

Zero To Hero



ESP8266

Manoj R. Thakur

Circuits4you.com

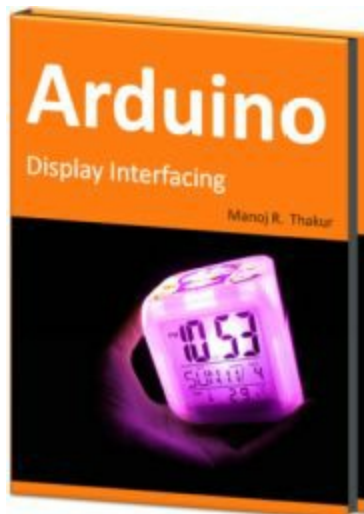
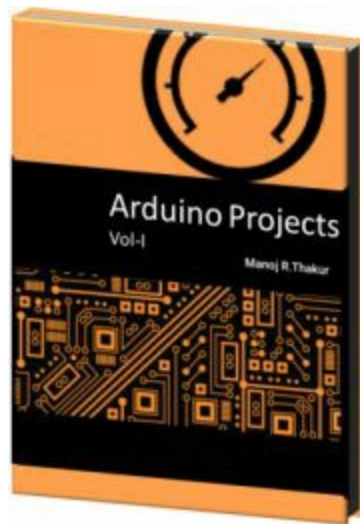
Zero to Hero ESP8266

Manoj R. Thakur

Date:
11/12/2016

Preface

I completed Master Degree in Electronics year 2008, I worked 5 years in MIT as Assistant Professor in Electronics Department and found that many students are struggling for knowing basic things and they are many times get bombarded with lot of information that makes them confuse, so I decided to write a book that focus more on practical approach and kept it like, read less and experiment more. My first e-book [Measurement Made Simple with Arduino](#) is the result of my experiences that got great success. Many found that its basis for all measurement needs.



This book is intended to focus on making use of ESP8266 with Arduino IDE. It is kept short and specific to the title of the book. IoT is recent trend in market and we have very less information on it. So this book takes

you from basics to advance level and guide you for correct path for IoT developments.

The second book “Arduino Projects Vol-1” is a Read Practical book, best thing about this book is it’s not just read or make, you can actually simulate before you make it. Most of the projects are provided with “Proteus” simulation files that are available on my website circuits4you.com.

Third book “Arduino Display interfacing” is great book on display interfacing techniques.

The book layout is kept very simple like experiment notes

1. Details of protocols used
 2. Interfacing details
 3. Circuit Design
 4. Programming
 5. Conclusions
-

Contents

1.

Introduction

2.

Hardware Setup ESP8266

3.

Software Setup ESP8266

4.

LED Blink Test

5.

Hardware and Software Serial

6.

Internal EEPROM

7.

ADC in Serial out

8.

HTML, CSS Basics

9.

Hello world! Web Server

10.

Temperature Monitoring Web Server

11.

LED Control Web Server

12.

IoT Home Automation

13.

Color Control Web Server

14.

HTTP Client

15.

Data logging to cloud server

16.

UDP ESP to ESP Communication

17.

Accessing ESP8266 over internet

18.

[Access ESP in VB.net](#)

19.

Plotting Graphs using ESP8266

20.

Using Google Gadgets

1. Introduction

1.1 Introduction to IoT

The Internet of things (IoT) is the internetworking of physical devices, vehicles (also referred to as "connected devices" and "smart devices"), buildings and other items—embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data. In 2013 the Global Standards Initiative on Internet of Things (IoT-GSI) defined the IoT as "the infrastructure of the information society." The IoT allows objects to be sensed and/or controlled remotely across existing network infrastructure, creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit. When IoT is augmented with sensors and actuators, the technology becomes an instance of the more general class of cyber-physical systems, which also encompasses technologies such as smart grids, smart homes, intelligent transportation and smart cities. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure. Experts estimate that the IoT will consist of almost 50 billion objects by 2020.

ESP8266 is a low cost small size IoT development platform. Low-power, highly-integrated Wi-Fi solution. A minimal of 7 external components are required, Wide temperature range: -40°C to +125°C, ESP8266 embedded with 8 Mbit flash and 80kbytes of RAM.

1.2 Introduction to ESP8266

‘Espressif Systems’ Smart Connectivity Platform (ESCP) is a set of high performance, high integration wireless SOCs, designed for space and power constrained mobile platform designers. It provides unsurpassed ability to embed WiFi capabilities within other systems, or to function as a standalone application, with the lowest cost, and minimal space requirement.

ESP8266EX offers a complete and self-contained WiFi networking solution; it can be used to host the application or to offload WiFi networking functions from another application processor. When ESP8266EX hosts the application, it boots up directly from an external flash. It has integrated cache to improve the performance of the system in such applications.

Alternately, serving as a WiFi adapter, wireless internet access can be added to any micro controller based design with simple connectivity (SPI/SDIO or I2C/UART interface).

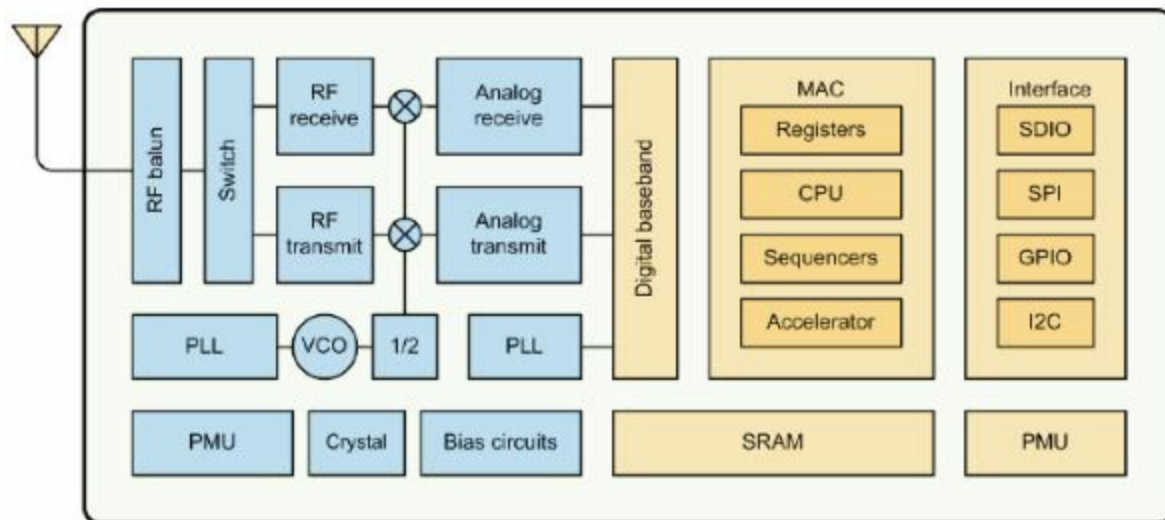


Figure 1.1: ESP8266EX Block Diagram

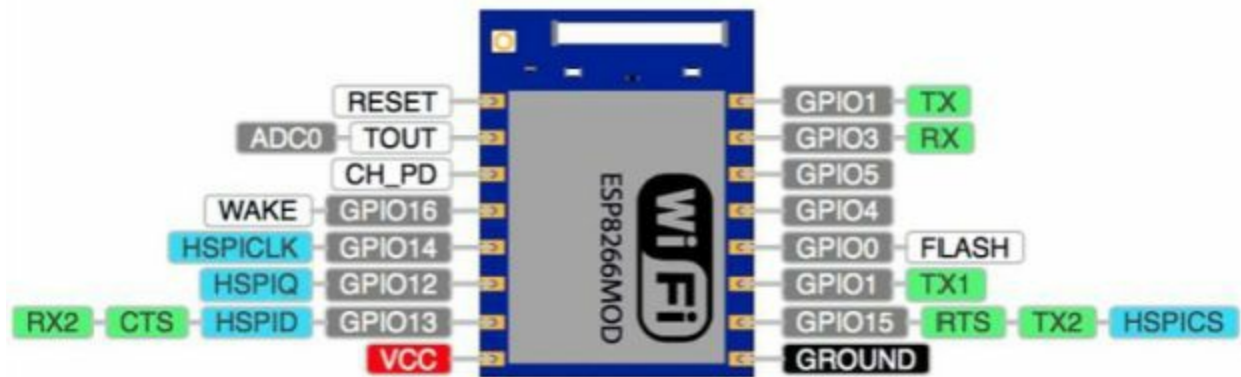
ESP8266EX is among the most integrated WiFi chip in the industry; it integrates the antenna switches, RF balun, power amplifier, low noise receive amplifier, filters, power management modules, it requires minimal external circuitry, and the entire solution, including front-end module, is designed to occupy minimal PCB area. ESP8266EX also integrates an enhanced version of Tensilica's L106 Diamond series 32-bit processor, with on-chip SRAM, besides the WiFi functionalities. ESP8266EX is often integrated with external sensors and other application specific devices through its GPIOs; sample codes for such applications are provided in the software development kit (SDK).

1.3 Features

- Integrated low power 32-bit MCU
- Integrated TCP/IP protocol stack
- Integrated TR switch, balun, LNA, power amplifier and matching network
- 802.11 b/g/n WiFi 2.4 GHz, support WPA/WPA2
- Support STA/AP/STA+AP operation modes
- 10-bit ADC, SDIO 2.0, (H) SPI, UART, I2C, I2S, IR Remote Control, PWM, GPIO
- Deep sleep power <10uA, Power down leakage current < 5uA
- Wake up and transmit packets in < 2ms
- Standby power consumption of < 1.0mW (DTIM3)
- +20 dBm output power in 802.11b mode
- Operating temperature range -40C ~ 125C
- FCC, CE, TELEC, WiFi Alliance, and SRRC certified

1.4 Hardware Overview

Commonly used ESP8266 modules are ESP-12 and ESP-01.



Note: On ESP-12 GPIO1 (TX1) is actually GPIO2 (TX)

Figure 1.2: ESP-12 Pin diagram

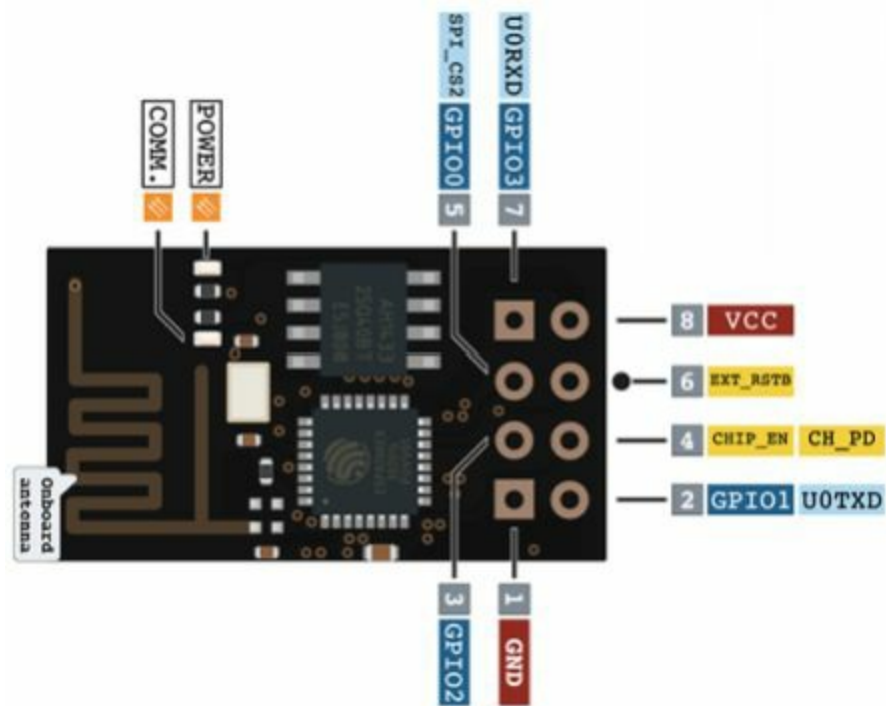


Figure 1.3: ESP-01 Pin diagram

There is no difference in programming ESP-12 or ESP-01, The difference is of only the number of IOs available on the board.

1.5 Pin Definition

Name	Type	Function
VCC	P	Power 3.0 ~ 3.6V
GND	P	Ground
RESET	I	External reset signal (Low voltage level: Active)
ADC(TOUT)	I	ADC Pin Analog Input 0 ~ 1V
CH_PD	I	Chip Enable. High: On, chip works properly; Low: Off, small current
GPIO0(FLASH)	I/O	General purpose IO, If low while reset/power on takes chip into serial programming mode
GPIO1(TX)	I/O	General purpose IO and Serial TXd
GPIO3(RX)	I/O	General purpose IO and Serial RXd
GPIO4	I/O	General purpose IO
GPIO5	I/O	General purpose IO
GPIO12	I/O	General purpose IO
GPIO13	I/O	General purpose IO
GPIO14	I/O	General purpose IO
GPIO15(HSPI_CS)	I/O	General purpose IO, Connect this pin to ground through 1KOhm resistor to boot from internal flash.

Note: TX1 and TX GPIO1 are internally connected, On board blue LED is also connected to this pin

1.5 Electrical Characteristics

Working Voltage: 3.3V

Maximum IO Driving Power I_{MAX} : 12 mA

Maximum IO Voltage Level V_{MAX} : 3.6V

Current Consumption: 100mAmp

2. Hardware Setup ESP8266

2.1 Introduction and selection of ESP8266 module

ESP8266 requires few external components to get started. These are reset circuit, Power supply and few configurations. In this chapter we see each hardware aspects of ESP8266.

2.1.1 ESP-12 Module or ESP-01 Module

ESP-12

Choosing WiFi IOT module is very easy, I recommend you to go with ESP-12 having connections on both sides to get more IOs and Interfaces.

ESP-12 that I use in most applications (Recommended)



Figure 2.1: ESP-12 Module

This module has in-build 10-bit ADC which can take 0 to 1V as input. All IOs of ESP8266 are 3.3V logic compatible. **Do not give 5V to any IO pin or supply of ESP8266.** ESP8266 becomes little hot it is normal. Do not use bottom pins of this module as these pins are connected to internal flash memory. These can be used when you want to run a program from SD card. Total 8 IO, 1 serial and single ADC lines are available on this module. **Remember that GPIO0 is combined with on board LED and Serial Tx.** You can use only one function of this pin. **GPIO15 requires 1K resistor towards ground to boot this device from internal flash.**

ESP-01

Useful where size requirement is small and you want to control one or two Relays (Device on/off). This module doesn't have ADC, so not suitable for sensor interface. There is no difference in programming these modules.



Figure 2.2: ESP-01 Module

2.2 Design of Power supply for ESP8266

ESP8266 operates at 3.3V and consumes around 100mA current, for designing of its power supply you can use LM1117-3.3V Low drop out (LDO) linear regulator which is having current capacity of 800mA. Remember that, this regulator is low drop out regulator. It is better to give 5V as input to the regulator to avoid over heating of regulator. You can create small heat sink on PCB using copper area.

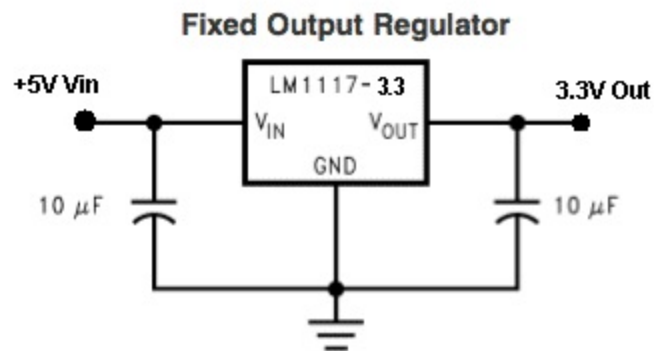


Figure 2.3: Power Supply 3.3V

At lower voltages below 4.7V as Vin device will get programmed but it will not run your code.

2.3 Reset Circuit

Reset circuit is required on only ESP12 module. It's not must but it is better to have it. Putting only 10K resistor as pull-up on reset pin is ok. When RST pin is made low device will reset.

This circuit is similar to the reset circuit of AVR controller. Capacitor (0.1uF) and Resistor (10K).

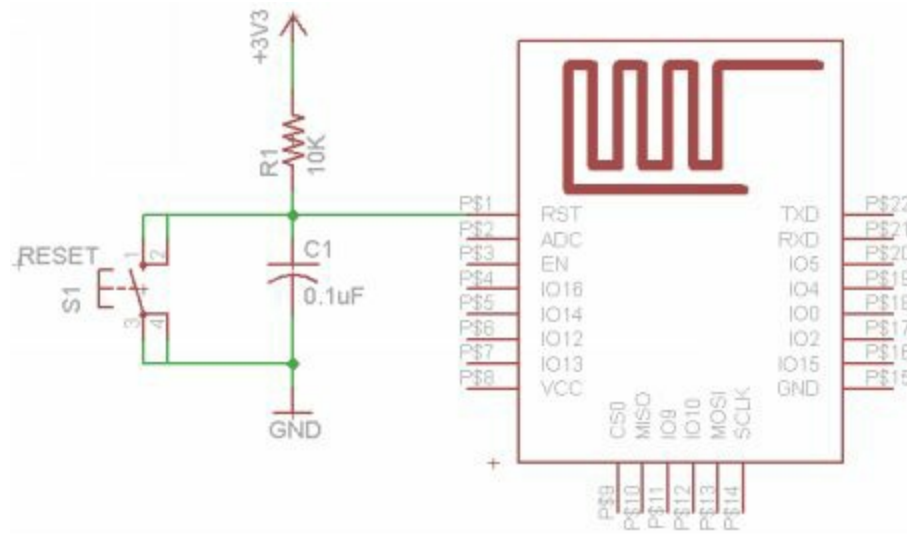


Figure 2.4: Reset Circuit

2.4 Chip Enable and Boot selection

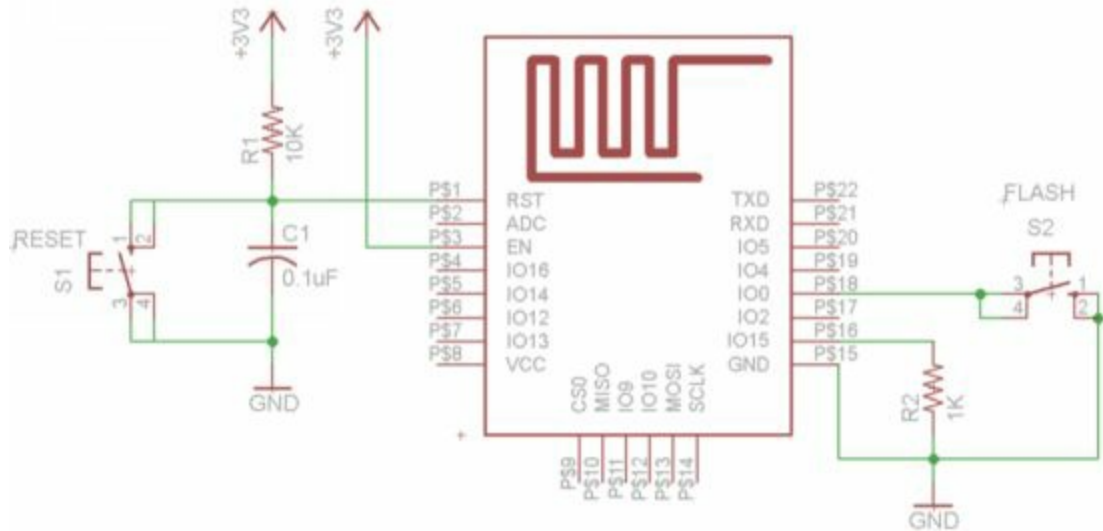


Figure 2.5: Boot and Chip Enable Circuit

GPIO15 (HSPI_CS), Connect this pin to ground through 1KOhm resistor to boot from internal flash. GPIO0 is used to put device in serial programming mode. When GPIO0 is low during power on or reset, this will make device go into serial programming mode.

To put device in serial programming mode

1. Press and hold FLASH (S2) Button.
2. Press and release RESET (S1) button while S2 is in pressed condition.
3. Release FLASH(S2) button after device reset.
4. These steps put ESP8266 in serial programming mode.

Once the device is programmed the internal default flash will be erased, default program can control ESP8266 using AT commands. Once you flash it using above steps you will not able to use AT commands on ESP8266. You have to re-flash with its original AT command boot code to use AT commands again.

In most cases I use direct flashing of ESP8266 this will make design simpler by eliminating need of external controller. Direct programming (flashing) of ESP8266 gives you many advantages over use of AT commands.

2.5 IO Level Conversion 3.3V to 5V interfacing

Most cases we need to interface 5V (TTL) logic level devices with ESP8266. For this you can simple use 1N4148 diode and one pull-up resistor.

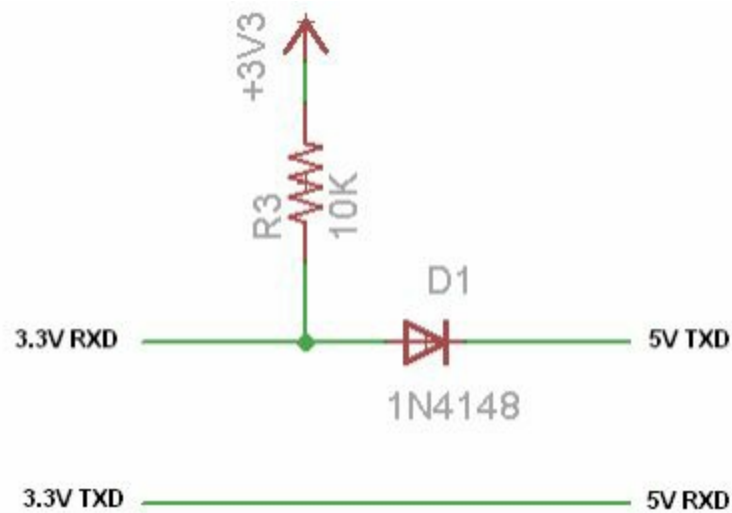


Figure 2.6: Logic Level Conversion for Serial

The above circuit working can be understood using figure 2.7.

1. When we give logic 1 i.e. 5V to TXD diode becomes reverse bias and ESP 3.3V RXD pin gets supply of 3.3V (Logic 1) from pull-up.
2. When we give logic low 0V to TXD pin the diode becomes forward bias and the voltage at its anode terminal drops below 0.7V i.e. logic zero.
3. For TXD pin of ESP we don't need any circuit as we can read 3.3V as logic 1 and 0V as logic 0 using any 5V controller.

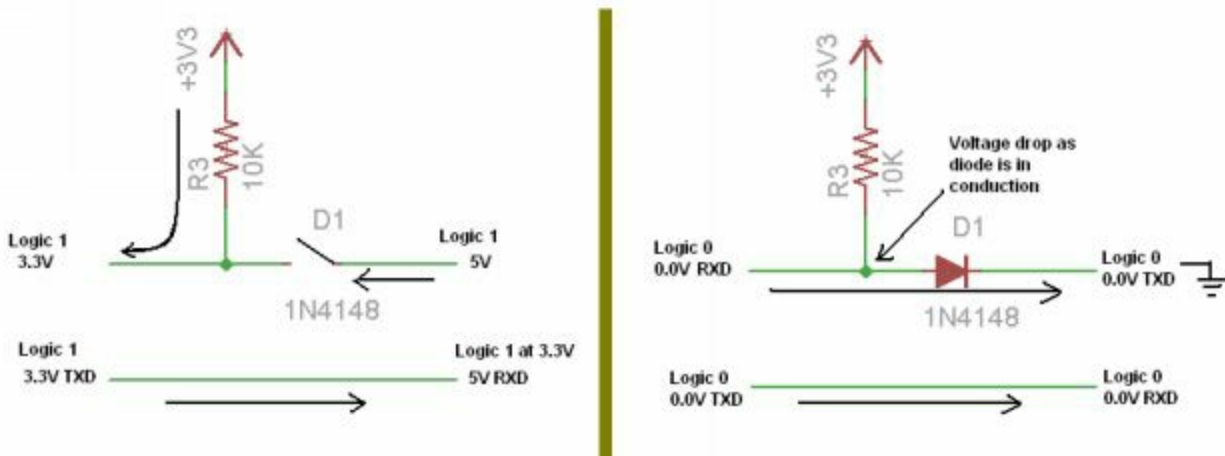


Figure 2.7: Logic level conversion concept

2.6 Serial Interface

For serial interface you have two options go with 3.3V USB to Serial converter or use above circuit (figure 2.6) to do level conversion and use 5V serial converter.

Serial interface is required to program the ESP flash. We take all our examples using only direct flashing method. We are not covering any AT command in this book. We can do a lot more using direct flashing technique.

Which USB to serial converter to use?

There are many USB to Serial converters are available in the market. I recommend you to go with FT232RL based converter as Chinese converter are low cost but they are unreliable and you may face driver issues.

FT232RL based USB to serial converter is best suitable with voltage levels selection jumpers is recommended. Keep USB to serial converter on 3.3V level if you are directly connecting

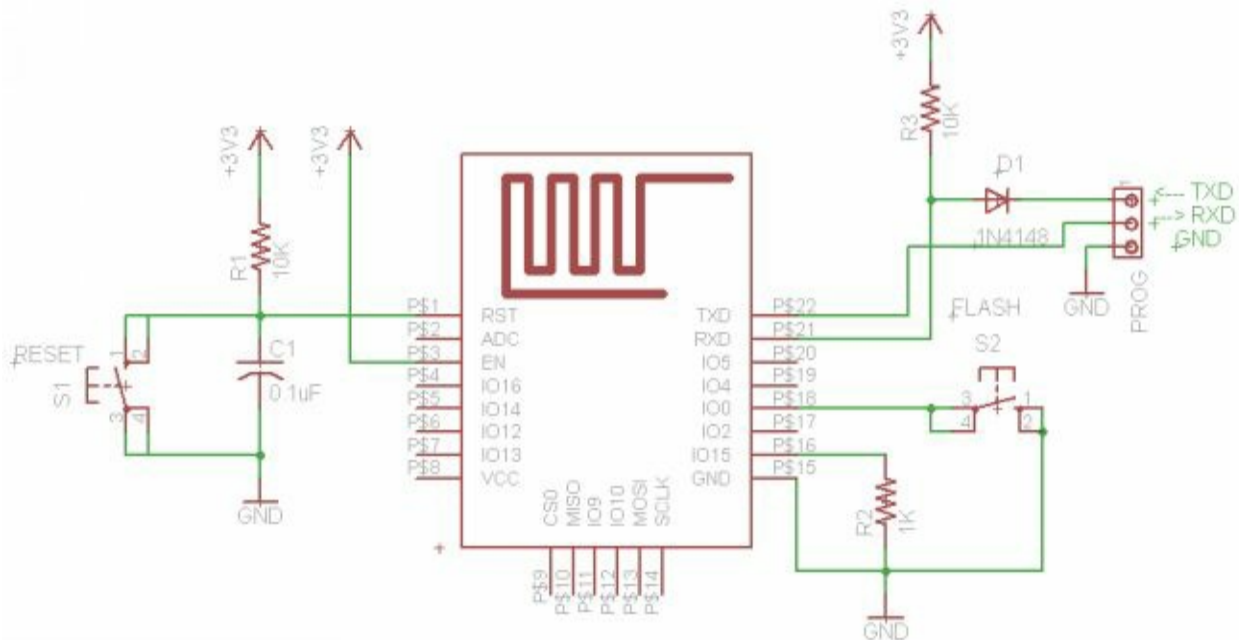


Figure 2.8: Serial Interface

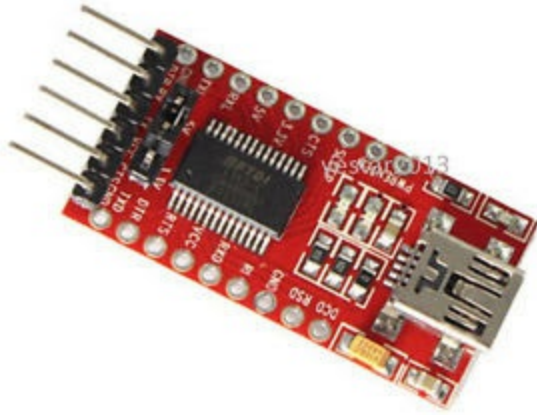


Figure 2.9: USB to Serial Converter

2.7 Complete hardware setup

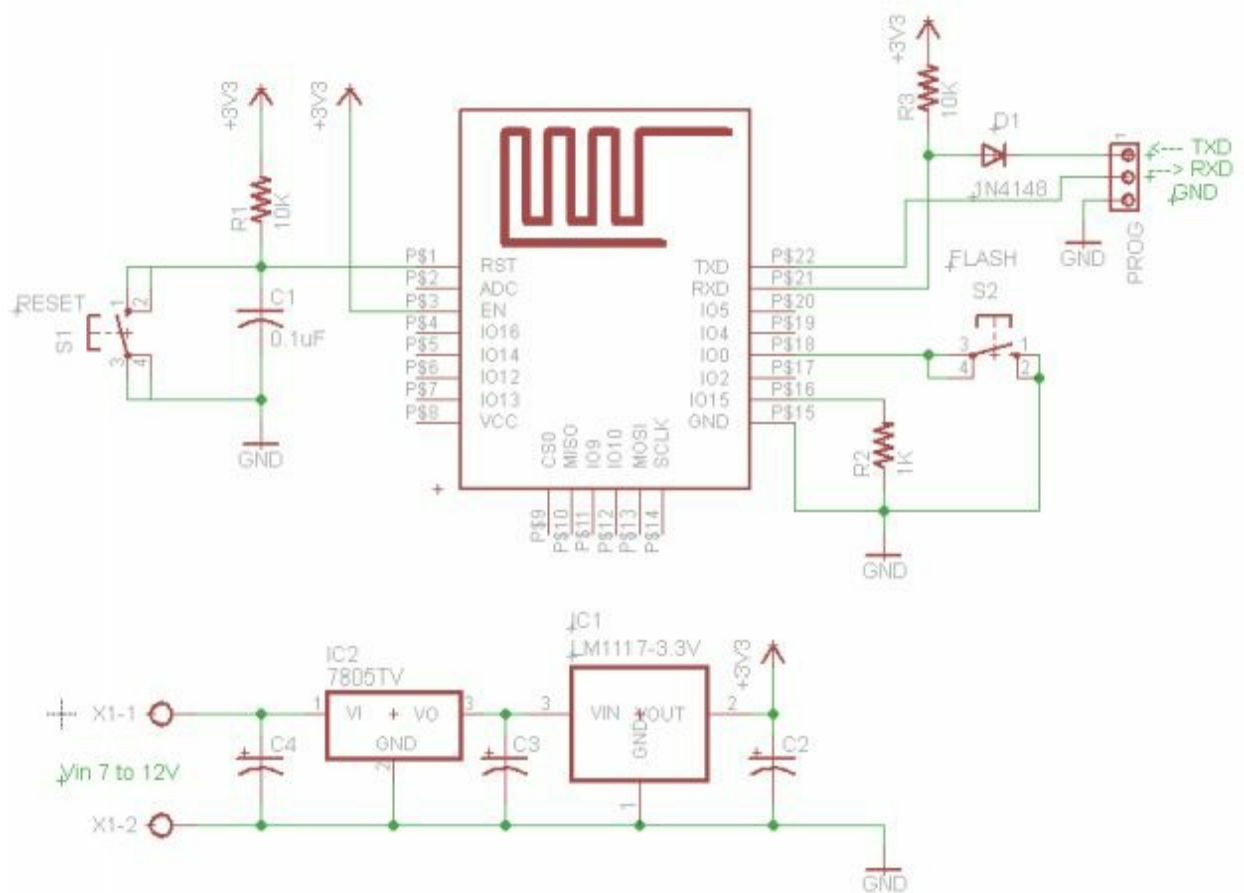


Figure 2.10: Complete Hardware setup for ESP8266

This is the complete hardware setup required to start with ESP8266. You can use 5V from USB also. For 3.3V Serial voltage levels no need of D1 and R3 resistors. To drive RGB LEDs, Relays use ULN2003 as driver IC. ESP8266 have PWM feature you can use it for fading the LEDs.

Let's move towards software requirements of ESP8266 and its setup.

3. Software Setup ESP8266

3.1 Introduction

ESP can be programmed using Arduino Software or using AT commands. In this book we are using Arduino software.

Use of AT commands requires additional controller to generate AT commands. Use of AT commands give you advantage of extra IOs of controller.

Direct flashing of ESP8266 using arduino gives many benefits such as no need of external controller, Program memory of 1 M bytes and RAM of 80K Bytes, Makes simple hardware and software.

Let me clear you first Programming of IoT esp8266 is done in Arduino software and uses same commands that we use for Arduino Uno.

3.2 Steps for software setup

Step 1: Download latest Arduino Software from www.arduino.cc

Starting with 1.6.4, Arduino allows installation of third-party platform packages using Boards Manager. Use 1.6.4 or higher version of Arduino software.

Step 2: Download Arduino Core for ESP8266 WiFi Chip from GitHub

[Download Link](https://github.com/esp8266/Arduino/) (<https://github.com/esp8266/Arduino/>)

Arduino Core for ESP8266 project brings support for ESP8266 chip to the Arduino environment. It lets you write sketches using familiar Arduino functions and libraries, and run them directly on ESP8266, no external microcontroller required.

ESP8266 Arduino core comes with libraries to communicate over WiFi using TCP and UDP, set up HTTP, mDNS, SSDP, and DNS servers, do OTA updates, use a file system in flash memory, work with SD cards, servos, SPI and I2C peripherals.

Step 3: Installing with Boards Manager

Starting with 1.6.4, Arduino allows installation of third-party platform packages using Boards Manager. ESP8266 packages are available for Windows, Mac OS, and Linux (32 and 64 bit).

Install Arduino 1.6.8 from the Arduino website.

Start Arduino and open Preferences window.

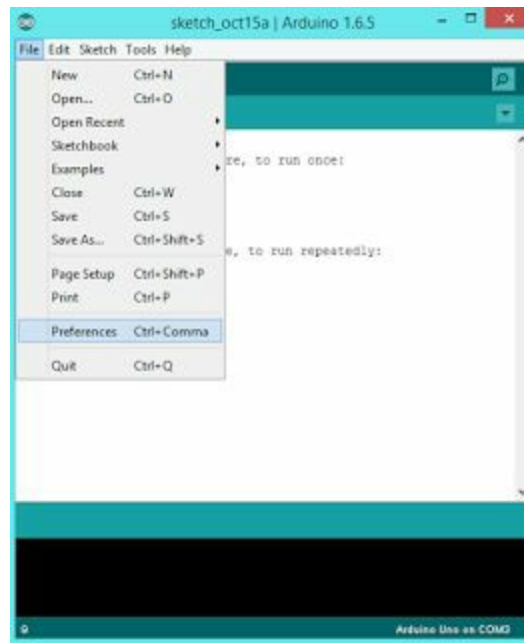


Figure 3.1: Preferences option

Enter

http://arduino.esp8266.com/stable/package_esp8266com_index.json into Additional Board Manager URLs field. You can add multiple URLs, separating them with commas.

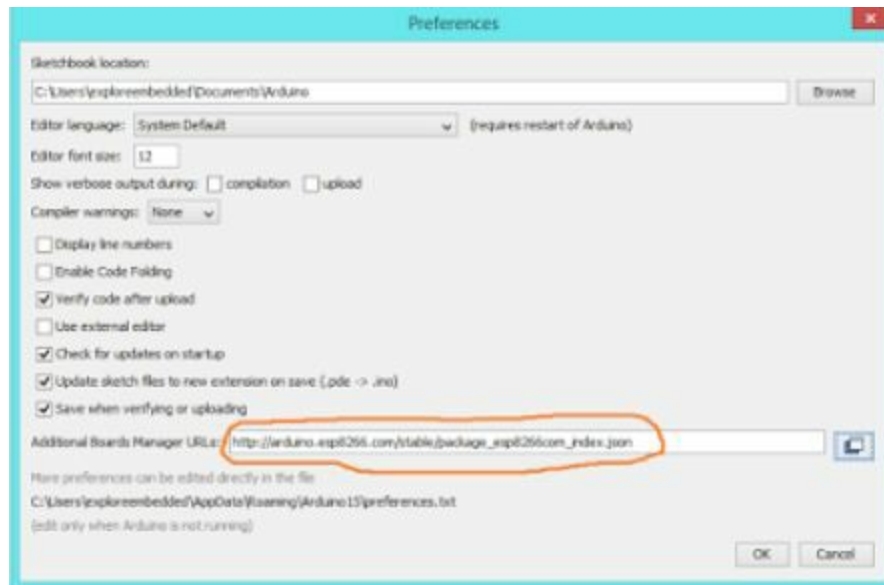


Figure 3.2: Preferences window

Open Boards Manager from Tools > Board menu and install esp8266 platform (and don't forget to select your ESP8266 board from Tools > Board menu after installation).

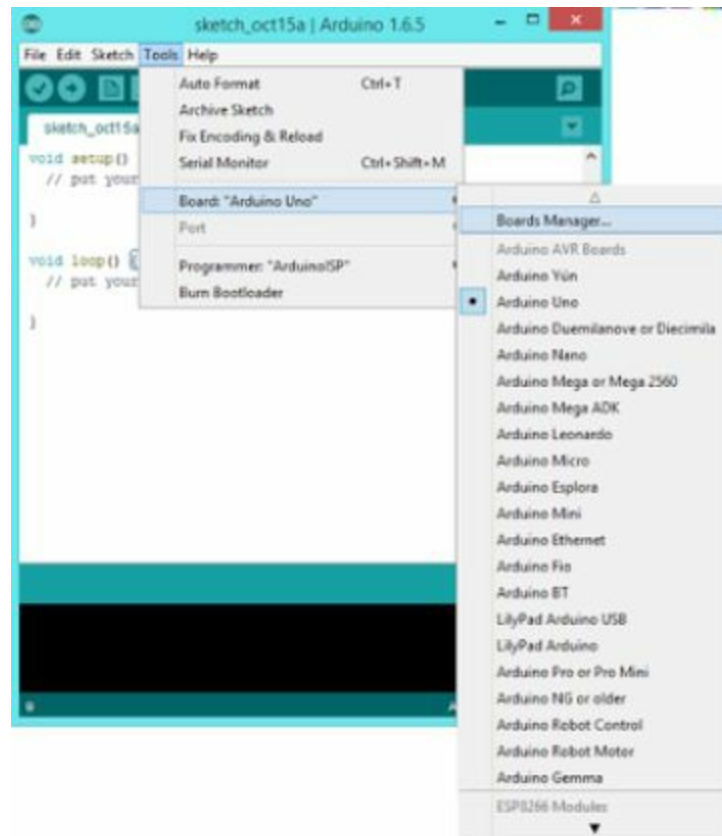


Figure 3.3: Board Manager Option

Select ESP8266 then press install



Figure 3.4: Boards Manager Window

This package installation is from internet, your computer must be connected to internet it is around 250 MBytes. Read details on GitHub.

After installation of ESP8266 packages for Arduino, Select board Generic ESP8266 or ESPino ESP12. Open examples >> ESP8266 >> LED blink and try compiling it.



Figure 3.5: LED Blink example of ESP8266

If it complies successfully skip step 4. To flash the program in ESP8266, press and hold Flash (S2) button and momentarily press Reset button and then release Flash button.

Press upload button from your Arduino window. Program will start uploading.

After uploading onboard blue LED will start to blink. If you passed this blink test you are ready to go further.

Step 4 (If required): After installing everything, you may find trouble in compiling missing .h files

Go to your arduino installation folder (where your arduino.exe is) in this folder create folder “portable”

Copy and paste copy from “C:\Documents and Settings\Administrator\Application Data\Arduino” (check that this address is shown while compilation error missing .h file)

Copy all contains (folder naming “packages” and “staging” and surrounding files)

Paste the copied contains in “portable” folder that you created in Arduino folder.

Check that compilation works now. If all ok go ahead to test it on hardware

4. LED Blink Test

4.1 Introduction

This is our first program for ESP8266. Assuming that you have prepared hardware as per Figure 4.1 (Complete Hardware setup for ESP8266) or you can use node MCU, ESP witty.

In this program we will blink on board LED i.e. present on GPIO2 (TX) pin.

Step 1: Hardware Requirement and Setup

1. USB to Serial converter

USB to serial converter is required if you are using your own build board, Node MCU and ESP witty have in build USB to serial converter.



Figure 2.9: USB to Serial Converter

Set jumper position to 3.3V if you want to directly connect this serial converter to ESP, for 5V you can use 1N4148 (D1) diode and 4.7K (R3) resistor for level conversion.

We need to connect only three lines to ESP Rx, Tx and GND. You can use +5V from USB as input to the LM1117-3.3V regulator. There is no need to connect reset pin to serial converter.

ESP witty is good choice for beginner to learn many things. It contents on board RGB LED, LDR and some switches. Our other programs are made assuming that you are using ESP Witty.

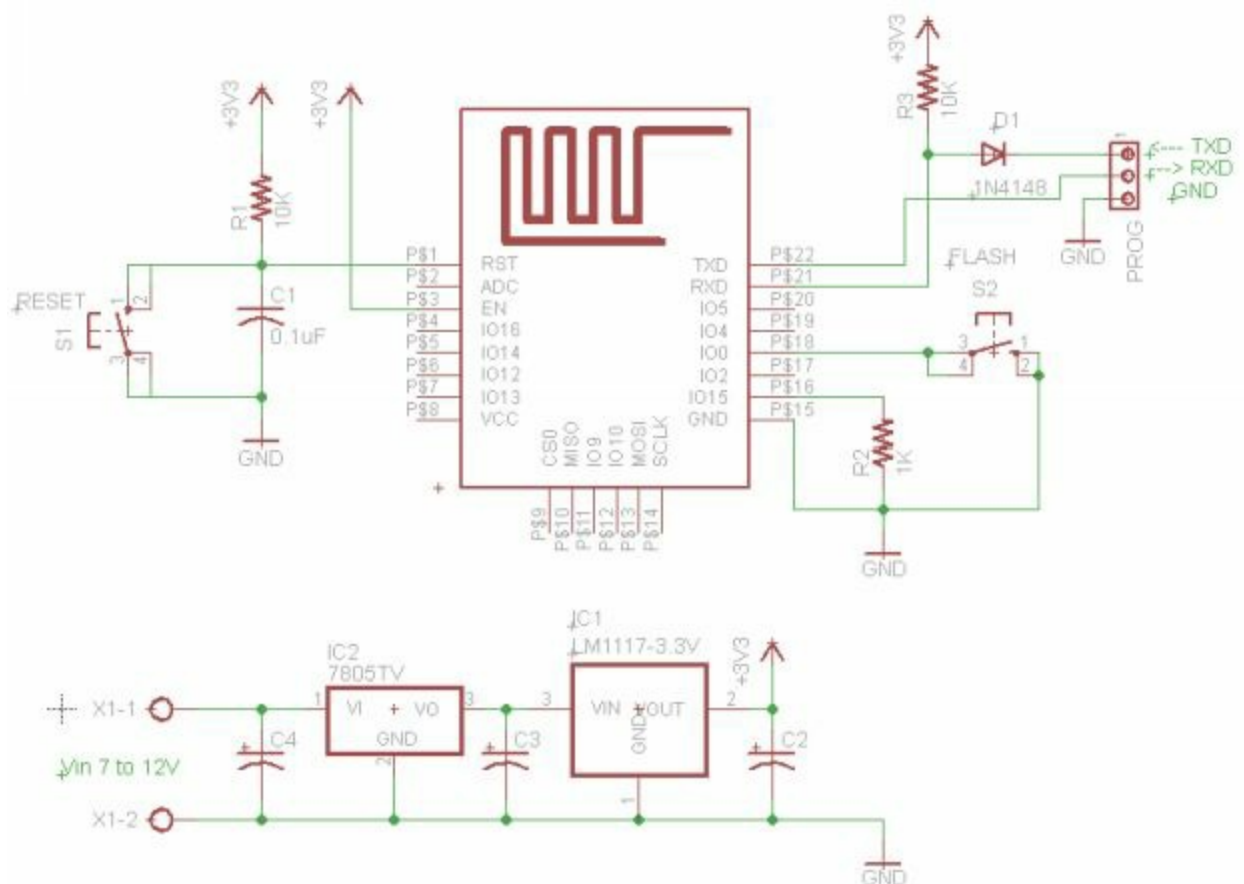


Figure 4.1 Hardware Setup

Step 2: Software

Arduino with ESP packages installed as we discussed in chapter 3. In this lesson we will go in depth as we are starting from zero.

Start your Arduino IDE, select board Generic ESP8266 Module from

Tools >> Boards >> Generic ESP8266 as shown in figure 4.2. You can select ESPino (ESP-12 Module) also, I observed that on some versions of Arduino uploading issue comes if Generic is selected, if you have some problem you can try with ESPino (ESP-12), Programming steps and code remains same for all.

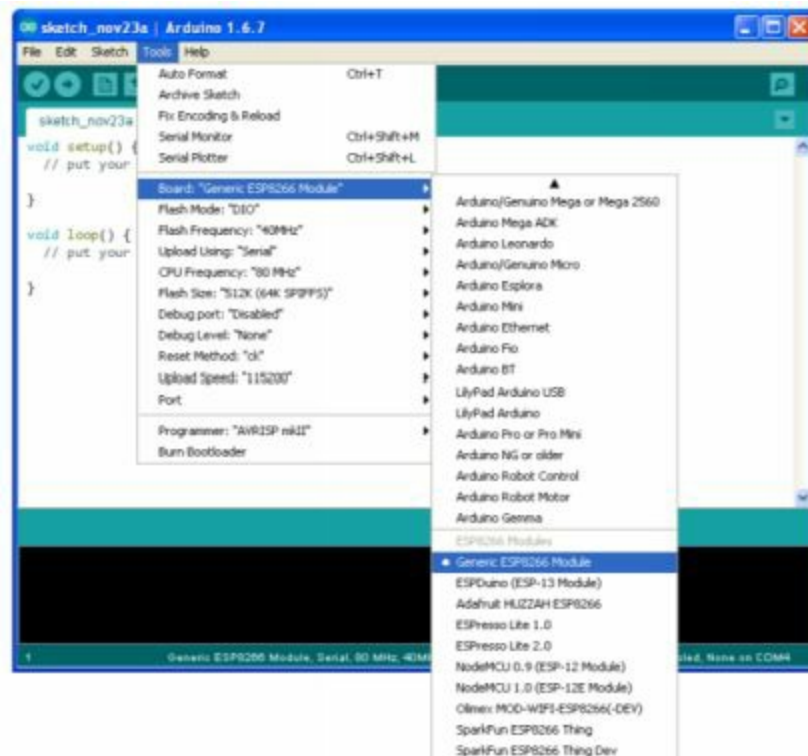


Figure 4.2: Board Selection

Assuming that you have installed USB to serial converter drivers, Select proper comport from Tools >> Port >> Com. Com port number depends on your laptop, to identify which is the correct one you need to unplug USB cable and observe that which com port number is absent and reconnect it and see which is new com port appears.

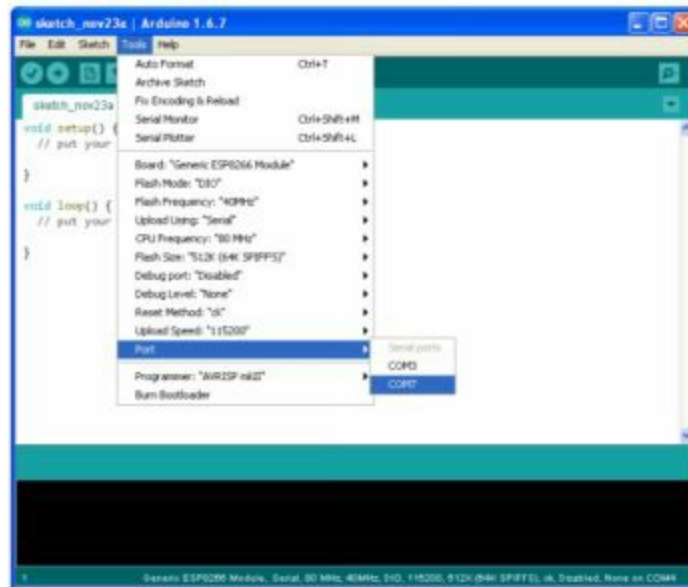


Figure 4.3: Comport Selection

4.2 Software Code for LED blink

ESP-12 has blue color on board LED. Which is connected in reverse i.e. Anode(+ve) of the LED is connected to VCC and cathode (-ve) is connected to ESP-12 GPIO2. It means that LED becomes on when we output LOW and off when we output HIGH. This pin is also Tx, you cannot use serial when using this pin as LED.

All programs are divided in three parts.

1. Define your connections and global variables.

Defining connection makes program understandable as we use names. The number after LED indicates that LED is on GPIO2.

```
#define LED 2 //Define connection of LED
```

2. Power on setup (runs only once at power on or reset)

At power on we initialize all IOs and Serial. We used LED as Output pin so initialize its pin Mode to output.

```
void setup () {  
  pinMode ( LED , OUTPUT ); // Initialize the LED_BUILTIN pin as an output  
}
```

3. Continuously running code (Main program)

In the third part we run a loop continuously. In this part we operate LED. To blink the LED we must turn it on and off for some time. Loop this LED on wait LED off wait.

```
void loop () {  
  digitalWrite ( LED , LOW ); // Turn the LED on (Note that LOW is the voltage level  
                             // but actually the LED is on; this is because  
                             // it is active low on the ESP-12)  
  delay (1000); // Wait for a second  
  digitalWrite ( LED , HIGH ); // Turn the LED off by making the voltage HIGH  
  delay (1000); // Wait for two seconds (to demonstrate the active low LED)  
}
```


Complete Program of LED blinking

Copy paste or write this code in Arduino IDE

```
/*
ESP8266 Blink
Blink the blue LED on the ESP-12 module

The blue LED on the ESP-12 module is connected to GPIO2
(which is also the TXD pin; so we cannot use Serial.print() at the same time)
*/

#define LED 2 //Define connection of LED

void setup () {
  pinMode ( LED , OUTPUT ); // Initialize the LED_BUILTIN pin as an output
}

// the loop function runs over and over again forever
void loop () {
  digitalWrite ( LED , LOW ); // Turn the LED on (Note that LOW is the voltage level
                             // but actually the LED is on; this is because
                             // it is active low on the ESP-12)
  delay (1000); // Wait for a second
  digitalWrite ( LED , HIGH ); // Turn the LED off by making the voltage HIGH
  delay (1000); // Wait for two seconds (to demonstrate the active low LED)
}
```

Compile the code by pressing this button



What happen if we make some mistakes in code? Let's see I have intentionally removed semicolon after line 21. And compiled the code.

```
digitalWrite ( LED , HIGH ) // Turn the LED off by making the voltage HIGH
```

In the below figure 4.5, You can observe that it high light the next line i.e. Line number 22 “*delay(1000);*” and says expected ‘;’ before “*delay*”.

By reading the errors you can identify the exact problems in the code. Always use line number reference to identify problems. Second important thing is separate the code with space and tabs to make it more understandable. You can observe there is space before start of loop() and tabs are given in the loop and setup.

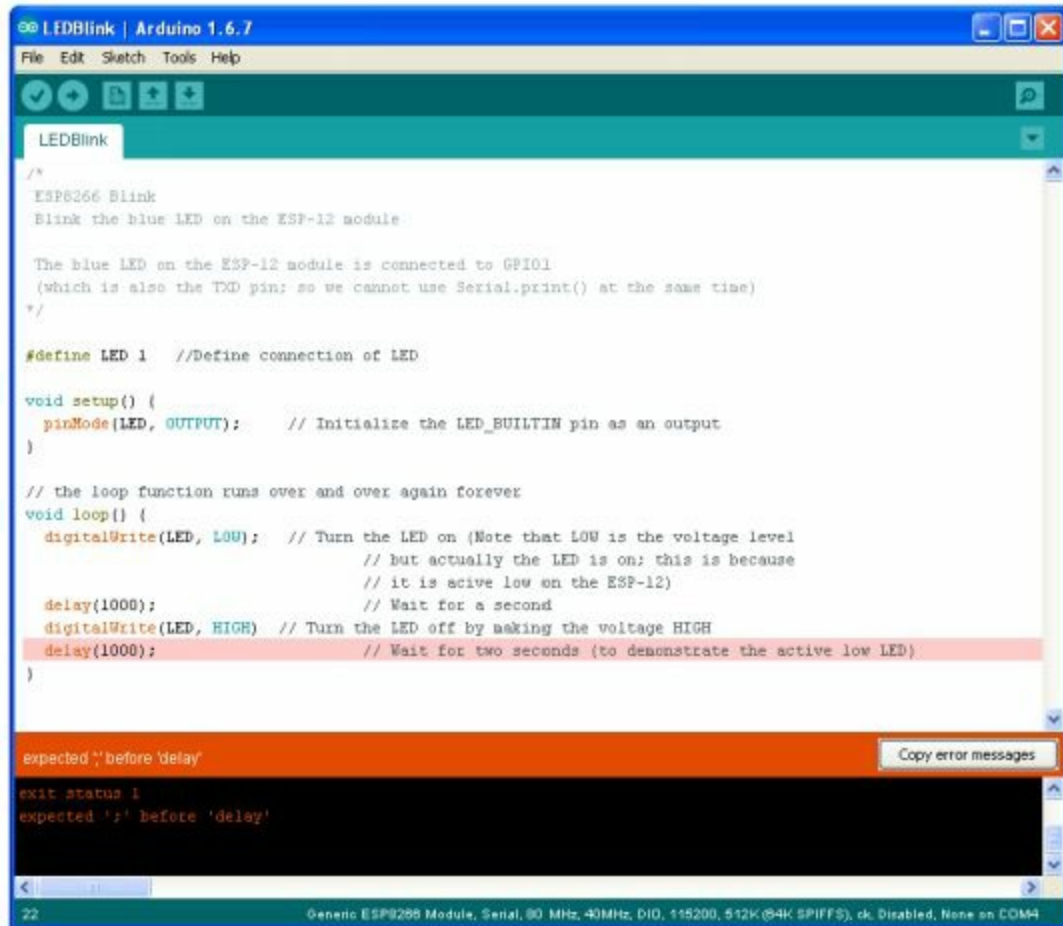


Figure 4.5: Compilation Error

4.3 Program upload to the board

To upload program to ESP on some boards you need to hold flash button while uploading and some boards require reset with holding of flash button.

To put device in serial programming mode

1. Press and hold FLASH (S2) Button.
2. Press and release RESET (S1) button while S2 is in pressed condition.
3. Release FLASH(S2) button after device reset or on some boards hold it.
4. These steps put ESP8266 in serial programming mode.

For ESP witty you have to hold the Flash button only while programming, No need to reset. Few boards may not require any user actions.


Press upload button  If you selected incorrect comport you may get these errors

Figure 4.6 error is only due to incorrect comport



Figure 4.6: Com port Error

Figure 4.7 error may occurs due to incorrect comport or Flash button is not pressed properly (i.e. device is not in serial programming mode)



Figure 4.7: Uploading error (synch failed)

If all is ok, you will see “Uploading...”, blue led will flash and finally “Done uploading”

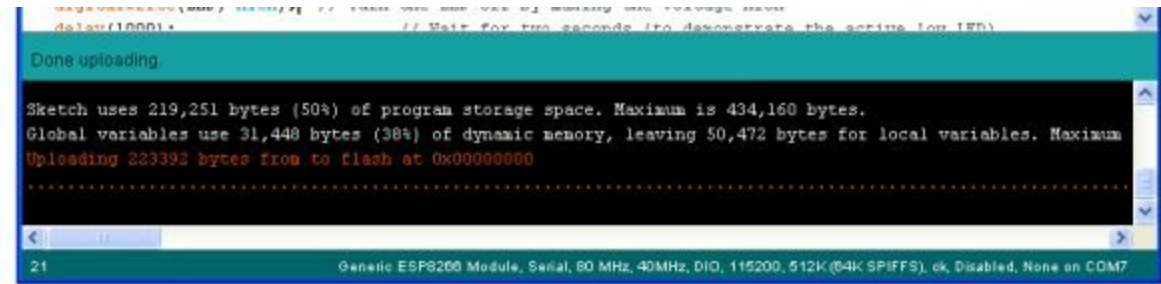


Figure 4.8: Uploading Successful

Once the uploading is successful press RST button on board and observe that on board blue LED blinks.

5. Hardware and Software Serial

5.1 Introduction

We are quickly moving towards higher level interface. Serial interface is common requirement all application development. We already seen how to do 3.3V to 5V level conversion for converting serial TTL to RS232 level from 3.3V you can use MAX3232 it operates at 3.3V levels.

Keeping hardware same as we discussed in previous sections. Let's move towards programming part. On ESP8266 we one hardware serial i.e. GPIO2 (Tx) and GPIO3 (Rx).

Hardware Serial Programming is similar to the Arduino Serial. Remember that few USB to Serial converter does not support higher baud rate. It is better to keep baud rate below 115200.

Serial interface is useful for debugging the programs by sending some debug info to serial.

5.2 Hardware Serial Interface

Assuming that you have already connected serial with your USB to Serial converter or You are using ESP Witty, Node MCU.

Serial communication on pins TX/RX uses TTL logic levels 3.3V. Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your ESP8266 board.

Serial is used for communication between the Arduino board and a computer or other devices. All ESP boards have at least one serial port (also known as a UART or USART): Serial. It communicates on RX and TX.

You can use the Arduino IDE environment's built-in serial monitor to communicate with an ESP board. Click the Tools>>Serial monitor button in the toolbar and select the same baud rate used in the call to begin().

Before we start our program lets understand commonly used serial commands

Serial.begin(9600);

This command is used to initialize serial port with 9600 baud rate. It is used only when you initialize serial i.e. in **Setup()** or when you want to change the baud rate. Baud rate is number of bits transmitted per second. Higher the baud rate higher the speed of communication.

Use lower baud rate if cable length is more. Standard baud rates are 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 250000.

```
void setup () {  
    Serial . begin (9600); // opens serial port, sets data rate to 9600 bps  
}  
  
void loop () {  
}
```

Serial.print(); and Serial.println();

Serial.print prints text on same line. Serial.println puts carriage return at the end of line.

Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is. For example:

`Serial.print(65)` gives "65"

`Serial.print(char(65))` gives "A", 65 is ASCII code of A

`Serial.print(1.23456)` gives "1.23"

`Serial.print('N')` gives "N"

`Serial.print("Hello world.")` gives "Hello world."

An optional second parameter specifies the base (format) to use; permitted values are BIN (binary, or base 2), OCT (octal, or base 8), DEC (decimal, or base 10), HEX (hexadecimal, or base 16). For floating point numbers, this parameter specifies the number of decimal places to use. For example:

`Serial.print(78, BIN)` gives "1001110"

`Serial.print(78, OCT)` gives "116"

`Serial.print(78, DEC)` gives "78"

`Serial.print(78, HEX)` gives "4E"

`Serial.println(1.23456, 0)` gives "1"

`Serial.println(1.23456, 2)` gives "1.23"

`Serial.println(1.23456, 4)` gives "1.2346"

`Serial.print("\t");` // prints a tab

`Serial.print("\n");` // inserts line break

Serial.write();

Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the characters representing the digits of a number use the **print()** function instead.

`Serial.print(65);` gives "65"

`Serial.print(char(65));` gives "A", 65 is ASCII code of A

`Serial.write(65);` gives "A", 65 is ASCII code of A

Serial.available();

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes).

Serial.read();

Reads incoming serial data as single byte.

5.3 Hardware Serial Program for ESP

Programming of ESP is same as arduino.

```
/*  
ESP8266 Serial Communication
```

```
The blue LED on the ESP-12 module is connected to GPIO2  
(which is also the TXD pin; so we cannot use Serial.print() at the same time)  
*/
```

```
void setup () {  
  Serial . begin (9600);  // Initialize the Serial interface with baud rate of 9600  
}  
  
// the loop function runs over and over again forever  
void loop () {  
  if( Serial . available ()>0)  //Checks is there any data in buffer  
  {  
    Serial . print ("We got:");  
    Serial . print (char( Serial . read ())); //Read serial data byte and send back to serial monitor  
  }  
  else  
  {  
    Serial . println ("Hello World.."); //Print Hello word every one second  
    delay (1000);           // Wait for a second  
  }  
}
```

Result:

Type some text and send, Try modifying above code and see results.

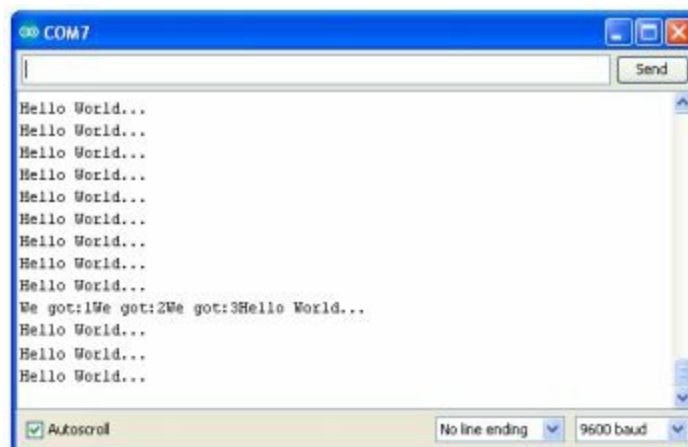


Figure 5.1: Serial monitor

5.4 Software Serial Program for ESP

Software serial uses timer, be careful when you are using software serial. Timer is also used for WiFi communication section if you don't give enough time to WiFi routines it will create problem of stack error or misbehavior. It is better to use software serial only when you need two serial ports and also avoid use of software serial for data reception.

Software serial can be implemented on any GPIO pin of ESP8266.

For this you need SoftwareSerial Library for ESP.

[Download from here Link](#)

To add library in Arduino IDE, Library path is different for ESP8266

C:\Program

Files\Arduino\portable\packages\esp8266\hardware\esp8266\2.1.0\libraries

Software serial library can be included in your program using below commands

```
#include <SoftwareSerial.h>
```

```
SoftwareSerial swSer(14, 12, false, 128);
```

Command format is **SoftwareSerial(rxPin, txPin, inverse_logic, buffer size);**

Program for software serial

We are using GPIO14 as Rx pin and GPIO15 as Tx pin

```
#include <SoftwareSerial.h>
```

```
SoftwareSerial swSer (14, 12, false, 128); //Define hardware connections
```

```
void setup () {
```

```
  Serial . begin (115200); //Initialize hardware serial with baudrate of  
  115200
```

```
  swSer . begin (115200); //Initialize software serial with baudrate of
```

115200

```
Serial . println ("\nSoftware serial test started");

for (char ch = ' '; ch <= 'z'; ch ++) { //send serially a to z on software
serial
    swSer . write ( ch );
}
swSer . println ("");

}

void loop () {
    while ( swSer . available () > 0) {           //wait for data at software serial
        Serial . write ( swSer . read ());        //Send data recived from software
serial to hardware serial
    }
    while ( Serial . available () > 0) { //wait for data at hardware serial
        swSer . write ( Serial . read ()); //send data recived from hardware
serial to software serial
    }

}
```

Results:

Observe both serials and send some data to both serial data, data send from hardware serial is sent out from software serial and vice versa.

I will not recommend use of software serial until it becomes must. It disturbs WiFi functionality.

6. Internal EEPROM

6.1 Introduction

ESP8266 have 512 bytes of internal EEPROM, it is useful when you want to store some settings, such as IP address of server, WEP key, SSID of WiFi.

In this lesson we will see writing data to EEPROM and reading from EEPROM

To write byte to EEPROM we need two commands

```
EEPROM . write ( addr , data );  
EEPROM . commit ();
```

To read single byte from EEPROM

```
Char Data;  
Data = EEPROM . read ( addr );
```

We have only read and write bytes commands, for writing String, Integer and other data types we have to split data into bytes first and rejoin it while reading.

6.2 Write data to EEPROM

We write some characters and String to EEPROM, This program only writes data to EEPROM we read it using another program. Bytes ABC are stored at address 0x00,0x01,0x02 respectively and string is stored from 0x0F.

Remember that ESP requires `EEPROM.commit()` command. Without this data will not be saved to EEPROM.

```
/*
 * EEPROM Write
 *
 * Stores values and string to EEPROM.
 * These values and string will stay in the EEPROM when the board is
 * turned off and may be retrieved later by another sketch.
 */

#include <EEPROM.h>

// the current address in the EEPROM (i.e. which byte
// we're going to write to next)
int addr = 0;

void setup ()
{
  EEPROM.begin(512); //Initialize EEPROM

  // write appropriate byte of the EEPROM.
  // these values will remain there when the board is
  // turned off.

  EEPROM.write ( addr , 'A'); //Write character A
  addr ++; //Increment address
  EEPROM.write ( addr , 'B'); //Write character A
  addr ++; //Increment address
  EEPROM.write ( addr , 'C'); //Write character A

  //Write string to eeprom
  String www = "www.circuits4you.com";
  for(int i =0; i < www.length (); i ++) //loop upto string lenght www.length() returns length of
string
  {
    EEPROM.write (0x0F+ i , www [ i ]); //Write one by one with starting address of 0x0F
  }
  EEPROM.commit (); //Store data to EEPROM
}

void loop ()
```



```
{  
  //We dont have anything in loop as EEPROM writing is done only once  
  delay (10);  
}
```

6.2 Read data from EEPROM

Now let's read data from EEPROM and show it on serial monitor. You can combine these two programs as per your need.

String is array of characters.

```
/*
 * EEPROM Read
 * Circuits4you.com
 */

#include <EEPROM.h>

// the current address in the EEPROM (i.e. which byte
// we're going to read from)
int addr = 0;

void setup ()
{
  EEPROM . begin (512); //Initialize EEPROM
  Serial . begin (9600); //Serial communication to display data
  // read appropriate byte of the EEPROM.
  Serial . println ("" ); //Goto next line, as ESP sends some garbage when you reset it
  Serial . print (char( EEPROM . read ( addr ))); //Read from address 0x00
  addr ++; //Increment address
  Serial . print (char( EEPROM . read ( addr ))); //Read from address 0x01
  addr ++; //Increment address
  Serial . println (char( EEPROM . read ( addr ))); //Read from address 0x02

  //Read string from eeprom
  String www ;
  //Here we dont know how many bytes to read it is better practice to use some terminating character
  //Lets do it manually www.circuits4you.com total length is 20 characters
  for(int i =0; i <20; i ++ )
  {
    www = www + char( EEPROM . read (0x0F+ i )); //Read one by one with starting address of
    0x0F
  }

  Serial . print ( www ); //Print the text on serial monitor
}

void loop ()
{
  //We dont have anything in loop as EEPROM reading is done only once
  delay (10);
}
```

Results

You will see what we have written to the EEPROM will appear on Serial terminal. ESP always sends some garbage to serial monitor when you reset it, skip first line.

In this example we have taken how to write String and read String from EEPROM, we are going to use String type data many times in this book.

7. ADC in Serial out

7.1 Introduction

The ESP8266 has in built single channel ADC that can measure between 0V and 1V with 10 bit resolution. To read higher voltages you need external voltage divider. ADC is available on ESP12.

7.2 Voltage Divider

Resistor voltage dividers are commonly used to create reference voltages, or to reduce the magnitude of a voltage so it can be measured.

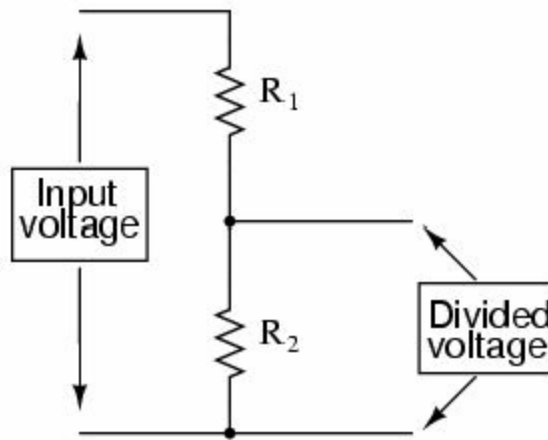


Figure 7.1: Voltage divider circuit

Formula to find V_{out} is. Assume $(R_1 + R_2) = 10K$ for 5V input or higher depending on input voltage level and find R_2 .

$$V_{out} = \frac{V_{in} \times R_2}{(R_1 + R_2)}$$

For 5V use $R_1 = 8K$ and $R_2 = 2K$

7.3 ADC Programming

ADC programming is similar to Arduino, Only difference is you have 0 to 1V with 10-bit resolution.

```
/*
 * ADC reading in ESP8266
 * circuits4you.com
 */

void setup () {
  Serial . begin (9600); //Initialize serial
}

void loop () {
  int sensor ;           //Define variable to read ADC data
  sensor = analogRead ( A0 ); //Read ADC
  Serial . print ("ADC Value:"); //Print Message
  Serial . println ( sensor ); //Print ADC value
  delay (500);           //Have some delay to make it visible
}
```


Results:

Observe the ADC value on serial monitor.

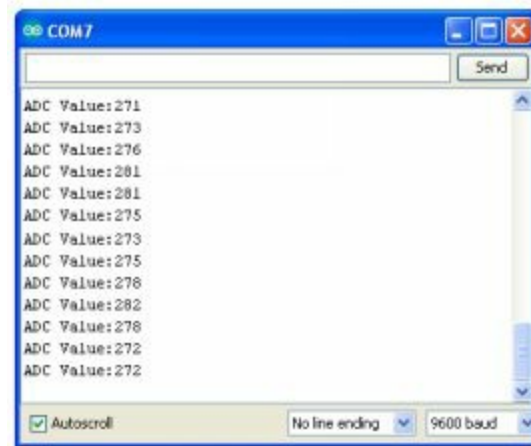


Figure 7.2: ADC readings on serial monitor

We have learned how to use ESP hardware and its programming, now we move towards IoT application developments.

Tips: Once the programming is started you can release “Flash” button on ESP Witty (during flashing of blue LED).

8. HTML, CSS Basics

8.1 Introduction

HTML is required in IoT, Internet is all about web pages. All web pages are created using HTML. You can find more information about HTML and CSS on w2schools website.

What is HTML?

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

We focus more on commonly used HTML tags.

8.2 Write your first HTML code

HTML tags are element names surrounded by angle brackets:

<tagname>content goes here...</tagname>

- HTML tags normally come in pairs like <p> and </p>
- The first tag in a pair is the start tag, the second tag is the end tag
- The end tag is written like the start tag, but with a forward slash inserted before the tag name

Tip: The start tag is also called the opening tag, and the end tag the closing tag.

Sample HTML program:

```
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>My First Heading</h1>
    <p>My first paragraph.</p>
  </body>
</html>
```

HTML is not case sensitive language. But remember that variable names must be same case.

Example:

var is not VAR

var is not Var

var is var

VAR is VAR

Var is Var

Write HTML Using Notepad or SublimeText

Web pages can be created and modified by using professional HTML editors.

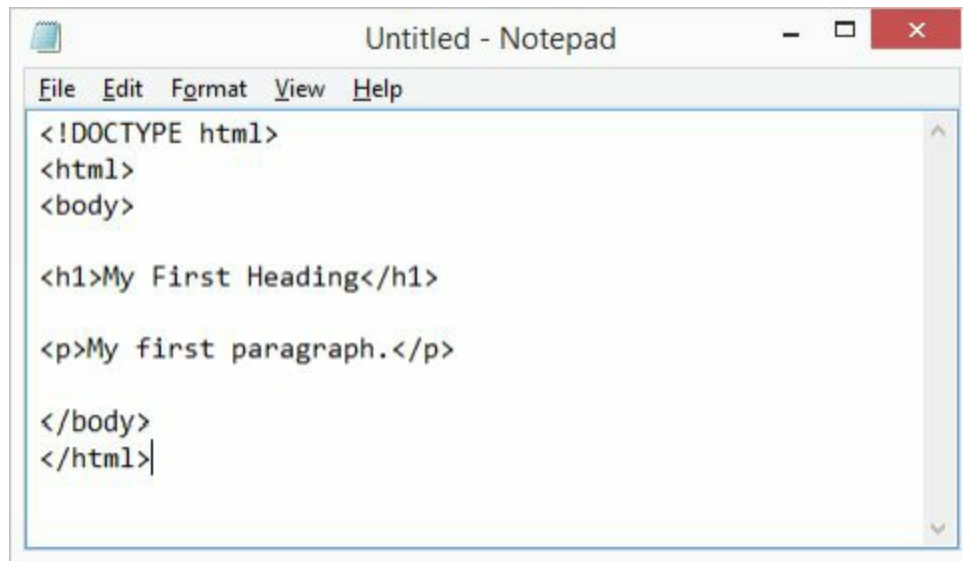
However, for learning HTML we recommend a simple text editor like

Notepad (PC) or TextEdit (Mac).

We believe using a simple text editor is a good way to learn HTML.

Follow the three steps below to create your first web page with Notepad or TextEdit.

Step 1: Open notepad and type simple HTML code

A screenshot of a Notepad window titled "Untitled - Notepad". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". The text area contains the following HTML code:

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

The code is written in a monospaced font. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Figure 8.1: Notepad HTML code writing

Step 2: Save your HTML

Save the file on your computer. Select **File > Save as** in the Notepad menu.

Name the file "**index.htm**" and set the encoding to **UTF-8** (which is the preferred encoding for HTML files). File extension must be .htm or .html

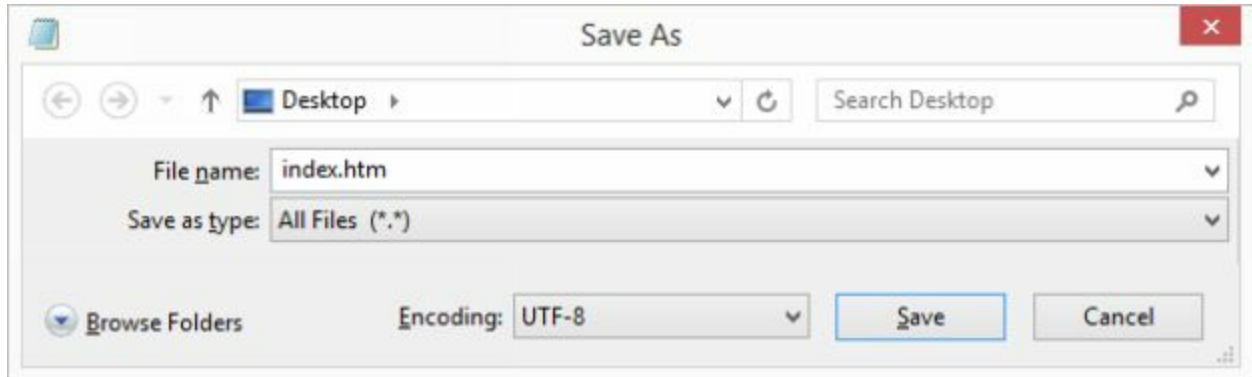


Figure 8.2: Save As window

Step 3: View the HTML Page in Your Browser

Open the saved HTML file in your favorite browser (double click on the file, or right-click - and choose "Open with").

The result will look much like this:



Figure 8.3: HTML Page results

8.3 HTML Documents and Tags

All HTML documents must start with a document type declaration: `<!DOCTYPE html>`.

The HTML document itself begins with `<html>` and ends with `</html>`.

The visible part of the HTML document is between `<body>` and `</body>`.

HTML Headings

HTML headings are defined with the `<h1>` to `<h6>` tags.

`<h1>` defines the most important heading. `<h6>` defines the least important heading:

Example:

```
<h1>This is heading 1</h1>
```

```
<h2>This is heading 2</h2>
```

```
<h3>This is heading 3</h3>
```

HTML Links

HTML links are defined with the `<a>` tag:

```
<a href="http://www.circuits4you.com">This is a link</a>
```

The link's destination is specified in the href attribute. Attributes are used to provide additional information about HTML elements.

HTML Images

HTML images are defined with the `` tag.

The source file (src), alternative text (alt), width, and height are provided as attributes:

```

```

Do Not Forget the End Tag some tags will not require end tag such as img.

HTML Tables

An HTML table is defined with the `<table>` tag.

Each table row is defined with the `<tr>` tag. A table header is defined with the `<th>` tag. By default, table headings are bold and centered. A table data/cell is defined with the `<td>` tag.

```

<table style="width:100%; border: 1px solid black;">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>

```

The <form> Element

The HTML <form> element defines a form that is used to collect user input. An HTML form contains form elements.

Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, and more.

The <input> Element

The <input> element is the most important form element.

The <input> element can be displayed in several ways, depending on the **type** attribute.

Here are some examples:

Type	Description
<input type="text">	Defines a one-line text input field
<input type="radio">	Defines a radio button (for selecting one of many choices)
<input type="submit">	Defines a submit button (for submitting the form)

8.4 Styling HTML with CSS

CSS stands for **Cascading Style Sheets**.

CSS describes how HTML elements are to be displayed on screen, paper, or in other media.

CSS saves a lot of work. It can control the layout of multiple web pages all at once.

CSS can be added to HTML elements in 3 ways:

Inline - by using the style attribute in HTML elements

Internal - by using a <style> element in the <head> section. We use this method as we are programming HTML pages in ESP8266.

External - by using an external CSS file

The most common way to add CSS, is to keep the styles in separate CSS files.

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
  color: blue;
  font-family: verdana;
  font-size: 300%;
}
p {
  color: red;
  font-family: courier;
  font-size: 160%;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

CSS and HTML is covered in details with applications in further examples. CSS is used to give styles

to HTML page such as colors, fonts and size, animations, shadows.

9. Hello world! Web Server

9.1 Introduction

A Web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients.

To implement web server on ESP, there are two ways to make your first web server first connect to your WiFi router or make ESP as access point.

9.2 Web Server Step by Step

As we know that all web servers have a web page to be served. First make a web page using HTML as we learned in pervious chapter and test it on your computer.

Step 1: Create a good looking Web page

Open your note pad and start writing HTML code. Save as index.htm.

```
<HTML>
  <HEAD>
    <TITLE>My first web page</TITLE>
  </HEAD>
  <BODY>
    <CENTER>
      <B>Hello World.... </B>
    </CENTER>
  </BODY>
</HTML>
```

<HEAD> and <TITLE> is used to give page title, which is visible in top of the browser.

<CENTER> tag is used for center alignment of text, is used to make text bold.

Test your web page

Open your web page in web browser. You can observe that at the top you see Title “My first web page”. And Web page with **Hello World...** message.

To see the changes in your HTML code simply change you HTML program and press refresh in browser. It will reflect immediately. This way you can make your webpage test it, then deploy it on ESP8266. It saves your lot of time.

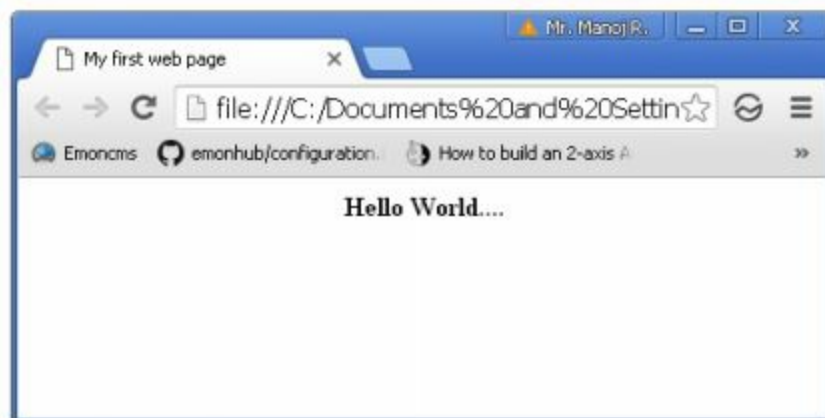


Figure 9.1: Your first webpage

Step 2: Creating web server on ESP8266

ESP can act as access point and it can connect to access point or both.

First we make program to connect to WiFi hot spot (Access Point)

Program to connect to Access point and Make web server

We need these libraries to make web server.

ESP8266WiFi.h is required for doing all WiFi related functionalities such as connection, AP, etc.

WiFiClient.h this file is required to send request to web browser

ESP8266WebServer.h it handles all HTTP protocols

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
```

Define your SSID and Password of your WiFi router, where the ESP connects

```
//SSID and Password of your WiFi router
const char* ssid = "your_ssid";
const char* password = "password";
```

Web server is on port 80, you can use other ports also, default HTTP port is 80, to open web page with different port number you have to enter port number after IP address. Ex. For port number 81 you have to type 192.168.2.2:81 in browser.

```
ESP8266WebServer server(80); //Server on port 80
```

There are two ways to make web server one is to connect to WiFi hot spot or make ESP as hot spot (Access Point).

This command is used to connect to your WiFi Access point. The term Access Point (AP) is same as WiFi Hot Spot. If the network is open you can remove password field from command.

```
WiFi . begin ( ssid , password );    //Connect to your WiFi router
```

After connection request we wait for WiFi to get connect. ESP8266 once connected and disconnected afterwards due to signal loss or any reason, there is no need to give this command again, it will try to connect again automatically. This is handled by its OS, you may find some stack errors displayed in serial monitor, and these errors come from its internal OS.

```
// Wait for connection
while ( WiFi . status () != WL_CONNECTED ) {
  delay (500);
  Serial . print (".");
}
```

To get IP address i.e. assigned to ESP8266 by your WiFi router use this command

```
WiFi . localIP ()
```

When client request a web page by entering ESP IP address which data to be sent is handled by subroutine and that subroutine name is defined in `server.on(path,subroutine_name)`.

```
server . on ( "/", handleRoot );    //Which routine to handle at root location
```

Example: If you have two pages you can define like this

```
Server.on("/",root);                //192.168.2.2 (IP of ESP)  this is root location
```

```
Server.on("/page1",First_page);    //"192.168.2.2/page1"  this is first page location
```

```
Server.on("/page2",Second_page);   //"192.168.2.2/page2"  this is second page location
```

You have three subroutines that handle client requests.

To start the server use this command

```
server . begin ();                  //Start server
```

In main loop we handle client request

```
server . handleClient ();      //Handle client requests
```

This subroutine is called when you enter IP address in web browser and hit enter. This routine sends the test “hello from esp8266” to web browser.

```
void handleRoot () {  
    server . send (200, "text/plain", "hello from esp8266!");  
}
```

Completer Program for Hello from esp8266

```
/*
 * Hello world web server
 * circuits4you.com
 */
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

//SSID and Password of your WiFi router
const char* ssid = "Circuits4you.com";
const char* password = "9422212729";

ESP8266WebServer server (80); //Server on port 80

//=====
// This routine is executed when you open its IP in browser
//=====
void handleRoot () {
  server . send (200, "text/plain", "hello from esp8266!");
}

//=====
//          SETUP
//=====
void setup (void){
  Serial . begin (9600);

  WiFi . begin ( ssid , password ); //Connect to your WiFi router
  Serial . println ("");

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

  //If connection successful show IP address in serial monitor
  Serial . println ("");
  Serial . print ("Connected to ");
  Serial . println ( ssid );
  Serial . print ("IP address: ");
  Serial . println ( WiFi . localIP ()); //IP address assigned to your ESP

  server . on ("/", handleRoot ); //Which routine to handle at root location

  server . begin (); //Start server
  Serial . println ("HTTP server started");
```



```
}  
//=====   
//          LOOP   
//=====   
void loop (void){  
    server . handleClient ();    //Handle client requests  
}
```

Results:

To see the result first get the IP address from serial monitor, Open serial monitor and press reset. It sends IP and shows its connection status, if it is not able to connect it will show “.....” dots in serial monitor. Check you ssid and password.

//SSID and Password of your WiFi router

```
const char* ssid = "your_ssid";
```

```
const char* password = "password";
```

Once connected it will show following



Figure 9.2: Serial monitor showing IP address

Open web browser and enter this IP (192.168.2.2), Make sure that your laptop or phone must be connected to the same network. You can see this web page in all the devices which are connected to the WiFi router, where the ESP8266 is connected.

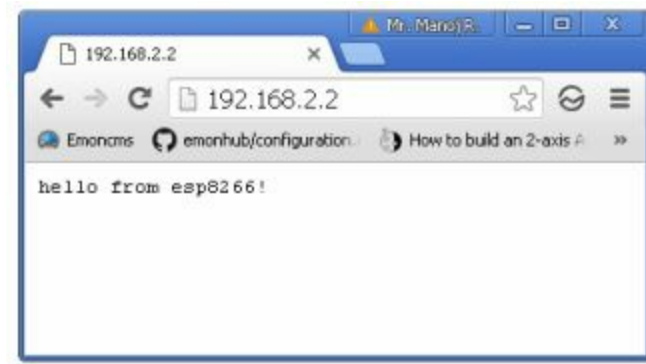


Figure 9.3: Web page from ESP8266

ESP as Access Point

You may find that ESP is also visible as hot spot in above example; you can hide its AP (Access point) using this command at the beginning of setup.

```
WiFi.mode(WIFI_STA); //This line hides the viewing of ESP as wifi network
```

In some application you may find that both AP and connection to WiFi router are useful for making configuration you use ESP AP and for data sending to cloud use WiFi connectivity in that case use this command and both connections. This way you can access ESP web page with two different IP address.

```
WiFi . mode ( WIFI_AP_STA ); //Both hotspot and client are enabled
```

Third way is only access point, default is all AP and STA are enabled, to get only AP use this command.

```
WiFi.mode(WIFI_AP); //Only Access point
```

To start ESP as Access point you have to use this simple command

```
WiFi . softAP ( ssid , password ); //Start HOTspot removing password will disable security
```

Completer Program for Hello from esp8266 as Access Point

```
/*
 * Hello world web server
 * circuits4you.com
 */
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

//SSID and Password to your ESP Access Point
const char* ssid = "ESPWebServer";
const char* password = "12345678";

ESP8266WebServer server (80); //Server on port 80

//=====
// This routine is executed when you open its IP in browser
//=====
void handleRoot () {
  server . send (200, "text/plain", "hello from esp8266!");
}

//=====
//          SETUP
//=====
void setup (void){
  Serial . begin (9600);
  Serial . println ("");
  WiFi . mode ( WIFI_AP );      //Only Access point
  WiFi . softAP ( ssid , password ); //Start HOTspot removing password will disable security

  IPAddress myIP = WiFi . softAPIP (); //Get IP address
  Serial . print ("HotSpt IP:");
  Serial . println ( myIP );

  server . on ( "/", handleRoot ); //Which routine to handle at root location

  server . begin ();              //Start server
  Serial . println ("HTTP server started");
}

//=====
//          LOOP
//=====
void loop (void){
  server . handleClient ();      //Handle client requests
}
```

Results:

To test above code after uploading program, Take your mobile turn on WiFi and in WiFi setting Scan for hot spot you will find “ESPWebServer” hot spot connect to it with password “12345678” as we have given in program. After connecting to ESP hot spot, Open web browser in mobile phone and enter IP 192..168.4.1 you will see “hello from esp8266!” message. IP address can be found in serial monitor.

Step 3: Upload your own HTML code as web page

We have learned how to create web server and its basics, now we want to upload our HTML web page. It's very simple, just replace "hello from esp8266" with HTML code.

```
server . send (200, "text/plain", "hello from esp8266!");
```

First we take webpage code in separate header file name it as "index.h", our web page is now a array of characters stored in variable **MAIN_page**. Do not use comments in this file. It is HTML data as a character array not a program. *Now HTML code is in a header file .h not .html file.*

index.h file

```
const char MAIN_page[] PROGMEM = R"=====(
<HTML>
    <HEAD>
        <TITLE>My first web page</TITLE>
    </HEAD>
    < BODY >
        <CENTER>
            <B>Hello World.... </B>
        </CENTER>
    </ BODY >
</HTML>
)===== ";
```

Now we import this header file in our program using #import "index.h". Make sure that this file must be with our arduino code file .ino

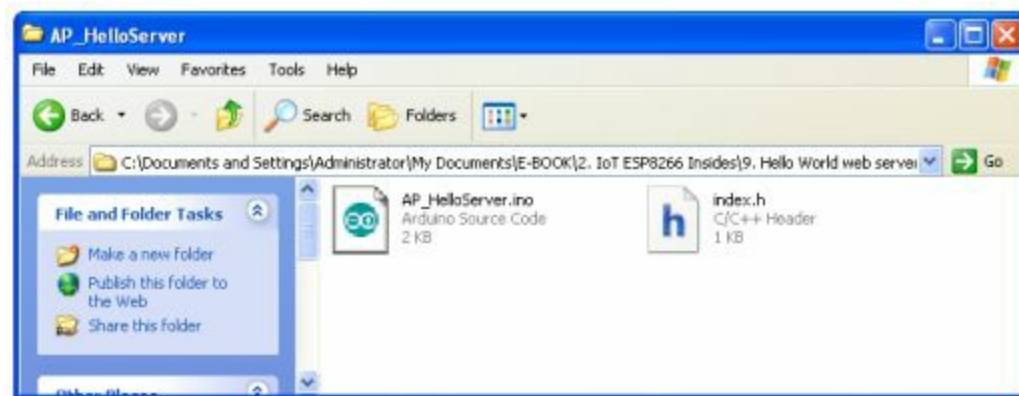


Figure 9.4: index.h file location

The changes in main programs are made in handleRoot subroutine which sends the web page to client, now we are sending html page change text/plain to text/html.

```
void handleRoot () {
    server . send (200, "text/plain", "hello from esp8266!");
}
```

Modified handleRoot subroutine

```
void handleRoot () {
    String s = MAIN_page ;
    server . send (200, "text/html", s);
}
```


Completer Program for HTML page Web Server and esp8266 as AP

```
/*
 * Hello world web server
 * circuits4you.com
 */
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

#include "index.h" //Our HTML webpage contents

//SSID and Password to your ESP Access Point
const char* ssid = "ESPWebServer";
const char* password = "12345678";

ESP8266WebServer server (80); //Server on port 80

//=====
// This routine is executed when you open its IP in browser
//=====
void handleRoot () {
  String s = MAIN_page ; //Read HTML contents
  server . send (200, "text/html", s ); //Send web page
}

//=====
//          SETUP
//=====
void setup (void){
  Serial . begin (9600);
  Serial . println ("" );
  WiFi . mode ( WIFI_AP ); //Only Access point
  WiFi . softAP ( ssid , password ); //Start HOTspot removing password will disable security

  IPAddress myIP = WiFi . softAPIP (); //Get IP address
  Serial . print ("HotSpt IP:");
  Serial . println ( myIP );

  server . on ( "/", handleRoot ); //Which routine to handle at root location

  server . begin (); //Start server
  Serial . println ("HTTP server started");
}

//=====
//          LOOP
//=====
void loop (void){
```

```
server . handleClient ();    //Handle client requests  
}
```

Results:

To test above code after uploading program, Take your mobile turn on WiFi and in WiFi setting Scan for hot spot you will find “ESPWebServer” hot spot connect to it with password “12345678” as we have given in program. After connecting to ESP hot spot, Open web browser in mobile phone and enter IP 192..168.4.1 you will see “**Hello World...**” message. IP address can be found in serial monitor. We kept everything as it is, you can try above code with connecting to your WiFi router also and combination of both.

10. Temperature Monitoring Web Server

10.1 Introduction

In this lesson we learn how to make dynamically changing HTML contents with cool looking Temperature indicator using CSS.

In this lesson we learn few new things commonly used in IoT applications.

Auto refresh webpage to make changes visible such as change in temperature. Content="3" is used to refresh web page every 3 seconds. You can change the timing as per your application needs. Use this command in HTML HEAD tag.

```
<meta http-equiv="refresh" content="3">
```

We have used cool CSS code that improves our web page looks. Go through HTML CSS code.

HTML Web page

Save this code with .HTM extension and try it in web browser also test your modifications, using “Inspect Element” feature of web browser.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="refresh" content="3">
<title>Temperature Monitoring Web Server</title>
<style>
div.card {
  width: 250px;
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
  text-align: center;
  border-radius: 5px;
}

div.header {
  background-color: #4CAF50;
  color: white;
  padding: 10px;
  font-size: 40px;
  border-radius: 5px;
}

div.container {
  padding: 4px;
}
</style>
</head>
<body>

<center><h2>Temperature Monitoring Web Server</h2>

<div class="card">
  <div class="header">
    <h1>33 &deg;C</h1>
  </div>

  <div class="container">
    <h2>Temperature</h2>
  </div>
</div>
</center>
</body>
</html>
```

10.2 Inspect Element Tool

Inspect element tool is very useful to try real time changes in web page colors, spacing, etc.

How to use Inspect Element?

Step 1: Open your web page in Chrome Web browser



Figure 10.1: Inspect Element

Right click on the element that you want to change. For example I want to change green color of temperature indicator. Then click on Inspect, you will see CSS code as shown in figure 10.2.



Figure 10.2: Inspect element window

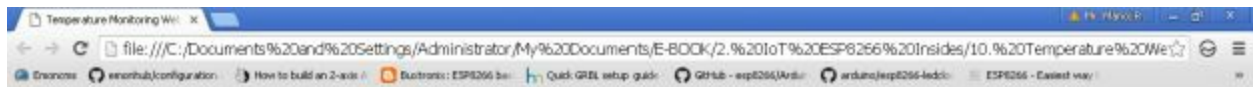
In above figure you can see HTML code on the left and CSS code on the right. Move your mouse on the left HTML code you will see highlighting of the object present in our web page. Once you find correct object which you want to modify click on that line ex. **<div class="header">**.

Step 2: Changing Color of the object

On the right hand side you will see CSS code in that **background-color: #4CAF50**. Click on the green color, It will open color palette and chose any one. You see a real time change in web page refer Figure 10.3. Note down this color code and modify it in your HTML CSS code.

Note: Disable auto refresh by modifying HTML code, It may disturb which using inspect element

```
div.header {  
    background-color: #E03C3C;  
    color: white;  
    padding: 10px;  
    font-size: 40px;  
    border-radius: 5px;  
}
```



Temperature Monitoring Web Server

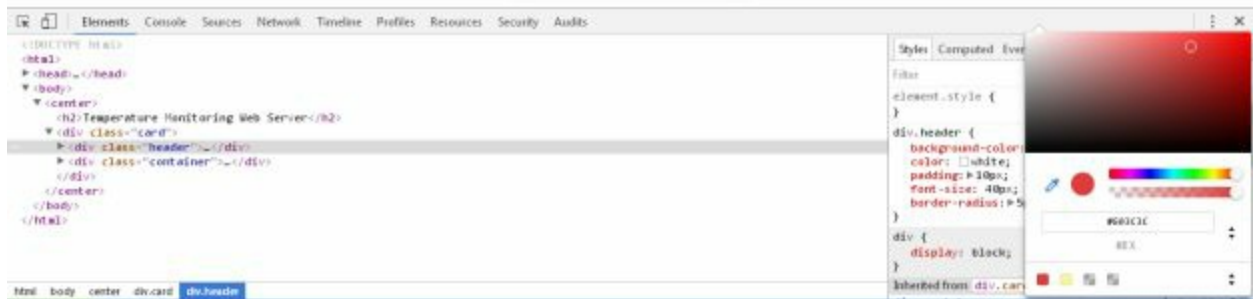


Figure 10.3: Color modification in web page

10.3 ESP8266 Programs

First create header file containing web page data as variable

Index.h

Make changes in HTML code replace temperature value with @@temp@@.

<h1>@@temp@@ °C</h1>

Do not use connects in header file. It is data not code.

```
const char MAIN_page [] PROGMEM = R "=====(
<!DOCTYPE html >
< html >
< head >
< meta http - equiv ="refresh" content ="3">
< title > Temperature Monitoring Web Server </ title >
< style >
div . card {
    width : 250px;
    box - shadow : 0 4px 8px 0 rgba (0, 0, 0, 0.2), 0 6px 20px 0 rgba (0, 0, 0, 0.19);
    text - align : center ;
    border - radius : 5px;
    background - color :# F5F7A0
}

div . header {
    background - color :# E03C3C ;
    color : white ;
    padding : 10px;
    font - size : 40px;
    border - radius : 5px;
}

div . container {
    padding : 4px;
}
</ style >
</ head >
< body >

< center >< h2 > Temperature Monitoring Web Server </ h2 >

< div class="card">
    < div class="header">
        < h1 > @@temp@@ & deg ; C </ h1 >
    </ div >

    < div class="container">
        < h2 > Temperature </ h2 >
    </ div >
</ div >
</ center >
```

```
</ body >  
</ html >
```

```
)=====";
```

ESP8266 program for Temperature Monitoring Web Server

To change temperature reading we have to change the HTML code every time the web browser requests web page. To make this we used String.replace command. It replaces text in HTML i.e. @@temp@@ with actual temperature readings.

```
s . replace ("@@temp@@", String ( Temperature ));
```

1. ESP as Access Point

```
/*
 * Temperature Monitoring web server ESP as Access Point
 * circuits4you.com
 */
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

#include "index.h"

//SSID and Password to your ESP Access Point
const char* ssid = "ESPWebServer";
const char* password = "12345678";

ESP8266WebServer server (80); //Server on port 80

//=====
// This routine is executed when you open its IP in browser 192.168.4.1
//=====
void handleRoot () {
  String s = MAIN_page ;
  int Temperature ;
  Temperature = analogRead ( A0 ); //Read Analog Voltage of ADC Pin
  Temperature = Temperature /10;    //10mV is 1 degree C

  //Convert Temperature int to String then Replace @@temp@@ in HTML with temperaure value
  s . replace ("@@temp@@", String ( Temperature ));
  server . send (200, "text/html", s ); //Send webpage to browser
}

//=====
//          SETUP
//=====
void setup (void){
  Serial . begin (9600);
  Serial . println ("");
  WiFi . mode ( WIFI_AP );    //Only Access point
  WiFi . softAP ( ssid , password ); //Start HOTspot removing password will disable security
```

```
IPAddress myIP = WiFi . softAPIP (); //Get IP address
Serial . print ("HotSpt IP:");
Serial . println ( myIP );

server . on ("/", handleRoot );    //Which routine to handle at root location

server . begin ();                //Start server
Serial . println ("HTTP server started");
}
//=====
//          LOOP
//=====
void loop (void){
  server . handleClient ();        //Handle client requests
}
```


2. Temperature Monitoring Web Server on Your network

Replace SSID and Password as per your network

```
/*
 * Temperature Monitoring Web Server Connects to your Router
 * circuits4you.com
 */
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

#include "index.h" //Web page

//SSID and Password of your WiFi router
const char* ssid = "Your_SSID";
const char* password = "Your_password";

ESP8266WebServer server (80); //Server on port 80

//=====
// This routine is executed when you open its IP in browser
//=====
void handleRoot () {
  String s = MAIN_page ;
  int Temperature ;
  Temperature = analogRead ( A0 ); //Read Analog Voltage of ADC Pin
  Temperature = Temperature /10;    //10mV is 1 degree C

  //Convert Temperature int to String then Replace @@temp@@ in HTML with temperature value
  s . replace ("@@temp@@", String ( Temperature ));
  server . send (200, "text/html", s ); //Send webpage to browser
}

//=====
//          SETUP
//=====
void setup (void){
  Serial . begin (9600);

  WiFi . begin ( ssid , password ); //Connect to your WiFi router
  Serial . println ("");

  // Wait for connection
  while ( WiFi . status () != WL_CONNECTED ) {
    delay (500);
    Serial . print ( ". ");
```

```

}

//If connection successful show IP address in serial monitor
Serial . println ("" );
Serial . print ("Connected to ");
Serial . println ( ssid );
Serial . print ("IP address: ");
Serial . println ( WiFi . localIP ()); //IP address assigned to your ESP

server . on ("/", handleRoot );    //Which routine to handle at root location

server . begin ();                //Start server
Serial . println ("HTTP server started");
}
//=====
//          LOOP
//=====
void loop (void){
  server . handleClient ();        //Handle client requests
}

```

10.4 ESP8266 Hardware connections for Temperature Sensor

We can connect Temperature sensor LM35 directly to the ESP8266, It gives 10mV per degree centigrade.

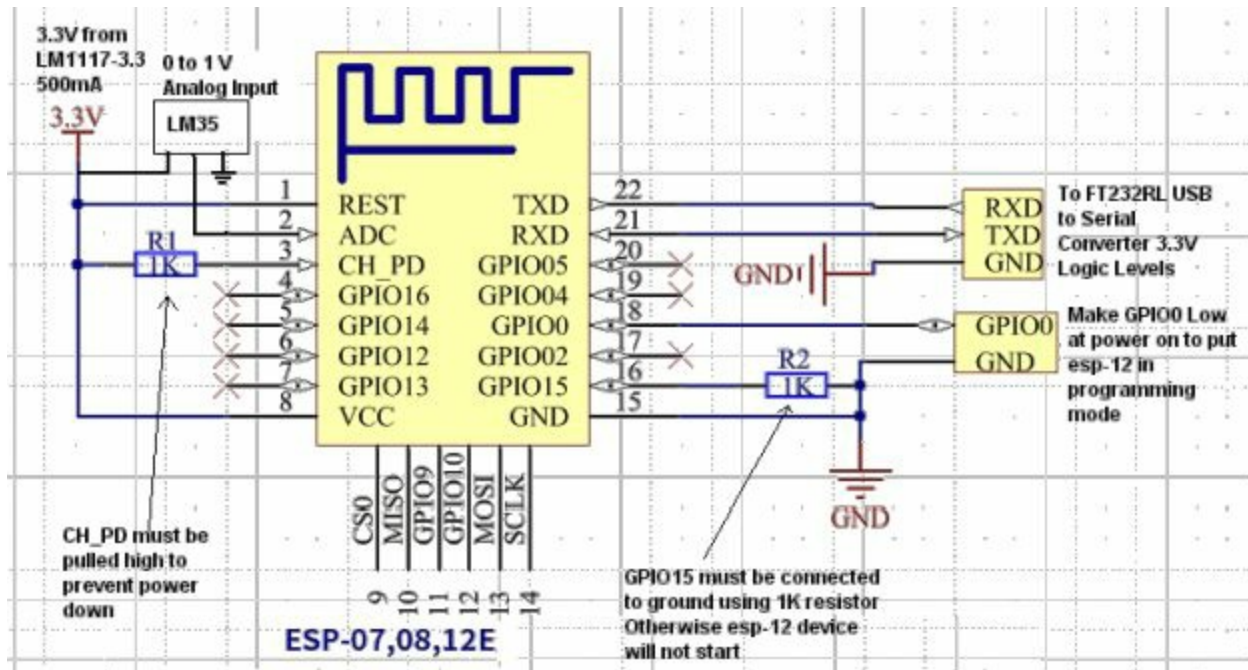


Figure 10.4: Circuit Connections of LM35 with ESP8266

Results

For ESP as access point, connect your laptop or phone to ESP using Wifi and open web browser enter 192.168.4.1 IP address you will see Temperature readings in web page. Similarly you can try second code to connect ESP to your network.



11. LED Control Web Server

11.1 Introduction

In this lesson we will see how to control LEDs, Relays using web on IoT ESP8266. For this lesson we are using HTML Forms and GET method. You can read more on the net we will keep our focus on implementation part of this on the ESP8266 with basic information of HTML GET and Forms.

1.2 Creating Web page with Form, Buttons

First we create a simple webpage with buttons on computer, test it and then use this webpage in ESP8266.

Index.htm

```
<!DOCTYPE html>
<html>
<head>
<title>LED Control Web Server</title>
</head>
<body>

<center><h2>LED Control Web Server</h2>
    <form method="get" action="/form">
        <input type="submit" name="button" value="ON">
        <input type="submit" name="button" value="OFF">
    </form>

    LED Status: @@status@@ </br>
    <a href="circuits4you.com">Visit us www.circuits4you.com</a>
</center>
</body>
</html>
```

The above code generates web page with two buttons ON and OFF. On clicking these buttons it will call “/form” web page with parameters “button=ON” or “button=OFF”. Carefully see this. It is name “button” is variable with value “ON” when you click on button and “OFF” when you click off button. We get this value in our ESP using **server.arg(variable name)**.

```
String t_state = server.arg("button");
```

We will replace @@status@@ with LED present status.



Figure 11.1: Web page output

Let's see what happen when we click on ON and OFF button

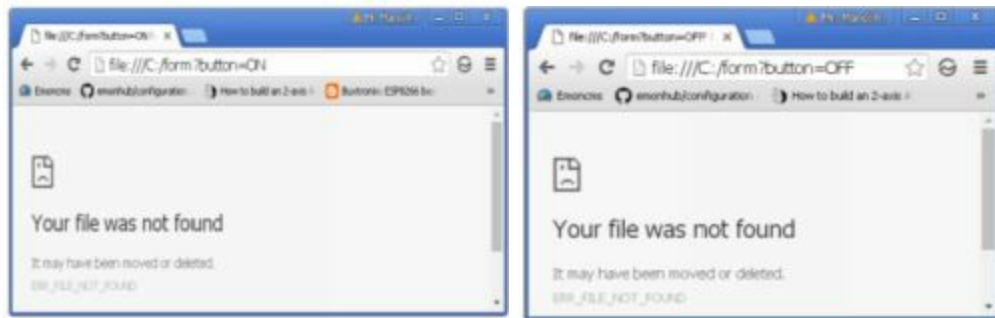


Figure 11.2: ON, OFF button click results

When we click on ON button the link becomes **/form?button=ON**. Web browser uses GET request to send data to server using this additional parameter. We are sending parameters to server i.e. **button=ON** or **button=OFF** we can read it in ESP using **server.arg("button")**. Based on these parameters we can make decision to turn LED on or off.

1.3 ESP8266 Program for LED on off web server

Webpage header file index.h

```
< form method ="get" action ="/form">
```

The form line action is where to take URL when we click on a button it takes you to **/form**

Ex. When we click on button.

192.168.2.4/form?button=ON

Index.h

```
const char MAIN_page [] PROGMEM = R "=====(
<!DOCTYPE html >
< html >
< head >
< title > LED Control Web Server </ title >
</ head >
< body >

< center >< h2 > LED Control Web Server </ h2 >
  < form method ="get" action ="/form">
    < input type ="submit" name ="button" value ="ON">
    < input type ="submit" name ="button" value ="OFF">
  </ form >

  LED Status : @@status@@ </ br >
  < a href ="circuits4you.com"> Visit us www . circuits4you . com </ a >
</ center >
</ body >
</ html >
)=====;
```

ESP Code

In this code you can see we have made three changes 1. Define LED connections and its pinMode. 2. Our server is at two locations root and form **server.on("/form",handleForm);** this handleForm is called when we click on button. There we read arguments of link and change LED status. 3. Web browser redirect using these two commands **server.sendHeader("Location", "/");** and **server.send(302, "text/html", "");** Read HTTP 302. It is redirect command to browser. We want to keep

user on same page.

```
/*
 * LED Control Web Server Connects to your Router
 * circuits4you.com
 */
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

#include "index.h" //Web page
#define LED 2 //Define LED connections, On board blue LED

//SSID and Password of your WiFi router
const char* ssid = "Circuits4you.com";
const char* password = "9422212729";

ESP8266WebServer server (80); //Server on port 80

String LED_state ="OFF";
//=====
// This routine is executed when you open its IP in browser
//=====
void handleRoot () {
    String s = MAIN_page ;

    //Convert Temperature int to String then Replace @@temp@@ in HTML with temperature value
    s . replace ("@@status@@", LED_state );
    server . send (200, "text/html", s ); //Send webpage to browser
}

//=====
// This routine is executed when we press ON/OFF button i.e. form
//=====
void handleForm () {
    String s = server . arg ("button"); //Get button state

    if( s == "ON")
    {
        digitalWrite ( LED , LOW ); //Turn LED on (LED is in reverse connection)
        LED_state ="ON"; //Change status to ON
    }
    if( s == "OFF")
    {
        digitalWrite ( LED , HIGH ); //Turn LED off (LED is in reverse connection)
        LED_state ="OFF"; //Change status to OFF
    }

    server . sendHeader ("Location", "/"); //Send web browser to root location
    server . send (302, "text/html", ""); //Send browser 302 redirect go back to root location
}
```

```

}
//=====
//          SETUP
//=====
void setup (void){
  Serial . begin (9600);

  WiFi . begin ( ssid , password );    //Connect to your WiFi router
  Serial . println ("" );

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

  //If connection successful show IP address in serial monitor
  Serial . println ("" );
  Serial . print ( "Connected to " );
  Serial . println ( ssid );
  Serial . print ( "IP address: " );
  Serial . println ( WiFi . localIP () ); //IP address assigned to your ESP

  //Change Pin mode once we have sent the ESP IP address
  pinMode ( LED , OUTPUT );
  digitalWrite ( LED , HIGH ); //LED Off State

  server . on ( "/" , handleRoot );    //Which routine to handle at root location
  server . on ( "/form" , handleForm ); //These request sent when we click on button

  server . begin ();                  //Start server
  Serial . println ( "HTTP server started" );
}
//=====
//          LOOP
//=====
void loop (void){
  server . handleClient ();           //Handle client requests
}

```

Result

Open web browser and test program by entering IP address that is assigned by your router to esp. Then press buttons and observe on board LED status.



Figure 11.3: ESP webpage in browser

To make it more interesting using CSS button modify your **index.h** file with below code

```
const char MAIN_page [] PROGMEM = R "=====(
<!DOCTYPE html >
< html >
< head >
< title > LED Control Web Server </ title >
< style >
.button {
  background - color : #4CAF50; /* Green */
  border : none ;
  color : white ;
  padding : 16px 32px;
  text - align : center ;
  text - decoration : none ;
  display : inline- block ;
  font - size : 16px;
  margin : 4px 2px;
  - webkit - transition - duration : 0.4s; /* Safari */
  transition - duration : 0.4s;
  cursor : pointer ;
}

.button1 {
  background - color : white ;
  color : black ;
  border : 2px solid #4CAF50;
}

.button1 : hover {
  background - color : #4CAF50;
  color : white ;
}

.button2 {
  background - color : white ;
  color : black ;
  border : 2px solid # f44336 ;
}

.button2 : hover {
  background - color :# f44336 ;
  color : white ;
}
</ style >
```

```
</ head >
< body >

< center >< h2 > LED Control Web Server </ h2 >
  < form method ="get" action ="/form">
    < button class="button button1" name ="button" value ="ON"> ON </ button >
    < button class="button button2" name ="button" value ="OFF"> OFF </ button >
  </ form >

  LED Status : @@status@@ </ br >
  < a href ="circuits4you.com"> Visit us www . circuits4you . com </ a >
</ center >
</ body >
</ html >

)=====";
```


Final results with CSS buttons



Figure 11.4: CSS buttons

12. IoT Home Automation

12.1 Introduction

In this lesson we are combining knowledge of previous lesson to make real application of IoT.

With advancement of Automation technology, life is getting simpler and easier in all aspects. In today's world Automatic systems are being preferred over manual system. Wireless Home Automation system(WHAS) using IoT is a system that uses computers or mobile devices to control basic home functions and features automatically through internet from anywhere around the world, an automated home is sometimes called a smart home. It is meant to save the electric power and human energy. The home automation system differs from other system by allowing the user to operate the system from anywhere around the world through internet connection.

12.2 Components Required

1. ESP-12 WiFi Module
2. LM1117-3.3V
3. Sugar Cube Relay – Qty.4
4. Resistors 10K, 1K,4.7K
5. Capacitor 1000uF, 10uF, 104 (0.1uF)
6. PBT-2 Connectors Qty. 5
7. ULN2003
8. 12V Power Supply

12.3 Circuit Diagram of Home Automation

From the circuit diagram it is very clear that we have used few components, LM1117-3.3V is used for providing power supply to the ESP-12 WiFi Module. ULN2003 provides relay driving.

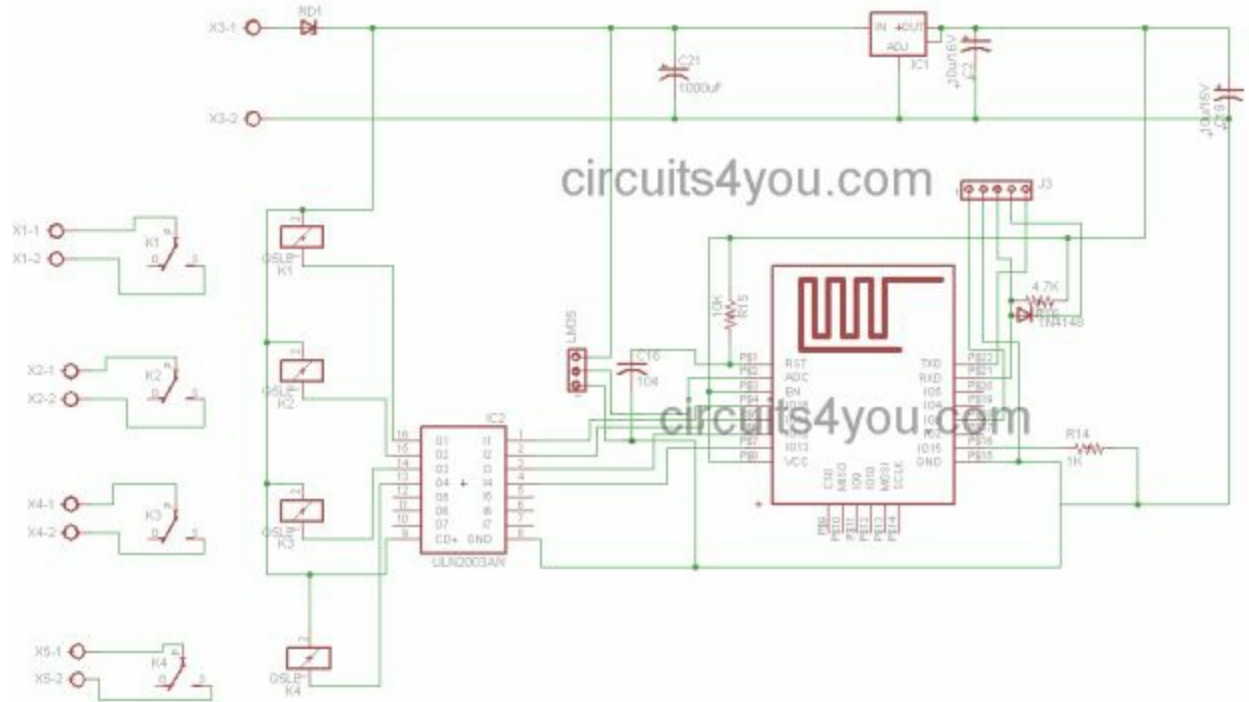


Figure 12.1: Home Automation Circuit

12.4 PCB Layout

[Download Printable PCB layout from here](#)

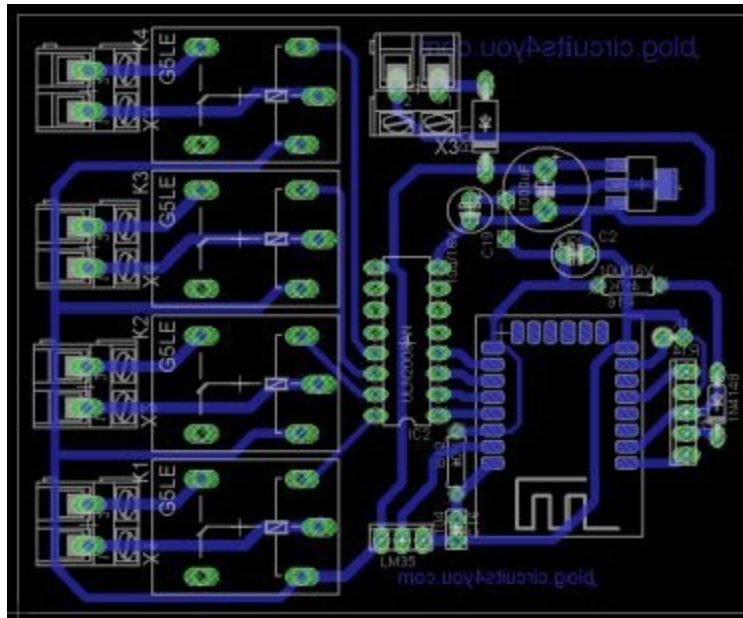


Figure 12.2: PCB Layout

12.5 Program for Home Automation

Program consists of two parts HTML Header and Main Code. Use of TABLE in html makes easy placement of FORM buttons. This program is combination of previous lessons to make real life application.

Index.h file

```
const char MAIN_page[] PROGMEM = R"=====(
<HTML>
<TITLE>
REMOTE LED ON/OFF CONTROL
</TITLE>
<BODY>
<CENTER>
<FORM method="post" action="/form">
<TABLE>
<TR><TD colspan=2><B>IoT Based Home Automation System</B></TD></TR>
<TR><TD>Temp: @@TEMP@@ C</TD><TD>Load Status</TD></TR>

<TR><TD>
<INPUT TYPE=SUBMIT VALUE="ON" name=load1>
<INPUT TYPE=SUBMIT VALUE="OFF" name=load1>
</TD>
<TD>@@L1@@</TD></TR>

<TR><TD>
<INPUT TYPE=SUBMIT VALUE="ON" name=load2>
<INPUT TYPE=SUBMIT VALUE="OFF" name=load2>
</TD>
<TD>@@L2@@</TD></TR>

<TR><TD>
<INPUT TYPE=SUBMIT VALUE="ON" name=load3>
<INPUT TYPE=SUBMIT VALUE="OFF" name=load3>
</TD>
<TD>@@L3@@</TD></TR>

<TR><TD>
<INPUT TYPE=SUBMIT VALUE="ON" name=load4>
<INPUT TYPE=SUBMIT VALUE="OFF" name=load4>
</TD>
<TD>@@L4@@</TD></TR>

<TR><TD colspan=2><B><CENTER><A href =
"http://circuits4you.com">www.circuits4you.com</a></CENTER></B></TD></TR>
</TABLE>
</FORM>
```



```

</CENTER>
</BODY>
</HTML>
)=====";

```

Main Program

```

/*
 * Copyright (c) 2015, circuits4you.com
 * All rights reserved.
 * Create a WiFi access point and provide a web server on it. */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

#include "index.h"

const int Load1 =16;
const int Load2 =14;
const int Load3 =12;
const int Load4 =13;

/* Set these to your desired credentials. */
const char * ssid = "HomeServer";
const char * password = "homeautomation";

ESP8266WebServer server (80);
String L1Status , L2Status , L3Status , L4Status , Temperature ;
//=====
//          handles main page 192.168.4.1
//=====
/* Go to http://192.168.4.1 in a web browser
 * connected to this access point to see it.
 */
void handleRoot () {
    String s = MAIN_page ;
    s . replace ("@@L1@@", L1Status );
    s . replace ("@@L2@@", L2Status );
    s . replace ("@@L3@@", L3Status );
    s . replace ("@@L4@@", L4Status );

    Temperature = String ( analogRead ( A0 )/10);
    s . replace ("@@TEMP@@", Temperature );
    server . send (200, "text/html", s );
}

//=====
//          Handle On off control

```

```
//=====
void handleForm () {
    String load1 = server . arg ("load1");
    String load2 = server . arg ("load2");
    String load3 = server . arg ("load3");
    String load4 = server . arg ("load4");

    Temperature = String ( analogRead ( A0 )/10); //Do calibration here

    //Change Load-1 State as per request
    if( load1 == "ON")
    {
        L1Status = "ON";
        digitalWrite ( Load1 , HIGH );    //Load1 Turned on
    }

    if( load1 == "OFF")
    {
        L1Status = "OFF";
        digitalWrite ( Load1 , LOW );    //Load1 Turned off
    }

    //Change Load-2 State as per request
    if( load2 == "ON")
    {
        L2Status = "ON";
        digitalWrite ( Load2 , HIGH );    //Load1 Turned on
    }

    if( load2 == "OFF")
    {
        L2Status = "OFF";
        digitalWrite ( Load2 , LOW );    //Load1 Turned off
    }

    //Change Load-3 State as per request
    if( load3 == "ON")
    {
        L3Status = "ON";
        digitalWrite ( Load3 , HIGH );    //Load1 Turned on
    }

    if( load3 == "OFF")
    {
        L3Status = "OFF";
        digitalWrite ( Load3 , LOW );    //Load1 Turned off
    }

    //Change Load-4 State as per request
```

```

if( load4 == "ON")
{
  L4Status = "ON";
  digitalWrite ( Load4 , HIGH );    //Load1 Turned on
}

if( load4 == "OFF")
{
  L4Status = "OFF";
  digitalWrite ( Load4 , LOW );    //Load1 Turned off
}

server . sendHeader ("Location", "");
server . send (302, "text/plain", "Updated-- Press Back Button"); //This Line Keeps It on Same
Page

delay (500);
}
//=====
//          Power on setup
//=====

void setup () {
  delay (1000);
  /* You can remove the password parameter if you want the AP to be open. */
  WiFi . softAP ( ssid , password );

  IPAddress myIP = WiFi . softAPIP ();
  server . on ( "/", handleRoot );
  server . on ( "/form", handleForm );

  server . begin ();
  pinMode ( Load1 , OUTPUT );
  pinMode ( Load2 , OUTPUT );
  pinMode ( Load3 , OUTPUT );
  pinMode ( Load4 , OUTPUT );
}

//=====
//          Main Program Loop
//=====

void loop () {
  server . handleClient ();
}
//=====

```

Results

Connect to WiFi network of ESP8266 named as “HomeServer” with password “homeautomation” as we have given in code. Open web browser and enter IP address 192.168.4.1. It will show web page as shown in below figure.

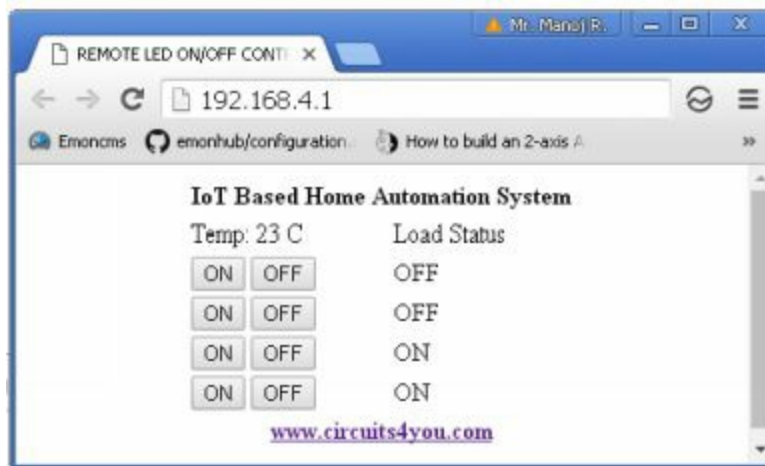


Figure 12.3: Output web page

Press buttons and observe relay operations. You can connect AC loads to relay. You can modify HTML web page using CSS knowledge to make it more attractive.

13. Color Control Web Server

13.1 Introduction

For this lesson we learn how to control RGB LED Color using web. For this demo we are using ESPWitty hardware. It comes with RGB LED and LDR. We can make any colors using combination of Red, Green and Blue color light. Depending on the value we receive from the web page we set the PWM/ analog value for each LED.

ESPWitty Connection Diagram

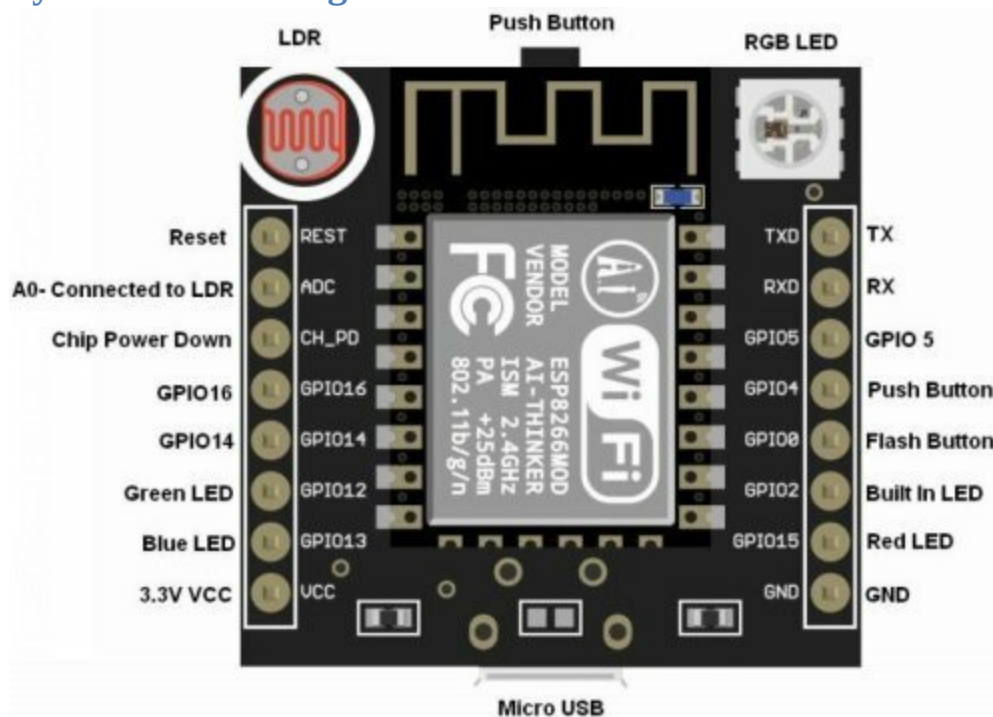


Figure 13.1: ESP8266 ESPWitty Board Connection Diagram

13.2 Programming

First we make webpage and test it in web browser. Then use that HTML code in our arduino IDE as header file.

HTML uses input type color for taking color as user input example.

```
<input type="color" name="color" value="@@color@@">
```

In this the default color value is set by value="@@color@@". The word @@color@@ is replaced by our program to appropriate color, and it is updated when user changes the set color.

As we have learned in previous chapters using GET method of HTML Forms we can get the input from users. The color type input return value in 24-bit hex format **#RRGGBB** for example "#FF0000". We need to separate it and convert it into long data type from string.

It can be done by following steps first remove the # and convert string to long integer using **strtoul(const char *str, char **endptr, int base);**. # is removed due to starting from color[1], then end pointer is null and we want base of HEX i.e. 16.

```
long number = (int) strtoul ( & color [1], NULL , 16); //Remove #  
  
// Split them up into r, g, b values  
long r = number >> 16;  
long g = ( number >> 8) & 0xFF;  
long b = number & 0xFF;
```

Then split them into red, green and blue colors. Then we apply it to RGB LED using **analogWrite** command.

```
//ESP supports analogWrite All IOs are PWM  
analogWrite ( RedLED , r );  
analogWrite ( GreenLED , g );  
analogWrite ( BlueLED , b );
```

Complete Program for RGB LED Color Control

Assuming that the testing of HTML is already finished. We move towards ESP programming.

index.h

```
const char MAIN_page [] PROGMEM = R "=====(
<!DOCTYPE html >
< html >
< head >
  < title > Color Picker </ title >
</ head >
< body >

< p >
Depending on browser support :< br >
A color picker can pop - up when you enter the input field .
</ p >

< form action ="\form" method ="GET">
  < B > Select your favorite color :</ B >
  < input type ="color" name ="color" value ="@@color@">
  < input type ="submit">
</ form >

< p >< b > Note :</ b > type ="color" is not supported in Internet Explorer 11 and earlier
versions or Safari 9.1 and earlier versions .</ p >

</ body >
</ html >
)=====";
```

Color.ino

```
/*
 * RGB LED Color control HTTP
 * Copyright (c) 2015, circuits4you.com
 * All rights reserved.
 * Connects to WiFi HotSpot. */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

#include "index.h"

const int RedLED =15;
const int GreenLED =12;
const int BlueLED =13;
```



```

/* Set these to your desired credentials. */
const char * ssid = "your ssid";
const char * password = "your password";

ESP8266WebServer server (80);
String setcolor = "#ff00ff"; //Set color for HTML
//=====
//          handles main page
//=====
void handleRoot () {
  String s = MAIN_page ;
  s . replace ("@@color@", setcolor );
  server . send (200, "text/html", s );
}

//=====
//          Handle Set Color
//=====
void handleForm () {
  String color = server . arg ("color"); //form?color=%23ff0000
  setcolor = color ; //Store actual color set for updating in HTML

  Serial . println ( color );          //See what we have recived
  //We get #RRGGBB in hex string
  // Get rid of '#' and convert it to integer, Long as we have three 8-bit i.e. 24-bit values
  long number = (int) strtol ( & color [1], NULL , 16);

  // Split them up into r, g, b values
  long r = number >> 16;
  long g = ( number >> 8) & 0xFF;
  long b = number & 0xFF;

  //ESP supports analogWrite All IOs are PWM
  analogWrite ( RedLED , r );
  analogWrite ( GreenLED , g );
  analogWrite ( BlueLED , b );

  server . sendHeader ("Location", "/");
  server . send (302, "text/plain", "Updated-- Press Back Button"); //302-Redirect This Line Keeps It
on Same Page

  delay (500);
}
//=====
//          Power on setup
//=====

void setup () {
  delay (1000);

```

```

Serial . begin (9600);

WiFi . begin ( ssid , password );    //Connect to your WiFi router
Serial . println ("" );

Serial . print ("Connecting");
// Wait for connection
while ( WiFi . status () != WL_CONNECTED ) {
    delay (500);
    Serial . print (".");
}

//If connection successful show IP address in serial monitor
Serial . println ("" );
Serial . print ("Connected to ");
Serial . println ( ssid );
Serial . print ("IP address: ");
Serial . println ( WiFi . localIP ()); //IP address assigned to your ESP

//Server
server . on ( "/", handleRoot );
server . on ( "/form", handleForm );

server . begin ();

//LED IO Directions
pinMode ( RedLED , OUTPUT );
pinMode ( GreenLED , OUTPUT );
pinMode ( BlueLED , OUTPUT );
}

//=====
//          Main Program Loop
//=====
void loop () {
    server . handleClient ();
}
//=====

```

Results

Open serial terminal and get the IP of your ESP8266, then open web browser and enter it. Try changing color, press submit button and observe RGB LED. It is better to use some white paper on LED to make proper color visibility.

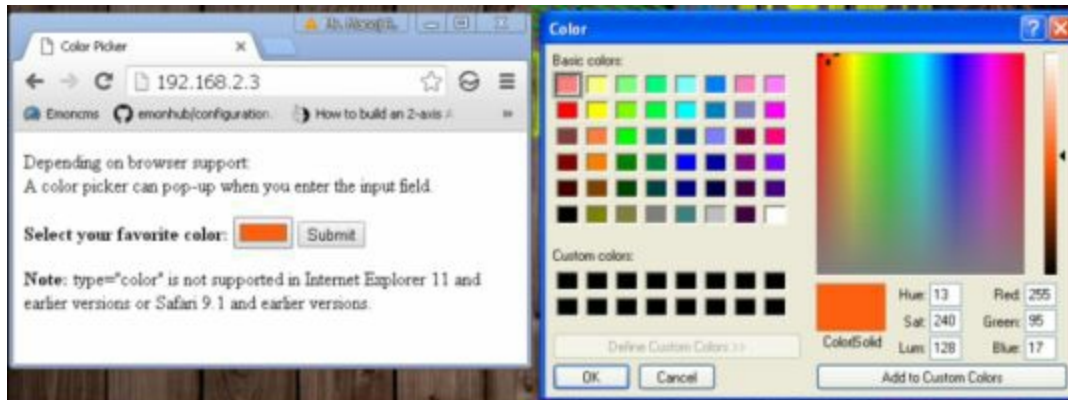


Figure 13.2: RGB Color Setting Web Page

14. HTTP Client

14.1 Introduction

We learn all about creating web server now we make ESP as web client to read web page contents. In most cases we don't need to read a web page as ESP can't show a web page anywhere. It is useful when we want to upload any data to a cloud server and read its response. You get all HTML code when you request a webpage. It is difficult to host complete webpage in RAM it may end in stack overflow error and get reset. As I have discussed its own OS monitors our programs. Let's see reading circuits4you.com

14.2 Programming

Steps for connecting to WiFi router are same. Make sure your WiFi have working internet access. HTTP client request goes in four steps.

1. Connect to web server with its host name (circuits4you.com).
2. Send GET request to web server to get web page.
3. Wait for data to arrive from web server.
4. Read and display web page HTML contents in serial terminal.

ESP code for reading a Web page

```
/*
 * HTTP Client
 * Copyright (c) 2015, circuits4you.com
 * All rights reserved.
 * Connects to WiFi HotSpot. */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

/* Set these to your desired credentials. */
const char * ssid = "Circuits4you.com";
const char * password = "9422212729";

//Web address to read from
const char * host = "circuits4you.com";
//=====
//          Power on setup
//=====

void setup () {
  delay (1000);
  Serial . begin (9600);

  WiFi . mode ( WIFI_STA );    //This line hides the viewing of ESP as wifi hotspot
  //WiFi.mode(WIFI_AP_STA); //Both hotspot and client are enabled
  //WiFi.mode(WIFI_AP);    //Only Access point

  WiFi . begin ( ssid , password ); //Connect to your WiFi router
  Serial . println ("" );

  Serial . print ("Connecting");
  // Wait for connection
  while ( WiFi . status () != WL_CONNECTED ) {
    delay (500);
    Serial . print (".");
  }

  //If connection successful show IP address in serial monitor
  Serial . println ("" );
  Serial . print ("Connected to ");
  Serial . println ( ssid );
  Serial . print ("IP address: ");
  Serial . println ( WiFi . localIP ()); //IP address assigned to your ESP
}

//=====
//          Main Program Loop
//=====
```



```

//=====
void loop () {
  WiFiClient client ;
  const int httpPort = 80; //Port 80 is commonly used for www
//-----
//Connect to host, host(web site) is define at top
if(! client . connect ( host , httpPort )){
  Serial . println ("Connection Failed");
  delay (300);
  return; //Keep retrying until we get connected
}

//-----
//Make GET request as per HTTP GET Protocol format
String Link ="GET /"; //Requeste webpage
Link = Link + " HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: close\r\n\r\n";
client . print ( Link );
delay (100);

//-----
//Wait for server to respond with timeout of 5 Seconds
int timeout =0;
while(! client . available () && ( timeout < 1000)) //Wait 5 seconds for data
{
  delay (10); //Use this with time out
  timeout ++;
}

//-----

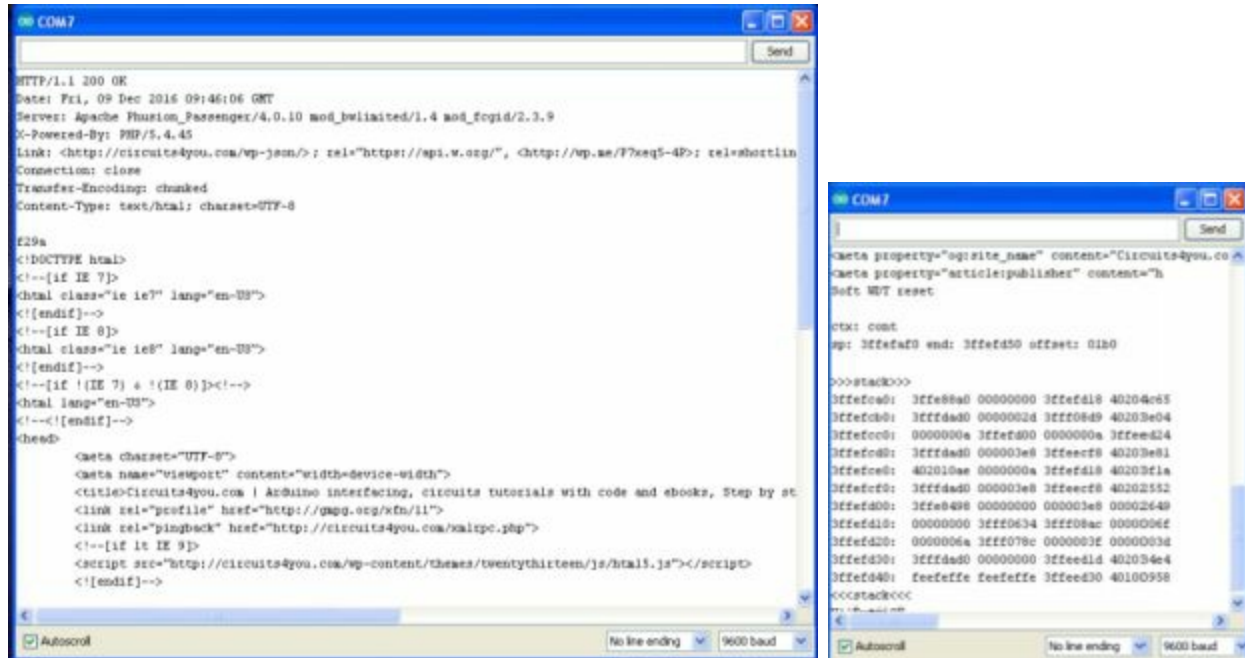
//If data is available before time out read it.
if( timeout < 500)
{
  while( client . available () ){
    Serial . println ( client . readStringUntil ('\n'));
  }
}
else
{
  Serial . println ("Request timeout..");
}

delay (5000); //Read Web Page every 5 seconds
}
//=====

```

Results

At the end of HTML code in serial terminal you will find some stack error with random hex and your ESP get WDT reset as it can't handle complete web page with little RAM.



The image shows two screenshots of a serial terminal window titled 'COM7'. The left screenshot displays an HTTP 200 OK response from a server. The right screenshot shows the end of the HTML document, followed by a 'Soft WDT reset' message and a memory stack dump.

```
COM7
Send

HTTP/1.1 200 OK
Date: Fri, 09 Dec 2016 09:46:06 GMT
Server: Apache/2.4.18 (Ubuntu)
X-Powered-By: PHP/5.4.45
Link: <http://circuits4you.com/wp-json/>; rel="http://api.w.org/", <http://wp.me/F7xq5-4P>; rel=shortlink
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

f29a
<!DOCTYPE html>
<!--[if IE 7]>
<html class="ie ie7" lang="en-US">
<![endif]-->
<!--[if IE 8]>
<html class="ie ie8" lang="en-US">
<![endif]-->
<!--[if !(IE 7) & !(IE 8)]><!-->
<html lang="en-US">
<!--<![endif]-->
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width">
<title>Circuits4you.com | Arduino interfacing, circuits tutorials with code and ebooks, Step by st
<link rel="profile" href="http://gmpg.org/xfn/11">
<link rel="pingback" href="http://circuits4you.com/xmlrpc.php">
<!--[if lt IE 9]>
<script src="http://circuits4you.com/wp-content/themes/twentythirteen/js/html5.js"></script>
<![endif]-->

<meta property="og:site_name" content="Circuits4you.co
<meta property="article:publisher" content="h
Soft WDT reset

ctx: cont
sp: 3ffefaf0 end: 3ffefd50 offset: 0180

>>>stack>>>
3ffefce0: 3ffe88a8 00000000 3ffefd18 40204e65
3ffefcb0: 3fffdad0 0000002d 3ffef849 40203e04
3ffefcc0: 0000000a 3ffef800 0000000a 3ffefed2a
3ffefcd0: 3fffdad0 000003e8 3ffefc78 40203e81
3ffefce0: 402010ae 0000000a 3ffefd18 40203faa
3ffefcf0: 3fffdad0 000003e8 3ffefc78 40202552
3ffefd00: 3ffe8498 00000000 000003e8 00002640
3ffefd10: 00000000 3ffef634 3ffef8ac 0000006f
3ffefd20: 0000006a 3ffef79c 0000003f 0000003d
3ffefd30: 3fffdad0 00000000 3ffefd1d 402034e4
3ffefd40: fefefefe fefefefe 3ffefd30 40100958
<<<stack<<<
```

Figure 14.1: HTML Contents read from a website, Right side Stack Error

It is possible to avoid this error by reading only specific line we need. Below code shows how to read a specific line. It is very useful you can use this to read a server date, time, OK, DONE, not ok like responses. When you upload a data to cloud server it response with some string you can use this method to do that.

ESP code for reading a Specific Line from Web page

```
/*
 * HTTP Client
 * Copyright (c) 2015, circuits4you.com
 * All rights reserved.
 * Connects to WiFi HotSpot. */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

/* Set these to your desired credentials. */
const char * ssid = "Circuits4you.com";
const char * password = "9422212729";

//Web address to read from
const char * host = "circuits4you.com";
//=====
//          Power on setup
//=====

void setup () {
  delay (1000);
  Serial . begin (9600);

  WiFi . mode ( WIFI_STA );    //This line hides the viewing of ESP as wifi hotspot
  //WiFi.mode(WIFI_AP_STA); //Both hotspot and client are enabled
  //WiFi.mode(WIFI_AP);    //Only Access point

  WiFi . begin ( ssid , password ); //Connect to your WiFi router
  Serial . println ("" );

  Serial . print ("Connecting");
  // Wait for connection
  while ( WiFi . status () != WL_CONNECTED ) {
    delay (500);
    Serial . print (".");
  }

  //If connection successful show IP address in serial monitor
  Serial . println ("" );
  Serial . print ("Connected to ");
  Serial . println ( ssid );
  Serial . print ("IP address: ");
  Serial . println ( WiFi . localIP ()); //IP address assigned to your ESP
}
```

```

//=====
//          Main Program Loop
//=====
void loop () {
  WiFiClient client ;
  const int httpPort = 80; //Port 80 is commonly used for www
  //-----
  //Connect to host, host(web site) is define at top
  if(! client . connect ( host , httpPort )){
    Serial . println ("Connection Failed");
    delay (300);
    return; //Keep retrying until we get connected
  }

  //-----
  //Make GET request as per HTTP GET Protocol format
  //String Link="GET /index.php"; //Requeste webpage
  String Link ="GET /"; //Requeste webpage
  Link = Link + " HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: close\r\n\r\n";
  client . print ( Link );
  delay (100);

  //-----
  //Wait for server to respond with timeout of 5 Seconds
  int timeout =0;
  while((! client . available () && ( timeout < 1000)) //Wait 5 seconds for data
  {
    delay (10); //Use this with time out
    timeout ++;
  }

  //-----
  String line [30]; //Line by line reading not having more than 10 lines

  //If data is available before time out read it.
  if( timeout < 500)
  {
    int lineno =0;
    while( client . available () && ( lineno <25)){
      //Serial.println(client.readStringUntil("\n"));
      line [ lineno ] = client . readStringUntil ("\n");
      lineno ++;
    }
    Serial . println ( line [1]); //Print a specific line Date time
    Serial . println ( line [23]); //Print a specific line Title
  }
  else
  {
    Serial . println ("Request timeout..");
  }
}

```

```
}
```

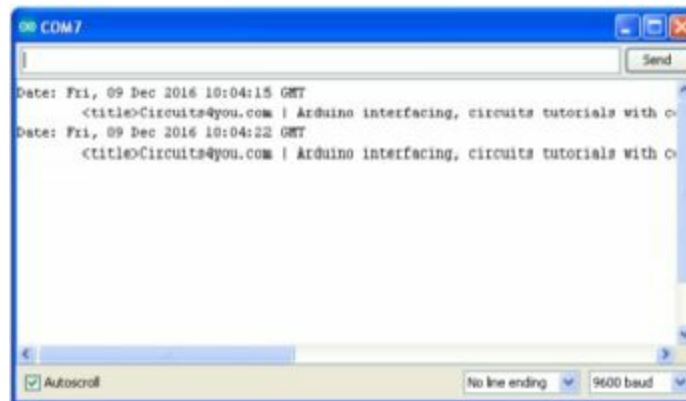
```
delay (5000); //Read Web Page every 5 seconds
```

```
}
```

```
//=====
```

Results

After modifying code to read a specific line, Stack error is gone and we are able to read server date time and web page title tag.



```
COM7
Date: Fri, 09 Dec 2016 10:04:15 GMT
<title>Circuits@you.com | Arduino interfacing, circuits tutorials with c
Date: Fri, 09 Dec 2016 10:04:22 GMT
<title>Circuits@you.com | Arduino interfacing, circuits tutorials with c
```

Autoscroll No line ending 9600 baud

Figure 14.2: Reading a Specific line from HTML page

15. Data logging to cloud server

15.1 Introduction

Based on previous lesson we can make GET request with additional data to upload data to cloud web server. You can read my next upcoming e-book on cloud server design to know more about creating your own cloud server for ESP8266.

15.2 Prepare Cloud Server

In this example we upload LDR analog read value of ESPWitty board on https://thingspeak.com/users/sign_up cloud server. First we sign up on thingspeak.com

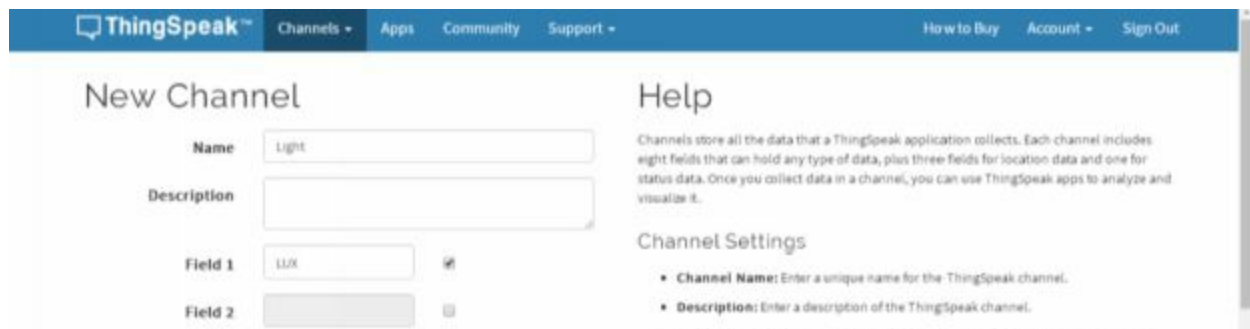
Step 1

Click on New Channel



Step 2

Name it light and Field LUX, keep all other fields blank and click Save Channel.



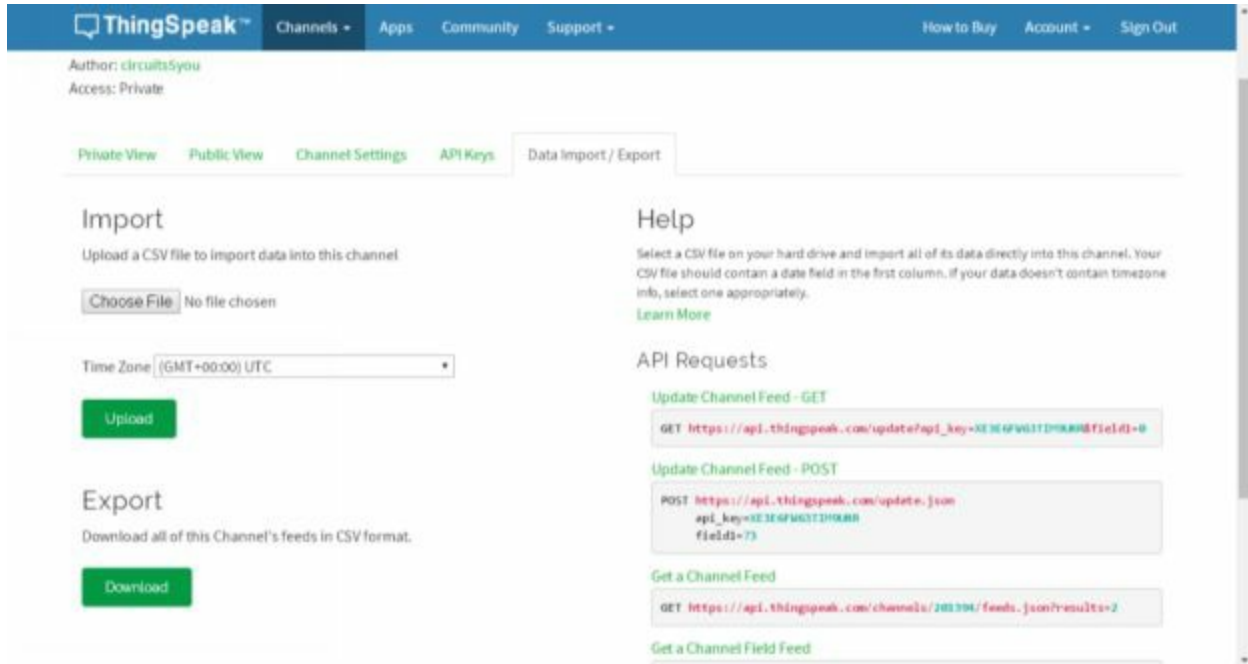
Step 3

Open Data Import export tab as shown in figure. See API Requests formats. Use Update channel feed using GET. **https://api.thingspeak.com/update?api_key=XE3E6FW63TIM9UHR&field1=0**

Do not share API key it makes decision of which user it is. It used by my account. In most cases I create own cloud server as per requirements. Second

parameter is **field1=0** here you can pass the ADC value ex. field1=1234.

It is possible to send the data to cloud server by simple entering Update channel feed - GET link **https://api.thingspeak.com/update?api_key=XE3E6FW63TIM9UHR&field1=0** in to web browser. It gives replay with the number of data sent count.



Step 4

Verify you got the data on your cloud dashboard. By entering different field1 values.

https://api.thingspeak.com/update?api_key=XE3E6FW63TIM9UHR&field1=1234

Open channel status tab and refresh it, it shows the recently sent data. Calling this above link from your ESP Web Client makes data upload possible to the cloud server, this is simplest method.



15.3 Cloud data uploading program

Change ssid and password as per your wifi network settings. Change the host to

```
const char * host = "api.thingspeak.com";
```

Change GET link as per your API key link given in thingspeak

```
String Link = "GET /update?api_key=XE3E6FW63TIM9UHR&field1=";
```

API Requests

Update Channel Feed - GET

```
GET https://api.thingspeak.com/update?api_key=XE3E6FW63TIM9UHR&field1=0
```

ESP8266 Program

```
/*
 * Cloud Server Data Upload
 * Copyright (c) 2015, circuits4you.com
 * All rights reserved.
 * Connects to WiFi HotSpot. */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

/* Set these to your desired credentials. */
const char * ssid = "your ssid";
const char * password = "your password";

//Cloud Web address to write data
const char * host = "api.thingspeak.com";
//=====
//          Power on setup
//=====

void setup () {
  delay (1000);
  Serial . begin (9600);

  WiFi . mode ( WIFI_STA ); //This line hides the viewing of ESP as wifi hotspot
  //WiFi.mode(WIFI_AP_STA); //Both hotspot and client are enabled
  //WiFi.mode(WIFI_AP);    //Only Access point

  WiFi . begin ( ssid , password ); //Connect to your WiFi router
  Serial . println ("" );

  Serial . print ("Connecting");
  // Wait for connection
  while ( WiFi . status () != WL_CONNECTED ) {
    delay (500);
    Serial . print (".");
  }

  //If connection successful show IP address in serial monitor
  Serial . println ("" );
  Serial . print ("Connected to ");
  Serial . println ( ssid );
  Serial . print ("IP address: ");
  Serial . println ( WiFi . localIP ()); //IP address assigned to your ESP
}

//=====
//          Main Program Loop
//=====
```

```

//=====
void loop () {
  WiFiClient client ;
  const int httpPort = 80; //Port 80 is commonly used for www
//-----
//Connect to host, host(web site) is define at top
if(! client . connect ( host , httpPort )){
  Serial . println ("Connection Failed");
  delay (300);
  return; //Keep retrying until we get connected
}

//-----
//Make GET request as pet HTTP GET Protocol format
String ADCData ;
int adcvalue = analogRead ( A0 ); //Read Analog value of LDR
ADCData = String ( adcvalue ); //String to interger conversion
String Link ="GET /update?api_key=XE3E6FW63TIM9UHR&field1="; //Requeste webpage
Link = Link + ADCData ;
Link = Link + " HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: close\r\n\r\n";
client . print ( Link );
delay (100);

//-----
//Wait for server to respond with timeout of 5 Seconds
int timeout =0;
while((! client . available () ) && ( timeout < 1000)) //Wait 5 seconds for data
{
  delay (10); //Use this with time out
  timeout ++;
}

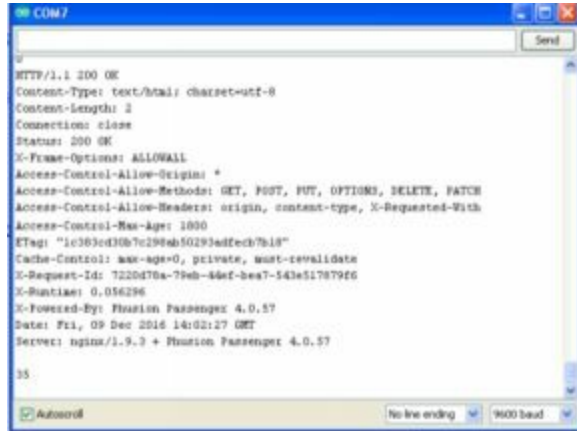
//-----
//If data is available before time out read it.
if( timeout < 500)
{
  while( client . available () ){
    Serial . println ( client . readString ()); //Response from ThingSpeak
  }
}
else
{
  Serial . println ("Request timeout..");
}

delay (5000); //Read Web Page every 5 seconds
}
//=====

```


Results

You will observe in serial terminal response from the thingspeak and Graph change on thingspeak.com. Try modifying program to send multiple values.



```
COM7
Send
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 2
Connection: close
Status: 200 OK
X-Frame-Options: ALLOWALL
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS, DELETE, PATCH
Access-Control-Allow-Headers: origin, content-type, X-Requested-With
Access-Control-Max-Age: 1000
ETag: "1c3b3ed30b7c290eb50293adfecb7b18"
Cache-Control: max-age=0, private, must-revalidate
X-Request-Id: 7220d70a-79eb-44ef-bea7-543e517879f6
X-RunTime: 0.056296
X-Powered-By: Phusion Passenger 4.0.57
Date: Fri, 09 Dec 2016 14:02:27 GMT
Server: nginx/1.9.3 + Phusion Passenger 4.0.57

35
Autoscroll No line ending 9600 baud
```



Figure 15.1: Serial monitor and Data upload to cloud server

16. UDP ESP to ESP Communication

16.1 Introduction

UDP (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP) used primarily for establishing low-latency and loss tolerating connections between applications on the Internet. Both UDP and TCP run on top of the Internet Protocol (IP) and are sometimes referred to as UDP/IP or TCP/IP. Both protocols send short packets of data, called datagrams. In some cases it may be required to communicate between two IoT devices. That can be done using UDP protocol.

There are two ways to make this possible

1. One ESP as access point and other connects to access point. We need only two ESP modules.
2. Both ESP connects to another access point and communicates through it. In this case we need WiFi router.

It is possible to have many ESP devices that can communicate to each other or all together using broadcast IP. In UDP it is possible to send single packet to all IoT devices or a group of devices having same port number. UDP is connectionless protocol so it is possible that we may have data loss.

In this lesson we are making wireless serial communication using two ESP boards.

16.2 TCP vs UDP

	TCP	UDP
Acronym for	Transmission Control Protocol	User Datagram Protocol or Universal Datagram Protocol
Connection	TCP is a connection-oriented protocol.	UDP is a connectionless protocol.
Function	As a message makes its way across the internet from one computer to another. This is connection based.	UDP is also a protocol used in message transport or transfer. This is not connection based which means that one program can send a load of packets to another and that would be the end of the relationship.
Usage	TCP is suited for applications that require high reliability, and transmission time is relatively less critical.	UDP is suitable for applications that need fast, efficient transmission, such as games. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients.
Use by other protocols	HTTP, HTTPs, FTP, SMTP, Telnet	DNS, DHCP, TFTP, SNMP, RIP, VOIP.
Ordering of data packets	TCP rearranges data packets in the order specified.	UDP has no inherent order as all packets are independent of each other. If ordering is required, it has to be managed by the application layer.
Speed of transfer	The speed for TCP is slower than UDP.	UDP is faster because error recovery is not

		attempted. It is a "best effort" protocol.
Reliability	There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent.	There is no guarantee that the messages or packets sent would reach at all.
Header Size	TCP header size is 20 bytes	UDP Header size is 8 bytes.
Common Header Fields	Source port, Destination port, Check Sum	Source port, Destination port, Check Sum
Streaming of data	Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.	Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent.
Weight	TCP is heavy-weight. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	UDP is lightweight. There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.
Data Flow Control	TCP does Flow Control. TCP requires three packets to set up a socket connection,	UDP does not have an option for flow control

	before any user data can be sent. TCP handles reliability and congestion control.	
Error Checking	TCP does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.	UDP does error checking but simply discards erroneous packets. Error recovery is not attempted.
Fields	1. Sequence Number, 2. AcK number, 3. Data offset, 4. Reserved, 5. Control bit, 6. Window, 7. Urgent Pointer 8. Options, 9. Padding, 10. Check Sum, 11. Source port, 12. Destination port	1. Length, 2. Source port, 3. Destination port, 4. Check Sum
Acknowledgement	Acknowledgement segments	No Acknowledgment
Handshake	SYN, SYN-ACK, ACK	No handshake (connectionless protocol)

16.3 Programming

Program is done in two parts one device acts as Access point and another connects to it and the data received from serial is sent to UDP and vice versa. It becomes wireless serial link.

Device one with AP. Access point IP is always 192.168.4.1.

```
/*
  Wireless Serial using UDP ESP8266
  Hardware: ESPWitty
  Circuits4you.com
  2016
  Master Board creates Access Point
*/

#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

const char * ssid = "circuits4you"; // your network SSID (name)
const char * pass = "password";     // your network password

unsigned int localPort = 2000;      // local port to listen for UDP packets
unsigned long sendPacket ( IPAddress & address );

IPAddress ServerIP (192,168,4,1);   //Enter Your PC IP address where VB.NET Application is
running
IPAddress ClientIP (192,168,4,2);   //Enter Your PC IP address where VB.NET Application is
running

// A UDP instance to let us send and receive packets over UDP
WiFiUDP udp ;

char packetBuffer [9]; //Where we get the UDP data
//=====
//                               Setup
//=====
void setup ()
{
  Serial . begin (9600);
  Serial . println ();

  WiFi . softAP ( ssid , pass ); //Create Access point

  Serial . println ("Starting UDP");
  udp . begin ( localPort );
  Serial . print ("Local port: ");
  Serial . println ( udp . localPort ());
}
```



```

//=====
//                                MAIN LOOP
//=====
void loop ()
{
  int cb = udp . parsePacket ();

  if (! cb ) {
    //If serial data is recived send it to UDP
    if( Serial . available ()>0)
    {
      udp . beginPacket ( ClientIP , 2000);    //Send UDP requests are to port 2000
      char a [1];
      a [0]=char( Serial . read ());
      udp . write ( a ,1);    //Serial Byte Read
      udp . endPacket ();
    }
  }
  else {
    // We've received a UDP packet, send it to serial
    udp . read ( packetBuffer , 1); // read the packet into the buffer, we are reading only one byte
    Serial . print ( packetBuffer );
    delay (20);
  }
}
//=====

```

Second Device Program

```

/*
  Wireless Serial using UDP ESP8266
  Hardware: ESPWitty
  Circuits4you.com
  2016
  Slave Board connects to another ESP access point
*/

#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

const char * ssid = "circuits4you"; // your network SSID (name)
const char * pass = "password";    // your network password

unsigned int localPort = 2000;    // local port to listen for UDP packets
unsigned long sendPacket ( IPAddress & address );

```

```
IPAddress ServerIP (192,168,4,1); //Enter Your PC IP address where VB.NET Application is
running
IPAddress ClientIP (192,168,4,2); //Enter Your PC IP address where VB.NET Application is
running
```

```
// A UDP instance to let us send and receive packets over UDP
WiFiUDP udp ;
```

```
char packetBuffer [9]; //Where we get the UDP data
```

```
//=====
```

```
//                      Setup
```

```
//=====
```

```
void setup ()
```

```
{
```

```
  Serial . begin (9600);
```

```
  Serial . println ();
```

```
  // We start by connecting to a WiFi network
```

```
  Serial . print ("Connecting to ");
```

```
  Serial . println ( ssid );
```

```
  WiFi . begin ( ssid , pass );
```

```
  while ( WiFi . status () != WL_CONNECTED ) {
```

```
    delay (500);
```

```
    Serial . print (".");
```

```
  }
```

```
  Serial . println ("");
```

```
  Serial . println ("WiFi connected");
```

```
  Serial . println ("IP address: ");
```

```
  Serial . println ( WiFi . localIP ());
```

```
  Serial . println ("Starting UDP");
```

```
  udp . begin ( localPort );
```

```
  Serial . print ("Local port: ");
```

```
  Serial . println ( udp . localPort ());
```

```
}
```

```
//=====
```

```
//                      MAIN LOOP
```

```
//=====
```

```
void loop ()
```

```
{
```

```
  int cb = udp . parsePacket ();
```

```
  if (! cb ) {
```

```
    //If serial data recived send it to other device through UDP
```

```
    if( Serial . available ()>0)
```

```
    {
```

```
      udp . beginPacket ( ServerIP , 2000); //Send UDP requests are to port 2000
```

```
    char a [1];
    a [0]=char( Serial . read ());
    udp . write ( a ,1);    //Serial Byte Read
    udp . endPacket ();
  }
}
else {
  // We've received a packet, read the data from it
  udp . read ( packetBuffer , 1); // read the packet into the buffer, we are reading only one byte
  Serial . print ( packetBuffer );
  delay (20);
}
}
```

//=====:

Results

Open both boards serial monitor windows and type data in one serial monitor. It will appear in second serial monitor and vice-versa. Use other serial monitor terminal Arduino may not allow two serial monitor windows. UDP is not reliable but ok for sensor data communication

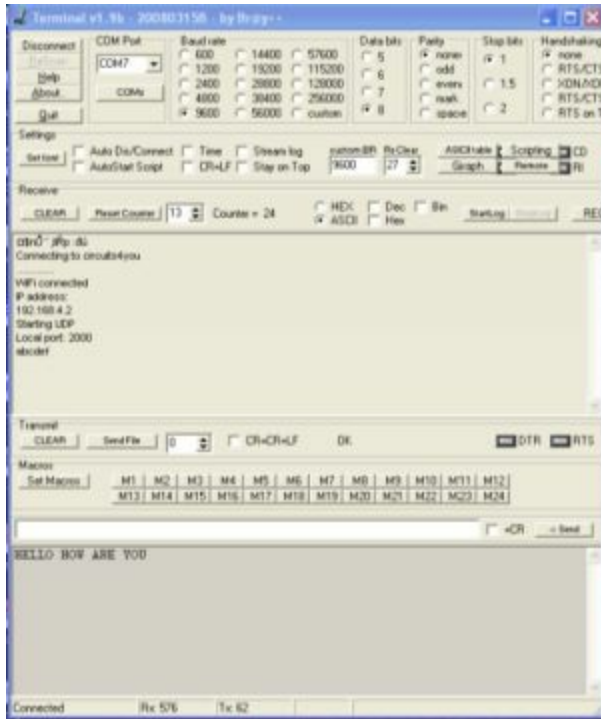


Figure 16.1: Wireless Serial using UDP

17. Accessing ESP8266 over internet

17.1 Introduction

In many cases you want to access your IoT ESP8266 over internet to control home appliances or monitor data, many cloud service provider give you one way access that is you can only monitor the data of your ESP, there is very few websites that may allow you to control your ESP. Personally I have not seen any website that allows control of IoT device example on/off buttons.

Before you start, here are a few points:

Note: Accessing of ESP over internet is possible only for some cases. This lesson is also helpful to access ESP in your own private network with different IP range.

127.0.0.1 is NOT your IP Address. This is an address used to access your own computer FROM your own computer. It's a LOCALHOST address and you can't use it to access your PC from another PC.

192.xx.xx.xx / 10.xx.xx.xx/ 169.xx.xx.xx is also NOT your IP Address. This is your computers or ESPs unique IP Address on your home network/private network. You can access your server on your own network only with this address. You will also need this address if you need to use port forwarding in your router.

Any other IP Address is likely to be your WAN or Internet IP Address. To be sure you have the correct one, go to the google and search "ip". It will tell you your real IP, and, if you are behind a proxy, the IP of your proxy. You need the real IP as this is the one people will use to access your server.

Does your ISP block port 80?

The simplest way to find out is to ring them up and ask. If they won't tell you or you are simply unsure, try changing ESP Web Server to another port and then see if you can access it. Port number is given after IP address ex. For port 8080 use 192.168.4.1:8080

Do you have a router? If so then check the following things:

Have you enabled port forwarding? This is the option that allows incoming

traffic to be diverted to your computer or ESP. You can enable port forwarding using NAT settings where you can find DMZ configuration, enter your ESP IP address and enable it and apply changes. Make sure that your ESP IP should not change on reconnection with your wifi router.

The screenshot displays the web interface of a DIGISOL DG-HR3400 300Mbps Wireless Broadband Router. The interface has a dark red header with the brand name and model. Below the header is a navigation bar with tabs for Setup, Wireless, Advanced, Maintenance, Status, and Help. The 'Advanced' tab is selected, and the left sidebar shows a list of configuration options including Access Control List, Port Triggering, DMZ, URL Block, IP/Port Filter, MAC Filter, DOS Settings, Dynamic DNS, QoS Setup, UPnP, Routing, and Virtual Server. The main content area is titled 'DMZ' and contains a description of a Demilitarized Zone. Below this is the 'DMZ Configuration' section, which includes a checkbox to 'Enable DMZ' (which is checked) and a text field for 'DMZ Host IP Address' set to '192.168.2.2'. At the bottom of the configuration section are 'Apply Changes' and 'Reset' buttons.

DIGISOL™ DG-HR3400 300Mbps Wireless Broadband Router						
	Setup	Wireless	Advanced	Maintenance	Status	Help
Access Control List	<div>DMZ</div> <p>A Demilitarized Zone is used to provide Internet services without sacrificing unauthorized access to its local private network. Typically, the DMZ host contains devices accessible to Internet traffic, such as Web (HTTP) servers, FTP servers, SMTP (e-mail) servers and DNS servers.</p> <div>DMZ Configuration</div> <div><input checked="" type="checkbox"/> Enable DMZ</div> <div>DMZ Host IP Address: <input type="text" value="192.168.2.2"/></div> <div><input type="button" value="Apply Changes"/> <input type="button" value="Reset"/></div>					

Figure 17.1: Enable port forward

In some cases doing above step is enough to get your ESP on the internet or in your WAN. For accessing ESP in WAN, enter IP assigned to your WiFi router by WAN network. You can fix your ESP IP address in WiFi router using DHCP Static IP configuration settings as shown in below figure.

The screenshot displays a router's configuration page with a sidebar on the left containing links: **Wireless**, **Local Network**, **Internet Setup**, **IPSec**, and **Mobile Settings**. The main content area is titled **LAN Interface Setup** and includes a **Helpful Hints...** section on the right. The **LAN Interface Setup** section contains fields for **IP Address** (192.168.2.1) and **Subnet Mask** (255.255.255.0), with an **Apply Changes** button below. The **DHCP Server Settings** section shows **DHCP Mode** set to **DHCP Server**, **IP Pool Range** from 192.168.2.2 to 192.168.2.254, **Max Lease Time** of 120 minutes, **Domain Name** as domain.local, and **DNS Server 1** as 192.168.2.1. It also includes **Apply Changes** and **Undo** buttons. The **DHCP Static IP Configuration** section features fields for **IP Address** (0.0.0.0) and **Mac Address** (00:00:00:00:00:00), with an **Apply Changes** button. The **DHCP Static IP Table** section has a table with columns **Select**, **IP Address**, and **MAC Address**.

Select	IP Address	MAC Address
--------	------------	-------------

Figure 17.2: Static IP for ESP

Do you have a firewall? If so check the following points:

Is it allowing incoming and outgoing connections on the port you are using ESP Web Server on? Is it allowing ESP Web Server to access the internet and act as a server?

If you cannot figure out how to use your firewall then temporally disable it while you are getting it to work for the outside world. Worry about the firewall when you get it working.

If it works by doing these things remember that your public IP changes after reconnecting to your ISP. Make sure use new IP.

Access of ESP over internet did not worked for me as I am having two WiFi devices. One that gets internet signals from internet service provider and second device that gives WiFi network in home.

But at my friends home by just putting IP in DMZ settings of ADSL router worked perfectly. He is having Internet connection through wired telephone line.

18. Access ESP in VB.net

18.1 Introduction

Accessing ESP in VB.Net is simple with use of UDP protocol. You can use serial too but we are making IoT applications.

UDP (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP) used primarily for establishing low-latency and loss tolerating connections between applications on the Internet. Both UDP and TCP run on top of the Internet Protocol (IP) and are sometimes referred to as UDP/IP or TCP/IP. Both protocols send short packets of data, called datagrams. In this lesson we are sending Analog value to VB.NET application and controlling RGB LED of ESPWitty.

18.2 Programming

VB.NET

Prepare VB.NET Form as shown in below figure. Or [download complete project file from here](#)

For this demo purpose VB.NET 2008 edition is used

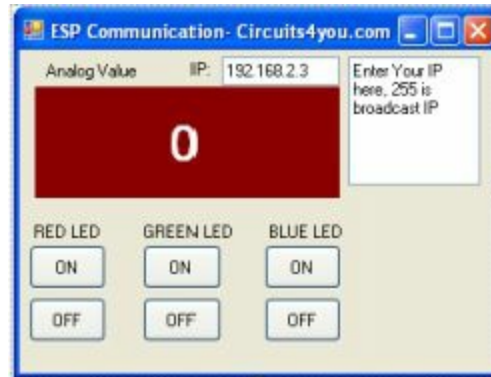


Figure 18.1: VB.net Application

VB.net program

```
Imports System.Data
Imports System.Net
Imports System.Text.Encoding
Imports Microsoft.VisualBasic
Imports System
Public Class Form1
    Dim publisher As New.Sockets.UdpClient(0)
    Dim subscriber As New.Sockets.UdpClient(2000) 'Port number used by UDP

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        subscriber.Client.ReceiveTimeout = 100
        subscriber.Client.Blocking = True
    End Sub

    Private Sub RED_ON_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RED_ON.Click
        publisher.Connect(IP_TXT.Text, 2000) '255 is broad cast IP
        Dim sendbytes() As Byte
        sendbytes = ASCII.GetBytes("a") 'Send data to ESP 'a' as RED LED on
        publisher.Send(sendbytes, sendbytes.Length)
    End Sub

    Private Sub RED_OFF_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RED_OFF.Click
        publisher.Connect(IP_TXT.Text, 2000) '255 is broad cast IP
        Dim sendbytes() As Byte
        sendbytes = ASCII.GetBytes("A") 'Send data to ESP 'b' for RED LED off
        publisher.Send(sendbytes, sendbytes.Length)
    End Sub

    Private Sub GREEN_ON_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles GREEN_ON.Click
        publisher.Connect(IP_TXT.Text, 2000) '255 is broad cast IP
        Dim sendbytes() As Byte
        sendbytes = ASCII.GetBytes("b") 'Send data to ESP 'b' as Green LED on
        publisher.Send(sendbytes, sendbytes.Length)
    End Sub

    Private Sub GREEN_OFF_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles GREEN_OFF.Click
        publisher.Connect(IP_TXT.Text, 2000) '255 is broad cast IP
        Dim sendbytes() As Byte
        sendbytes = ASCII.GetBytes("B") 'Send data to ESP 'B' for Green LED off
        publisher.Send(sendbytes, sendbytes.Length)
```

End Sub

```
Private Sub BLUE_ON_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BLUE_ON.Click
    publisher.Connect(IP_TXT.Text, 2000) '255 is broad cast IP
    Dim sendbytes() As Byte
    sendbytes = ASCII.GetBytes("c") 'Send data to ESP 'c' as Blue LED on
    publisher.Send(sendbytes, sendbytes.Length)
End Sub
```

```
Private Sub BLUE_OFF_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BLUE_OFF.Click
    publisher.Connect(IP_TXT.Text, 2000) '255 is broad cast IP
    Dim sendbytes() As Byte
    sendbytes = ASCII.GetBytes("C") 'Send data to ESP 'C' for Blue LED off
    publisher.Send(sendbytes, sendbytes.Length)
End Sub
```

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Timer1.Tick
    Try
        Dim ep As IPEndPoint = New IPEndPoint(IPAddress.Any, 0) 'Get data from any IP address
        Dim rcvbytes() As Byte = subscriber.Receive(ep)

        adc_lbl.Text = ASCII.GetString(rcvbytes) 'Show ADC Value

    Catch ex As Exception 'On error skip
        'Timer1.Enabled = False
        'adc_lbl.Text = "Restart Application"
        'MsgBox(ex.Message)
    End Try
End Sub
End Class
```

ESP8266 Program

Program is well commented. ESP gives too much flexibility, you can have Access point with web server, web client with wifi router connections and UDP at the same time on ESP. ESP has no limits on the creation of multiple protocols and connections.

Most of the time it is useful to have

1. ESP as access point with Web server for setting the Connection parameters, names.
2. ESP connection to WiFi router for accessing internet to upload data on cloud server.
3. Device to device communication with UDP protocol.

```
/*
  UDP Communication with VB.NET
  Hardware: ESPWitty
  Circuits4you.com
  2016
*/

#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

const char * ssid = "your ssid"; // your network SSID (name)
const char * pass = "your password"; // your network password

unsigned int localPort = 2000; // local port to listen for UDP packets
unsigned long sendPacket ( IPAddress & address );

IPAddress ServerIP (192,168,2,2); //Enter Your PC IP address where VB.NET Application is
running

// A UDP instance to let us send and receive packets over UDP
WiFiUDP udp ;

//Hardware connections of LEDs
const int RedLED =15;
const int GreenLED =12;
const int BlueLED =13;

char packetBuffer [9]; //Where we get the UDP data
//=====================================================
//                               Setup
```

```

//=====
void setup ()
{
  Serial . begin (9600);
  Serial . println ();

  // We start by connecting to a WiFi network
  Serial . print ("Connecting to ");
  Serial . println ( ssid );
  WiFi . begin ( ssid , pass );

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

  Serial . println ("WiFi connected");
  Serial . println ("IP address: ");
  Serial . println ( WiFi . localIP ());

  Serial . println ("Starting UDP");
  udp . begin ( localPort );
  Serial . print ("Local port: ");
  Serial . println ( udp . localPort ());

  //LED IO Directions
  pinMode ( RedLED , OUTPUT );
  pinMode ( GreenLED , OUTPUT );
  pinMode ( BlueLED , OUTPUT );
}
//=====
//                               MAIN LOOP
//=====
void loop ()
{
  int cb = udp . parsePacket ();

  if (! cb ) {
    //Serial.println("no packet yet");
    //Send ADC Data to VB.net application
    sendPacket ( ServerIP );
    delay (500); //We don't want to overload the network with lots of UDP packets
    //Removing this delay will hang your PC
  }
  else {
    Serial . print ("Packet received, length= ");
    Serial . println ( cb );
    // We've received a packet, read the data from it
    udp . read ( packetBuffer , 1); // read the packet into the buffer, we are reading only one byte
  }
}

```



```

Serial . print ("Packet Data = ");
Serial . println ( packetBuffer );

if( packetBuffer [0]=='a')
{ digitalWrite ( RedLED , HIGH ); } //Red LED on
if( packetBuffer [0]=='A')
{ digitalWrite ( RedLED , LOW ); } //Red LED off

if( packetBuffer [0]=='b')
{ digitalWrite ( GreenLED , HIGH ); } //Green LED on
if( packetBuffer [0]=='B')
{ digitalWrite ( GreenLED , LOW ); } //Green LED off

if( packetBuffer [0]=='c')
{ digitalWrite ( BlueLED , HIGH ); } //Blue LED on
if( packetBuffer [0]=='C')
{ digitalWrite ( BlueLED , LOW ); } //Blue LED off

delay (20);
}
}

//=====
// send an UDP request to the time server at the given address
//=====
unsigned long sendPacket ( IPAddress & address )
{
  //Serial.println("sending NTP packet...");
  char cVtg [5];
  int vtg ;
  vtg = analogRead ( A0 ); //Read analog value
  sprintf ( cVtg , "%04d", vtg ); //Convert integer to 4 byte ascii

  //Send UDP packet
  udp . beginPacket ( address , 2000); //Send UDP requests are to port 2000
  udp . write ( cVtg ,4); //Analog Read Voltage
  udp . endPacket ();
}

```

Results

Modify ssid, password and enter your PC IP address in code. Make sure that your firewall may block port 2000, temporarily disable it. Start your ESP and see the serial monitor window and get IP address of ESP and put it into VB.NET application. As shown in below figure 18.2. It will show constantly updating ADC value and you can control RGB led using buttons. UDP is useful for high speed communication.

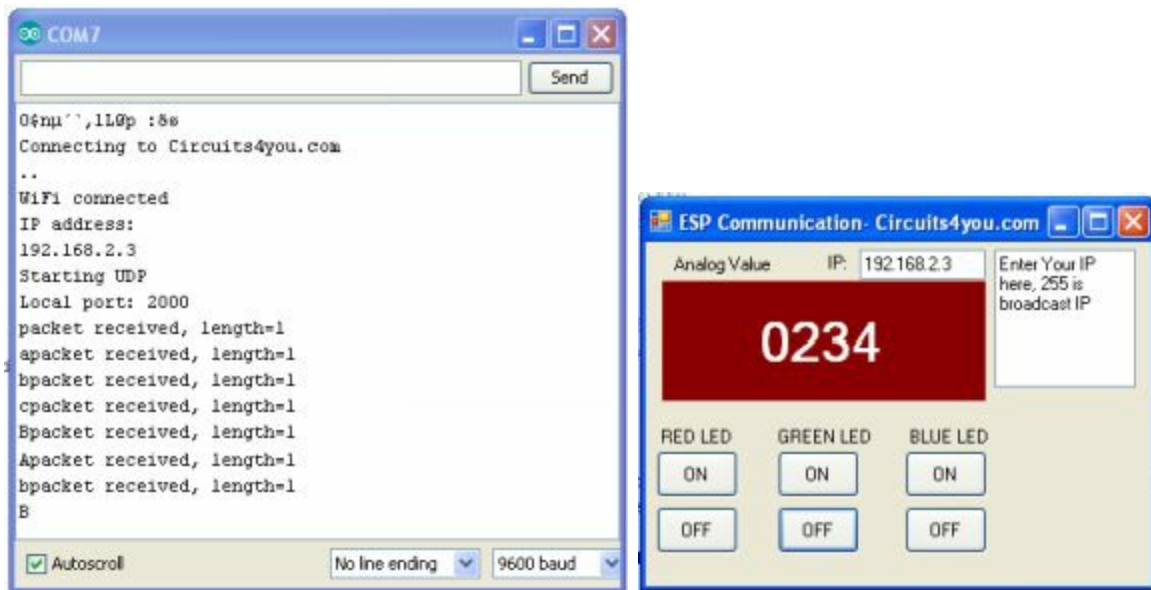


Figure 18.2: Serial monitor and VB.net application

19. Plotting Graphs using ESP8266

19.1 Introduction

Plotting graphs like cloud server on ESP as web server is possible using **Scalable Vector Graphics (SVG)** is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. Let's make ESP as Web server and plot analog value graph on our HTML web page.

19.2 HTML Webpage with SVG Graph

First we test html web page on computer and understand SVG basics. Create index.htm web page and open in web browser. Have a look into HTML code you will find SVG commands are similar to old CPP graphics.h commands and combination of CSS.

Index.htm

```
<!DOCTYPE html>
<html>
<head>
    <title>Plotting Graphs</title>
</head>
<body>
<center>
<svg width="510" height="210">
    <rect x="5" y="5" rx="20" ry="20" width="500" height="200"
    style="fill:#A0FFA0;stroke:black;stroke-width:2;opacity:0.8" />

    <text x="220" y="25" fill="black">ADC Value</text>

    <line x1="5" y1="105" x2="505" y2="105" style="stroke:green;stroke-width:1;opacity:0.3"/>
    <line x1="5" y1="52" x2="505" y2="52" style="stroke:green;stroke-width:1;opacity:0.3"/>
    <line x1="5" y1="157" x2="505" y2="157" style="stroke:green;stroke-width:1;opacity:0.3"/>
    <line x1="5" y1="105" x2="50" y2="50" style="stroke:black;stroke-width:1;opacity:1"/>

    <circle cx="50" cy="50" r="4" stroke="green" stroke-width="1" fill="yellow" />
    Sorry, your browser does not support inline SVG.
</svg>

</br>
<a href="http://circuits4you.com">visit us: circuits4you.com</a>
</center>
</body>
</html>
```

Results of html web page

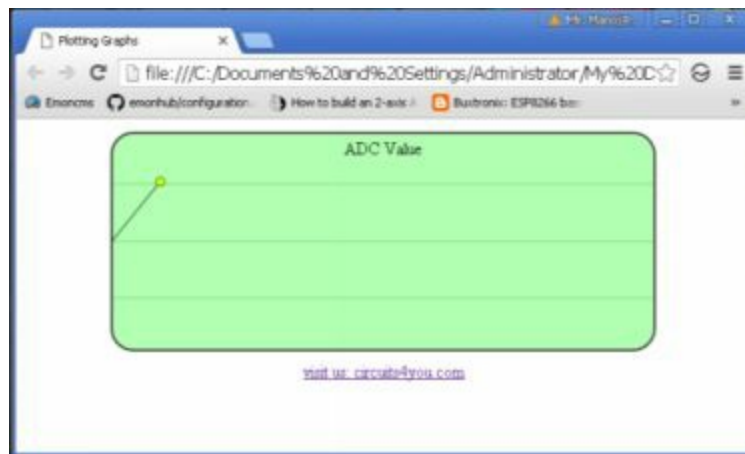


Figure 19.1: HTML Web page results

Try modifying html web page and understand svg.

Another way to use SVG is as image in HTML code as we want dynamic graph we create svg image dynamically and call it in html web page. First understand the SVG file and how to use it in HTML. Create tow files **test.svg** and **index.htm**.

test.svg

You can use notepad to create this file and save with extension .svg.

```
<svg xmlns="http : //www .w3.org/2000/svg" width="510" height="210">
  <rect x="5" y="5" rx="20" ry="20" width="500" height="200"
    style="fill:#A0FFA0;stroke: black ;stroke-width: 2 ;opacity: 0 .8" />

  <text x="220" y="25" fill="black">ADC Value</text>

  <line x1="5" y1="105" x2="505" y2="105" style="stroke: green ;stroke-width: 1 ;opacity: 0 .3"/>

  <line x1="5" y1="52" x2="505" y2="52" style="stroke: green ;stroke-width: 1 ;opacity: 0 .3"/>

  <line x1="5" y1="157" x2="505" y2="157" style="stroke: green ;stroke-width: 1 ;opacity: 0 .3"/>

  <line x1="5" y1="105" x2="50" y2="50" style="stroke: black ;stroke-width: 1 ;opacity: 1"/>

  <circle cx="50" cy="50" r="4"
    stroke="green" stroke-width="1" fill="yellow" />

  Sorry , your browser does not support inline SVG .
</svg>
```

Index.htm

```
<!DOCTYPE html>
<html>
<head>
  <title>Plotting Graphs</title>
</head>
<body>
<center>
  
</br>
<a href="http://circuits4you.com">visit us: circuits4you.com</a>
</center>

</body>
</html>
```


Keep both index.htm and test.svg file at same location and open index.htm in web browser you will find same results as shown in figure 19.1.

19.3 ESP8266 Programming

Based on knowledge of SVG lets make ESP as web server to display analog values on graphs. Also have a look on java script based readily available graph tool.

In HTML program we have used SVG and Auto refresh using HTML syntax

```
< meta http - equiv ="refresh" content ="5">
```

Graph routine plots graph using mathematical tricks. Program consists of two parts HTML header and ESP main code

Index.h

```
const char MAIN_page [] PROGMEM = R "=====(
<!DOCTYPE html >
< html >
< head >
    < title > Plotting Graphs </ title >
    < meta http - equiv ="refresh" content ="5">
</ head >
< body >
< center >
    < img src ="test.svg">
</ br >
< a href ="http://circuits4you.com"> visit us : circuits4you . com </ a >
</ center >

</ body >
</ html >
)=====";
```

ESP8266 Program

Graph.ino

```
/*
 * SVG Graph Plotting ESP8266
 * Copyright (c) 2016, circuits4you.com
 * All rights reserved.
 * Connects to WiFi HotSpot. */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

#include "index.h"
```

```

/* Set these to your desired credentials. */
const char * ssid = "Circuits4you.com";
const char * password = "9422212729";

ESP8266WebServer server (80);

const int led = 13; //blink blue LED every time web page is refreshed

int x =5, y =135, x1 [25], x2 [25], yy [25], y2 [25], p =0; //Graph variables
//=====
//          handles main page
//=====
void handleRoot () {
    digitalWrite ( led , 1);
    String s = MAIN_page ;
    server . send (200, "text/html", s );
    digitalWrite ( led , 0);
}
//=====
//          SVG Graph Drawing
//=====
void drawGraph () {
    String out = "<svg xmlns=\"http://www.w3.org/2000/svg\" width=\"510\" height=\"270\">\n";
    out += "<rect x=\"5\" y=\"5\" rx=\"20\" ry=\"20\" width=\"500\" height=\"260\"
style=\"fill:#A0FFA0;stroke:black;stroke-width:2;opacity:0.8\" />\n";
    out += "<text x=\"220\" y=\"25\" fill=\"black\">ADC Value</text>\n";

    //Graph background green lines
    out += "<line x1=\"5\" y1=\"67\" x2=\"505\" y2=\"67\" style=\"stroke:green;stroke-
width:1;opacity:0.3\"/>\n";
    out += "<line x1=\"5\" y1=\"135\" x2=\"505\" y2=\"135\" style=\"stroke:green;stroke-
width:1;opacity:0.3\"/>\n";
    out += "<line x1=\"5\" y1=\"202\" x2=\"505\" y2=\"202\" style=\"stroke:green;stroke-
width:1;opacity:0.3\"/>\n";

    char temp [100];
    int i ;

    //We need to keep track of value in a array
    //Keep previous values
    x1 [ p +1]= x ;
    yy [ p +1]= y ;

    //Graph x axis update every refresh
    if( x <480)
    { x = x +20; p ++;}
    else
    { x =5; p =0;}

    //Graph y axis comes from analog read value

```

```
y =(1023- analogRead ( A0 ))/4; //we have maximum height of 256, Zero starts from top so invert it
```

```
//We need to keep track of value in a array
```

```
//New Values
```

```
x2 [ p ]= x ;
```

```
y2 [ p ]= y ;
```

```
for( i =0; i <= p ; i ++ ) //Draw graph
```

```
{
```

```
    sprintf ( temp , "<line x1=\"%d\" y1=\"%d\" x2=\"%d\" y2=\"%d\" style=\"stroke:black;stroke-  
width:1;opacity:1\" />\n", x1 [ i ], yy [ i ], x2 [ i ], y2 [ i ] );
```

```
    out += temp ;
```

```
    sprintf ( temp , "<circle cx=\"%d\" cy=\"%d\" r=\"4\" stroke=\"green\" stroke-width=\"1\"  
fill=\"yellow\" />\n", x2 [ i ], y2 [ i ] );
```

```
    out += temp ;
```

```
}
```

```
out += " </svg>\n";
```

```
server . send ( 200, "image/svg+xml", out );
```

```
}
```

```
//=====
```

```
//          POWER ON SETUP
```

```
//=====
```

```
void setup ( void ) {
```

```
    pinMode ( led , OUTPUT );
```

```
    digitalWrite ( led , 0);
```

```
    Serial . begin (9600);
```

```
    WiFi . begin ( ssid , password );
```

```
    Serial . println ("" );
```

```
    // Wait for connection
```

```
    while ( WiFi . status () != WL_CONNECTED ) {
```

```
        delay (500);
```

```
        Serial . print ( "." );
```

```
    }
```

```
    Serial . println ("" );
```

```
    Serial . print ("Connected to ");
```

```
    Serial . println ( ssid );
```

```
    Serial . print ("IP address: ");
```

```
    Serial . println ( WiFi . localIP ());
```

```
    server . on ( "/", handleRoot );
```

```
    server . on ( "/test.svg", drawGraph );
```

```
    server . begin ();
```

```
    Serial . println ("HTTP server started");
```

```
}
```

```
//=====
```

```
//          LOOP
//=====
void loop ( void ) {
    server . handleClient ();
}
//=====
```

Results

Open web browser and enter ESP8266 IP. You will see below results. Auto refresh flicker is annoying for our graph it refreshes complete web page this can be avoided using javascript graphs. But this graph is useful to show static graphs.



20. Using Google Gadgets

20.1 Introduction

At Google, gadgets are HTML and JavaScript applications that can be embedded in web pages and other apps, including Sites. These gadgets offer the ability to include external and dynamic content within your site, such as miniature applications and database-driven lists, incorporated with text and images for a seamless user experience.

Generically, gadgets are small utilities that generate or pull external information into web pages. In its simplest form, a gadget is a small .xml file that retrieves information with the ability to make it available in multiple web pages at once. In Sites, including a gadget results in an iframe that acts as the conduit for this external information. Some gadgets are no more than that, iframes that pass through information from another web site.

Gadgets are graphs, dials, maps and much more. You can use GPS with ESP to show your location on Google map. In this lesson we are displaying our analog value on google gauge gadget.

Visit [Google Gadget Gallery](https://developers.google.com/chart/interactive/docs/gallery)
<https://developers.google.com/chart/interactive/docs/gallery>

Gauge

A gauge with a dial, rendered within the browser using SVG or VML.



Figure 20.1: Google Dial Gauge Gadget

20.2 Programming

Before we start on ESP we first make HTML web page with google gauge and understand it.

Visit

<https://developers.google.com/chart/interactive/docs/gallery/gauge>

understanding google gauge gadget.

google
for

Index.htm

```
<html>
<head>
<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
  google.charts.load('current', {'packages':['gauge']});
  google.charts.setOnLoadCallback(drawChart);
  function drawChart() {

    var data = google.visualization.arrayToDataTable([
      ['Label', 'Value'],
      ['Memory', 80],
      ['CPU', 55],
      ['Network', 68]
    ]);

    var options = {
      width: 400, height: 120,
      redFrom: 90, redTo: 100,
      yellowFrom: 75, yellowTo: 90,
      minorTicks: 5
    };

    var chart = new google.visualization.Gauge(document.getElementById('chart_div'));

    chart.draw(data, options);

    setInterval(function() {
      data.setValue(0, 1, 40 + Math.round(60 * Math.random()));
      chart.draw(data, options);
    }, 13000);
    setInterval(function() {
      data.setValue(1, 1, 40 + Math.round(60 * Math.random()));
      chart.draw(data, options);
    }, 13000);
  }
</script>
</head>
</html>
```

```
    }, 5000);  
    setInterval(function() {  
        data.setValue(2, 1, 60 + Math.round(20 * Math.random()));  
        chart.draw(data, options);  
    }, 26000);  
    }  
</script>  
</head>  
<body>  
    <div id="chart_div" style="width: 400px; height: 120px;"></div>  
</body>  
</html>
```

For running above code you need active internet connection on your PC. It will show three gauges as shown in figure 20.1. Now we modify the code to have only one gauge showing analog value.

ESP8266 Programming

All our web server programs are split into two parts html web page and ESP program lets make these two files.

Index.h

```
const char MAIN_page[] PROGMEM = R"=====(
<html>
<head>
<meta http-equiv="refresh" content="2">
<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
  google.charts.load('current', {'packages':['gauge']});
  google.charts.setOnLoadCallback(drawChart);
  function drawChart() {

    var data = google.visualization.arrayToDataTable([
      ['Label', 'Value'],
      ['Analog', @@value@@],
    ]);

    var options = {
      width: 400, height: 400,
      redFrom: 900, redTo: 1024,
      yellowFrom:750, yellowTo: 900,
      minorTicks: 5,
      max: 1024
    };

    var chart = new google.visualization.Gauge(document.getElementById('chart_div'));
    chart.draw(data, options);
  }
</script>
</head>
<body>
  <center>
<div id="chart_div" style="width: 400px; height: 400px;"></div>
<a href="http://circuits4you.com">visit us: circuits4you.com</a>
  </center>
</body>
</html>
)=====";
```

Googlegadget.ino

ESP main code

```

/*
 * Using Google Gadgets with ESP8266
 * Copyright (c) 2016, circuits4you.com
 * All rights reserved.
 * Connects to WiFi HotSpot. */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

#include "index.h"

/* Set these to your desired credentials. */
const char * ssid = "Circuits4you.com";
const char * password = "9422212729";

ESP8266WebServer server (80);

//=====
//          handles main page
//=====
void handleRoot () {
  String s = MAIN_page ;
  int analog = analogRead ( A0 ); //Read ADC value
  s . replace ("@@value@@", String ( analog )); //Update it in HTML Code
  server . send (200, "text/html", s );
}
//=====
//          POWER ON SETUP
//=====
void setup ( void ) {
  Serial . begin (9600);
  WiFi . begin ( ssid , password );
  Serial . println ("" );

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

  Serial . println ("" );
  Serial . print ("Connected to ");
  Serial . println ( ssid );
  Serial . print ("IP address: ");
  Serial . println ( WiFi . localIP ());

  server . on ("/", handleRoot );

  server . begin ();
  Serial . println ("HTTP server started");
}

```

```
}  
//=====   
//          LOOP   
//=====   
void loop ( void ) {  
    server . handleClient ();  
}  
//=====
```

Results

Open web browser and enter the ESP IP address. You will see gauge showing analog values.



Figure 20.2: Final Results