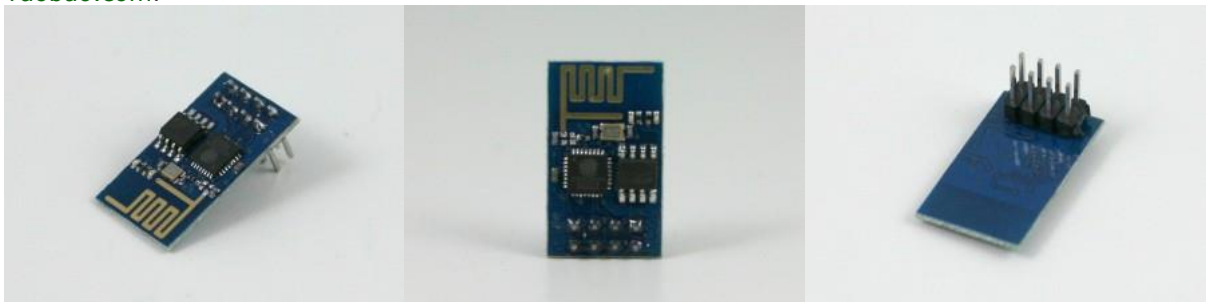


First Impression on the ESP8266 Serial-to-WiFi Module

Oct 2nd, 2014 by ray

Check my new blog post on the **ESP8266 Toy**

Continuing from my previous blog post about [Hi-Link HLK-RM04 module](#), I have finally received the [ESP8266](#) Serial-to-WiFi module that I've been waiting for. As I said previously, with the popularity of IoT devices, there is an increasing demand for low-cost and easy-to-use WiFi modules. ESP8266 is a new player in this field: it's tiny (25mm x 15mm), with simple pin connections (standard 2x4 pin headers), and best of all, it's extremely cheap, less than US\$3 from [Taobao.com](#)!



What is Serial-to-WiFi? Simply put, it means using serial TX/RX to send and receive Ethernet buffers, and similarly, using serial commands to query and change configurations of the WiFi module. This is quite convenient as it only requires two wires (TX/RX) to communicate between a microcontroller and WiFi, but more importantly, it offloads WiFi-related tasks to the module, allowing the microcontroller code to be very light-weighted.

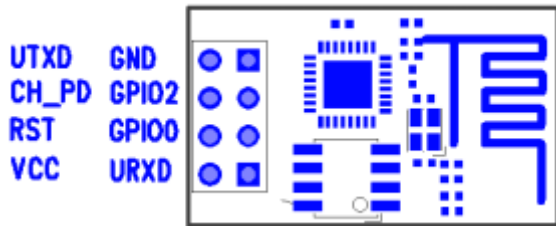
There are already a lot of excitements and resources you can find online about ESP8266. I've included a few links below:

- [English Datasheets and Information](#) (nurdspace.nl)
- [Community Forum](#)
- [Hack A Day project](#)

These are great resources to reference if you need help working with ESP8266. Below I document my own experience. I've also bought a few extra and put them available on the [Rayshobby Shop](#) for anyone who is interested in buying the module and don't want to wait for the long shipping time from China 😊

Check my new blog post on the **ESP8266 Toy**

Pin Connections. ESP8266 is sold in several different versions. The one I received is the version with 2×4 male pin headers, and PCB antenna. In terms of the form factor, it looks a lot like the nRF24L01 2.4G RF transceiver. Here is a diagram of the pins:

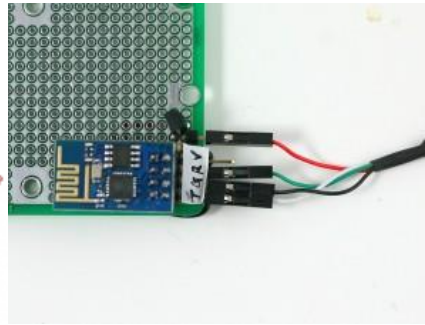
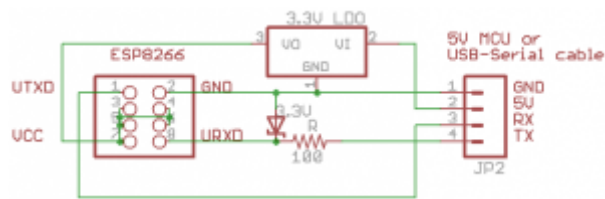


Connect the top two pins (**UTXD, GND**) and bottom two pins (**VCC, URXD**) to the **RXD, GND, VCC, TXD** pins of a microcontroller. Note that **VCC must be no more than 3.6V**. The **middle four pins** are should be **pulled up to VCC** for normal operation. However, if you need to upgrade the firmware of the module, you need to pull the GPIO0 pin to ground — that way upon booting ESP8266 will wait for a new firmware to be uploaded through serial. This is how you can upgrade the firmware in the future.

A few quick notes for connection:

- The typical operating voltage is 3.3V (acceptable range is 1.7V to 3.6V). As the module can draw up to 200 to 300mA peak power, make sure the power supply can deliver at least 300mA. For example, the 3.3V line from a USB-serial cable would be barely sufficient, in that case it's better to use a LDO to derive 3.3V from the 5V line.
- When using the module with a 5V microcontroller, such as a standard Arduino, make sure to use a level shifter on the URXD pin — a simple resistor-zener level shifter is sufficient. Again, this is to prevent over-voltage.

A schematic will make it clear. See below. In my case, I soldered the components and a matching female 2×4 pin header to a perf-board. This way I can easily plug in and unplug ESP8266. Again, if you are using a 3.3V microcontroller, you can do away with the LDO and zener diode.



Experiments using a USB-Serial Cable. Before connecting to a microcontroller, it's a good idea to use a USB-Serial cable (such as the inexpensive [PL2303 USB-serial converter](#)) to check out the basic functions of the module. Connect the PL2303 cable with ESP8266 according to the schematic above. Then open a serial monitor (such as `asgtkterms` in Linux and `putty` in Windows) with 11520 baud rate (my ESP8266 seems to be set to 115200 bard rate; earlier versions use 57600). Then you can use a list of **AT commands** to talk to ESP8266. The AT commands are pretty well documented on [this page](#). Below are some example input (shown in bold font) and output that show how to reset the module, list available WiFi networks, check the WiFi network it's connected to, list IP address, and firmware version etc.

AT

OK

AT+RST

OK

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

wdt reset

... ..

chksum 0x46

csum 0x46

ready

AT+CWLAP

+CWLAP:(0,"",-49)

+CWLAP:(3,"freely",-49)

OK

AT+CWJAP?

+CWJAP:"freely"

OK

AT+CIFSR

192.168.1.130

AT+GMR

00150900

OK

A Simple Demo using Arduino.

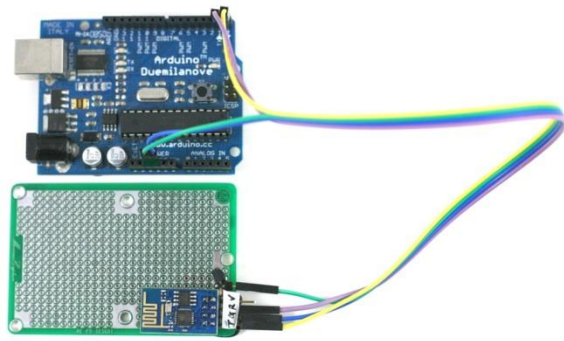
Next, I connected ESP8266 to an Arduino. Because Arduino is already using the TX/RX pins for bootloader, make sure to **unplug ESP8266 while flashing the Arduino**, otherwise you may not be able to upload a sketch successfully. Also, you can't use TX/RX for printing debugging information, since ESP8266 will be using them to communicate with Arduino. Instead, you can use another pair of pins (e.g. D7 and D8) as software serial pins, and use a PL2303 serial cable to monitor the output. This will help print debugging information.

I've also experimented with using software serial to communicate with ESP8266, but that has failed — ESP8266 requires 115200 baud rate, and that's a little beyond the capability of software serial. So you have to stick with the hardware TX/RX pins.

- **Download ESP8266 Arduino Example** (Updated Dec 14, 2014)

(**Update:** the code has been revised on Dec 14, 2014 to improve robustness, particularly for some of the latest ESP8266 firmwares. Right now there is still an issue that if the browser is closed before the transfer is completed, it may leave ESP8266 in the **notorious 'busy s' mode**, the only solution to which is to do a hard reset. If using the module in real products, make sure you have a way to use a microcontroller pin to reset the power of the module, thus providing a way to hard reset the module. It looks like future firmwares may be able to address this in software.)

The demo program first configures ESP8266 to log on to your WiFi network (SSID and password are given as macro defines at the beginning), then it sets ESP8266 as HTTP server with port number 8080, and it listens to incoming request. If you open a browser and type in `http://x.x.x.x:8080` where x.x.x.x is the module's IP address (printed to soft serial pins), you will see the output which is a list of analog pin values, and the page refreshes every 5 seconds. So this is a basic Hello World example that shows how ESP8266 can be configured as an IoT server, responding to incoming requests.



Challenges. While my initial experiments with ESP8266 have been quite successful, I've also encountered minor issues that took me a while to figure out. For example, while the AT commands are well documented, they don't seem extremely consistent — some commands allow question marks at the end, some don't. I also see variations of the returned values from running the AT commands: sometimes there is an extra end of line character, sometimes there is none. These basically require a robust software library to handle all possible cases.

Overall I would say ESP8266 is a very promising WiFi module for IoT, particularly open-source IoT gadgets, because of its low cost, compact size, and the community development. It seems the manufacturer has also open source the firmware code, and thus the minor issues can probably be easily fixed through a firmware upgrade.

We have a small number of ESP8266 modules available **in stock**, and will continue to offer them if there are sufficient interests. Thanks!