Instruction Manual

-Think before you connect.

"No one gets a sudden rise, not even the sun, it takes time to rise to the peak of success"
- Edmund Hillary

suryasg847@gmail.com

Instructions

- **Do not open the starter kit until instructed**
- **All the boards in the kits are tested.**
- **Follow the instruction carefully while connecting peripherals on breadboard**
- **Do not place Node MCU board on metallic surface (Specially on Laptop)**
- **Do not short 5v or 3.3 v and Ground.**
- **Cross check the connection on the breadboard before connecting NodeMCU.**

ESP8266 Wi-Fi Module

- ESP8266 is a self contained SOC
- Integrated TCP/IP protocol stack that can give any access to your Wi-Fi network
- It offers a complete and self-contained Wi-Fi networking solution
- Is capable of either hosting an application or offloading all Wi-Fi networking functions from another application processor.
- High degree of on-chip integration
- Powerful enough on-board processing and storage capability
- To be integrated with the sensors and other application specific devices through its GPIOs

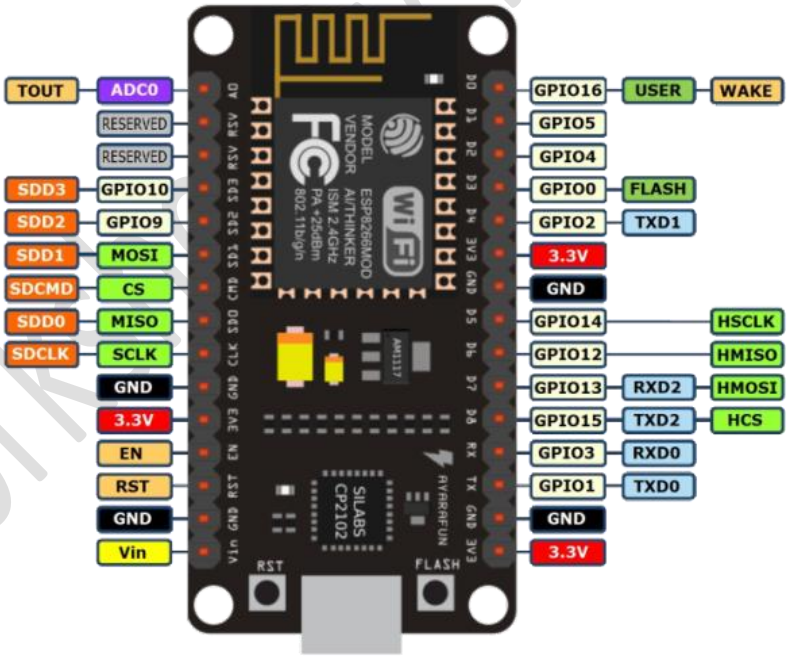

Figure: Node MCU Pin outs.

I hear and I forget. I see and I remember. I do and I understand.

-Confucius

suryasg847@gmail.com

List of Key words

Keywords	Syntax	Example	Description

Data types
int
float
String
boolean
char
byte
array
	int variable_name;
	float variable_name;
	string variable_name;
	booleanvariable_name;
	charvariable_name;
	bytevariable_name;
	arrayvariable_name[size];
		int m; // int m=8
		float p; // float p=5.035
		string n; // n="college"
		boolean b;// b=1 or 0
		char p; //p='x'
		byte r;
		int array a[10];// 'a' can store 10 elements from 0 to 9
			To define integer numbers. (16 bits in size)
			to define floating point numbers(32 bits)
			to define array of characters
			defines simple logical true/false(high or low)
			To define characters (signed or unsigned)
			unsigned number from 0-255
			collection of identical elements of same type

pinMode()	**pinMode**(Var1,OUTPUT/INPUT)	**pinmode**(ledpin ,OUTPUT); **pinmode**(13, INPUT);	Configures the specified pin to behave either as an input or output
digitalRead()	**digitalRead**(pin);	**int** v1=12; Val= **digitalRead**(v1);	Reads the value from a specified digital pin , either HIGH or LOW
digitalWrite()	**digitalWrite**(pin, value);	**digitalWrite**(13, HIGH);	Write HIGH or LOW value to digital pin
analogRead()	**analogRead**(pin);	**int** m=**analogRead**(A0); **int** n=**analogRead**(A1);	Reads the analog value from a specified analog pin(A0-A5)
analogWrite()	**analogWrite**(pin);	**int**m=**analogWrite**(A0); **int** n=**analogWrite**(A1);	Write the analog value from a specified analog pin(A0-A5)

delay()	**delay**(value);
	millis(value);
	micros(value);
		delay(1000); //one sec dealy
		millis(1000); //one milli sec dealy
		micros(1000); //one micro sec dealy
			Pauses the program for the amount of time specified

Map()	**map(**variable, alow, ahigh,mlow ,mhigh**);**	**y=map(**x,0,1023,1,200**);**	Re-maps a number from one range to another
Serial.begin()	**Serial.begin**(baud rate)	**Serial.begin**(9600)	Sets the data rate in bits per second (baud) for serial data transmission.(9600,14400 etc)

Serial.print()	**Serial.print(**data**)**	**Serial.print(**78**) //gives 78**
		Serial.print("college"**) //prints College**
		Intv**=3; Serial.print(**v**) // prints 3**
			Prints data to the serial port as human readable ASCII text

Serial.println()	**Serial.println()**	**Serial.println(**56**);**	Prints the value 56 fallowed by a carriage return character(ASCII 13 or '\n') i.e new line character
Serial.read()	**Variable=Serial.read(**variable**)**	**v=Serial.read(**p**);** **m=Serial.read(**A0**);**	Reads incoming serial data
Serial.available()	**Serial.available()**	**if (Serial.available()** > 0**)** **{** **int**incomingByte **=** **Serial.read();** **}**	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.write()	**Serial.write(val)**	**Serial.write(45);** **int**bytesent=**Serial.write("HELLO");**	Writes the binary data to the serial port And it will return the number of bytes written on serial monitor
Serial.println(val, format)	**Serial.println(val, format)**	**Serial.print(**78, BIN**)** **Serial.print(**78, OCT**)** **Serial.print(**78, DEC**)** **Serial.print(**78, HEX**)**	gives "1001110" gives "116" gives "78" gives "4E
Arrays	Data_typearray_name[size];	**int** mypins[]**=**{2,4,8}; **int** myval[4]**=**{2,4,-8,3}**;**	An array defines the pin numbers of 2,4,8 which acts as input or output An array initializes four values of type **int**
lcd.begin()	**lcd.begin(**cols,rows**);**	**lcd.begin(16, 2);**//defines for 16,2 LCD display	Initializes the interface to the lcd screen **Note**: use #include <LiquidCrystal.h> to interface lcd with arduino
lcd.print()	**lcd.print(**data**)**	**lcd.print(**"Hello world"**)**	Prints text to LCD
lcd.cursor()	**lcd.cursor(**col,row**)**	**lcd.cursor(5,1**)//points to 5th coloumn,2nd row	an underscore (line) at the position to which the next character will be written

suryasg847@gmail.com

Keywords	Syntax	Example	Description
IPAddress()	**IPAddress**(address)	**IPAddress**ip(192, 168, 0, 2);	Defines an IP address. It can be used to declare both local and remote addresses.
WiFi.config()	**WiFi.config**(ip); **WiFi.config**(ip, dns); **WiFi.config**(ip, dns, gateway); **WiFi.config**(ip, dns, gateway, subnet);	**IPAddress**ip(192, 168, 0, 2); **WiFi.config**(ip);	WiFi.config() allows you to configure a static IP address as well as change the DNS, gateway, and subnet addresses on the WiFi shield. **DNS**: The Domain Name System (DNS) is a hierarchical decentralized naming system for computers, services, or other resources connected to the Internet or a private network. **Gateway:** A gateway is a network node connecting two networks that use different protocols. **Subnet:** A subnetwork or subnet is a logical subdivision of an IP network.
WiFi.begin()	**WiFi.begin**(); **WiFi.begin**(ssid); **WiFi.begin**(ssid, pass); **WiFi.begin**(ssid, keyIndex, key);	**WiFi.begin**(); **WiFi.begin**(ssid); **WiFi.begin**(ssid, pass); **WiFi.begin**(ssid, keyIndex, key);	Initializes the WiFi library's network settings and provides the current status. **Ssid**: Wifi name(Wifi Name) **Pass**: password of the ssid(wifi router Pass)
WiFiServer()	**Server**(port); port: the port to listen on (int)	**WiFiServer** server(80);	Creates a server that listens for incoming connections on the specified port.
WiFi.status()	**WiFi.status**();	**WiFi.status**();	Return the connection status.
server.begin();	**server.begin**()	**server.begin**();	Tells the server to begin listening for incoming connections.
WiFiClient()	**WiFiClient**()	**WiFiClient** client; **client.connect(**server, 80)	Creates a client that can connect to to a specified internet IP address and port as defined in client.connect().
server.available()	**server.available**()	**WiFiClient** client = **server.available**();	Gets a client that is connected to the server and has data available for reading. The connection persists when the returned client object goes out of scope;
client.connect()	**client.connect**() **client.connect**(ip, port) **client.connect**(URL, port)	**client.connect**(server, 80)	Connects to a specified IP address and port. The return value indicates success or failure. Also supports DNS lookups when using a domain name. **ip**: the IP address that the client will connect to (array of 4 bytes) **URL**: the domain name the client will connect to (string, ex.:"arduino.cc") **port:** the port that the client will connect to (int)
client.read()	**client.read**()	**client.read**()	Read the next byte received from the server the client is connected to
client.println()	**client.println**() **client.println**(data) **client.print**(data, BASE)	**client.println**("Hello World")	Print data, followed by a carriage return and newline, to the server a client is connected to. Prints numbers as a sequence of digits, each an ASCII character
WiFi.SSID()	**WiFi.SSID**(); **WiFi.SSID**(wifiAccessPoint)	char ssid[] = "yourNetwork"; **WiFi.begin**(ssid);	Gets the SSID of the current network **wifiAccessPoint**: specifies from which network to get the information
WiFi.localIP()	**WiFi.localIP**();	**IPAddress**ip; ip = **WiFi.localIP**(); **Serial.println**(ip);	Gets the WiFi shield's IP address
WiFi.RSSI()	**WiFi.RSSI**(); **WiFi.RSSI**(wifiAccessPoint);	long rssi = **WiFi.RSSI**(); **Serial.print**("RSSI:"); **Serial.println**(rssi);	Gets the signal strength of the connection to the router
WiFi.subnetMask()	**WiFi.subnet**();	**IPAddress** subnet;	Gets the WiFi shield's subnet mask

		subnet = WiFi.subnetMask();	
WiFi.gatewayIP()	**WiFi.gatewayIP**();	**IPAddress** gateway; gateway = **WiFi.gatewayIP**(); **Serial.print**("GATEWAY: "); **Serial.println**(gateway);	Gets the WiFi shield's gateway IP address.
WiFi.encryptionType()	**WiFi.encryptionType**(); **WiFi.encryptionType**(wifiAccessPoint);	**byte** encryption = **WiFi.encryptionType**(); **Serial.print**("Encryption Type:"); **Serial.println**(encryption,HEX);	Gets the encryption type of the current network
WiFi.scanNetworks();	**WiFi.scanNetworks**();	**byte** numSsid = **WiFi.scanNetworks**(); **Serial.print**("SSID List:"); **Serial.println**(numSsid);	Scans for available WiFi networks and returns the discovered number
WiFi.macAddress(mac);	**WiFi.macAddress**(mac);	**WiFi.macAddress**(mac); **Serial.print**("MAC: "); **Serial.print**(mac[5],**HEX**); **Serial.print**(":"); **Serial.print**(mac[4],**HEX**); **Serial.print**(":"); **Serial.print**(mac[3],**HEX**); **Serial.print**(":"); **Serial.print**(mac[2],**HEX);** **Serial.print**(":"); **Serial.print**(mac[1],**HEX);** **Serial.print**(":"); **Serial.println**(mac[0],**HEX**);	Gets the MAC Address of your WiFi shield **mac**: a 6 byte array to hold the MAC address
client.flush()	**client.flush**()	**client.flush**()	This function discards any bytes that have been written to the client but not yet read.
client.write()	**client.write**(val) **client.write**(buf, len)	Client myClient; **myClient.write**("Hi there");	Write data to the server the client is connected to. This data is sent as a byte or series of bytes. **val**: a value to send as a single byte (byte or char) **buf**: an array to send as a series of bytes (byte or char) **len**: the length of the buffer
client.stop()	**client.stop**()	**client.stop**()	Disconnect the client from the server. It closes the established connection.

MQTT return codes at a glance.

Return Code	Return Code Response
0	Connection Accepted
1	Connection Refused, unacceptable protocol version
2	Connection Refused, identifier rejected
3	Connection Refused, Server unavailable
4	Connection Refused, bad user name or password
5	Connection Refused, not authorized