# ESP8266 Wi-Fi Transceiver Adapter

# Introduction

The ESP8266 is a low cost Wi-Fi System on Circuit (SoC) module with a Wi-Fi and UART interface and a rich array of other features. The device provides a processing unit with various registers and memory capabilities. This means the device is able to hosts its own application as a standalone device with just power hooked up. This Wi-Fi module provides General Purpose Input Output (GPIO) pins, which can be hooked up to external devices to collect data or send control signals. Configuration settings of the ESP8266 are done using AT commands provided by the data sheet. This is a neat and powerful device for the low cost it comes at.

Unfortunately this device has few issues when it comes to resources and information on the use of certain features. This is a new device that has come straight out of China with just a single data sheet all in Chinese. The data sheet was converted unprofessionally to German and then to English. Some of the configuration details to allow the use of certain features are still a mystery. Word is out that the device is “top secret”. There is great community of people working together to discover the full potentials of this device. There is even in fact a website dedicated to discovering its full features at Hackaway.io. The great thing about these devices is the fact that they are reprogrammable and newer and better versions of “hacked firmware” and additional libraries are being released all the time. The issues right now for people writing new firmware, is trying to understand the underling hardware and its registers which have been made hidden.

Luckily for the purpose of this project there is just enough information to configure and setup a TCP/UDP connection that would send and capture packets.

One of the downsides of this module is its power consumption. The device during an idle operation mode can consume up to 400 mA of current and 600 mA during communication mode. It is recommended that power supply should not come from a microcontroller as they can’t supply those sorts of current ranges. For some people using a microcontroller worked and for some it didn’t. One must consider that if a microcontroller draws that much current, to feed this Wi-Fi module, that it can damage the microcontroller. If there is nothing to lose then why not try. For this project hooking up the ESP8266 to the ATMega328 gave no hassle at all and worked well. The USB supplies enough current to drive both the microcontroller and Wi-Fi adapter. When using a battery as a power supply, the battery must be good in that provides the necessary current.

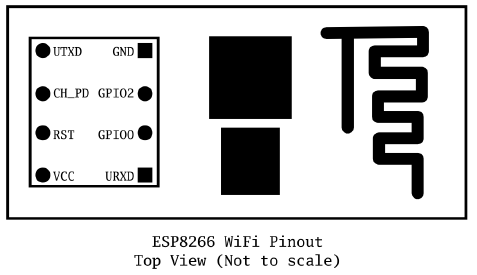
# Purpose of Module

The purpose of the module is to establish a Wi-Fi interface on the robot and make the robot behave as a TCP client within a Wi-Fi network. The Wi-Fi standard will be used to send and received TCP/IP packets to and from the robot. The aim to is be able send and receive data from anywhere around using any browser on the internet.

# Specifications

|  |  |
| --- | --- |
| **Parameters** | **Column1** |
| Protocol | 802.11 b/g/n |
| System on Circuit (Soc) | 32 bit CPU, Memory and Wi-Fi |
| GPIO Pins | 2 |
| Reset Pin | Yes |
| Wi-Fi Direct (P2P) | Yes |
| Voltage | 3.3 V |
| Current | 400~600 mA |
| CPU | 32-bit micro-CPU |
| Memory | ROM & SRAM |
| UART | 3.3 level UART interface |

# Pin Layout

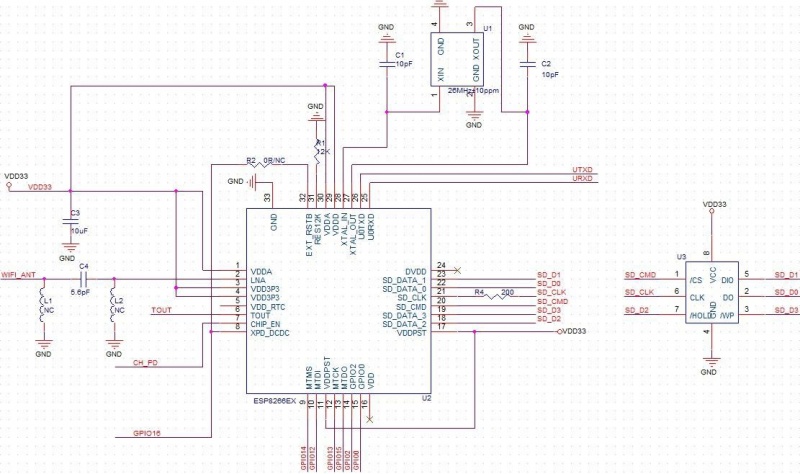


Below shows a table of the pin configurations of the module.

|  |  |
| --- | --- |
| **Pins** |  |
| VCC | 3.3 V |
| GND | Ground |
| TXD | Transmit Pin |
| RXD | Receive Pin |
| CH\_PD | Chip Power Down |
| GPIO0 | General Purpose I/O 0 |
| GPIO2 | General Purpose I/O 2 |
| RST | Reset |

Note: It is important the module only get a maximum of 3.3 V volts as anything greater can destroy the module. It does not provide input impedances to protect against higher voltages. Before hooking this up to a microcontroller, which usually supplies 5 V, there must be a voltage level shifter or a voltage regulator to reduce the voltage. As mentioned earlier it is not recommended to hook this up to the microcontroller because of the current constraints. For some people it does work and for others it doesn’t. In the case of this project it does work using a 9 V battery with a Voltage regulator (5 V).

# Architecture



# Applications

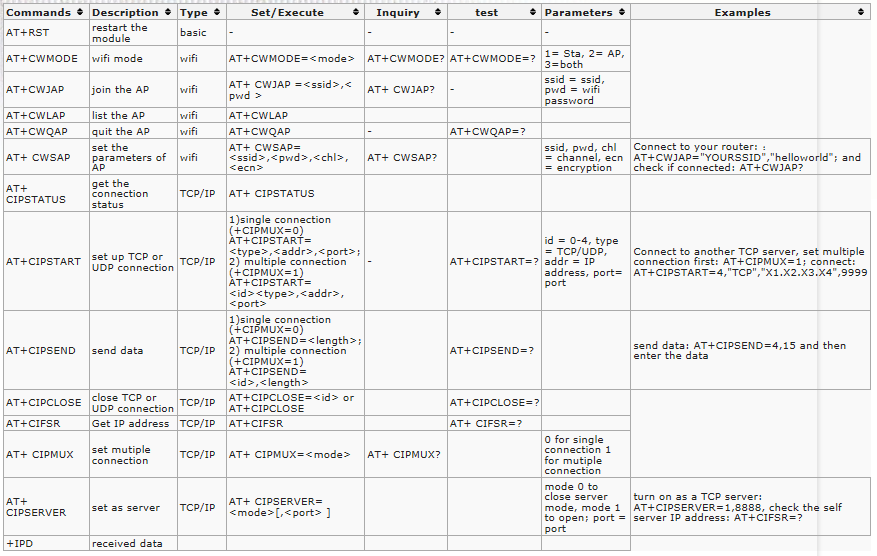
This Wi-Module can be used in various applications that require a Wi-Fi connection. Some of these applications include:

* Mesh network
* industrial wireless control
* Baby Monitor
* Network Camera
* Sensor networks

For the purpose of this project, the module is being used to connect the robot to the internet.

# AT Commands

Below shows a table of all the various commands that can be used in setting up a Wi-Fi connection. The Wi-Fi module would first have to connect to a Wi-Fi modem or router before using Post and Get requests for particular URLs.



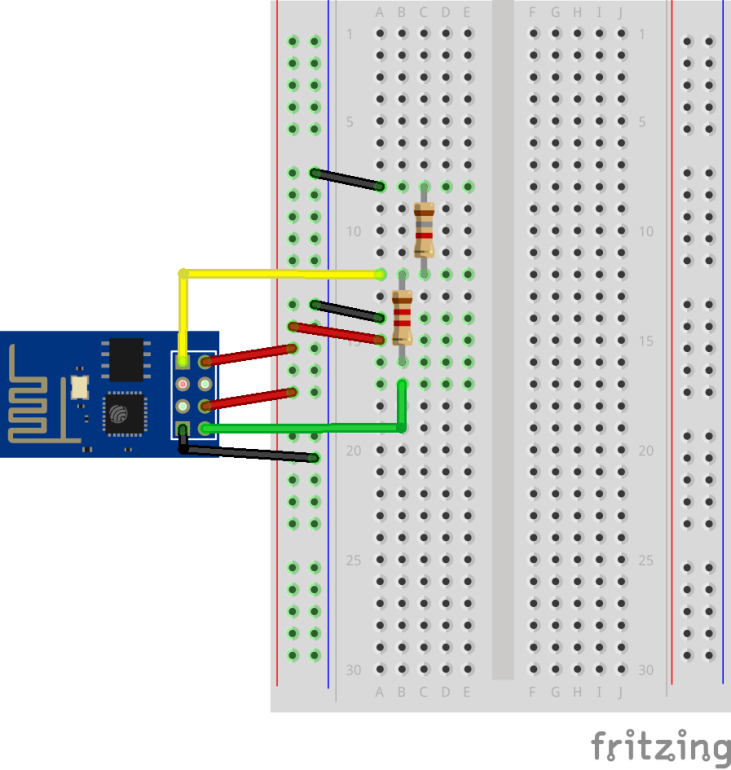
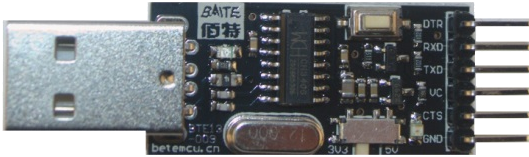
# Operation Test Procedure

This is a section where by the ESP8266 was tested to see if it works and get a sense of how it works. The circuit setup is very easy where by the USB to Serial adapter is connected to the Wi-Fi adapter with the addition of a small circuit. The setup will need a voltage divider circuit at the RX pin of the Wi-Fi which will reduce the voltage to 3 V. The RX pin of the Wi-Fi uses a maximum voltage of 3.3 V. The Wi-Fi module will receive a power supply of 3.3 V as the USB to Serial adapter offers that option. The main reason for this test is to test out AT commands by sending them through the serial monitor of the Arduino IDE. Any data received would be captured and displayed in the serial monitor.

## Components

* CH340 CH340G USB TTL Serial Adapter.
* 1.2 kΩ and 1.8 kΩ Resistors
* Breadboard.
* ESP8266 Wi-Fi Adapter.

## Schematics



**Switched to 3.3 V**

## Connection Configurations Using AT commands

**Connecting to Wi-Fi modem or router**

1. Open up the Serial monitor of the Arduino IDE and set the baud. These modules come with different baud rates depending on what firmware is installed. In the case of this project the given baud rate is 9600 Mbps. Power up the ESP8266 module. The result below indicates the module is ready to use.

´tpjä¤¤ôØ$<$$8üp`>äì•$¼p¬

[System Ready, Vendor:www.ai-thinker.com]

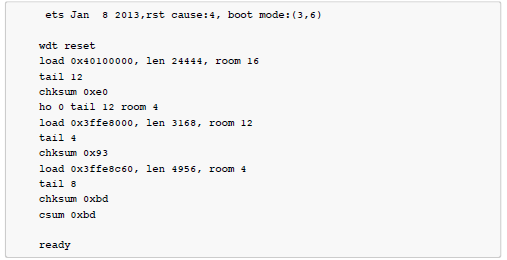
1. To test that AT commands work with the firmware the command “AT” must be entered. Note that it is not case sensitive. The response:



1. Reset the module to default settings using the command:



The response should be:



1. Enable the module to act as station by entering:

AT+CWMODE = 1

This means the module is being configured to act as a wireless client to an access point (AP).

Response:



1. Connect the module to an access point using:



Where <access\_point\_name> is the name of the wireless modem or router and the <password> is the associated password.

Response:



**Connecting as a TCP Client to internet**

A connection to a Wi-Fi modem only need be done once. Connecting as a TCP client to the internet the following commands need to always be entered.

1. To enable multiple connections to internet the following command must be entered:



Response:



1. Next command specifies which channel (0-4), the protocol (TCP/UDP) the IP address (domain name) and the port number that will be used.

AT+CIPSTART=0,”TCP”,”tixgy.com”,80

Response:



1. The next command is used to specify a channel and how much data will be sent to that channel.

AT+CIPSEND=0,100

Channel 0 was chosen and 100 bytes of data was chosen to be sent through that channel.

Response:



This is an indication that the module is waiting to send 100 bytes of data.

1. The following data can put in:

> GET http://www.tixgy.com/DITrobot.php HTTP/1.0

The data entered is a GET request that will fetch all the html code of the specified URL. The HTTP/1.0 indicates that HTTP 1.0 should be used which is upper layer protocol of TCP.

Note: When entering data less than the specified amount of data chosen to send, the return key must be hit a couple of times to fill in the empty spaces so the specified amount is reached.

Response:

AT+CIPSEND=0,50

> GET [http://www.tixgy.com/DITrobot.php HTTP/1.0](http://www.tixgy.com/DITrobot.php%20HTTP/1.0)

SEND OK

+IPD,0,1460:HTTP/1.1 200 OK

Date: Mon, 01 Jun 2015 16:08:45 GMT

Server: Apache

X-Powered-By: PHP/5.2.17

refresh: 10;

Content-Length: 2777

Connection: close

Content-Type: text/html

<html>

<head><title>DIT robot control interface</title>

<body>

<table border="1">

<tr>

<td>ID</td>

<td>Battery Power (%)</td>

<td>Temperature (C)</td>

<td>Pressure (Pa)</td>

<td>Altitude (m)</td>

<td>Date/Time</td>

</tr>

Data can be given parameters that can be passed to a webpage. This done by appending at the end of the URL: “?param1=some\_number&param2=some\_number”. Param1 and Param2 are defined parameters to the website. Each of these parameter values is made to equal som\_ number. An example of sending parameters to a website:

GET http://www.tixgy.com/DITrobot.php/?param1=1&param2=2&param3=3&param4=4 HTTP/1.0

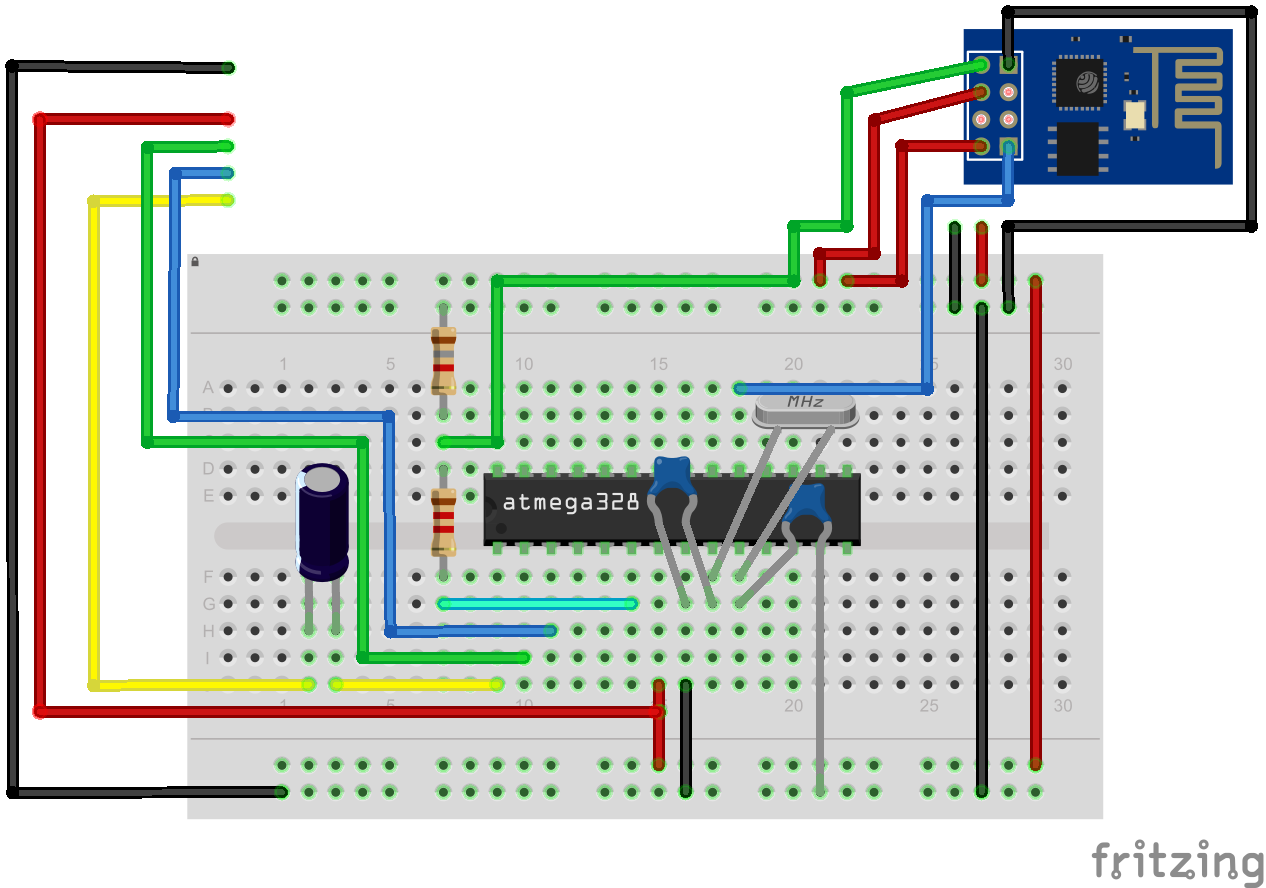
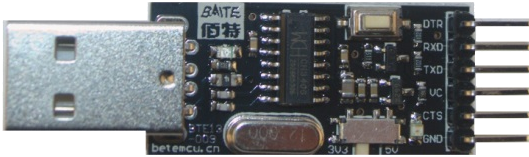
# Implementing ESP8266 to ATMega328-PU

For this part of the project the AT commands need to be written to the ESP8266 module using the ATMega328-PU. This can be done in the code using the Serial Library to send the AT commands. The port then can send data and receive data by listening at the port. Below demonstrates how this can be done using the schematic diagram and two functions. A virtual UART serial port is used to connect the ESP8266 Wi-Fi adapter to the ATMega328-PU.

## Components

* 1.2 kΩ and 1.8 kΩ Resistors
* Breadboard.
* ESP8266 Wi-Fi Adapter.

## Schematics



## Code Using AT commands for ATMega328-Pu Microcontroller

**connectToWiFiModem()**

This function is used to call the appropriate AT commands to connect the ESP8266 to a WiFi modem. The SSID and password are random and these should be replaced by real ones of a modem. This functions only needs to be called once in the program.

void connectToWiFiModem**()**

**{**

Serial**.**write**(**"AT+RST\r\n"**);** //Reset ESP8266 to default configurations.

delay**(**3000**);** //Create a delay to ensure reset in fully complete.

Serial**.**write**(**"AT+CWMODE=1"**);** //Configure the module to be a station.

Serial**.**write**(**"\r\n"**);**

//Command to connect to Wi-Fi modem or Wi-Fi server.

//The command is not printed out in one line because of interferences

//caused by the "" in the command.

//Connect to WiFi modem.

Serial**.**write**(**"AT+CWJAP="**);**

Serial**.**write**(**'"'**);**

Serial**.**write**(**"ILoveIreland"**);**

Serial**.**write**(**'"'**);**

Serial**.**write**(**','**);**

Serial**.**write**(**'"'**);**

Serial**.**write**(**"1234"**);**

Serial**.**write**(**'"'**);**

Serial**.**write**(**"\r\n"**);**

delay**(**3000**);** //3 Second to ensure the module fully connected.

**}**

**SendDataViaWiFi**

This function is used to connect to a website by which it can send and receive data. The function will configure the ESP8266 for multiple connections, connecte to a website on a particular port using a channel 0, declare how much data it will send and lastly send the data. This function needs to be called every time data is sent.

void SendDataViaWiFi**(**float batteryPower**,** short temperature**,** long pressure**,** float altitude**)**

**{**

delay**(**3000**);**

//Used to set-up multiple connections. Must be always set during boot up.

Serial**.**write**(**"AT+CIPMUX=1\r\n"**);**

delay**(**3000**);**

//Connect to tixgy.com on channel 0 using TCP on port 80.

//The command is not printed out in one line because of interferences

//caused by the "" in the command.

Serial**.**write**(**"AT+CIPSTART="**);**

Serial**.**write**(**"0,"**);** //Channel number

Serial**.**write**(**'"'**);**

Serial**.**write**(**"TCP"**);** //Chosen protocol

Serial**.**write**(**'"'**);**

Serial**.**write**(**','**);**

Serial**.**write**(**'"'**);**

Serial**.**write**(**"tixgy.com"**);** //Website

Serial**.**write**(**'"'**);**

Serial**.**write**(**','**);**

Serial**.**write**(**"80\r\n"**);** //Port number.

//3 Second delay created to ensure previous command completes.

delay**(**3000**);**

//Specifies the number of bytes send to channel 0.

Serial**.**write**(**"AT+CIPSEND=0,110\r\n"**);**

//The various strings with the parameters are concatenated.

//They are concatenated this way because the parameters change.

//GET http://www.tixgy.com/DITrobot.php?param1=" + 300 + "&param2=" + 400

// + " HTTP/1.0;

String request **=** String**(**"GET http://www.tixgy.com/DITrobot.php?param1="**);**

request **=** request **+** batteryPower**;**

request **=** request **+** "&param2="**;**

request **=** request **+** temperature**;**

request **=** request **+** "&param3="**;**

request **=** request **+** pressure**;**

request **=** request **+** "&param4="**;**

request **=** request **+** altitude**;**

request **=** request **+** " HTTP/1.0"**;**

Serial**.**print**(**request**);**

//The newline and carriage return are entered in case data entered is

//shorter than the specified value. These fill in the empty spaces.

Serial**.**write**(**"\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n"**);**

//Used to display display the received.

**while(**Serial**.**available**()>**0**){**

Serial**.**print**((**char**)**Serial**.**read**());**

**}**

Serial**.**write**(**"\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n"**);**

**}**

# References

<https://nurdspace.nl/ESP8266>

<https://www.youtube.com/watch?v=9QZkCQSHnko>

<http://en.wikipedia.org/wiki/Wi-Fi_Direct>

<https://www.sparkfun.com/products/13252>

<http://wiki.iteadstudio.com/File:ESP8266_Schematic.jpg>

<http://rancidbacon.com/files/kiwicon8/ESP8266_WiFi_Module_Quick_Start_Guide_v_1.0.4.pdf>