SNS COLLEGE OF TECHNOLOGY

Coimbatore-35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with ‘A++’ Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

19ECT213- IoT SYSTEM ARCHITECTURE

II ECE / IV SEMESTER

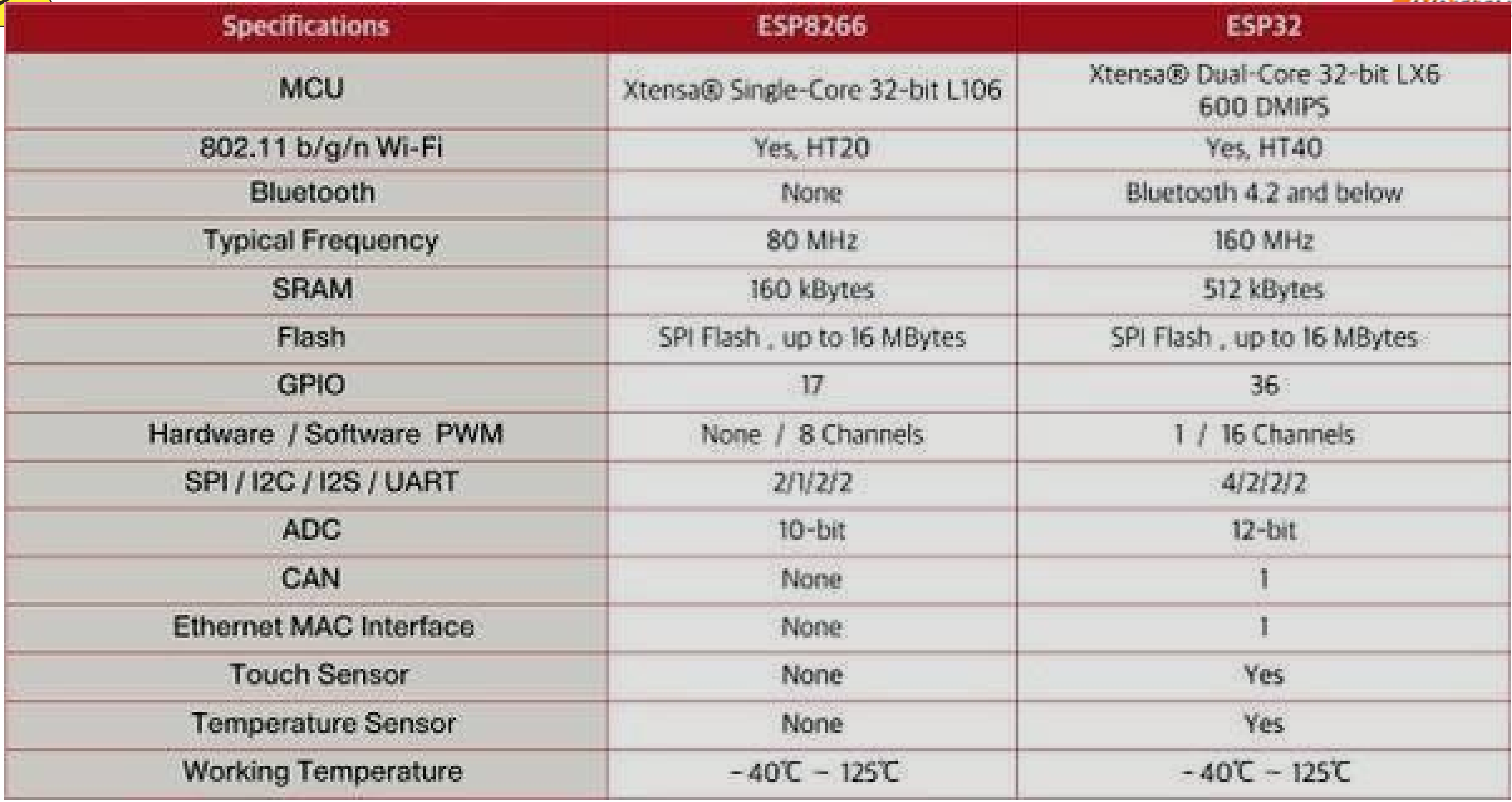
UNIT 3 – ACTUATORS AND IOT NETWORKING DEVICES

TOPIC 6 –ESP8266 Wi-Fi Module

ESP8266 Vs NodeMCU

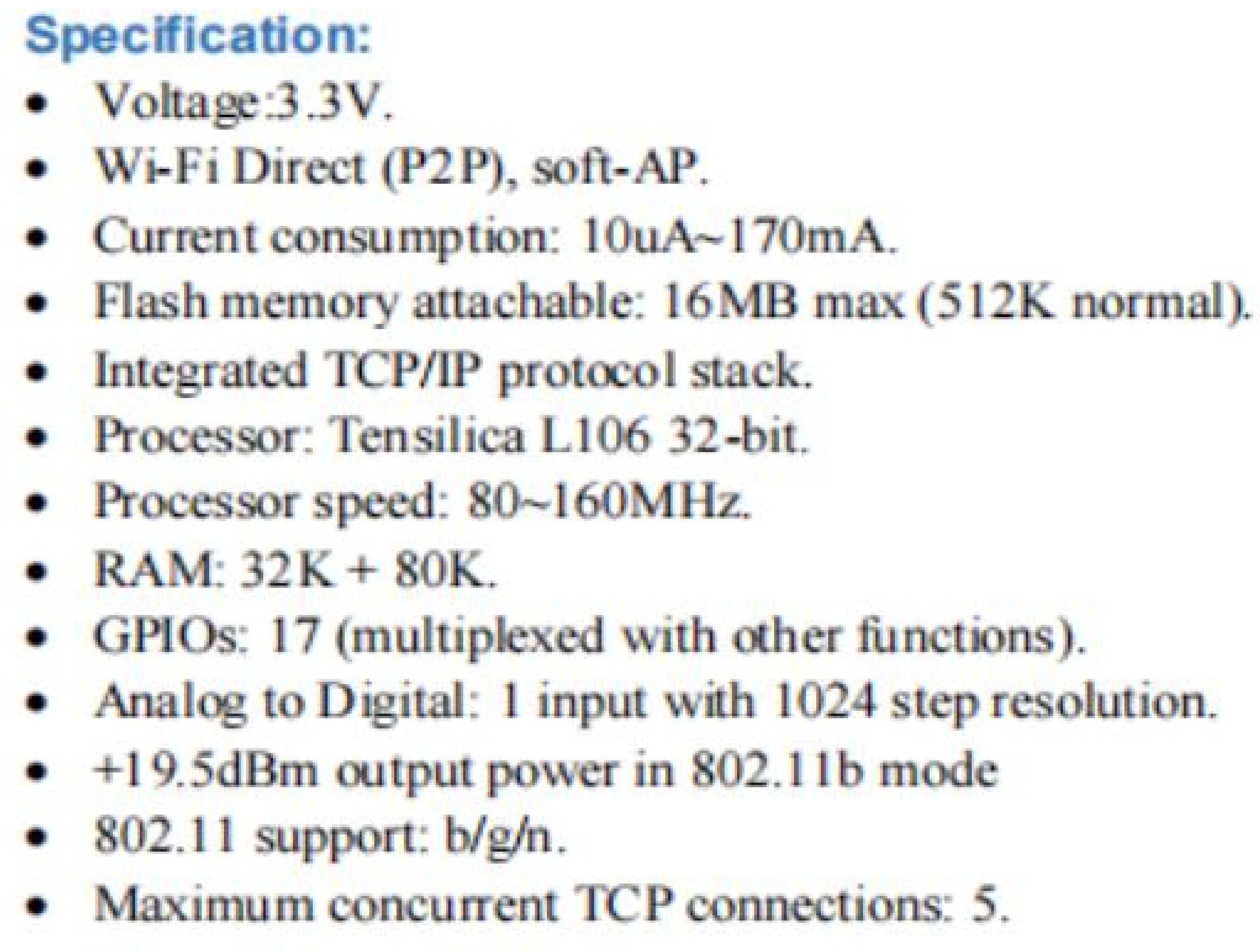
- **ESP8266**: This is a tiny, low-power Wi-Fi microcontroller chip made by Espressif Systems. It's the core component and the brains behind the operation.
- **NodeMCU**: This is an open-source firmware (software) that specifically works with the ESP8266 chip. It provides a user-friendly development environment, making it easier to program the ESP8266.
- An analogy:
- Think of ESP8266 as a powerful engine in a car.
- NodeMCU is like the dashboard and steering wheel - it provides a way to interact with the engine (ESP8266) and control its functions

ESP8266 Vs NodeMCU



Specifications	ESP8266	ESP32
MCU	Xtensa® Single-Core 32-bit L106	Xtensa® Dual-Core 32-bit LX6 600 DMIPS
802.11 b/g/n Wi-Fi	Yes, HT20	Yes, HT40
Bluetooth	None	Bluetooth 4.2 and below
Typical Frequency	80 MHz	160 MHz
SRAM	160 kBytes	512 kBytes
Flash	SPI Flash , up to 16 MBytes	SPI Flash , up to 16 MBytes
GPIO	17	36
Hardware / Software PWM	None / 8 Channels	1 / 16 Channels
SPI / I2C / I2S / UART	2/1/2/2	4/2/2/2
ADC	10-bit	12-bit
CAN	None	1
Ethernet MAC Interface	None	1
Touch Sensor	None	Yes
Temperature Sensor	None	Yes
Working Temperature	- 40℃ ~ 125℃	- 40℃ ~ 125℃

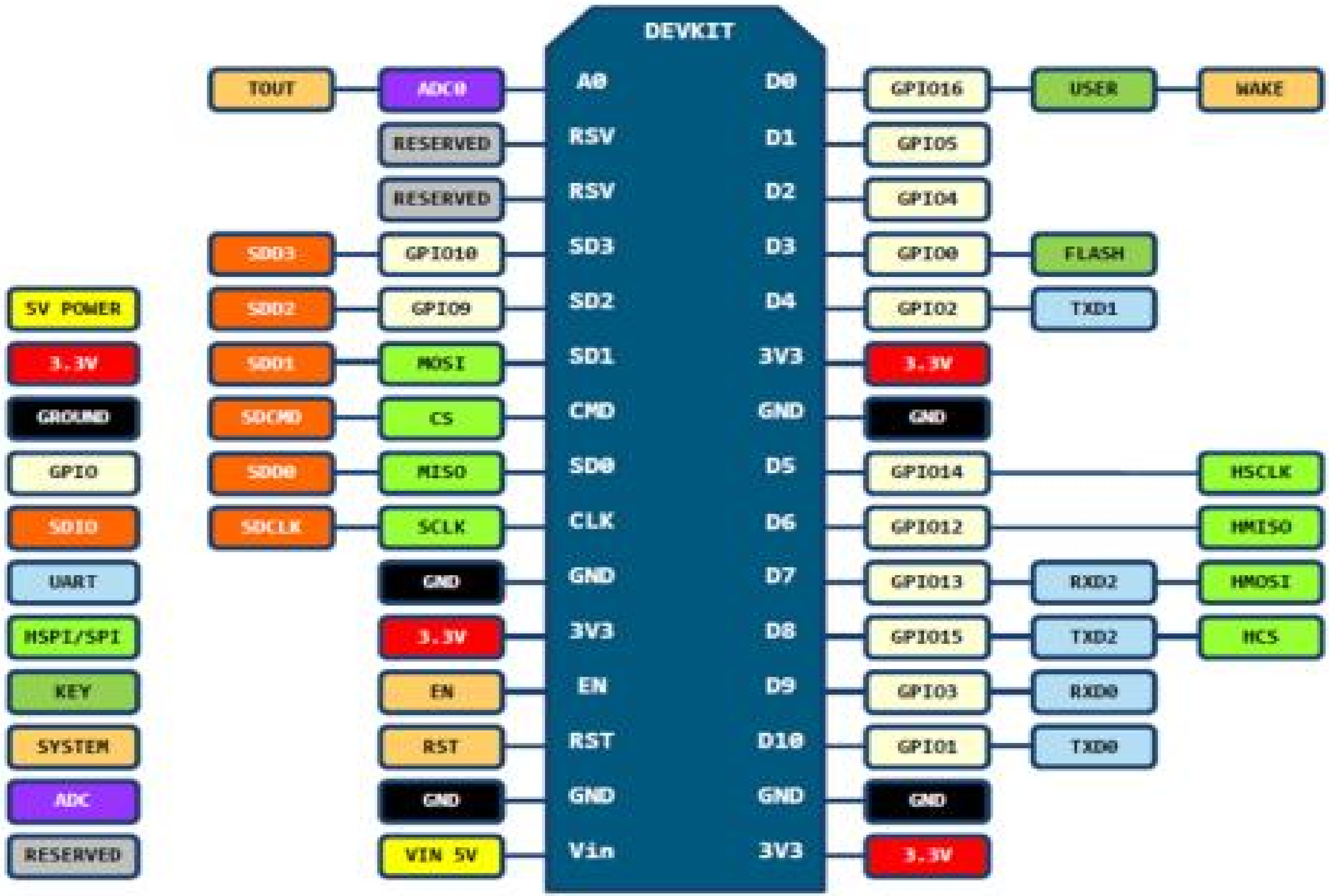
ESP8266 NodeMCU WiFi Development Board



Specification:

- Voltage:3.3V.
- Wi-Fi Direct (P2P), soft-AP.
- Current consumption: 10uA~170mA.
- Flash memory attachable: 16MB max (512K normal).
- Integrated TCP/IP protocol stack.
- Processor: Tensilica L106 32-bit.
- Processor speed: 80~160MHz.
- RAM: 32K + 80K.
- GPIOs: 17 (multiplexed with other functions).
- Analog to Digital: 1 input with 1024 step resolution.
- +19.5dBm output power in 802.11b mode
- 802.11 support: b/g/n.
- Maximum concurrent TCP connections: 5.

ESP8266



TOUT	ADC0	A0	D0	GPIO16	USER	WAKE
	RESERVED	RSV	D1	GPIO5		
	RESERVED	RSV	D2	GPIO4		
SDD3	GPIO10	SD3	D3	GPIO0	FLASH	
SDD2	GPIO9	SD2	D4	GPIO2	TXD1	
SDD1	MOSI	SD1	3V3	3.3V		
SDCMD	CS	CMD	GND	GND		
SDD0	MISO	SD0	D5	GPIO14		HSCLK
SDCLK	SCLK	CLK	D6	GPIO12		HMISO
	GND	GND	D7	GPIO13	RXD2	HMOSI
	3.3V	3V3	D8	GPIO15	TXD2	HCS
	EN	EN	D9	GPIO3	RXD0	
	RST	RST	D10	GPIO1	TXD0	
	GND	GND	GND	GND		
	VIN 5V	Vin	3V3	3.3V		

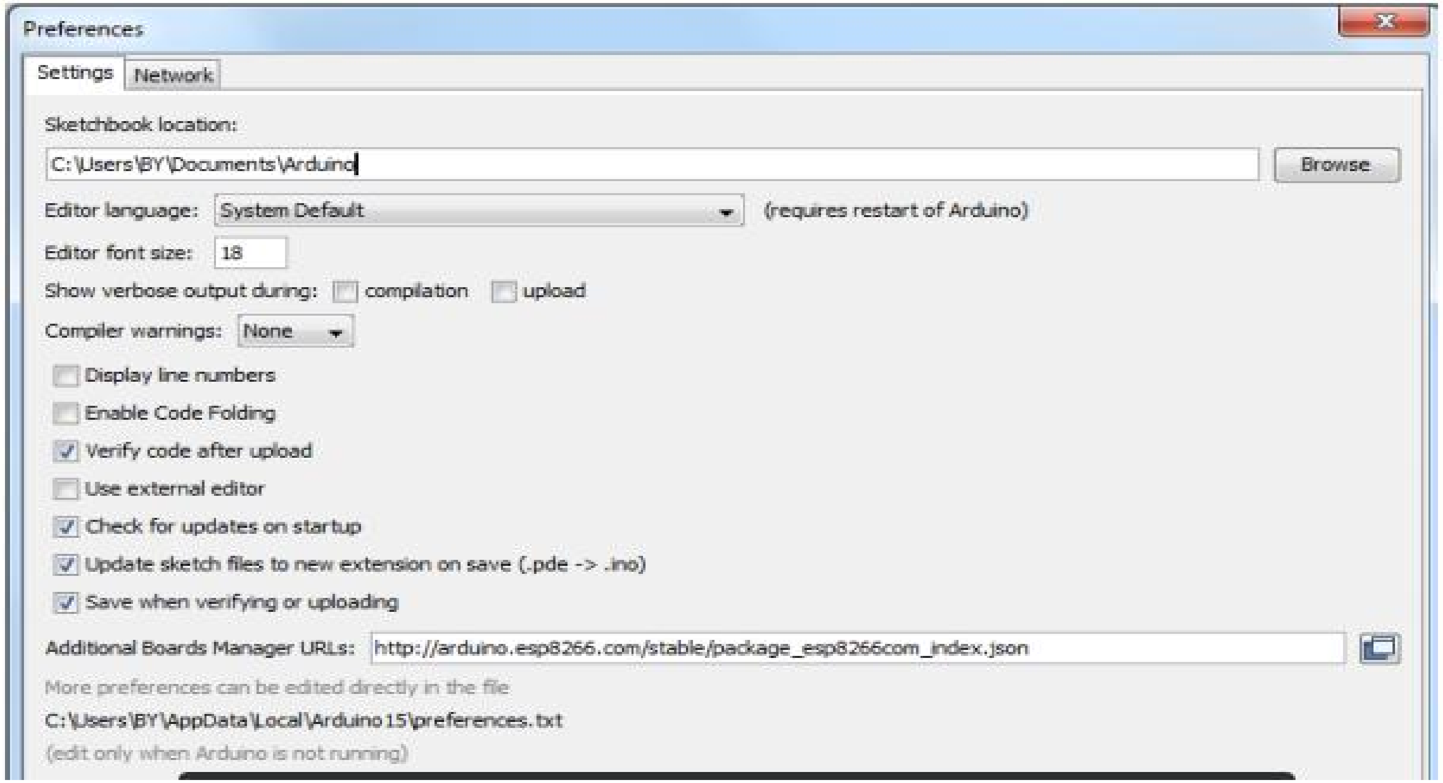
3.1 Install the Arduino IDE 1.6.4 or greater

Download Arduino IDE from Arduino.cc (1.6.4 or greater) - don't use 1.6.2 or lower version! You can use your existing IDE if you have already installed it.

You can also try downloading the ready-to-go package from the ESP8266-Arduino project, if the proxy is giving you problems.

3.2 Install the ESP8266 Board Package

Enter ***http://arduino.esp8266.com/stable/package_esp8266com_index.json*** into *Additional Board Manage* field in the Arduino v1.6.4+ preferences.



Preferences

Sketchbook location: C:\Users\BY\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 18

Show verbose output during: □ compilation □ upload

Compiler warnings: None

□ Display line numbers

□ Enable Code Folding

☑ Verify code after upload

□ Use external editor

☑ Check for updates on startup

☑ Update sketch files to new extension on save (.pde -> .ino)

☑ Save when verifying or uploading

Additional Boards Manager URLs: http://arduino.esp8266.com/stable/package_esp8266com_index.json

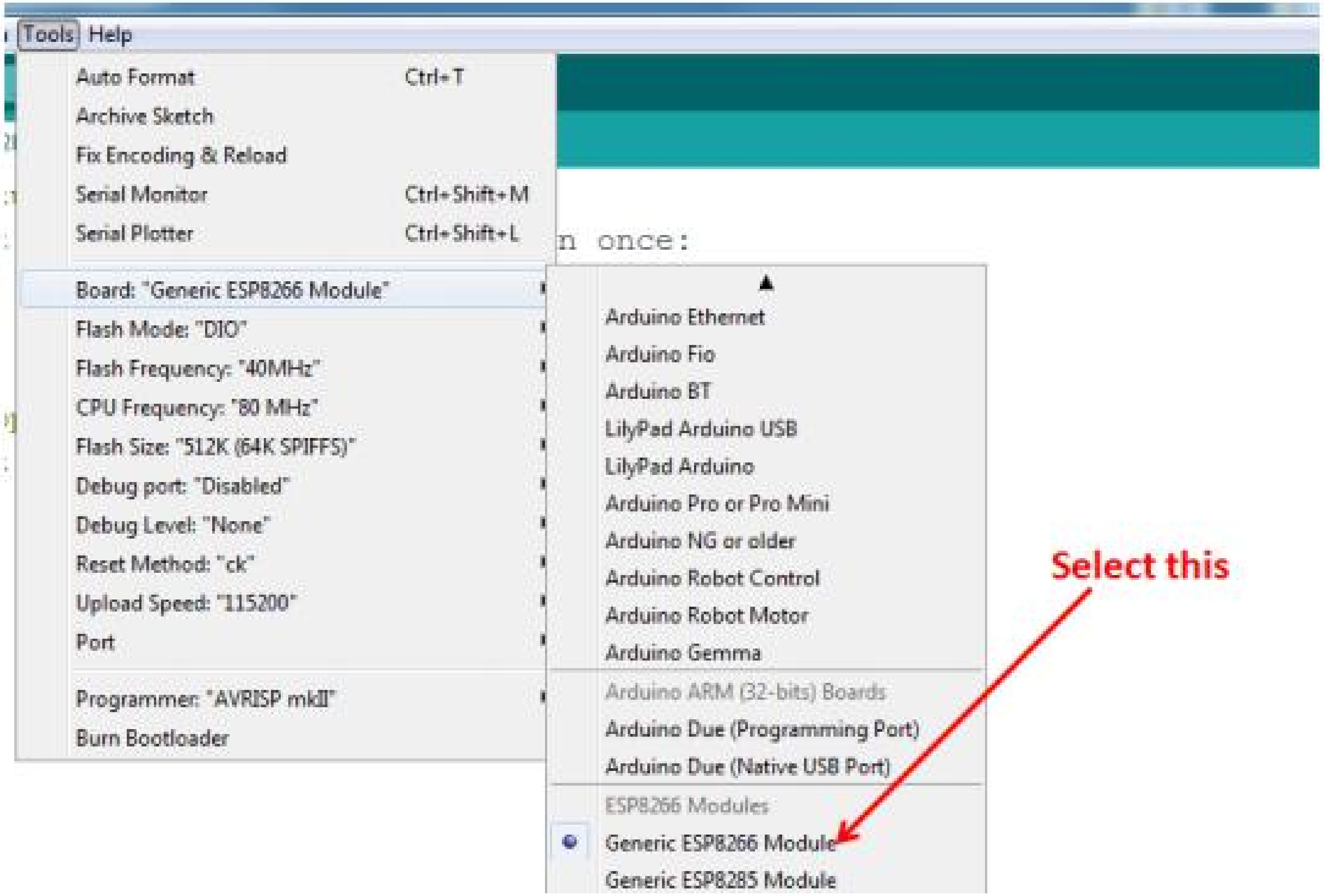
More preferences can be edited directly in the file

C:\Users\BY\AppData\Local\Arduino15\preferences.txt

(edit only when Arduino is not running)

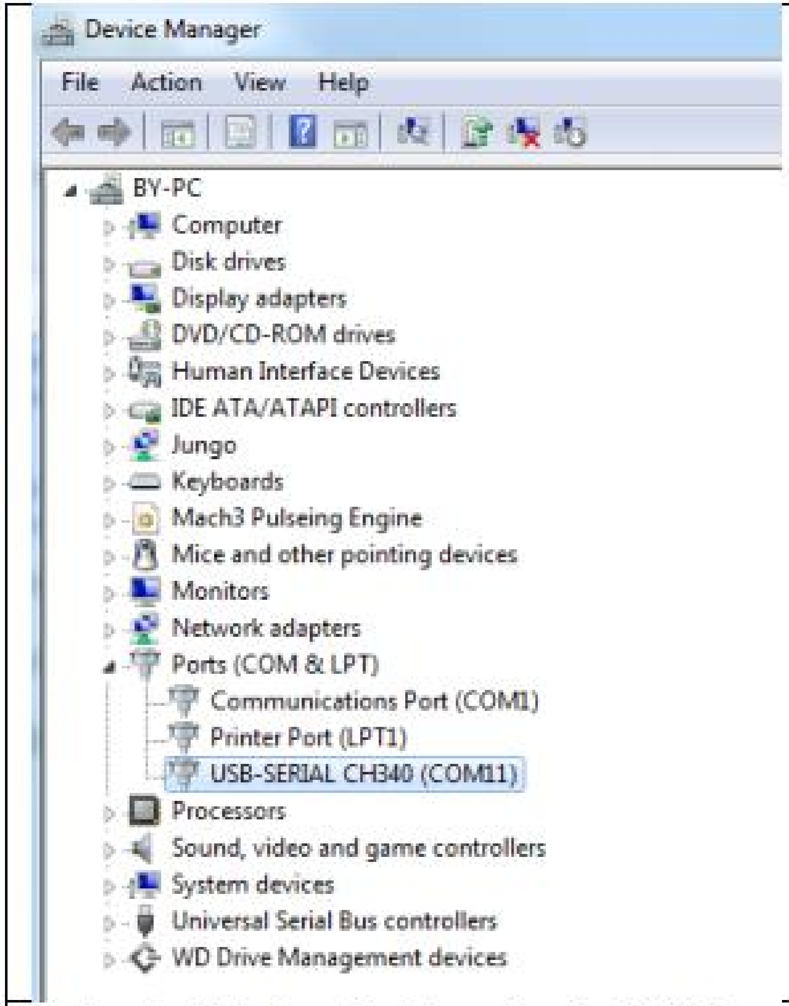
3.3 Setup ESP8266 Support

When you've restarted Arduino IDE, select 'Generic ESP8266 Module' from the 'Tools' -> 'Board:' dropdown me

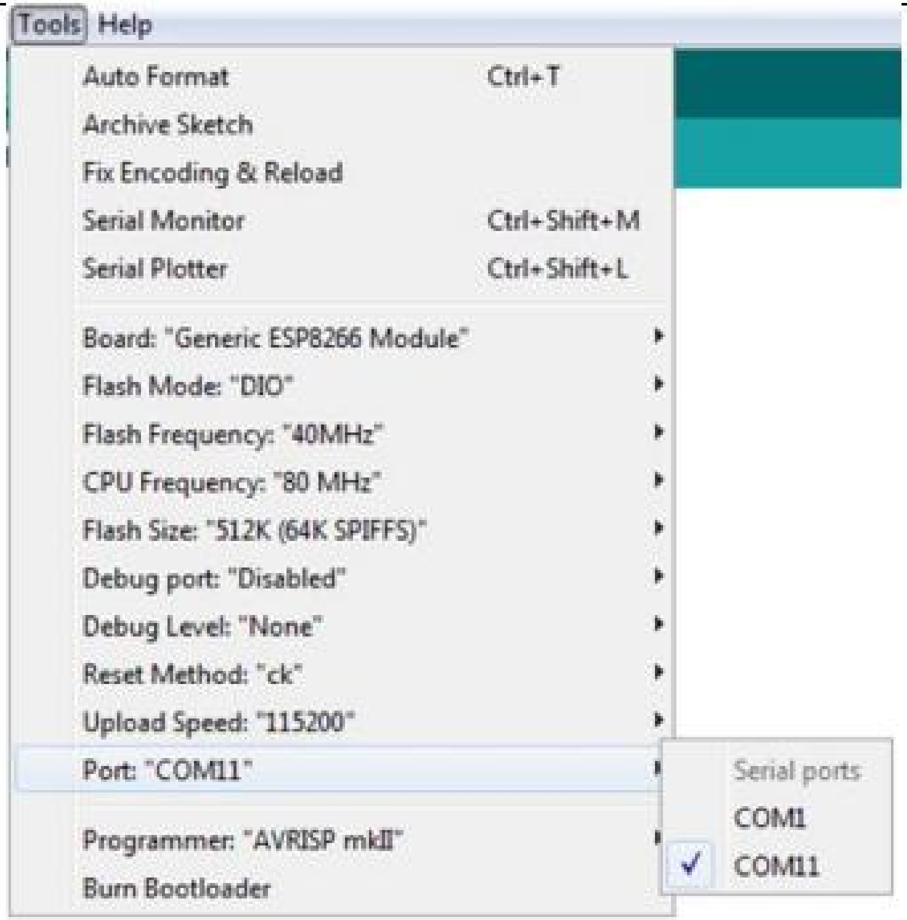


Board: "Generic ESP8266 Module"
Flash Mode: "DIO"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Flash Size: "512K (64K SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
Reset Method: "ck"
Upload Speed: "115200"
Programmer: "AVRISP mkII"

Select this



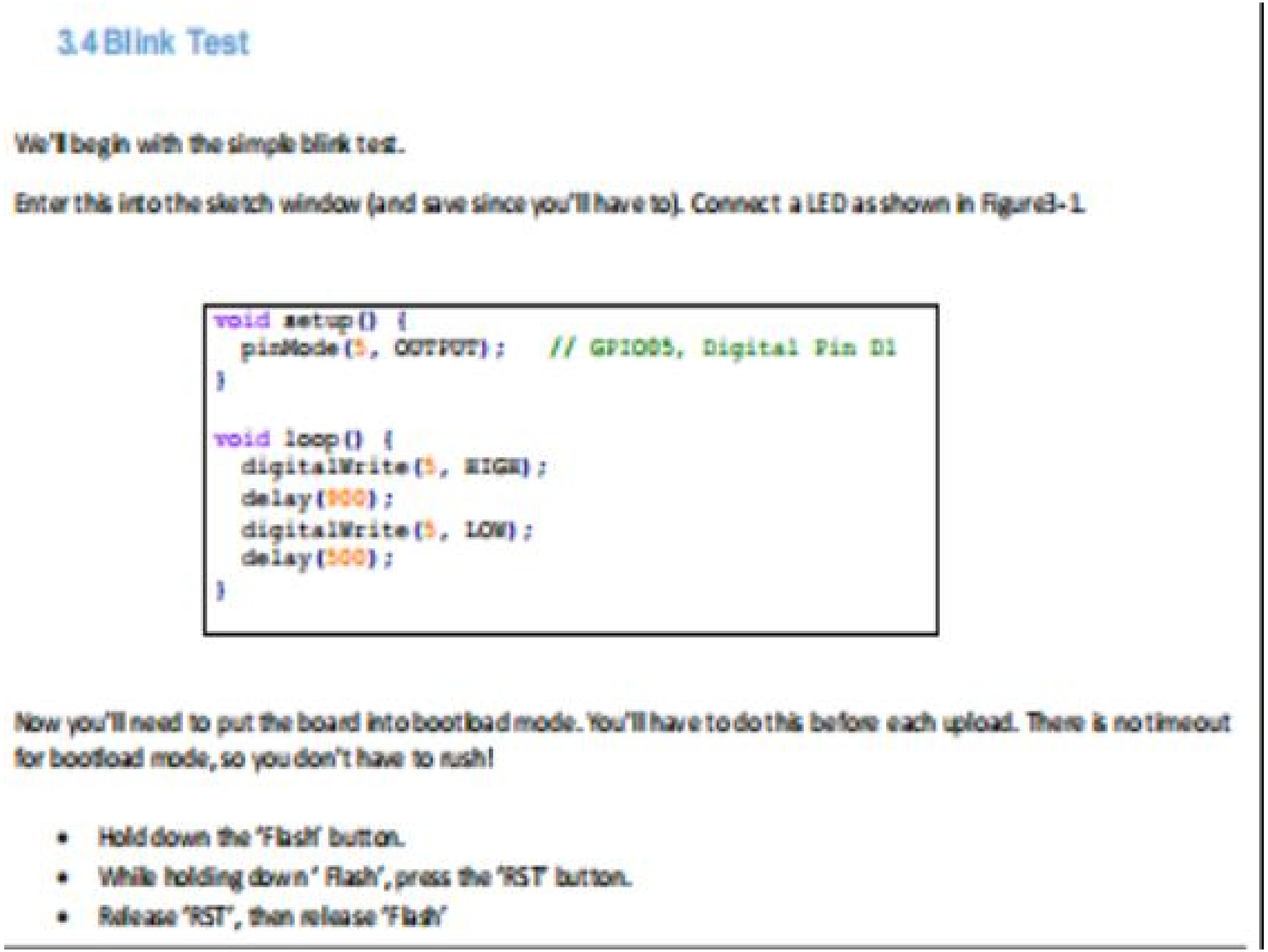
Find out which Com Port is assign for CH340



Board: "Generic ESP8266 Module"
Flash Mode: "DIO"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Flash Size: "512K (64K SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
Reset Method: "ck"
Upload Speed: "115200"
Port: "COM11"
✓ COM11
Programmer: "AVRISP mkII"

Select the correct Com Port as indicated on 'Device Manager"

Connecting via WiFi



3.4 Blink Test

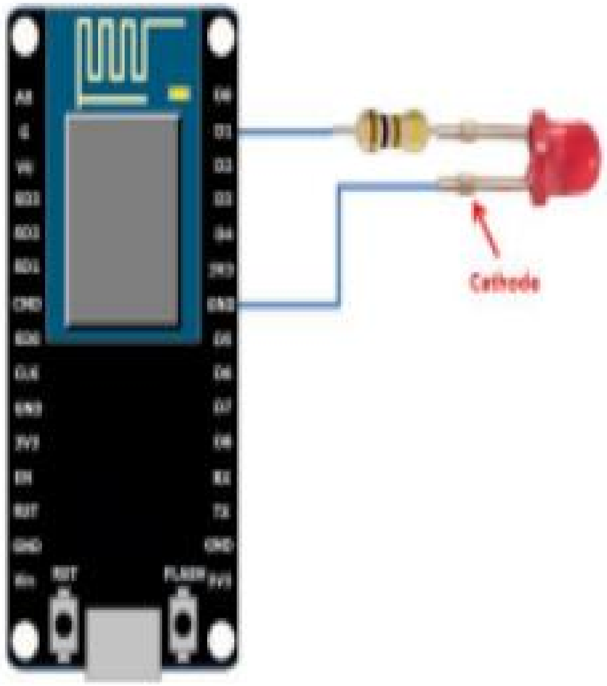
We'll begin with the simple blink test.

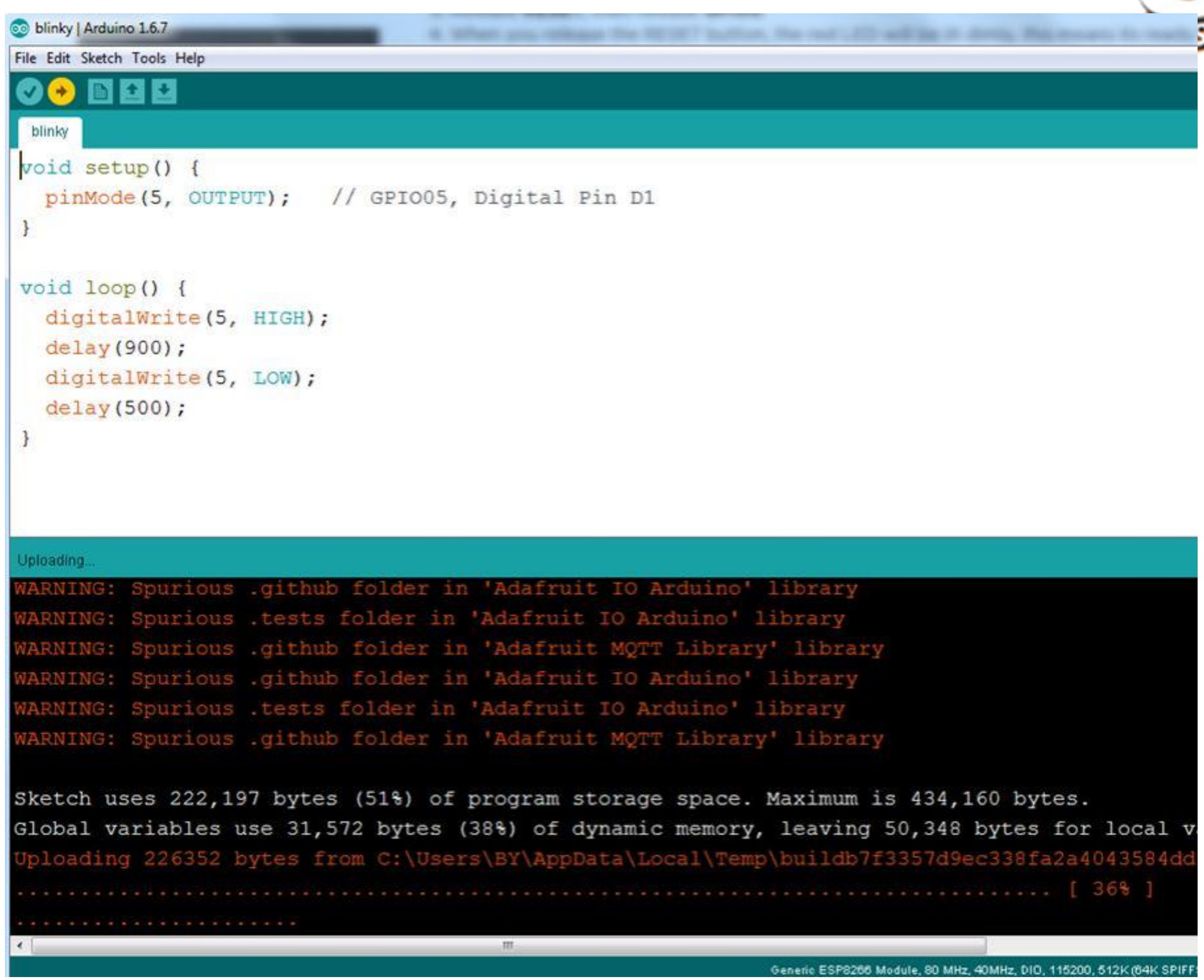
Enter this into the sketch window (and save since you'll have to). Connect a LED as shown in Figure3-1.

```
void setup() {
  pinMode(5, OUTPUT);    // GPIO05, Digital Pin D1
}

void loop() {
  digitalWrite(5, HIGH);
  delay(900);
  digitalWrite(5, LOW);
  delay(500);
}
```

Now you'll need to put the board into bootload mode. You'll have to do this before each upload. There is no timeout for bootload mode, so you don't have to rush!

- Hold down the 'Flash' button.
- While holding down ' Flash', press the 'RST' button.
- Release 'RST', then release 'Flash'



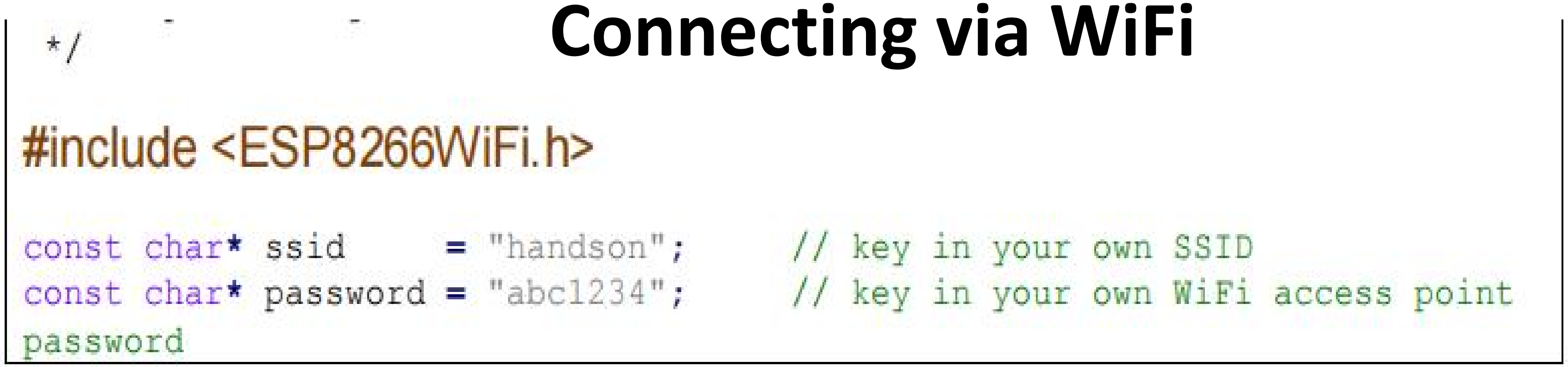


```
void setup() {
  pinMode(5, OUTPUT);   // GPIO05, Digital Pin D1
}

void loop() {
  digitalWrite(5, HIGH);
  delay(900);
  digitalWrite(5, LOW);
  delay(500);
}
```

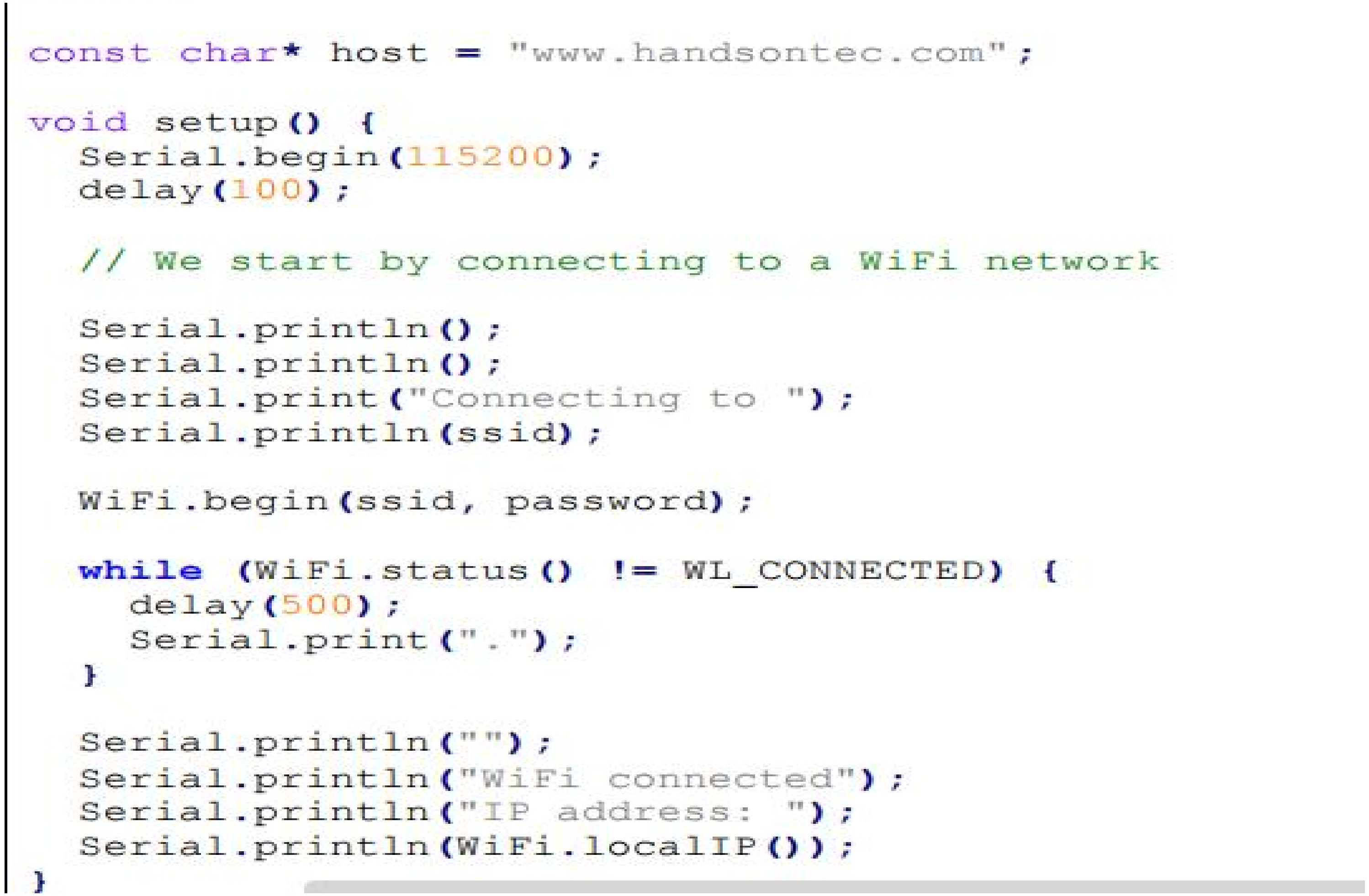
```
WARNING: Spurious .github folder in 'Adafruit IO Arduino' library
WARNING: Spurious .tests folder in 'Adafruit IO Arduino' library
WARNING: Spurious .github folder in 'Adafruit MQTT Library' library
WARNING: Spurious .github folder in 'Adafruit IO Arduino' library
WARNING: Spurious .tests folder in 'Adafruit IO Arduino' library
WARNING: Spurious .github folder in 'Adafruit MQTT Library' library

Sketch uses 222,197 bytes (51%) of program storage space. Maximum is 434,160 bytes.
Global variables use 31,572 bytes (38%) of dynamic memory, leaving 50,348 bytes for local v
Uploading 226352 bytes from C:\Users\BY\AppData\Local\Temp\buildb7f3357d9ec338fa2a4043584dd
................................................................................ [ 36% ]
......................
```

Generic ESP8266 Module, 80 MHz, 40MHz, DIO, 115200, 512K (64K SPIFF

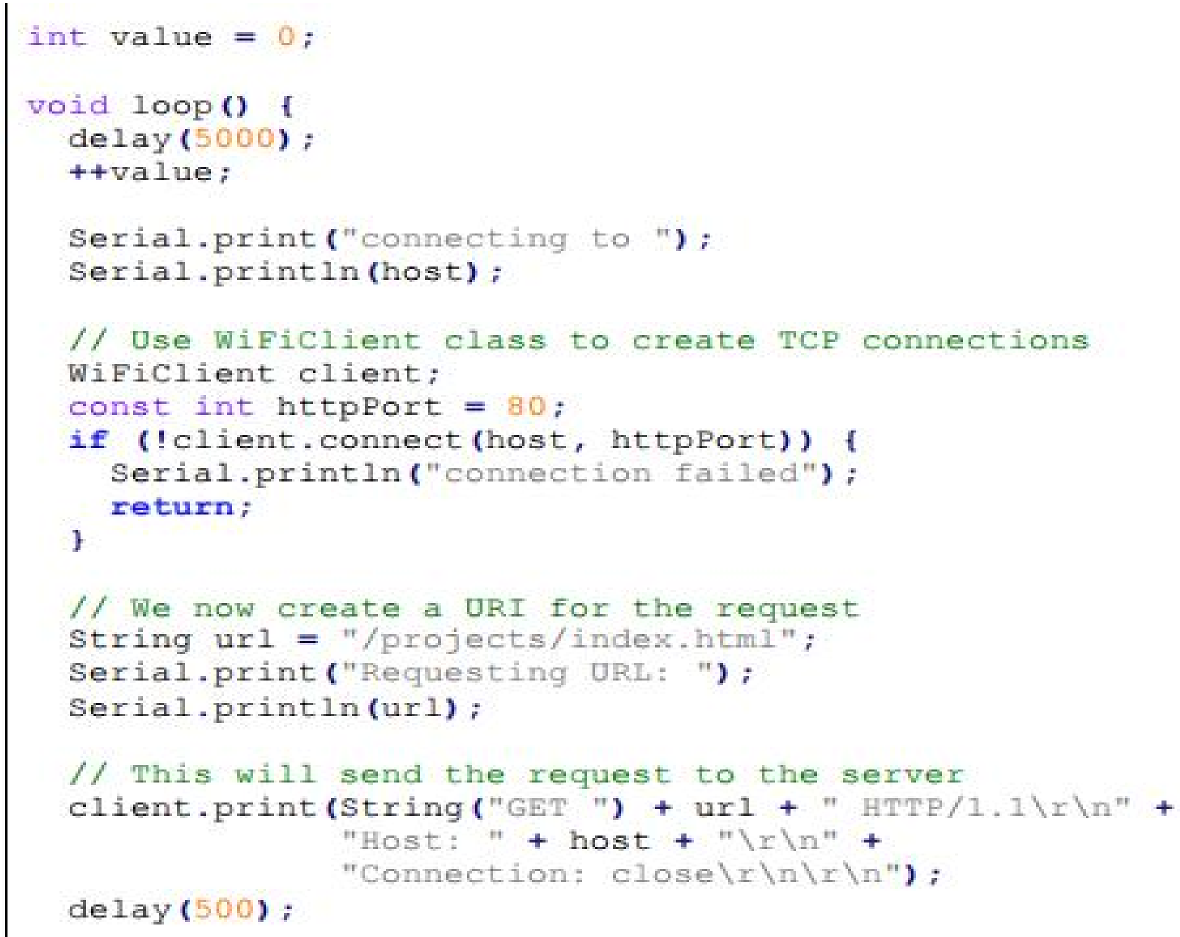
Connecting via WiFi



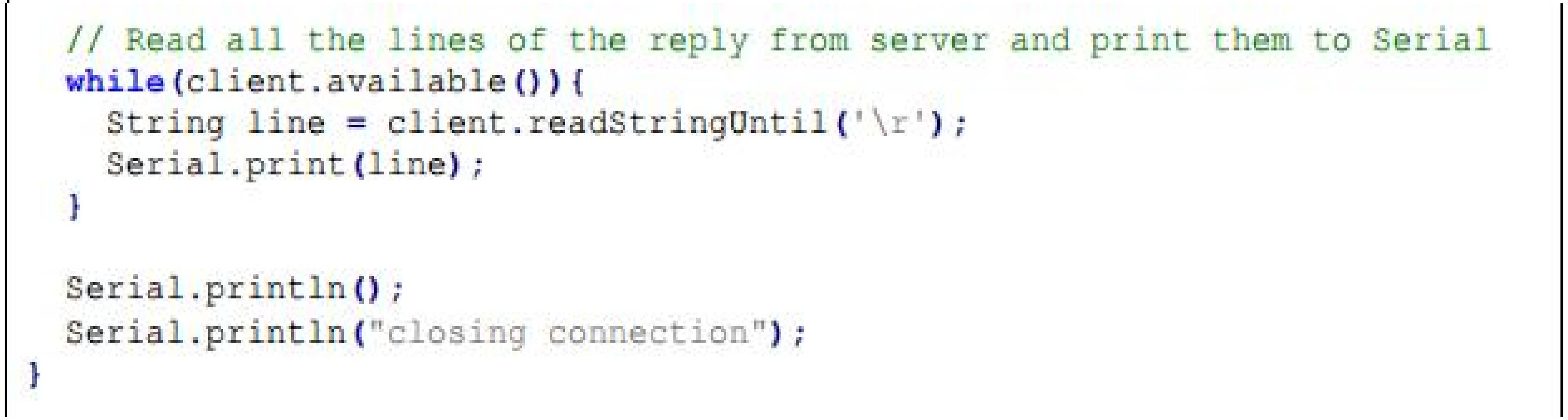
```
*/

#include <ESP8266WiFi.h>

const char* ssid     = "handson";       // key in your own SSID
const char* password = "abc1234";       // key in your own WiFi access point
password
```



```
const char* host = "www.handsontec.com";

void setup() {
  Serial.begin(115200);
  delay(100);

  // We start by connecting to a WiFi network

  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

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

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



```
int value = 0;

void loop() {
  delay(5000);
  ++value;

  Serial.print("connecting to ");
  Serial.println(host);

  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

  // We now create a URI for the request
  String url = "/projects/index.html";
  Serial.print("Requesting URL: ");
  Serial.println(url);

  // This will send the request to the server
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  delay(500);
```



```
  // Read all the lines of the reply from server and print them to Serial
  while(client.available()){
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  Serial.println();
  Serial.println("closing connection");
}
```