ECE 4999

Independent Project:

Wi-Fi Communication Using ESP8266 & PIC32

Mikhail Rudinskiy

Completed for Dr. Bruce Land

Introduction

The purpose of this independent project was to explore the capabilities of the new ESP8266 wireless module. The ESP8266 is a low cost wireless module with a complete AT command library. This allows for easy integration with a Wi-Fi network through serial communication. The ESP8266 was integrated with a PIC32 mirocontroller on a Microstick II development board. Two modes of the ESP8266 were explored: as a station and access point. As a station, the system connects to a wireless network and reads weather data from the internet. This data is parsed and certain measures are displayed on an LCD screen. As an access point, the ESP hosts a small html website while the PIC32 reads the temperature on demand from a temperature sensor. This mode demonstrates the Wi-Fi chip's ability to host and receive data simultaneously. Overall, the ESP8266 SOC is an easy to use Wi-Fi radio that can be easily added to most microcontroller projects using the serial protocol.

Hardware Design-ESP8266

The central component of this project is an ESP8266 Wi-Fi Module. This radio on chip features an easy to use AT command set through a serial protocol. The chip itself also supports SPI communication; however, the most common version of the module (ESP-01) and firmware is designed for serial communication (Fig. 1). The ESP8266 module is equipped with a microcontroller which allows it to operate without an external microcontroller. The ESP-01 version is equipped with two GPIO pins. Some other configurations of the board have a larger number of GPIO pins and would be preferred if the onboard microcontroller is utilized. Another consideration when selecting an ESP module is whether breadboard compatibility is required. The ESP-01 version is not designed for such application.

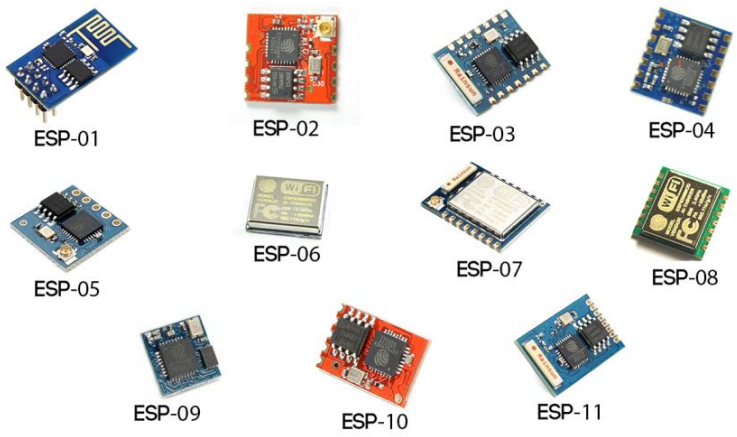
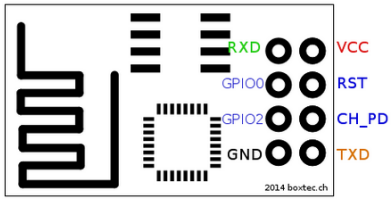


Figure 1 The ESP8266 is available in various module configuration [1]. The main difference is the selection of pinouts from the chip.

Connections on the ESP board module are detailed in Figure 2 and the wiring overview is available in Figure 3. The breadboard implementation of this project was assembled without current-limiting resistors and performed well.



Label	Signal
VCC	3.3V (max 3.6V) Versorgungsspannung
GND	Ground
TXD	Transmit Data (3.3V Pegel)
RXD	Receive Data (3.3V Pegel!)
CH_PD	Chip Power down: (LOW = Power down aktiv)
GPIO0	General Purpose I/O 0
GPIO2	General Purpose I/O 2
RST	Reset (LOW = Reset aktiv)

Figure 2 The ESP pinout configuration is described above. The module uses 3.3V logic [2].

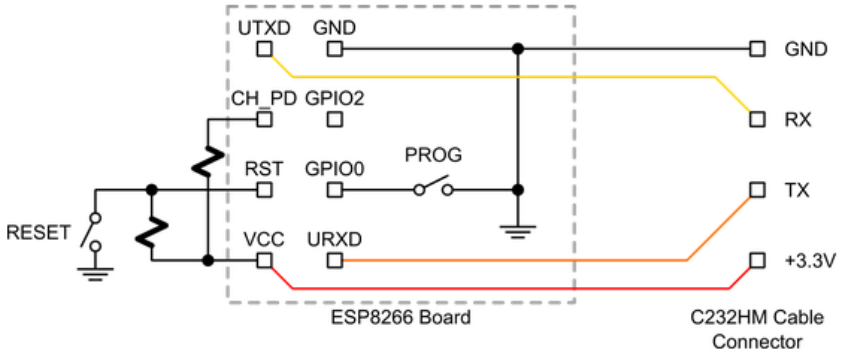


Figure 3 Wiring overview including firmware flashing provision [3].

The ESP8266 module comes with firmware preinstalled. It is useful to update the firmware to the latest version. This requires running the esp8266_flasher.exe and installing the latest firmware.bin file. A thorough explanation of the required steps is outlined by Dave Vandenbout in *ESP8266: Reflash Dance!* [3]. Reflashing the ESP8266 board requires momentarily pulling the RST pin to ground and then connecting GPIO0 to ground (Note: GPIO0 & GPIO2 should normally be tied to VCC). This ESP8266 is running on firmware version v0.9.2.2 with a default baud rate of 9600 (although some other firmware versions use other bauds by default).

Hardware Design-PIC 32

This project utilizes a PIC 32 bit mircontroller PIC32MX250F128B on a Microstick II development board. This model of the PIC supports 2 UART communication channels both of which are used in this project. One channel communicates with the Wi-Fi module while the other one is used for debugging purposes.

Hardware Design-Temperature Sensor

The TI LM35 is a linear temperature sensor which increases 10mV for every °C (Fig. 4). It is connected to an ADC pin on the PIC. The temperature sensor should be connected to a 5V tolerant pin on the microcontroller.

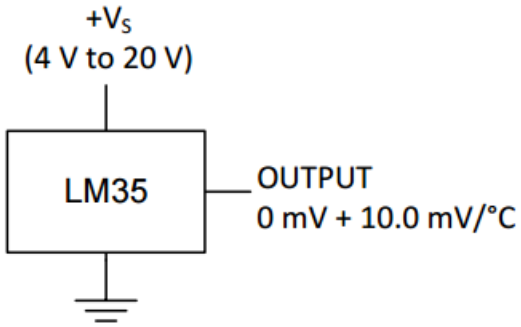


Figure 4 Temperature sensor configuration [4].

Hardware Design-LCD Display

The LCD used in this project is an Electronic Assembly DOGM163W-A 3.3V monochromatic display. It is used in the 4bit write only configuration. This means that the LCD is addressed with only 4 data wires rather than the normal 8 wires necessary to write a bit. Instead of writing 8 bits, the system write a nibble of data. An enable and reset wire are also necessary for this configuration. This configuration simplifies the wiring to the microcontroller. Figure 5 displays the wiring diagram.

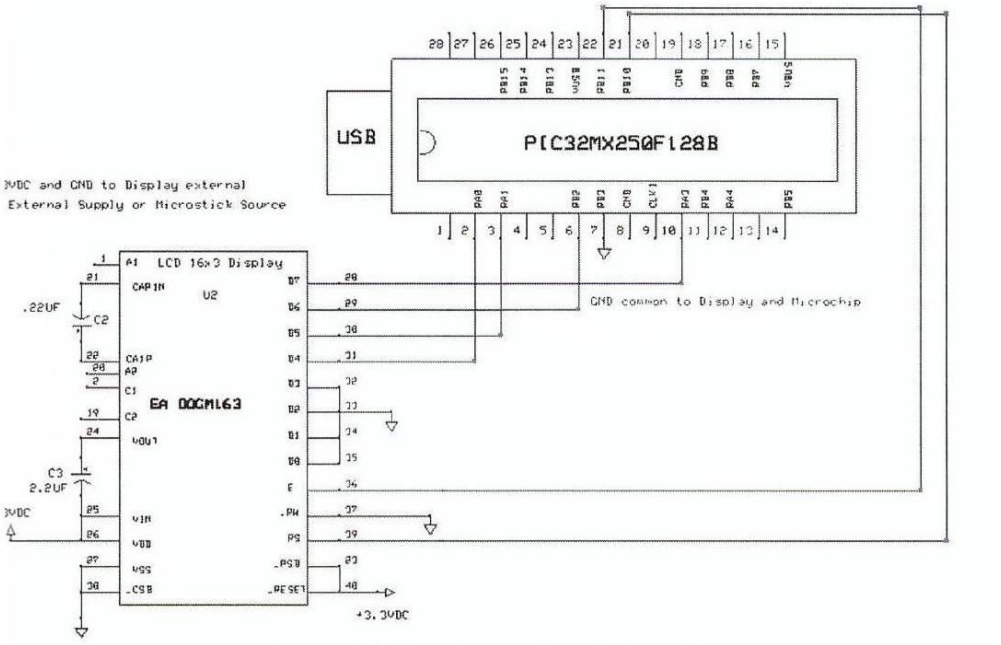


Figure 5 LCD connection [5].

Hardware Design-Other

The complete system is depicted in Figure 6. Some miscellaneous objects include a power supply and 3.3V regulator. The photograph also includes a 3.3V TTL logic board. The following connections are also present on the system (along with LCD):

- PIC pin 18 – ESP TX
- PIC pin 17 – ESP RX
- PIC pin 16 – TTL RX
- PIC pin 15 – TTL TX
- PIC pin 26 – Temperature Sensor

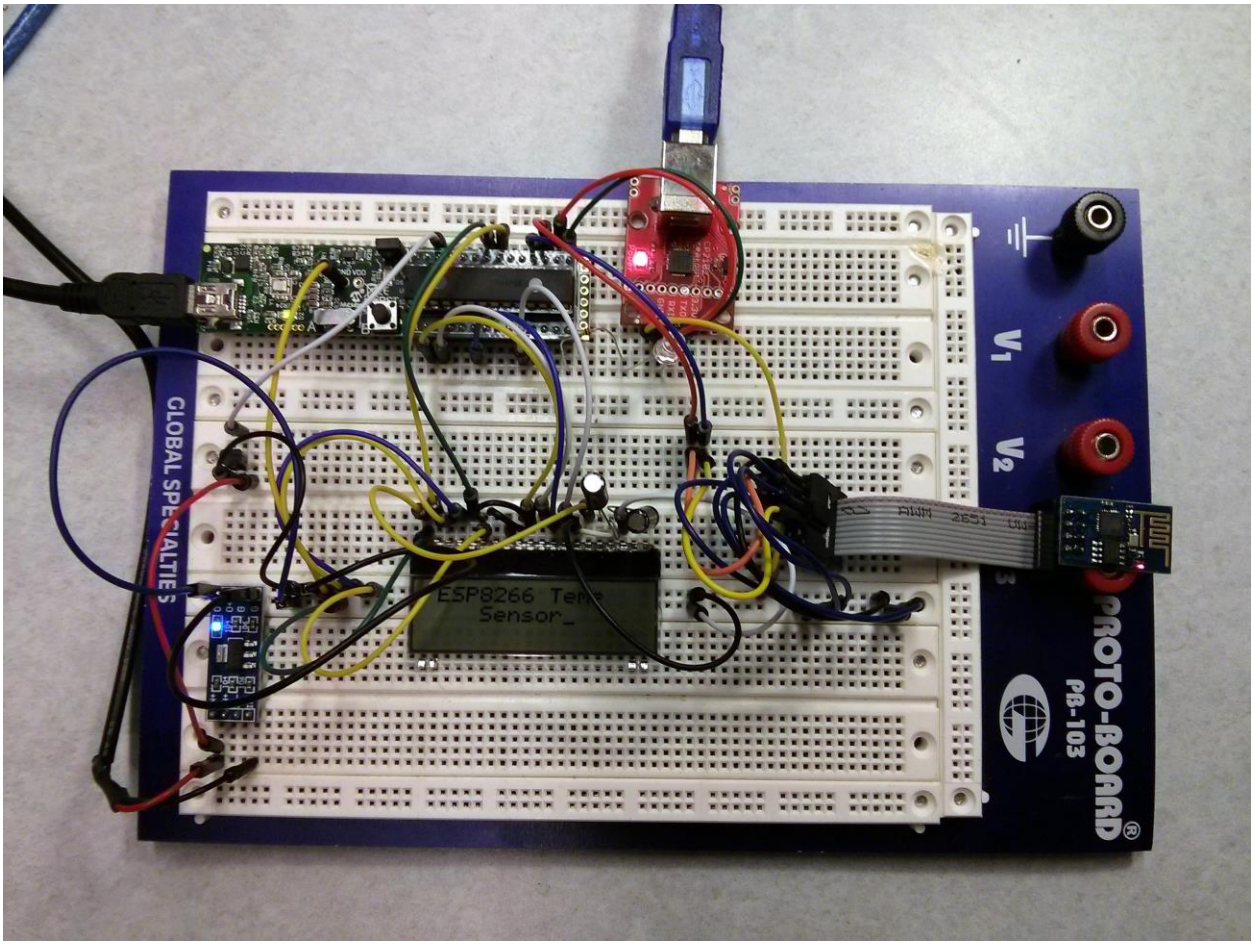


Figure 6 Complete System.

Software Design – Overview

The ESP 8266 module works based on AT commands. This version of firmware operates at 9600 baud and is terminated with a \r\n. The following are common commands used to communicate with the module. A complete list of commands can be found in Appendix A.

- AT+RST – Reset the module
- AT+CWLAP – List available access points
- AT+CWJAP=”SSID”,”Password”- Join access SSID with Password
- AT+CWMODE=? – Set module as a 1) Station 2)Access point 3) Access point and station
- AT+CIPMUX – Allow single or multiple connections to be made
- AT+CIPSERVER=1,80 – Create server on port 80
- AT+CIPSEND=0,10 – Send 10 bytes to ID 0
- AT+CIPCLOSE=0 – Close connection

Two separate programs were created to demo the ESP8266. The first of these programs is titled “WeatherStation”; its purpose is to connect to a network and read data off the internet. Since

both the Wi-Fi module and debug terminal are based on UART communication, writing to the UART is detailed first.

Software Design – UART

The code CONU1.C and CONU2.C act as the UART libraries. UART1 is used to communicate with the debug terminal while UART2 is used to write to the Wi-Fi Module. The code is initialized with the following setup:

```
void initU1( void)
{
U1BRG = BRGVAL; // BAUD Rate Setting for 9600
U1MODE = U_ENABLE;
U1STA = U_TXRX; //set transmit polarity inverse and tranmit break bit
U1MODEbits.BRGH = 0; // standard mode
}
```

The putU1 command puts a single character to the UART transmit register:

```
int putU1( int c)
{
while ( U1STAbits.UTXBF);   // wait while Tx buffer full
U1TXREG = c;
return c;
}
```

Multiple characters can be places into the transmit buffer by calling the single putU1 command multiple times using the putsU1 command.

```
void putsU1( char *s)
{
        while( *s)          // loop until *s == '\0' the  end of the string
                putU1( *s++);       // send the character and point to the next one
        putU1('\r'); // terminate with a cr / line feed
        putU1('\n');
}
```

Alternatively, a certain length of characters can be places with the putsU1point command.

```
void putsU1point( char *s, int i)
{
   int count=0;
   while (count<i) {
      putU1(*s++); //put new character
      count++; // increment counter
   }
   putU1('\r');
   putU1('\n');
}
```

Only UART2 (the Wi-Fi module UART) needs to be read back. This is done with an interrupt based approach. The following code reads the receive buffer and stores it into a character array. Although other more efficient data structures exist to store the received characters, because of the small number of characters dealt with, this method was sufficient.

```
void __ISR(_UART_2_VECTOR, ipl2) UART2BHandler(void) { //if character becomes available
   s[u]= (char) (U2RXREG & 0x00ff); //read character and store it in an array
   u++;
   // Clear the UART RX interrupt flag bit
   IFS1bits.U2RXIF = 0;
}
```

Software Design – LCD

The LCD writes nibbles based on a library developed by Thomas Kiablo [5]. The libraries functions most used in this implementation are the character and string write functions as detailed below.

```
void write_character_LCD( char mychar) {
        char * pt;      //need to make a pointer
        pt = &mychar;
        update_Display( pt,1);  //display single character using pt and numofchar =1
}

void write_string_LCD( char* chpointer) {
        while (*chpointer != '\0'){
            LCDWrite(*chpointer,1);
            chpointer++;
        }
}
```

Software Design – Main WeatherStation.c

The main program assigns the appropriate peripherals of the PIC microcontroller to its external pins. It then disables the analog features of the pins and sets up the interrupt to receive data from the Wi-Fi module. The LCD is initialized and a message is written to it. Finally, the function of code dealing with the ESP8266 module is called.

The code begins by writing the AT+RST command to the module and waiting for a return. A delay statement is used to wait for the return. The reset command in particular may take a long time to return. If the module does not return an affirmative response, a message will be displayed to the user.

```
putsU2("AT+RST");
do {
   delay(2000);
   if (strstr(s, "OK") != NULL) {
      putsU1("ESP8266 Reset & Ready");
      clear_display(); //clear the LCD display
```

```
      position_cursor(0); //position cursor to row 1
      write_string_LCD((char*) "Reset & Ready"); //outputs Message
      break;
    } else if (strstr(s, "FAIL") != NULL) {
      putsU1("Reset Failed");
      clear_display(); //clear the LCD display
      position_cursor(0); //position cursor to row 2
      write_string_LCD((char*) "Reset Failed"); //outputs Message
      break;
    }
  } while (1);
  clrstr(s);
```

Once the module has been reset, the code checks if the module is already connected to an access point with the AT+CJWAP? command. The return is monitored and a flag is raised if the module is not connected.

```
putsU2("AT+CWJAP?"); // check if already connected to AP
do {
   delay(500);
   putsU1(s);
   if (strstr(s,SSID) != NULL) {
      putsU1("already connected to wifi ap");
      position_cursor(16); //position cursor to row 1
      write_string_LCD((char*) "Alrdy connected to wifi ap"); //outputs Message
      connection=1;
      break;
   } else if (strstr(s, "ERROR") != NULL) {
      putsU1("connecting to access point...");
      //clear_display(); //clear the LCD display
      position_cursor(16); //position cursor to row 1
      write_string_LCD((char*) "connecting to   access point..."); //outputs Message
      connection=0;
      break;
   }
    else
    {
       break;
    }
} while (1);
clrstr(s);
```

If the module needs to be connected, a string is concatenated with the AT+CWJAP command, access point name and access point password as follows:

```
char cmd[100];
clrstr(cmd);
if (connection == 0) { //if not connected
   strcat(cmd, "AT+CWJAP=\"");
   strcat(cmd, SSID);
```

```
strcat(cmd, "\",\"");
strcat(cmd, password);
strcat(cmd, "\"");
putsU2(cmd);
```

The string is written to the UART buffer and the return is monitored.

```
do {
   delay(2000);
   putsU1(s);
   if (strstr(s, "OK") != NULL) {
      putsU1("connected to wifi ap");
      clear_display(); //clear the LCD display
      position_cursor(0); //position cursor to row 1
      write_string_LCD((char*) "WiFi Connected!"); //outputs Message
      connection =1;
      break;
   } else if (strstr(s, "FAIL") != NULL) {
      //putsU2("AT+CWQAP"); //quite access point
      putsU1("failed to connect, let me try again");
      clear_display(); //clear the LCD display
      position_cursor(0); //position cursor to row 1
      write_string_LCD((char*) "Connect Failed  Let me try again"); //outputs Message
      putsU2("AT+CWQAP");
      delay(100);

      clrstr(s);
      putsU2(cmd);
      delay(100);
```

Next the mode and mux of the module are set. Setting the mode to 3 configures it as an access point and station (so it can be used in either mode). Setting the mux to 1 allow the module to make up to 4 connections. The commands associated with these actions are AT+CWMODE=3 and AT+CIPMUX=1. The code to place these commands and monitor the return is very similar to those previously stated.

Connecting to the online weather station requires the following command:

```
putsU2("AT+CIPSTART=4,\"TCP\",\"api.openweathermap.org\",80");
```

This sets a connection ID=4 and initiates a TCP connection to the listed site on port 80. Once the connection is started, the AT+CIPSEND command is used forecast the number of byte that will be sent to the ESP Wi-Fi station.

```
putsU2("AT+CIPSEND=4,91");
```

The code waits for the Wi-Fi module to return a “>” signifying that it is ready to receive a transmission. The transmission is placed to the module.

```
putsU2("GET /data/2.5/station/find?lat=42&lon=-76&cnt=30 HTTP/1.0\r\nHost: api.openweathermap.org\r\n");
```

This line samples 30 nearest weather stations around the specified latitude and longitude. Many of these stations are self-submitted by users to the openweather server. Other search modes exist to retrieve weather; a user can call a specific station, search by city, zip code, etc. These commands would require only minor modification of the get string.

Data that is received from the ESP8266 is headed by a +IPD. This program waits for an arbitrary amount of time until the whole transmission is received. Future iterations should monitor the end of the transmission instead.

Once the data is received and stored in a string, parameters of interested are retrieved from the string by a keyword search. The following is an example of windspeed.

```
char * windspeed;
windspeed = strstr(s, "wind") + 15;
position_cursor(32);
write_string_LCD((char*) "windspeed:   mps");
position_cursor(43);
write_array_LCD((char*) windspeed, 2);
```

As a final step, this individually parsed data is written to the LCD screen.

Software Design – Temperature Hosting

This version of the project hosts a simple HTML website and which reads a temperatures sensors. Whenever a query is received, the system performs a series of ADCs, averages them, and sends the data to the user. Although the configuration was only tested on an intranet system, it should be easily scalable to an internet application. The only difference is that the firewall settings of the router need to be configured to port incoming requests to the internal IP address of the ESP8266 device. It was not possible to test this on Cornell's routers.

The project described below connects to a Wi-Fi network, configures the ESP module to appropriate settings and displays the internal IP assigned by the router to the ESP8266 device. This IP address is displayed on the LCD screen. In your browser, type the IP address specified followed by a ":80". Make sure that your computer is connected to either same wireless network as the ESP8266 or directly to the access point network created by the module.

Section of the following code is are similar to that described in the first program.

Software Design – ADC Temperature Reading

This code reads 4 temperature values and averages them. The result is displayed on the LCD and stored to be used be sent to the user.

```
void ReadTemp(void){
  // Take an ADC reading on pin
   initADC(pin); //Initialize ADC for AN0 pin 2
   adcvalue=averagevalue(); //read and average 4 samples
```

```
    temp = (float) adcvalue * 330.0 / 1024.0;
    sprintf(tempvalue, "%.1f", temp);

    //write the temperature out to the LCD
    //clear_display(); //clear the LCD display
    position_cursor(32); //position cursor to row 1
    write_string_LCD((char *) "temp:     C");
    position_cursor(38);
    write_string_LCD((char*) tempvalue);
}
```

Software Design – ESP8266 Initialization

The initialization of the module is similar with the exception of the server initialization command.

```
putsU2("AT+CIPSERVER=1,80");
```

This command starts a server on port 80. The program also reads and displays the internal IP address assigned by the router using:

```
putsU2("AT+CIFSR");
```

Software Design – Main TemperatureSensor.c

Once the ESP module is initialized, the system waits for the “IPD” command to be received. This means that the ESP module has received some communication (most likely a request from the user). The code then reads the temperature sensor and concatenates an HTML string.

```
strncpy(website,"<!DOCTYPE html><html>"\
        ,strlen("<!DOCTYPE html><html>"));
   strcat(website,"<body style=\"background-color:lightblue\">"); …
```

The string length is calculated and also sent to the Wi-Fi module. It then waits for the “>” command to be returned from the ESP. Once this is reaffirmed, the HTML string is written.

```
    putsU2(website); //write the HTML to the ESP8266
    delay(200);
```

The delay is used to verify that the whole string is written fully and the ESP module is not overrun with the next command. Finally, the connection to the ESP8266 is closed

```
    putsU2("AT+CIPCLOSE=0");
```

A user see a website as displayed in figure 7.

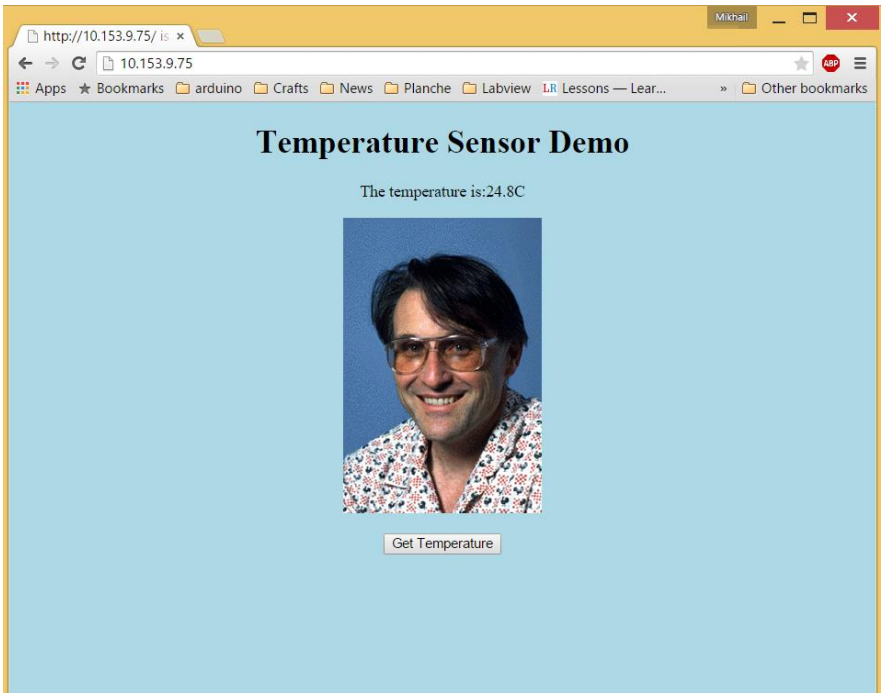


Figure 7 Browser view of HTML site.

Results/Conclusion

The system is able to send and receive data over the internet. The ESP module does throw different exceptions when a command does not go through as anticipated. This requires having a logic structure able to handle each situation. This program is able to handle the exceptions for the basic Wi-Fi operation. It will take more time to fully implement solutions for higher level situation. As a next step, it is also beneficial to implement an interrupt that is able to detect when the ESP module has returned a string. This will eliminate the need for delays in the program. From an HTML perspective, having a javascript run buttons will allow for the system to act as a remote controller. Sending a button call over the internet to a microcontroller can have very beneficial applications.

Overall, the system is able to read weather and temperature data over the internet both as a station and as a host. The ESP8266 module is easy to use once the communication structure is established.

Software Code Acknowledgments

The following sources contributed to the knowledge and code used in this project.

- AllAboutEE YouTube channel: https://www.youtube.com/user/AllAboutEE/videos
- Thomas Kibalo: Beginner's Guide to Programming the PIC 32
- Lucio Di Jasio: Programming the 32-bit Microcontroller in C

References

[1] [Online]. Available: http://randomnerdtutorials.com/getting-started-with-esp8266-wifi-transceiver-review/.

[2] "BoxTec," [Online]. Available: http://playground.boxtec.ch/doku.php/wireless/esp8266.

[3] D. Vanedenbout, "XESS," [Online]. Available: http://www.xess.com/blog/esp8266-reflash/.

[4] "LM35 Precision Centigrade Temperature Sensor," [Online]. Available: http://www.ti.com/lit/ds/symlink/lm35.pdf.

[5] T. Kibalo, Beginner's Guide to Programming the PIC32, Milford: Electronic Products, 2013.

Appendix A http://firatdeveci.com/wp-content/uploads/Electronics/ESP8266_AT_Command.pdf

Espressif: AT Instruction Set

Status	Released
Current version	V0.20
Author	Xu Jingjie
Completion Date	2014.11.28
Reviewer	
Completion Date	

[√] CONFIDENTIAL
[ ] INTERNAL [ ] PUBLIC

Version Info

Date	Version	Author	Comments/Changes
2014.6.27	0.1	XuJingjie	Draft
2014.7.11	0.11	XuJingjie	Unvarnished transmission added
2014.8.12	0.15	XuJingjie	1、Added Timeout and IP settings for AP 2、Edited description for server functions 3、Support DNS
2014.9.25	0.18	XuJingjie	1、Added upgrade through network 2、Added CWLAP
2014.11.10	0.19	XuJingjie	Added UDP
2014.11.27	0.20	XuJingjie	1、Added set and get APIP/APMAC/STAIP /STAMAC 2、Added start and stop DHCP

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member Logo is a trademark of the Wi-Fi Alliance.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Table of Contents

1 Overview

This is the documentation for Espressif AT command instruction set and usage.

Instruction set is divided into: Basic AT commands, Wifi function, AT commands, TCP / IP Toolbox AT commands.

Note: Please make sure that correct BIN(\esp_iot_sdk\bin\at) is already in the chip (ESP8266) before the AT commands listed in this documentation can be used.

2 Instruction Description

Each instruction set contains four types of AT commands.

Type	Instruction Format	Description
Test	AT+<x>=?	Query the Set command or internal parameters and its range values.
Query	AT+<x>?	Returns the current value of the parameter.
Set	AT+<x>=<…>	Set the value of user-defined parameters in commands and run.
Execute	AT+<x>	Runs commands with no user-defined parameters.

Note:

1. Not all AT instruction has four commands.
2. [] = default value, not required or may not appear
3. String values require double quotation marks, for example:

AT+CWSAP="ESP756190","21030826",1,4

4. Baud rate = 115200
5. AT instruction ends with "\r\n"

3 AT Instruction Listing

Instructions	Description
Basic	
AT	Test if AT startup
AT+RST	Restart
AT+GMR	View version info
AT+GSLP	Enter deep-sleep mode
ATE	AT commands echo
Wi-Fi	
AT+CWMODE	WIFI mode（station/softAP/station+softAP）
AT+CWJAP	Connect to AP
AT+CWLAP	Lists available APs
AT+CWQAP	Disconnect from AP
AT+CWSAP	Set parameters under AP mode
AT+CWLIF	Get stations' ip which are connected to ESP8266 softAP
AT+CWDHCP	Enable/Disable DHCP
AT+CIPSTAMAC	Set mac address of ESP8266 station
AT+CIPAPMAC	Set mac address of ESP8266 softAP
AT+CIPSTA	Set ip address of ESP8266 station
AT+CIPAP	Set ip address of ESP8266 softAP
TCP/IP	
AT+CIPSTATUS	Get connection status
AT+CIPSTART	Establish TCP connection or register UDP port
AT+CIPSEND	Send data
AT+CIPCLOSE	Close TCP/UDP connection
AT+CIFSR	Get local IP address
AT+CIPMUX	Set multiple connections mode
AT+CIPSERVER	Configure as server
AT+CIPMODE	Set transmission mode
AT+CIPSTO	Set timeout when ESP8266 runs as TCP server
AT+CIUPDATE	For OTA（upgrade through network）
Data RX	
+IPD	Data received from network

4 Basic AT Instruction Set

4.1 Overview

Basic	
Instruction	**Description**
AT	Test AT startup
AT+RST	Restart module
AT+GMR	View version info
AT+GSLP	Enter deep-sleep mode
ATE	AT commands echo or not

4.2 Instructions

4.2.1 AT – Test AT startup

AT – Test AT startup	
Type：execute Instruction： **AT**	Response： OK
	Param description：null

4.2.2 AT+RST – Restart module

AT+RST – Restart module	
Type ：execute Instruction： **AT+RST**	Response： OK
	Param description：null

4.2.3 AT+GMR – View version info

AT+GMR – View version info	
Type ：execute Instruction： **AT+GMR**	Response： <number> OK
	Param description： < number > version info，length：8 bytes

Note	For example, response is 0017xxxxxx, then 0017 means the AT version.

4.2.4 AT+GSLP – Enter deep-sleep mode

AT+GSLP – Enter deep-sleep mode	
Type :set Instruction: **AT+GSLP=<time>**	Response: <time> OK
	Param description: < time > ms , set the sleep time of ESP8266 in ms. ESP8266 will wake up after X ms in deep-sleep.
Note	Hardware has to support deep-sleep wake up (XPD_DCDC connects to EXT_RSTB with 0R).

4.2.5 ATE – AT commands echo

ATE – AT commands echo	
Type :set Instruction: **ATE**	Response: OK
	Param description: ATE0 : Disable echo ATE1 : Enable echo

5 WIFI functions

5.1 Overview

WIFI	
Instruction	**Description**
AT+CWMODE	WIFI mode（station/softAP/station+softAP）
AT+CWJAP	Connect to AP
AT+CWLAP	Lists available APs
AT+CWQAP	Disconnect from AP
AT+CWSAP	Set parameters under AP mode
AT+CWLIF	Get station’s ip which is connected to ESP8266 softAP
AT+CWDHCP	Enable/Disable DHCP
AT+CIPSTAMAC	Set mac address of ESP8266 station

AT+CIPAPMAC	Set mac address of ESP8266 softAP
AT+CIPSTA	Set ip address of ESP8266 station
AT+CIPAP	Set ip address of ESP8266 softAP

5.2 Instructions

5.2.1 AT+CWMODE – WIFI mode

AT+CWMODE - WIFI mode（station/softAP/station+softAP）	
	Response： OK
	Param description： The same as above.
Type：test Function: Get value scope of wifi mode.	Response： +CWMODE:(value scope of <mode>)
Instruction： **AT+CWMODE=?**	OK Param description： <mode>1 means Station mode 2 means AP mode 3 means AP + Station mode
Type：query Function: Query ESP8266’s current wifi mode.	Response： +CWMODE:<mode> OK
Instruction：	Param description：

The same as above.

AT+CWMODE?

Type：set
Function:
Set ESP8266 wifi mode
Instruction：

AT+CWMODE=<mode>

5.2.2 AT+CWJAP – Connect to AP

AT+CWJAP – Connect to AP

Type：query Function: Query AP’s info which is connect by ESP8266. Instruction： **AT+ CWJAP?**	Response： + CWJAP:<ssid> OK
	Param description： <ssid> string, AP’s SSID

Type：set Function: Set AP's info which will be connect by ESP8266. Instruction： **AT+ CWJAP =<ssid>,< pwd >**	Response： OK ERROR Param description： <ssid> string, AP's SSID <pwd> string, MAX: 64 bytes

5.2.3 AT+CWLAP – List available APs

AT+CWLAP - Lists available APs	
Type：set Function: Search available APs with specific conditions. Instruction： **AT+ CWLAP =** **<ssid>,< mac >,<ch>**	Response： + CWLAP: <ecn>,<ssid>,<rssi>,<mac> OK ERROR Param description： The same as below.
Type ：execute Function: Lists all available APs. Instruction： **AT+CWLAP**	Response： + CWLAP: <ecn>,<ssid>,<rssi>,<mac> OK ERROR Param description： < ecn >0 OPEN 1 WEP 2 WPA_PSK 3 WPA2_PSK 4 WPA_WPA2_PSK <ssid> string, SSID of AP <rssi> signal strength <mac> string, MAC address

5.2.4 AT+CWQAP – Disconnect from AP

AT+CWQAP - Disconnect from AP	
Type：test Function: Only for test Instruction：	Response： OK Param description：

AT+CWQAP=?	
Type：execute Function: Disconnect from AP. Instruction： **AT+ CWQAP**	Response： OK Param description：

5.2.5 AT+CWSAP – Configuration of softAP mode

AT+ CWSAP – Configuration of softAP mode	
Type：Query Function: Query configuration of softAP mode. Instruction： **AT+ CWSAP?**	Response： + CWSAP:<ssid>,<pwd>,<chl>,<ecn> Param description： The same as below.
Type：Set Function: Set configuration of softAP mode. Instruction： **AT+ CWSAP=** **<ssid>,<pwd>,<chl>,** **<ecn>**	Response： OK ERROR Note: This CMD is only available when softAP mode enable, and need to follow by AT+RST to make it works. Param description： <ssid> string, ESP8266 softAP' SSID <pwd> string, MAX: 64 bytes <chl> channel id < ecn >0 OPEN 2 WPA_PSK 3 WPA2_PSK 4 WPA_WPA2_PSK

5.2.6 AT+CWLIF – IP of stations

AT+ CWLIF– ip of stations which are connected to ESP8266 softAP	
Type：execute Function: Get ip of stations which are connected to ESP8266 softAP	Response： <ip addr> OK Param description：

Instruction: **AT+CWLIF**	<ip addr> ip address of stations which are connected to ESP8266 softAP

5.2.7 AT+CWDHCP – Enable/Disable DHCP

AT+ CWDHCP – Enable/Disable DHCP	
Type :set Function: Enable/Disable DHCP. Instruction: **AT+CWDHCP=<mode>,<en>**	Response: OK
	Param description: <mode> 0 : set ESP8266 softAP 1 : set ESP8266 station 2 : set both softAP and station <en> 0 : Enable DHCP 1 : Disable DHCP

5.2.8 AT+CIPSTAMAC – Set mac address of station

AT+ CIPSTAMAC – Set mac address of ESP8266 station	
Type :query Function: Get mac address of ESP8266 station. Instruction: **AT+CIPSTAMAC?**	Response: +CIPSTAMAC:<mac> OK
	Param description: <mac> string, mac address of ESP8266 station
Type :set Function: Set mac address of ESP8266 station. Instruction: **AT+CIPSTAMAC=<mac>**	Response: OK
	Param description: <mac> string, mac address of ESP8266 station

5.2.9 AT+CIPAPMAC – Set mac address of softAP

AT+ CIPAPMAC – Set mac address of ESP8266 softAP	
Type：query Function: Get mac address of ESP8266 softAP. Instruction: **AT+CIPAPMAC?**	Response: +CIPAPMAC:<mac> OK Param description: <mac> string, mac address of ESP8266 softAP
Type：set Function: Set mac address of ESP8266 softAP. Instruction: **AT+CIPAPMAC=<mac>**	Response: OK Param description: <mac> string, mac address of ESP8266 softAP

5.2.10 AT+ CIPSTA – Set ip address of station

AT+ CIPSTA – Set ip address of ESP8266 station	
Type：query Function: Get ip address of ESP8266 station. Instruction: **AT+CIPSTA?**	Response: +CIPSTA:<ip> OK Param description: <ip> string, ip address of ESP8266 station
Type：set Function: Set ip address of ESP8266 station. Instruction: **AT+CIPSTA=<ip>**	Response: OK Param description: <ip> string, ip address of ESP8266 station

5.2.11 AT+ CIPAP – Set ip address of softAP

AT+ CIPAP – Set ip address of ESP8266 softAP	
Type：query Function:	Response: +CIPAP:<ip>

Get ip address of ESP8266 softAP. Instruction: **AT+CIPAP?**	OK
	Param description: <ip> string, ip address of ESP8266 softAP
Type : set Function: Set ip address of ESP8266 softAP. Instruction: **AT+CIPAP=<ip>**	Response: OK
	Param description: <ip> string, ip address of ESP8266 softAP

6 TCP/IP Related

6.1 Overview

TCP/IP	
Instruction	**Description**
AT+ CIPSTATUS	Get connection status
AT+CIPSTART	Establish TCP connection or register UDP port
AT+CIPSEND	Send data
AT+CIPCLOSE	Close TCP/UDP connection
AT+CIFSR	Get local IP address
AT+CIPMUX	Set multiple connections mode
AT+CIPSERVER	Configure as server
AT+CIPMODE	Set transmission mode
AT+CIPSTO	Set timeout when ESP8266 runs as TCP server

6.2 TCP/IP

6.2.1 AT+ CIPSTATUS – Information about connection

AT+ CIPSTATUS – Information about connection	
Type：execute Function: Get information about connection. Instruction:	Response： STATUS:<stat> + CIPSTATUS:<id>,<type>,<addr>,<port>,<tetype> OK
AT+ CIPSTATUS	Param description： <stat> 2: Got IP 3: Connected 4: Disconnected <id> id of the connection (0~4), for multi-connect <type> string, “TCP” or “UDP” <addr> string, IP address. <port> port number <tetype> 0: ESP8266 runs as client 1: ESP8266 runs as server

6.2.2 AT+CIPSTART – Start connection

<table>
<tr><th colspan="2">AT+CIPSTART – Establish TCP connection or register UDP port, start connection</th></tr>
<tr><td rowspan="2">Type：test
Function:
Get the information of param.
Instruction:
AT+CIPSTART=?</td><td>Response：
1) If AT+CIPMUX=0
+CIPSTART:(<type>),(<IP address>),(<port>)[,(<local port>),(<mode>)]
+CIPSTART:(<type>),(<domain name>),(<port>)[,(<local port>),(<mode>)]

OK
2) If AT+CIPMUX=1
+CIPSTART:(id),(<type>),(<IP address>),(<port>)[,(<local port>),(<mode>)]
+CIPSTART: (id), (<type>),(<domain name>),(<port>)[,(<local port>),(<mode>)]</td></tr>
<tr><td>Param description：null</td></tr>
<tr><td>Type：Set
Function:
Start a connection as client.
Instruction:

1)Single connection
(+CIPMUX=0)

AT+CIPSTART=

<type>,<addr>,<port>

[,(<local port>),(<mode>)]

2)Multiple connection (+CIPMUX=1)
AT+CIPSTART=

<id><type>,<addr>,<port>

[,(<local port>),(<mode>)]</td><td>Response：
OK
or
ERROR
If connection already exists, returns ALREAY CONNECT</td></tr>
<tr><td></td><td>Param description：
<id> 0-4 , id of connection
<type> string, “TCP” or “UDP”
<addr> string, remote ip
<port> string, remote port
[<local port>] for UDP only
[<mode>] for UDP only
0 : destination peer entity of UDP will not change.
1 : destination peer entity of UDP can change once.
2 : destination peer entity of UDP is allowed to change.

Note:
[<mode>] can only be used when [<local port>] is set.</td></tr>
</table>

6.2.3 AT+CIPSEND – Send data

AT+CIPSEND – Send data	
Type：test Function: Only for test. Instruction: **AT+CIPSEND=?**	Response： OK Param description： null
Type：Set Function: Set length of the data that will be sent. For normal send. Instruction: 1)For single connection: (+CIPMUX=0) **AT+CIPSEND=<length>** 2) For multiple connection: (+CIPMUX=1) **AT+CIPSEND=** **<id>,<length>**	Wrap return “>” after set command. Begins receive of serial data, when data length is met, starts transmission of data. If connection cannot be established or gets disconnected during send, returns ERROR If data is transmitted successfully, returns SEND OK Note: This CMD Param description： <id> ID no. of transmit connection <length> data length, MAX 2048 bytes
Type：execute Function: Send data. For unvarnished transmission mode. Instruction: **AT+CIPSEND**	Response： Wrap return “>” after execute command. Enters unvarnished transmission, 20ms interval between each packet, maximum 2048 bytes per packet. When single packet containing “+++” is received, it returns to command mode. This command can only be used in unvarnished transmission mode which require to be single connection mode.

6.2.4 AT+CIPCLOSE – Close TCP or UDP connection

AT+CIPCLOSE – Close TCP or UDP connection	
Type：test Function:	Response：

Only for test. Instruction: **AT+CIPCLOSE=?**	OK
Type：Set Function: Close TCP orUDP connection. Instruction: For multiply connection mode **AT+CIPCLOSE=<id>**	Response： No errors, returns OK If connection <id> is disconnected, returns Link is not Param description： <id> ID no. of connection to close, when id=5, all connections will be closed. (id=5 has no effect in server mode)
Type：execute Instruction: For single connection mode **AT+CIPCLOSE**	Response： OK or If no such connection, returns ERROR Prints UNLINK when there is no connection

6.2.5 AT+CIFSR – Get local IP address

AT+CIFSR – Get local IP address	
Type：Test Function: Only for test. Instruction: **AT+CIFSR=?**	Response： OK
Type：Execute Function: Get local IP address. Instruction: **AT+ CIFSR**	Response： + CIFSR:<IP address> + CIFSR:<IP address> OK ERROR Param description： <IP address> IP address of ESP8266 softAP IP address of ESP8266 station

6.2.6 AT+ CIPMUX – Enable multiple connections

AT+ CIPMUX – Enable multiple connections or not	
Type：Query Function: Get param config. Instruction: **AT+ CIPMUX?**	Response： + CIPMUX:<mode> OK
	Param description： The same as below.
Type：Set Function: Set connection mode. Instruction: **AT+ CIPMUX=<mode>**	Response： OK If already connected, returns Link is builded
	Param description： <mode>0 single connection 1 multiple connection
Note	This mode can only be changed after all connections are disconnected. If server is started, reboot is required.

6.2.7 AT+ CIPSERVER – Configure as TCP server

AT+ CIPSERVER – Configure as TCP server	
Type：Set Function: Set TCP server. Instruction: **AT+ CIPSERVER= <mode>[,<port>]**	Response： OK
	Param description： <mode> 0 Delete server (need to follow by restart) 1 Create server <port> port number, default is 333
Note	1、Server can only be created when AT+CIPMUX=1 2、Server monitor will automatically be created when Server is created. 3、When a client is connected to the server, it will take up one connection, be gave an id.

6.2.8 AT+ CIPMODE – Set transfer mode

AT+ CIPMODE – Set transfer mode

Type：Query Function: Query transfer mode. Instruction: **AT+ CIPMODE?**	Response： + CIPMODE:<mode> OK Param description： The same as below.
Type：Set Function: Set transfer mode. Instruction: **AT+CIPMODE=<mode>**	Response： OK If already connected, returns Link is builded Param description： <mode>0 normal mode 1 unvarnished transmission mode

6.2.9 AT+ CIPSTO – Set server timeout

AT+ CIPSTO – Set server timeout	
Type：Query Function: Query server timeout. Instruction: **AT+CIPSTO?**	Response： + CIPSTO:<time> OK Param description： The same as below.
Type：Set Function: Set server timeout. Instruction: **AT+CIPSTO=<time>**	Response： OK Param description： < time> server timeout, range 0~7200 seconds

6.2.10 AT+ CIUPDATE – Update through network

AT+ CIUPDATE – update through network	
Type：execute Function: Start upgrade. Instruction: **AT+ CIUPDATE**	Response： +CIPUPDATE:<n> OK Param description： <n> 1 found server 2 connect server 3 got edition

	4 start update

6.2.11 +IPD – Receive network data

+IPD – Receive network data	
1)Single connection: **(+CIPMUX=0)** **+IPD,<len>:<data>** 2) Multiple connection **(+CIPMUX=1)** **+IPD,<id>,<len>:<data>**	NOTE: When the module receives network data, it will send the data through the serial port using +IPD command Param description: <id> id no. of connection <len> data length <data> data received

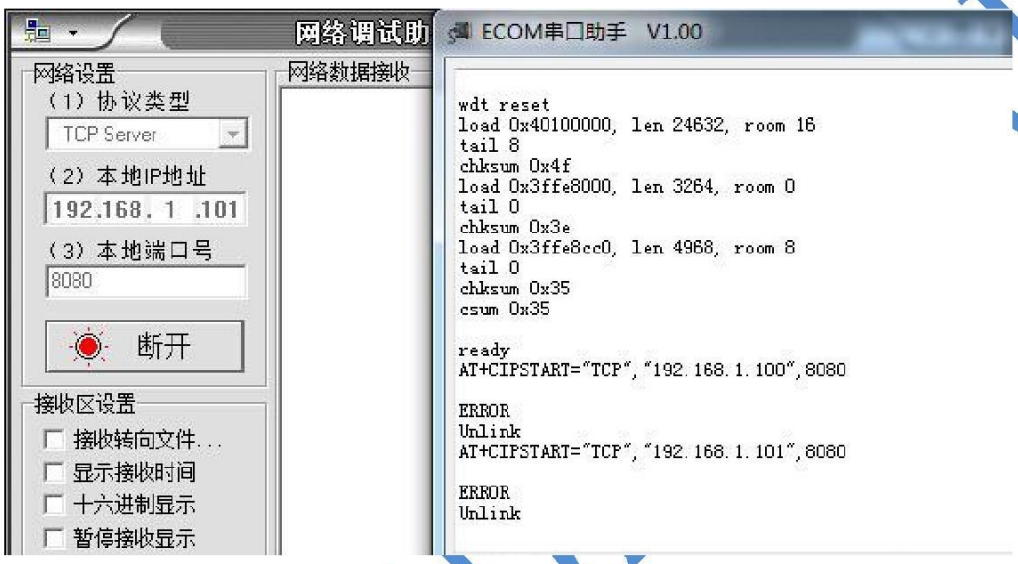
7 Q&A

If you have any question about AT instructions, please contact us (support-at@espressif.com) with information as follows:

(1) Version info of AT

Using “AT+GMR” to get the version info.

(2) Screenshot of the test steps, for example:



(3) Log:

ets Jan 8 2013,rst cause:1, boot mode:(3,3)

load 0x40100000, len 26336, room 16
tail 0
chksum 0xde
load 0x3ffe8000, len 5672, room 8
tail 0
chksum 0x69
load 0x3ffe9630, len 8348, room 8
tail 4
chksum 0xcb
csum 0xcb
SDK version:0.9.1
addr not ack when tx write cmd
mode : sta(18:fe:34:97:d5:7b) + softAP(1a:fe:34:97:d5:7b)