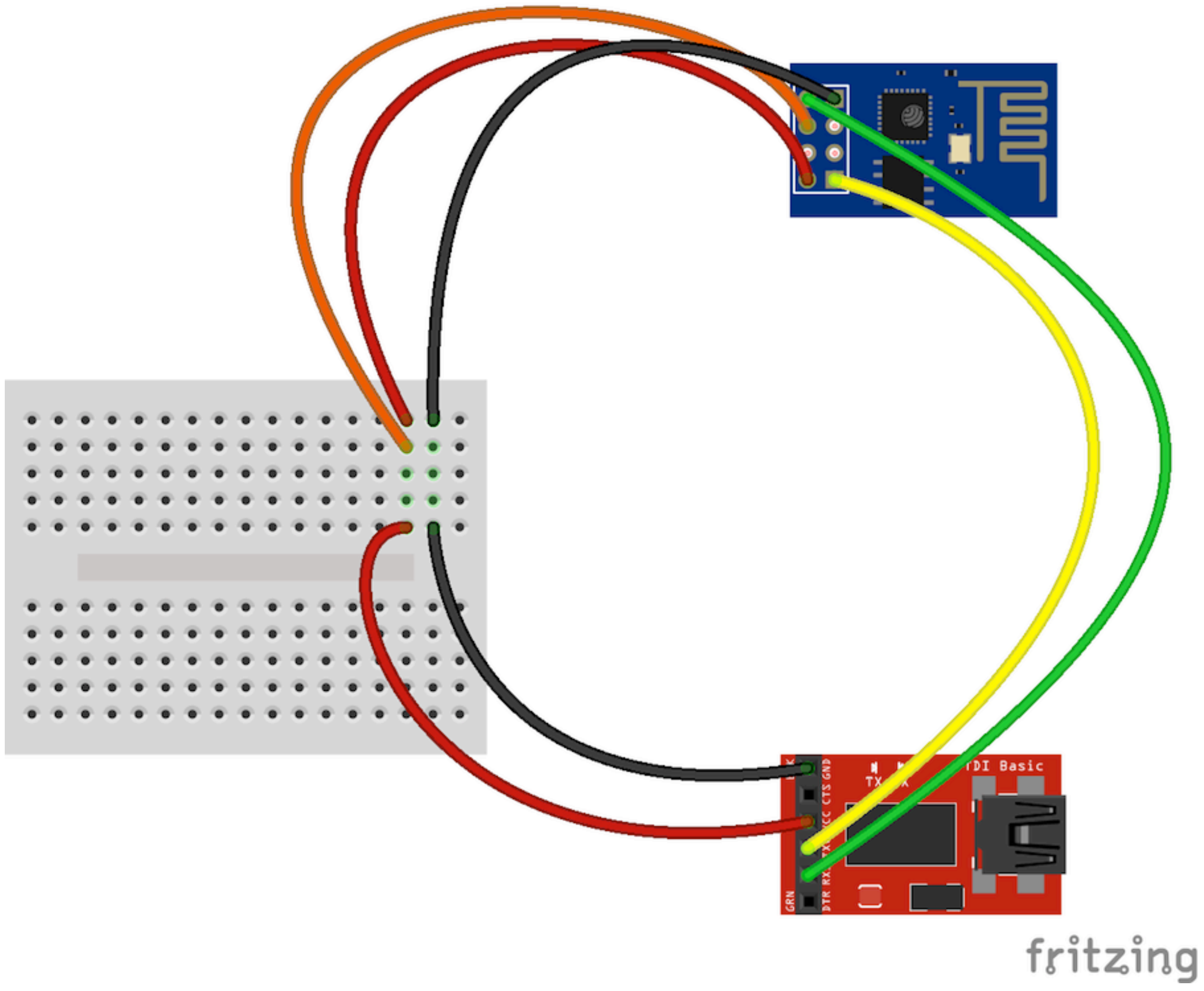
ESP8266 WiFi Pinout
Top View (Not to scale)

When power is applied to the module you should see the red power light turn on and the blue serial indicator light flicker briefly.

With FTDI 3.3V Board (Legit)

If you have a 3.3V FTDI Serial to USB board you can get started without fear of destroying your new ESP8266 WiFi module. Do note that many FTDI boards have a solder jumper to convert from 3.3V to 5V operation so ensure it is set to enable 3.3V operation.

Here are the connections required to enable communication with the module over serial:



With Arduino Uno R3 (Quick but not very legit)

DO NOT DO THIS! You may break your Arduino or your ESP8266 WiFi module.

This method is a bad idea for a number of reasons:

- The ESP8266 does not have 5V tolerant inputs–you could destroy your WiFi module.
- The ESP8266 may draw more current than the 3.3V regulator on your Arduino can supply–you could damage your Arduino.
- The operation of the ESP8266 outside of stated limits may be unstable and unreliable–you could damage your sanity.

This method does have one redeeming feature:

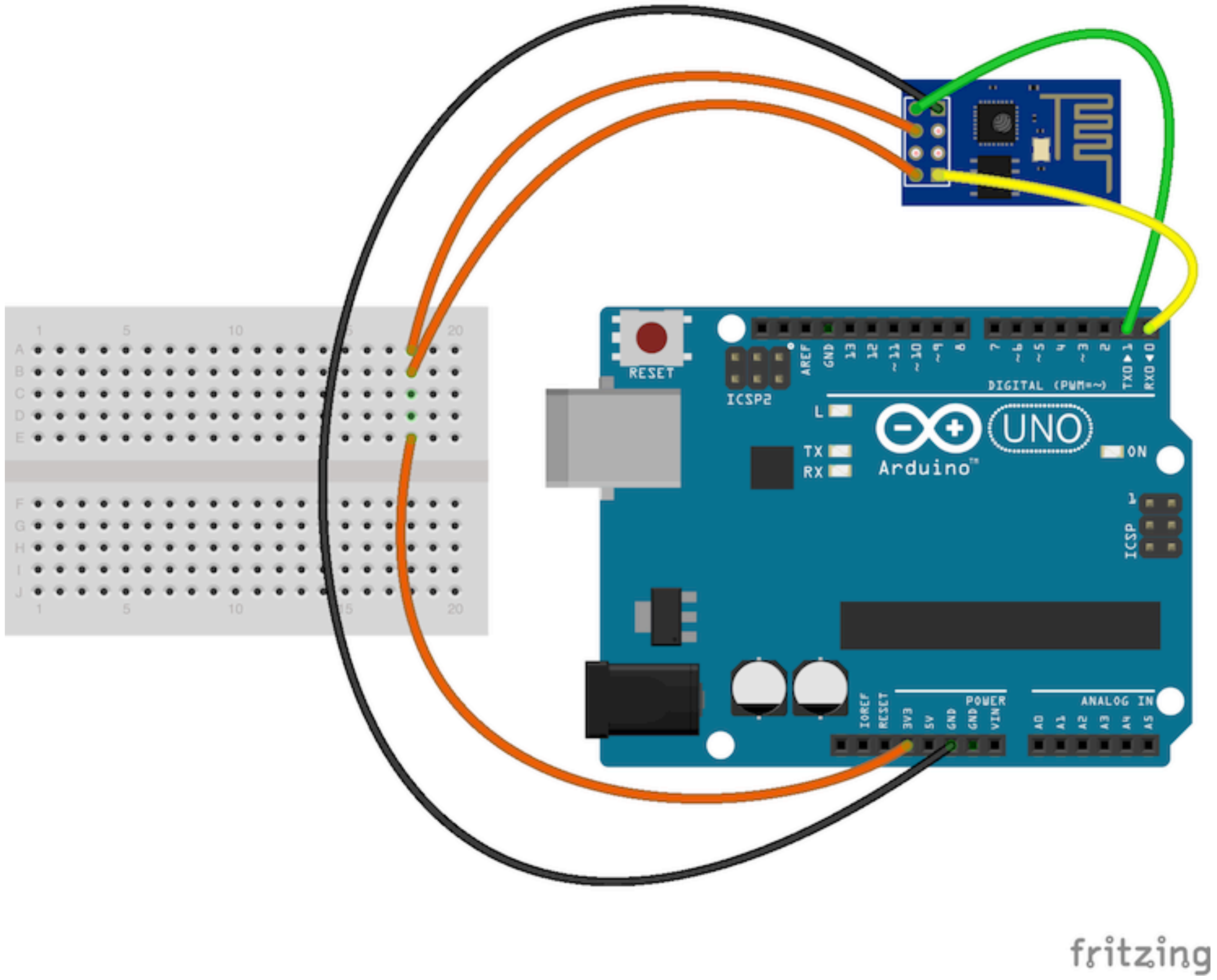
- If you already have an Arduino Uno R3 it's really easy to get started and (mostly) worked for me. :)

We're essentially going to use your Arduino as a power supply and USB to serial converter so you need to ensure there is no sketch that interferes with the serial connection. There are a couple of ways to achieve this:

- Upload an "empty" sketch that doesn't use the serial connection, e.g. `BareMinimum` from the examples menu; or,

- Remove the microcontroller from the Arduino board (if it is a DIP form factor–don't undersolder an SMD one).

Here are the connections required to enable communication with the module over serial in this configuration:



Serial Control

With the hardware connections in place you can communicate with the WiFi module through a serial terminal.

Using Arduino IDE Serial Monitor

If you already have the Arduino IDE installed the easiest way to get started is to use the built-in Serial Monitor:

1. Plug in the WiFi module.
2. Choose the correct serial port from the `Tools > Serial Port` menu.
3. Open the Serial Monitor via the `Tools` menu or “magnifying glass” icon on the editor window.
4. For the default firmware version (00160901), ensure `Carriage return` is selected in the line ending pop-up menu at the bottom of the serial monitor. (For later versions it must be set to `Both NL & CR`.)
5. For the default firmware version, ensure the communication speed is set to `115200 baud`. (For later versions or if it has been modified it will need to be `9600 baud` or similar.)

You’re now ready to send your first commands!

Using GNU Screen

It’s possible to use GNU Screen out of the box with the default version of the firmware (00160901) which expects Carriage-Return-only line endings, e.g. (on OS X):

```
        screen /dev/tty.usbserial-AB12345 115200
```

Unfortunately the updated firmware versions require Carriage-Return-and-New-Line line endings and there appears to be no way to configure `screen` to send both with one key press. Instead, you need to press `<enter>` or `Ctrl-M` then follow that with `Ctrl-J`.

You might have more success with something like `minicom` or `picocom` with later firmware versions.

Now *you’re* ready to send your first commands!

First Commands

1. Ensure `AT` commands are received correctly (the `AT` seems not to be case sensitive but the rest of any command *is* case sensitive):

   ```
       AT
   ```

   Response:

   ```
       OK
   ```

   If you don’t get this response check:

   - Hardware connections (the blue LED on the board should flash). Try swapping RX & TX.
   - Correct baudrate–should be 115200 on the default firmware version (00160901).
   - Correct line endings–should be Carriage Return only for default firmware version (00160901).

   If it still doesn’t work, give up, you’ve probably got better things to do with your life…

2. Ensure the module is in a known state by issuing a reset command:

```
AT+RST
```

Response:

```
 ets Jan  8 2013,rst cause:4, boot mode:(3,6)

wdt reset
load 0x40100000, len 24444, room 16
tail 12
chksum 0xe0
ho 0 tail 12 room 4
load 0x3ffe8000, len 3168, room 12
tail 4
chksum 0x93
load 0x3ffe8c60, len 4956, room 4
tail 8
chksum 0xbd
csum 0xbd

ready
```

Or similar. The important part is the `ready` on the last line.

3. Check the firmware version:

   ```
   AT+GMR
   ```

   The stock firmware produces this output:

   ```
   00160901
   ```

4. Enable the module to act as both a “Station” and an “Access Point”:

   ```
   AT+CWMODE=3
   ```

   Response:

   ```
   OK
   ```

   (You *may* need to perform another reset after changing this setting.)

5. List surrounding WiFi networks.

   First, if you’re on (at least) the `00160901` firmware you need to work around an apparent firmware bug that hangs on listing WiFi networks if the last joined network is no longer available.

   The following command ensures no network details are stored for connection (you should get an `OK`) response:

```
AT+CWJAP="",""
```

(You can check to see what the currently stored network details are with:

```
AT+CWJAP?
```

which should generate the following output if no network details are stored:

```
+CWJAP:""
```

With a later firmware or if the last joined network is still in the vicinity you shouldn't need to do this pre-step.)

Now you send the *actual* command to list networks in the vicinity:

```
AT+CWLAP
```

You should get a response like:

```
+CWLAP:(0,"",0)
+CWLAP:(3,"WiFiArtThouRomeo",-80)
+CWLAP:(3,"Free Public WiFi",-51)
+CWLAP:(3,"Guest",-91)

OK
```

Sometimes you might get `ERROR` or no response–in which case you'll have to try doing the USB plug/unplug cycle and try again. (Or the same with the ground jumper.)

6. Join a suitable WiFi access point:

```
AT+CWJAP="<access_point_name>","<password>"
```

You should get an `OK` response.

For example, with the above list of access points you might use:

```
AT+CWJAP="Guest","MyVoiceIsMyPasswordSoWhyDoINeedThis?"
```

You can check if the module has been allocated a IP address with:

```
AT+CIFSR
```

You should get your current IP address in response, e.g:

```
192.168.1.2
```

Acting as a TCP Client

You can connect to an internet server (e.g. a web server) with the following method:

1. Enable multiple connections (you need an `OK` response):

   ```
   AT+CIPMUX=1
   ```

2. Specify which connection channel you wish to connect on (`0` - `4`), the protocol type (TCP/UDP), the IP address (or domain if you have DNS access) and the port number using the `CIPSTART` command:

   ```
   AT+CIPSTART=4,"TCP","google.com",80
   ```

   You should receive the response `OK` followed by `Linked` when the connection is open:

   ```
   OK
   Linked
   ```

3. Next you need to specify how much data you wish to send (after specifying which channel). In this example we're going to send "`GET / HTTP/1.0\r\n\r\n`" which is 18 bytes:

   ```
   AT+CIPSEND=4,18
   ```

   This time, instead of an "OK" response your will get a `>` prompt:

   ```
   >
   ```

   This indicates the module is waiting for you to send the 18 bytes of data.

   Here it gets a bit messy if you're using the Arduino serial monitor as you have to swap between the line ending the module requires ("Carriage return" only) and what the HTTP server is expecting ("Both NL & CR"). Change the setting to *Both NL & CR* and send the following (you will need to press `Send` a second time to send the empty line the HTTP server expects):

   ```
   GET / HTTP/1.0
   ```

   The module should respond with:

   ```
   SEND OK
   ```

   Now you should change the line ending setting to *Carriage return* only so you can send additional commands.

   The module should provide a second response once the web server responds:

   ```
   +IPD,4,530:
   ```

The `4` indicates it's data from connection channel 4 and the `530` indicates there's 530 bytes of data. You should now see the data:

```
HTTP/1.0 302 Found
Cache-Control: private
Content-Type: ...
```

Woo! You just put a thing on the internet…time to go find a VC!

You'll likely get an `OK` response or two and then eventually an indication that the server has closed the connection:

```
Unlink
```

Acting as a TCP Server

You can enable the module to *accept* TCP connections (i.e. act as a server) in the following manner:

1. Connect to a WiFi access point.
2. Enable multiple connections.

   ```
   AT+CIPMUX=1
   ```

3. Find out the IP address of the module:

   ```
   AT+CIFSR
   ```

   Note the response, e.g.:

   ```
   192.168.1.2
   ```

4. Set the module to listen (first parameter, `mode` is set to `1`) for a connection on a specific port (in this case port 1336):

   ```
   AT+CIPSERVER=1,1336
   ```

5. From another device on the same network connect to the listening port, e.g. with telnet:

   ```
   telnet 192.168.1.2 1336
   ```

   The module should display:

   ```
   Link
   ```

   Type some text into the telnet session, e.g.:

```
KiwiconAte!
```

The module should display the following (`0` being the connection channel, `13` being the length of the data received:

```
+IPD,0,13:KiwiconAte!
OK
```

You can send data in response with the `CIPSEND` command as used previously, e.g. (`0` is the channel, `8` is the length of the data):

```
AT+CIPSEND=0,8
```

The module will display the prompt:

```
>
```

Then you can send the data, e.g.:

```
WhatEvs
```

And then the module will respond with:

```
SEND OK
```

The telnet session should now display:

```
WhatEvs
```

You can then end the telnet session with by pressing `Ctrl-]` and `q<enter>`, the module will display:

```
Unlink
```

Acting as a WiFi Access Point

In addition to connecting to WiFi Access Points the module can also act *as* an Access Point–this means you can connect devices to the module without any other network infrastructure in place. Ideal for a local private shared "drop box" perhaps…

1. The module comes with an access point pre-defined (SSID of "ESP_…") but you can define your own with:

```
AT+CWSAP="NoWorriESSID","password",3,0
```

The first parameter is the SSID name; the second parameter is the password; the third the WiFi channel–pick one not used in your area; and, the final parameter is the encryption standard to use. An encryption value of `0` turns encryption off which means the password is ignored–but it still can't be an empty value. I couldn't get any encryption to work though (it would always create an unencrypted network) you might have more luck–possibly with a more recent firmware…

2. To actually enable the network to be created you need to set the "WiFi mode" of the module to "AP" (`2`) or "Both" (`3`):

   ```
   AT+CWMODE=3
   ```

   Now you will be able to connect to your module as an access point from another device (e.g. a laptop or a phone).

3. You can list the IP address etc of any device connected to the network with:

   ```
   AT+CWLIF
   ```

   Which generates the response:

   ```
   192.168.4.100, [...]
   ```

4. Now you can run the server example from above and connect–note that the module always has the IP `192.168.4.1` when acting as an AP.

Updating the firmware

You'll be pleased to know you're not stuck with the ample list of features that ship with your module, no, because you can update the firmware of your module! Yes, that means you can get *new* features & *new* bugs too…

Finding New Official Firmware

It's not particularly easy to find new official firmware versions–the links are spread over forums, wikis and Google Drive accounts which don't look particularly legit.

It seems like the best source of details is the "Updates" section of the "Firmware" section on this page: http://www.electrodragon.com/w/Wi07c#Firmware

This ridiculous URL (from the link entitled "Find most upated firmware on this google link" on the above ElectroDragon page) seems to list all the official firmware updates: https://drive.google.com/folderview?id=0B3dUKfqzZnlwR2pVWjg3NXRVdW8&usp=drive_web&tid=0B3dUKfqzZnlwRjFaNTUzZFptbzg#list

Consider uploading one of the following versions:

- `ESP8266_flasher_V00170901_00_Cloud Update Ready.zip`
- `ESP8266_AT_V00180902_02_baudrate watchdog added.zip`

(A note on version numbers: the version number `00160901` is made up of two parts: `0016` is the

SDK version and `0901` is the AT command set version.)

Once you've updated the firmware you also have the option of "Cloud Updates" but I've not managed to get any cloud update to work yet: http://blog.electrodragon.com/cloud-updating-your-wi07c-esp8266-now/

Firmware Update Tools

While there are other tools for updating the firmware, for this example we'll use `esptool` https://github.com/themadinventor/esptool/ which is cross platform and GPL licensed.

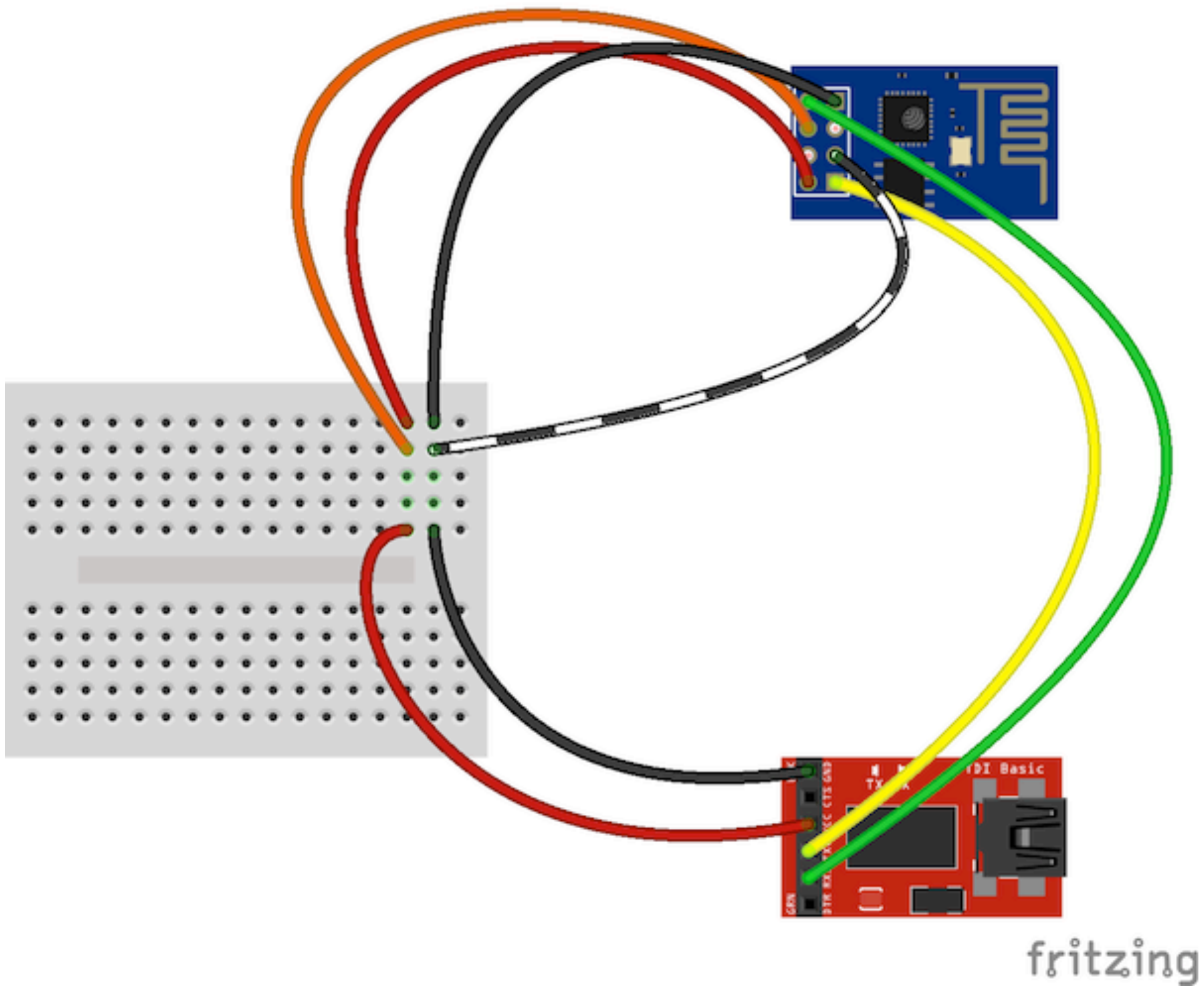
Download the script file from https://raw.githubusercontent.com/themadinventor/esptool/master/esptool.py.

Hardware Connections and Performing the Update

In order to update the firmware you need to configure the hardware to enter firmware update mode:

1. Disconnect the module from your computer.
2. Connect GPIO0 to ground (0 Volts) ("pulled low") to enable firmware update mode.
3. Reconnect the module to your computer.

The module should now be in firmware update mode.



(Note that the `esptool` docs say to ensure “GPIO2 is pulled high” but this doesn’t appear to be necessary.)

Execute this command to perform the firmware update (the `.bin` file is from the `ESP8266_AT_V00180902_02_baudrate watchdog added.zip` archive):

```
    python esptool.py --port /dev/tty.usbserial-ABC12345 write_flash 0x000000 "
v0.9.2.2 AT Firmware.bin"
```

You should receive progress updates during the upload process:

```
    Connecting...
    Erasing flash...
    Writing at 0x0007ec00... (100 %)

    Leaving...
```

If you have problems connecting and have been using the Arduino IDE Serial Monitor ensure it isn’t open when trying to perform the upgrade. (Ask me how I know…)

Once the update has completed:

1. Disconnect the module from your computer.
2. Remove the connection from GPIO0 to ground.
3. Reconnect the module to your computer.

You should now have a new firmware version installed and functioning.

Note:

- The startup behaviour of the module is now different–it displays different output.
- The module now defaults to communicating at 9600 baud.
- The module now expects both carriage return and new line characters at the end of every line.
- New AT commands will be available and some bugs may have been fixed!

Arduino and ESP8266

Up until now you’ve been manually controlling the WiFI module via the serial console. It’s obviously possible to control the module programmatically via an Arduino sketch. At present there doesn’t seem like a “preferred” choice for an Arduino library to abstract the functionality but a few searches will find you Arduino examples to start with and presumably over time the Arduino community will settle on a preferred library.

ESP8266 SDK and Custom Firmware

An official Software Development Kit (SDK) has been released for the System-on-Chip (SoC) controller which powers the ESP8266 WiFi module. Using the SDK it’s possible to add extra features to the AT command firmware or even create a standalone firmware.

Here’s a couple of custom firmwares to check out…

NodeMcu

Lua based firmware for WiFi-SoC ESP8266: https://github.com/nodemcu/nodemcu-firmware

Very cool and easy to get started with.

Download the firmware:

```
    wget https://github.com/nodemcu/nodemcu-firmware/raw/master/0.9.2/512k-flas
h/nodemcu_512k.bin
```

Flash the firmware to the module:

```
    python esptool.py --port /dev/tty.usbserial-ABC123456 write_flash 0x000000
nodemcu_512k.bin
```

Connect to the module:

```
    screen /dev/tty.usbserial-ABC123456 9600
```

Do the obvious…

```
    print("hello")
```

For more details look at:

- https://github.com/nodemcu/nodemcu-firmware#readme
- https://github.com/nodemcu/nodemcu-firmware/wiki/nodemcu_api_en

AT Command GPIO control

There is a custom firmware with AT commands added to read and write the General Purpose Input/Output (GPIO) pins available–this enables you to control LEDs and read buttons with AT commands.

For more details see: http://blog.electrodragon.com/esp8266-gpio-test-edited-firmware/

More resources

- Espressif Systems (Designer): https://espressif.com/en/products/esp8266/
- Espressif Systems Github: https://github.com/espressif
- ESP8266 Community Forum: http://www.esp8266.com/
- NURDspace on ESP8266: https://nurdspace.nl/ESP8266
- First setup of ESP8266 SDK: https://nurdspace.nl/ESP8266/First_setup
- ElectroDragon on ESP8266: http://www.electrodragon.com/w/Wi07c

Finally…

Have fun!

Find updates to this guide and more information at http://www.labradoc.com/i/follower/p/notes-esp8266.

This getting started guide is brought to you by Kiwicon 8 and rancidbacon.com!