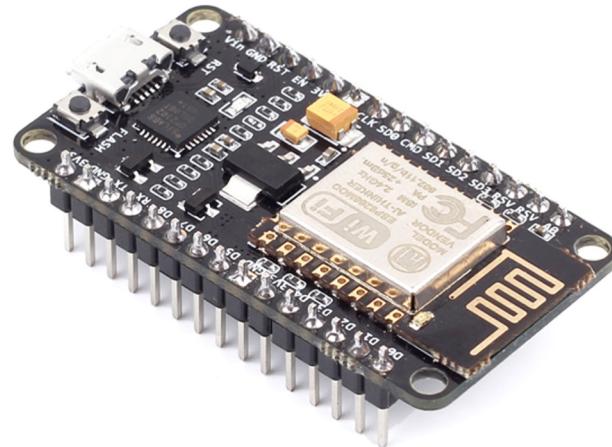
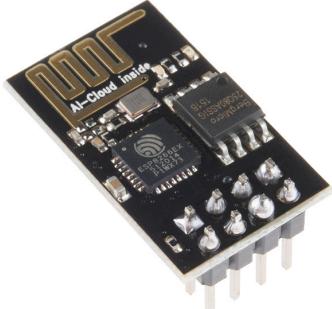


# Embedded Systems

Master 's Degree in Informatics Engineering



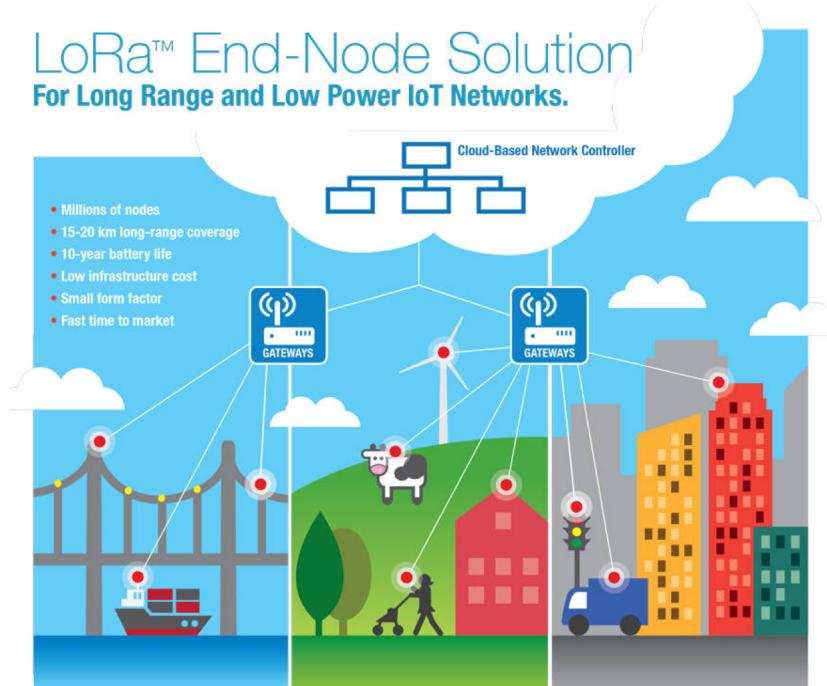
# ESP-01 & NodeMCU Platform





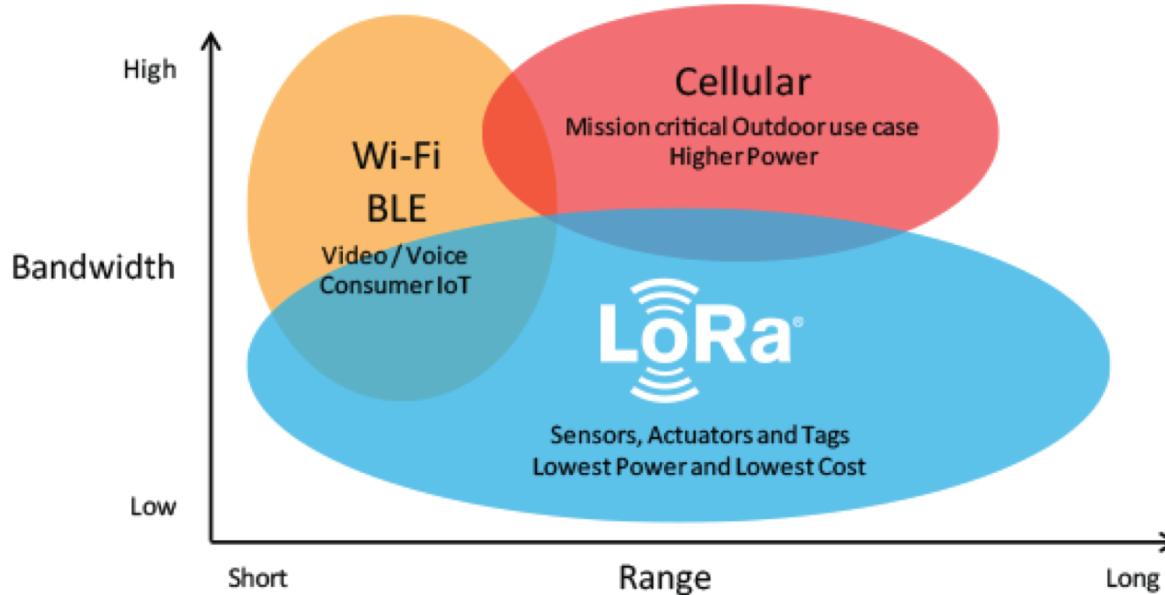
# LoRa – Long Range Radio

- LoRa (Long Range) is a low-power wide-area network (LPWAN) technology (more than 10 km in rural areas)
- The LoRa Alliance is an association created in 2015 to support LoRaWAN (~ 500 members)
- The LoRa physical layer protocol is proprietary





# LoRa – Long Range Radio



LoRa uses unlicensed frequencies that are available worldwide. These are the most widely used frequencies:

- 868 MHz for Europe
- 915 MHz for North America
- 433 MHz band for Asia



# Key Feature of LoRa



### Long Range

Connects devices up to 30 miles apart in rural areas and penetrates dense urban or deep indoor environments



### Low Power

Requires minimal energy, with prolonged battery lifetime of up to 10 years, minimizing battery replacement costs



### Secure

Features end-to-end AES128 encryption, mutual authentication, integrity protection, and confidentiality



### Standardized

Offers device interoperability and global availability of LoRaWAN networks for speedy deployment of IoT applications anywhere



### Geolocation

Enables GPS-free tracking applications, offering unique low power benefits untouched by other technologies



### Mobile

Maintains communication with devices in motion without strain on power consumption



### High Capacity

Supports millions of messages per base station, meeting the needs of public network operators serving large markets

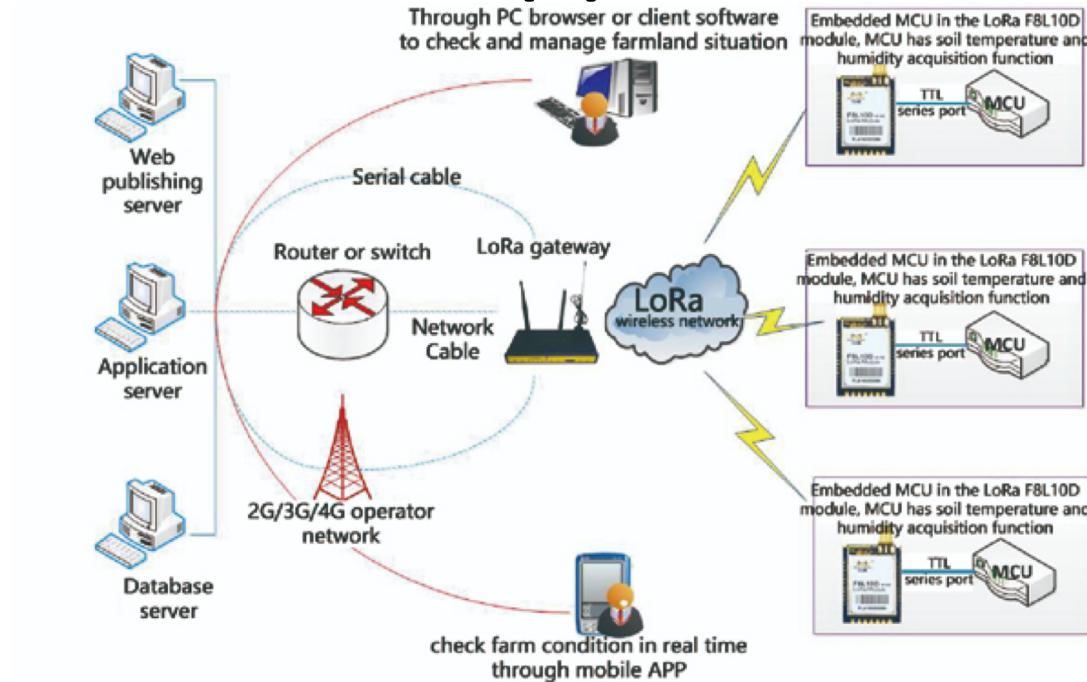


### Low Cost

Reduces infrastructure investment, battery replacement expense, and ultimately operating expenses



# LoRa – Applications

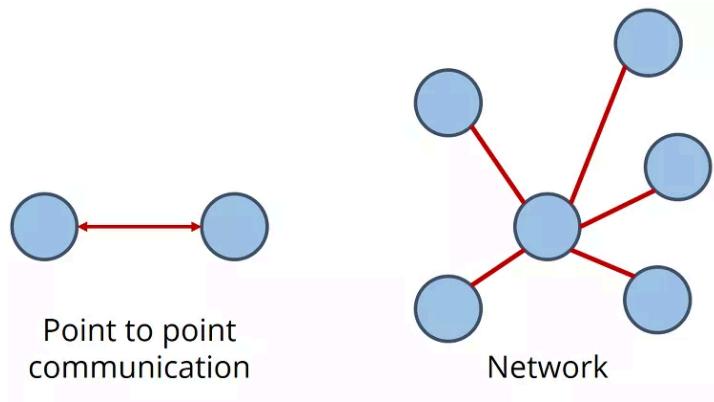


LoRa long range and low power features, makes it perfect for battery-operated sensors and low-power applications in:

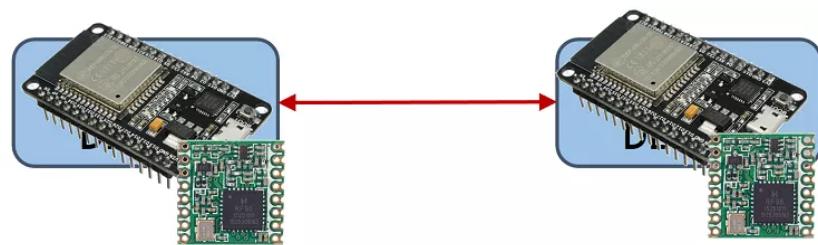
- Internet of Things (IoT)
- Smart home
- Machine-to-machine communication
- Etc.



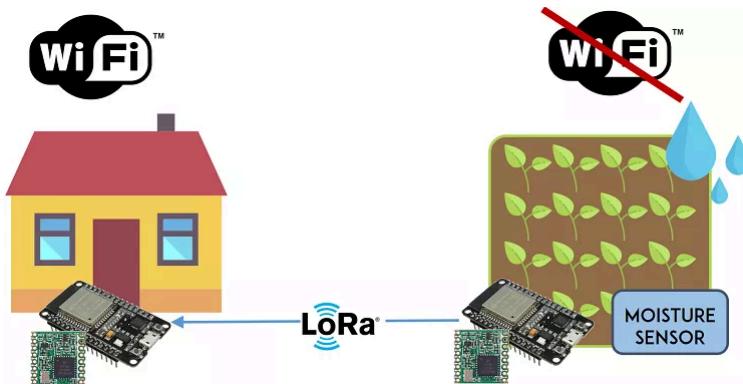
# LoRa - Topologies



**Point to Point Communication:** two LoRa enabled devices talk with each other using RF signals.



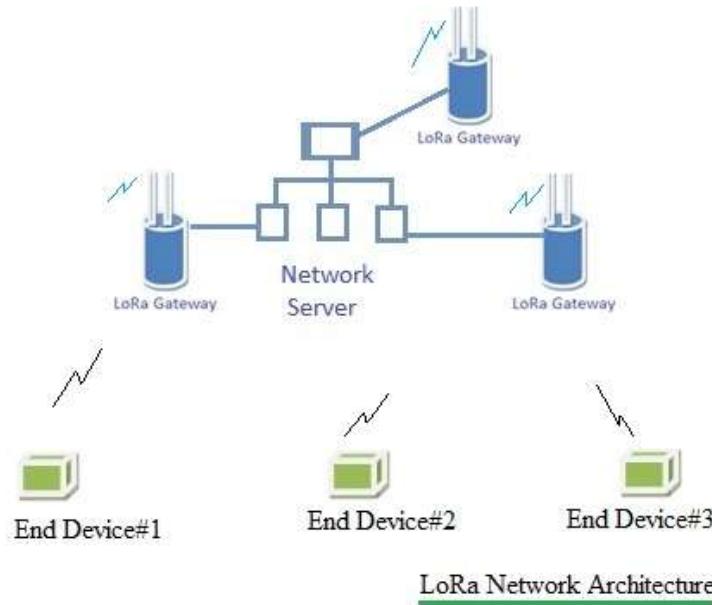
**Network or LoRa WAN:** It allows accessing to gateways





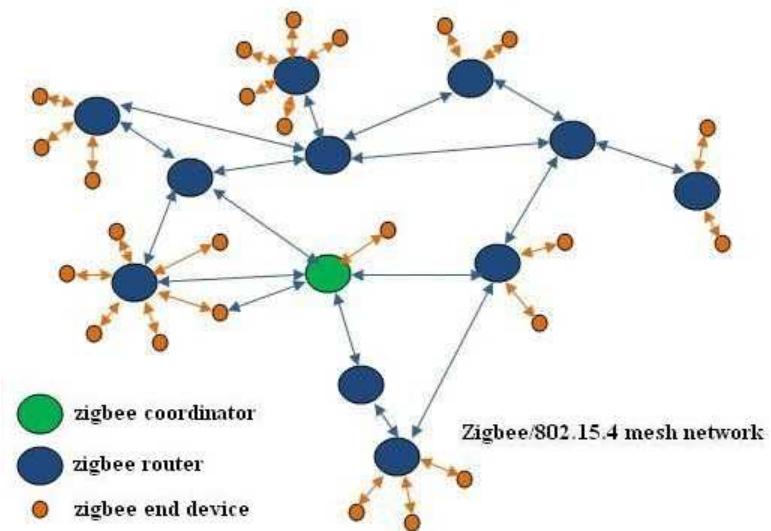
# LoRa vs ZigBee

## Network Architecture



Gateway, servers and end devices.  
Wide Area Network

- 2-5 Km (urban areas)
- 15 Km (suburban areas)



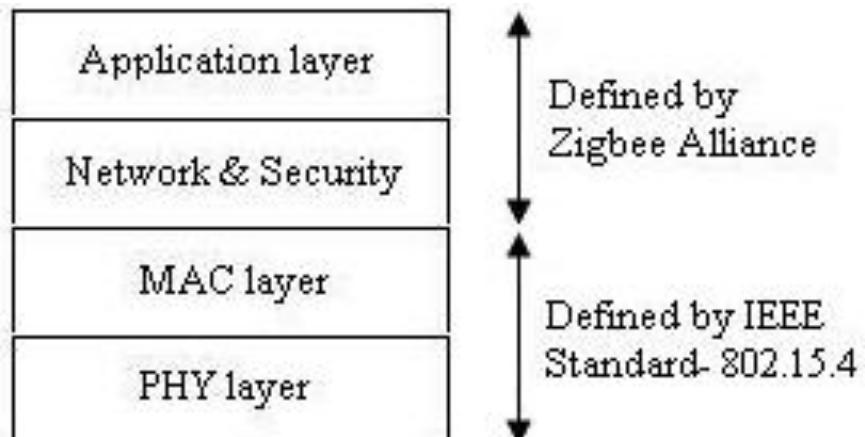
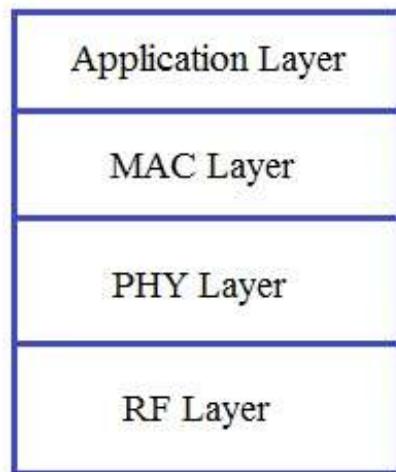
Coordinator, routers and end devices.  
Low rate wireless personal area network

- 10 to 100 meters



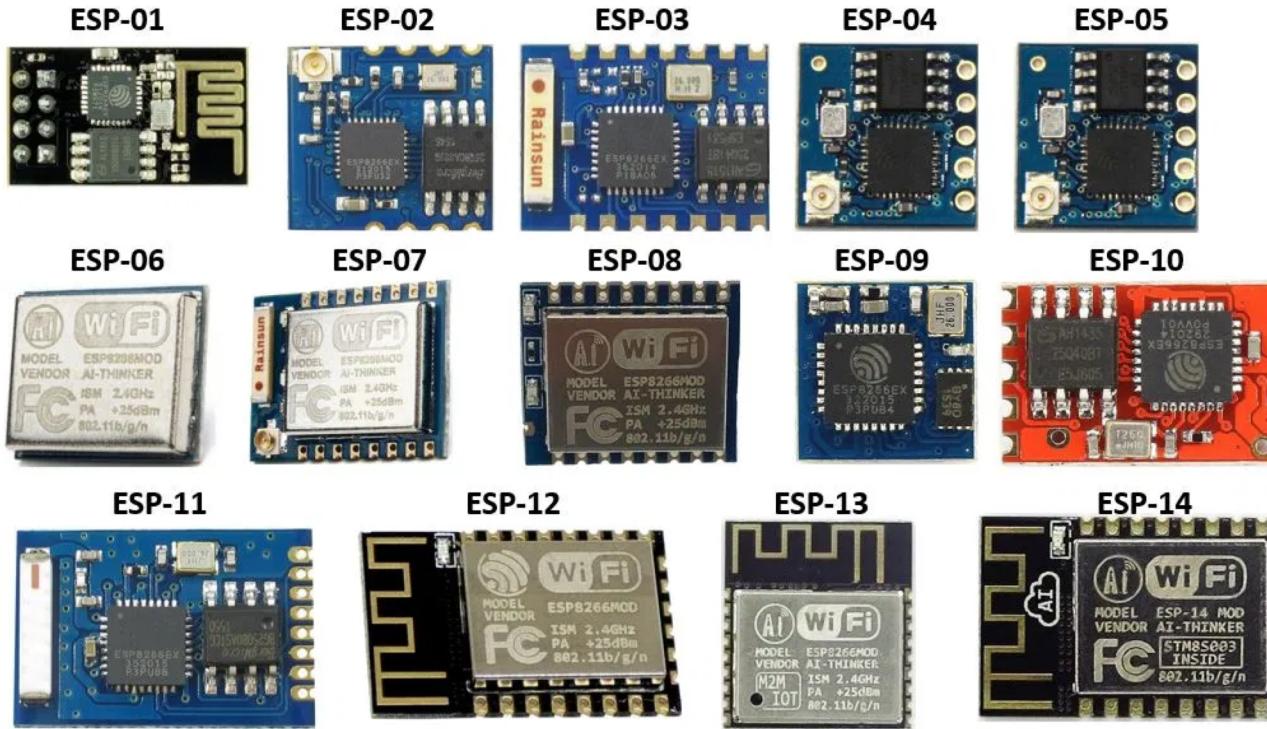
# LoRa vs ZigBee

## Protocol Stack





# ESP family



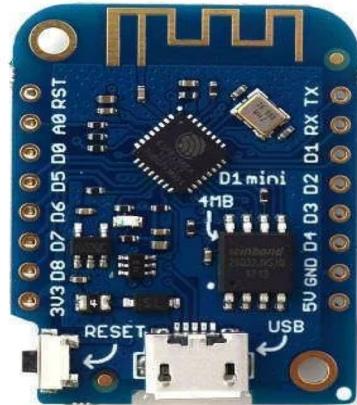
Ai-Thinker series based around ESP8266 and , referred as ESP-xx modules ranges from 01 to 14.



## Wemos D1 mini -- 2017 versions

US / UK Prices -- includes shipping to US / UK

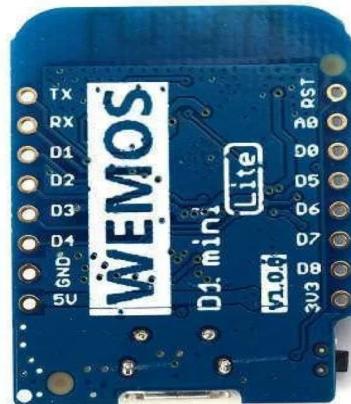
mini V3.0.0  
ESP8266 4MB  
\$4.21 / £ 3.23



mini Lite V1.0.0  
ESP8285 1MB  
\$3.71 / £ 2.84



mini Pro V1.1.0  
ESP8266 16MB  
ceramic antenna  
\$6.81 / £ 5.41





# NodeMCU family



NodeMCU v0.9 / V1



NodeMCU v1.0 / V2

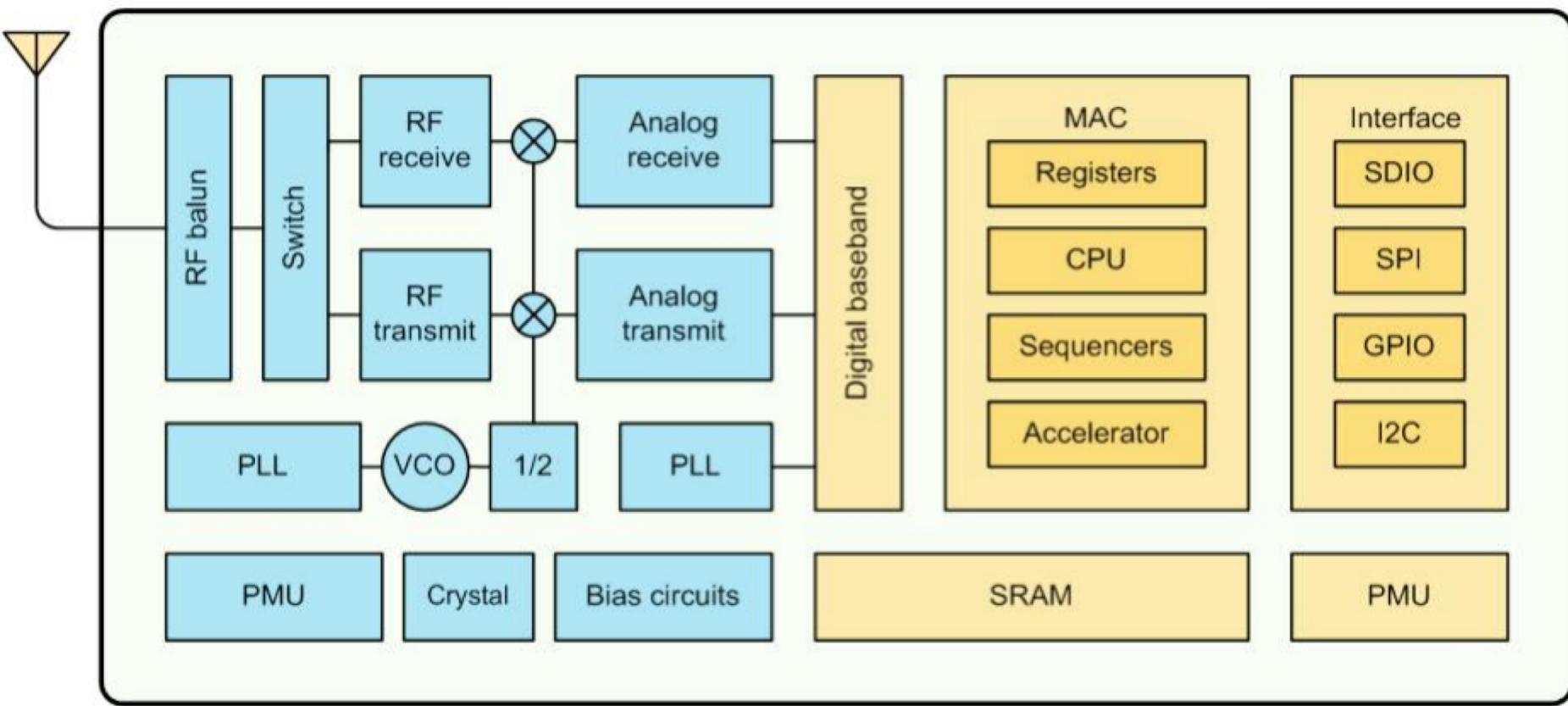


NodeMCU v1.0 / V3

NodeMCU development board has a total 30 pins in which 14 pins are active, uses ESP-12 module, onboard reset and flash button, 3.3 voltage regulator, Micro USB, USB to UART Bridge and some other components.



# ESP8266-12E Block Diagram





# ESP8266-12E pinout

**ESP8266 Modul  
ESP12E**

RESET	RESET	RES	1
ADC	ADC	ADC	2
CH PD	EN	EN	3
WAKE	GPIO16	D0	4
PWM	SCK	GPIO14	D5
PWM	MISO	GPIO12	D6
Rx0	MOSI	GPIO13	D7
VCC	VCC	VCC	8

Hilfreiche Übersicht, bei der Programmierung über die Arduino IDE

Pinnummer
NodeMCU - Pin Bezeichnung
GPIO-Port
Wichtig für Programmierung
Wichtig für Betrieb oder Startvorgang
Spezielle Funktion Arduino ESP-LIB
Interne Funktionen

LED an GPIO2 (D4)
22 D10 GPIO1 TXD0
21 D9 GPIO3 RXD0
20 D1 GPIO5 SCL
19 D2 GPIO4 SDA PWM
18 D3 GPIO0 FLASH
17 D4 GPIO2 TX1
16 D8 GPIO15 MTD0 PWM SS
15 GND GND GND

**Hinweise:**

- Bei der Programmierung muss GPIO0 (Flash) beim Bootvorgang auf LOW gezogen werden.
- GPIO15 muss beim Start immer auf LOW liegen
- GPIO2 beim Start High oder unbeschaltet
- EN / CH\_PD muss zum aktivieren es Moduls auf High liegen
- Die Arduino-IDE programmiert das Modul über den eingebauten Bootloader per TXD0 und RXD0
- Maximale Belastbarkeit pro Pin 6 mA (max. 12mA)
- Betriebsspannung 3,3V



# ESP-01

- Low cost, compact and powerful Wi-Fi Module
- Power Supply: +3.3V only
- Built-in low power 32-bit MCU @ 80MHz
- 512kB Flash Memory
- Can be used as Station or Access Point or both combined
- Supports Deep sleep (<10uA)
- Supports serial communication hence compatible with many development platform like Arduino
- Can be programmed using Arduino IDE or AT-commands or Lua Script





# What is NodeMCU?

- Open-source
- WI-FI enabled
- Based on the ESP8266-12
- Arduino-like hardware IO (**3.3V**)
- Nodejs style network API
- MicroPython/LUA programming\*
- Extremelly low cost <5€ !!



\* <https://nodemcu.readthedocs.io/en/master/> > LUA

\*\* [https://www.nodemcu.com/index\\_en.html](https://www.nodemcu.com/index_en.html)

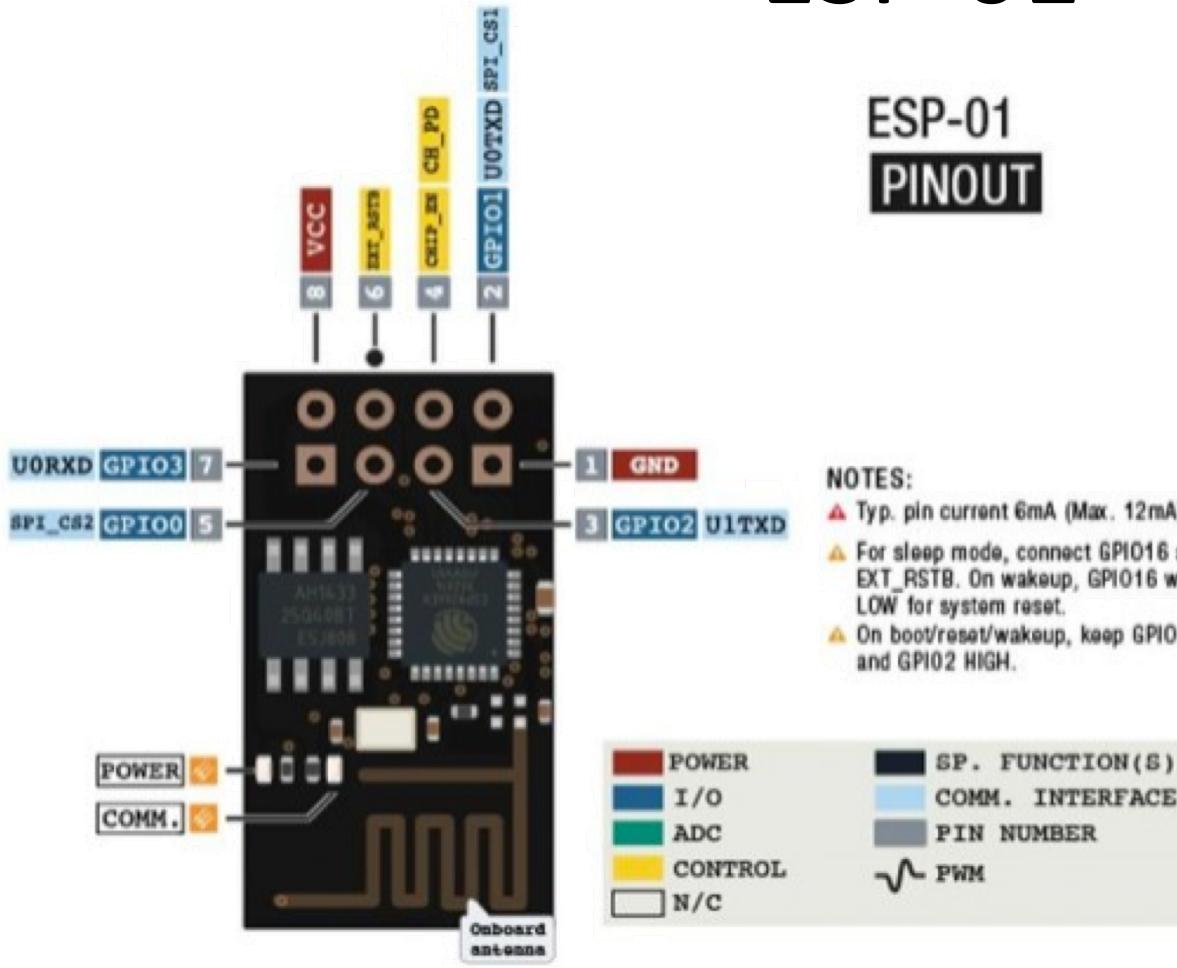


# ESP-01 vs NodeMCU?

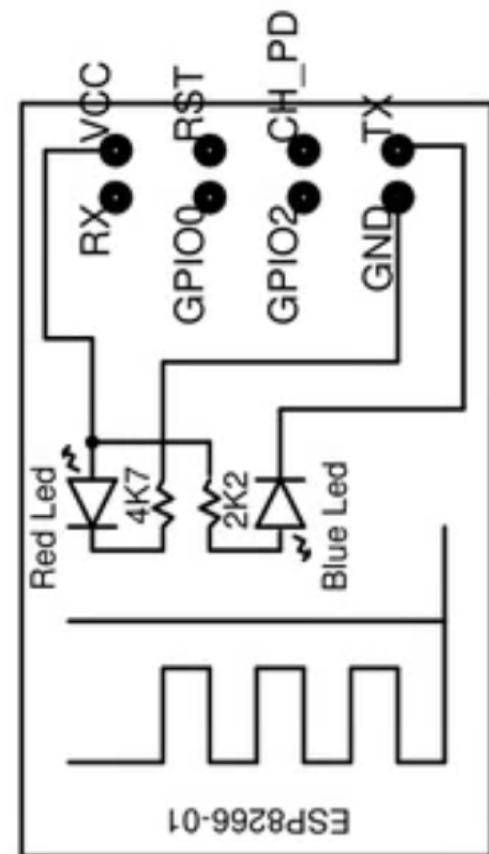
	ESP-01	NodeMCU v1.0
CPU	32-bit Tensilica Xtensa LX106	
CPU Clock	80MHz/160MHz	
Instruction SRAM	64KBytes (<36KB Station Mode)	
Data SRAM	96KBytes	
Flash Memory	1MByte	4MBytes
GPIO Pins	2	11
ADC	-	1
USB-to-Serial	-	CH340G
UART/SPI/I2C	1/1/1	
WiFi Built-In	802.11 b/g/n	



# ESP-01

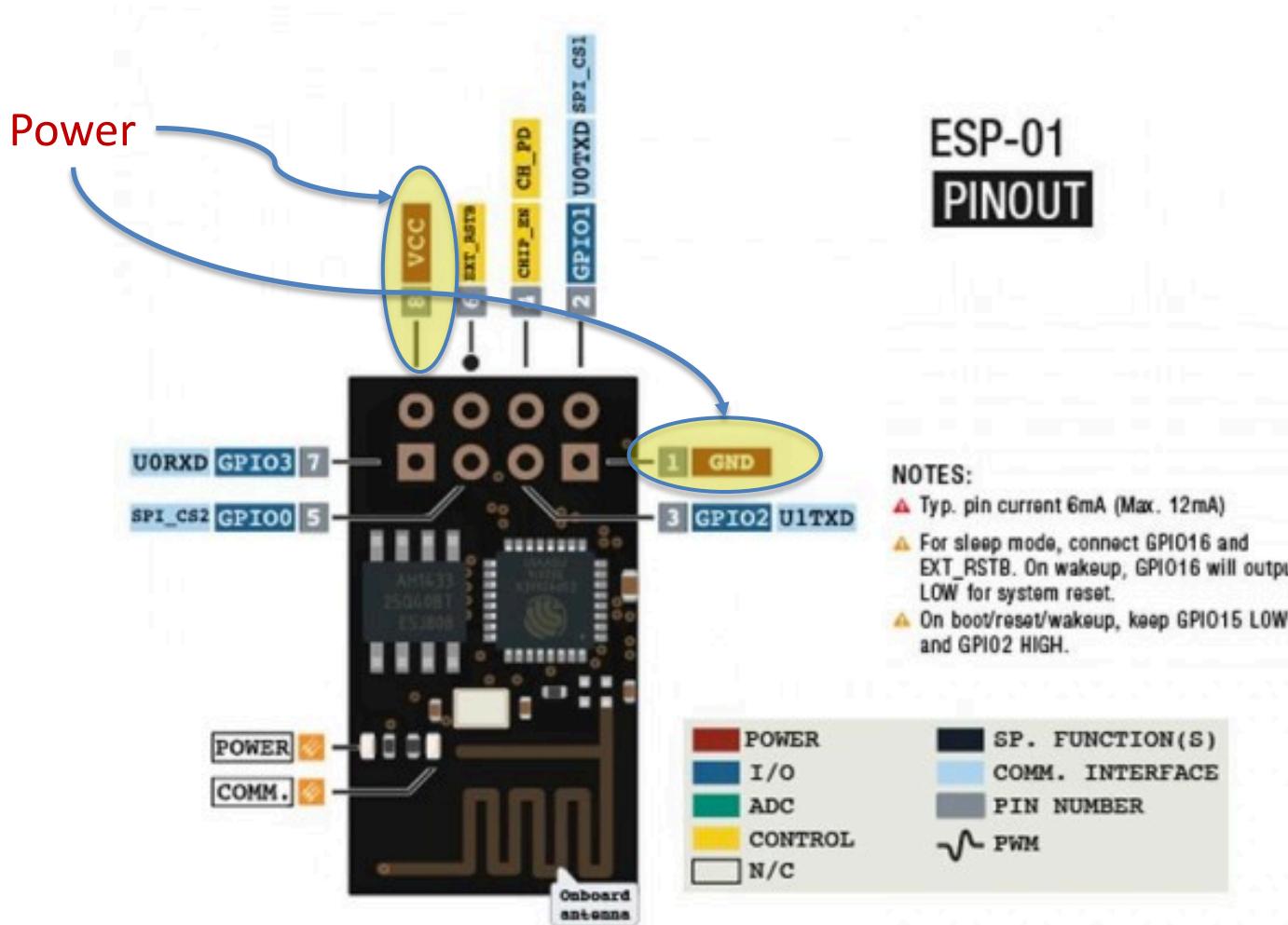


## ESP-01 PINOUT





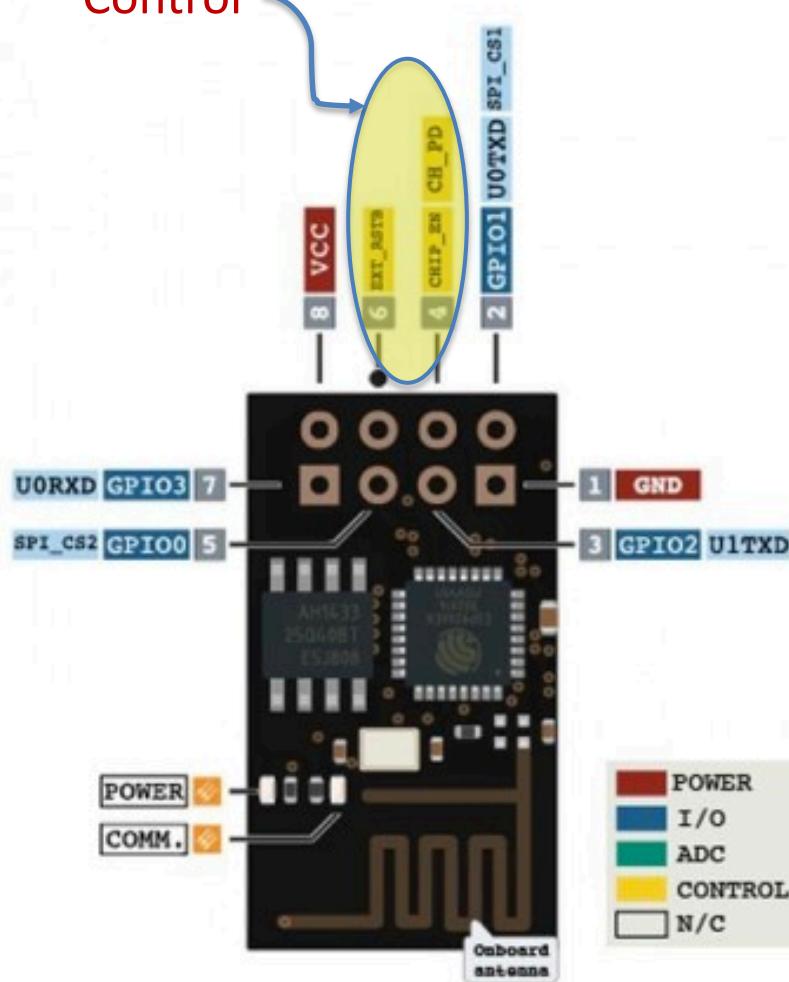
# ESP-01





# ESP-01

Control



## ESP-01 PINOUT

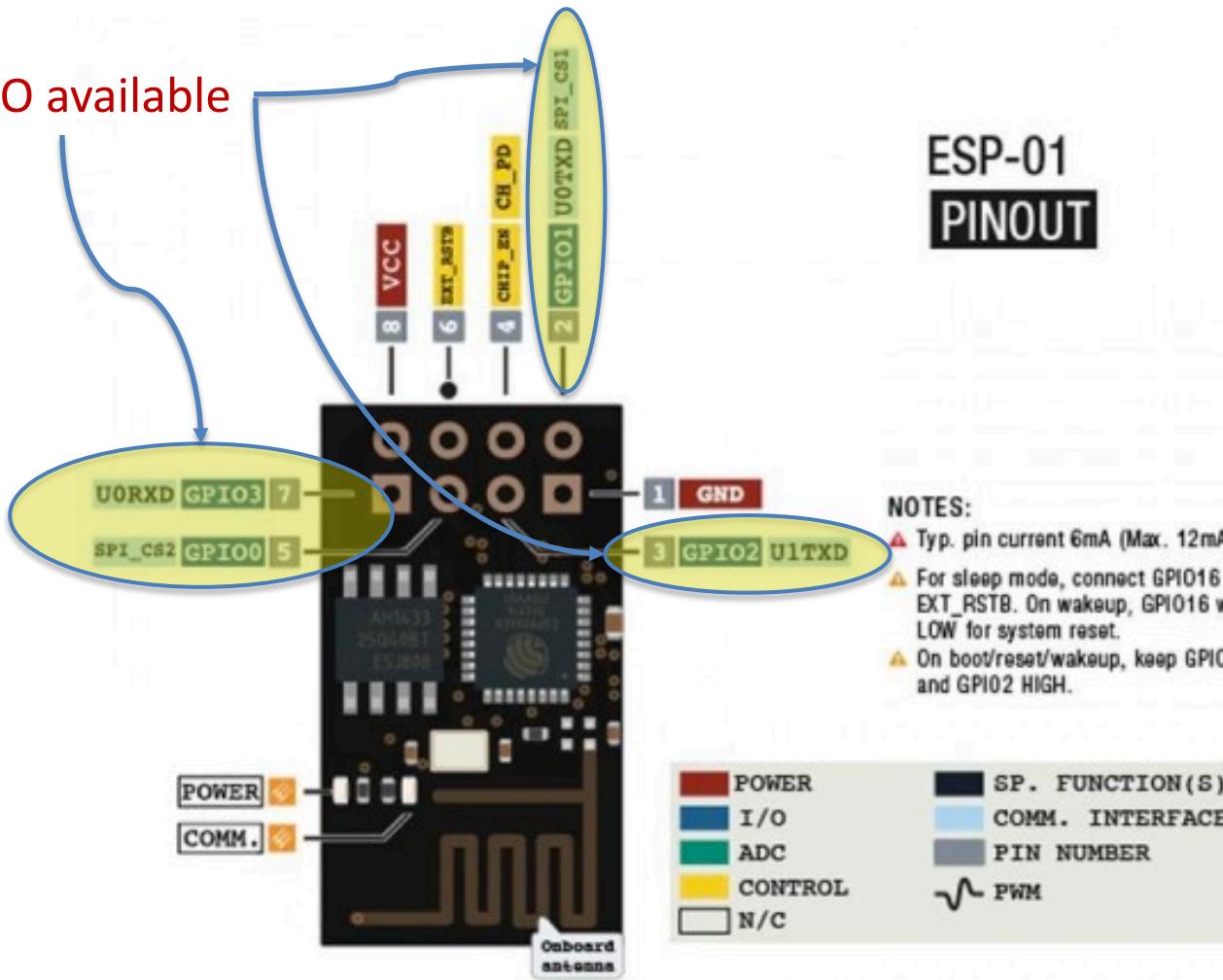
### NOTES:

- ▲ Typ. pin current 6mA (Max. 12mA)
- ▲ For sleep mode, connect GPIO16 and EXT\_RSTB. On wakeup, GPIO16 will output LOW for system reset.
- ▲ On boot/reset/wakeup, keep GPIO15 LOW and GPIO2 HIGH.



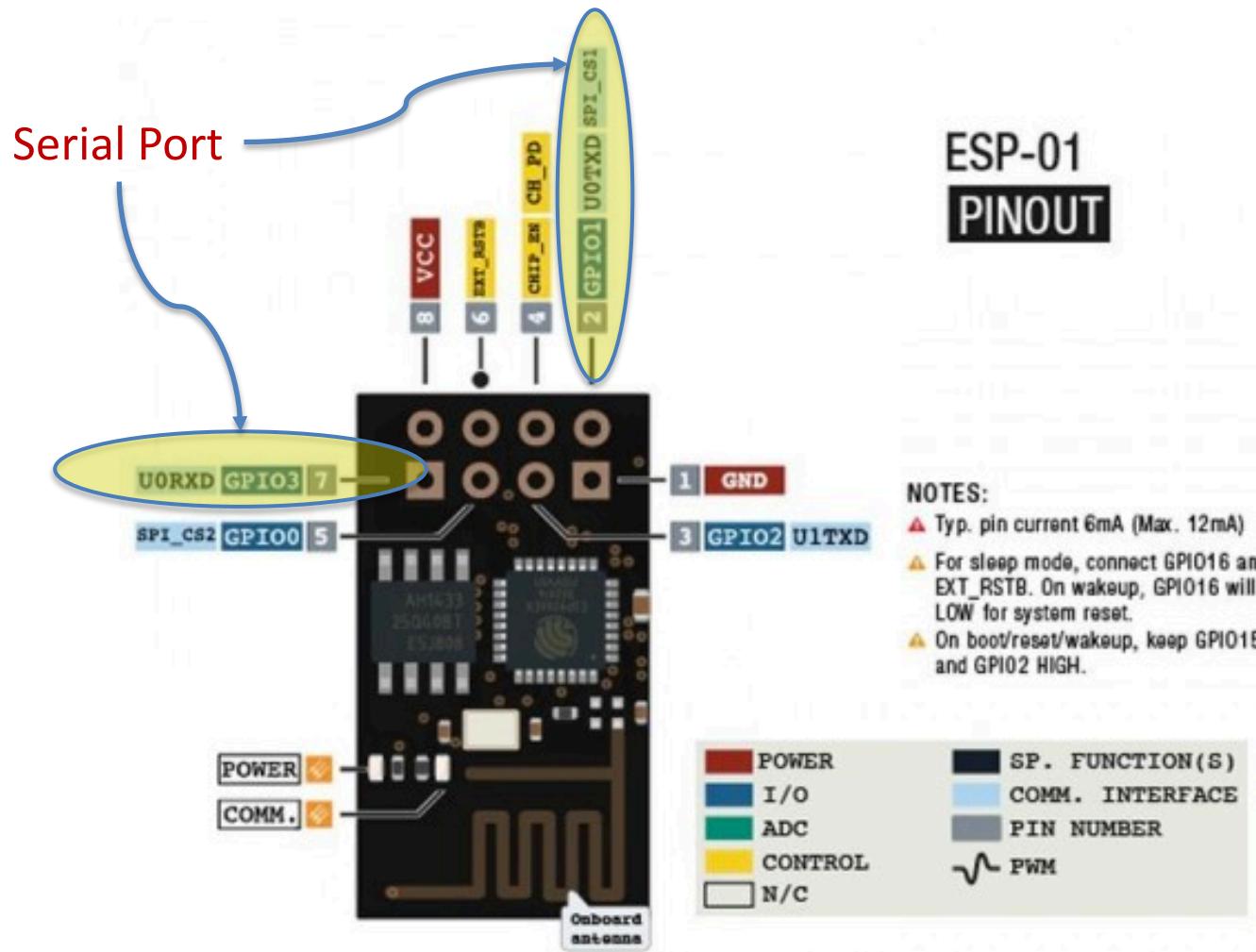
# ESP-01

4 GPIO available



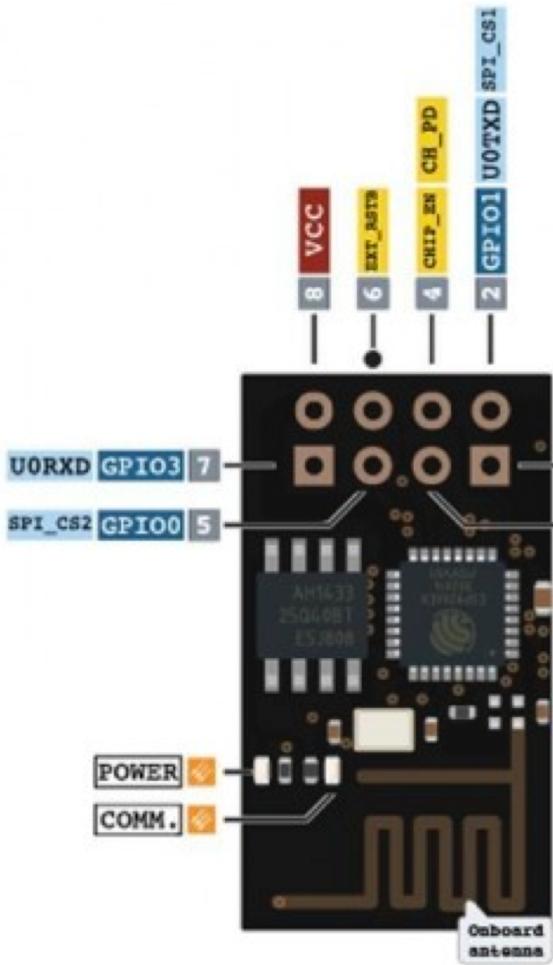


# ESP-01

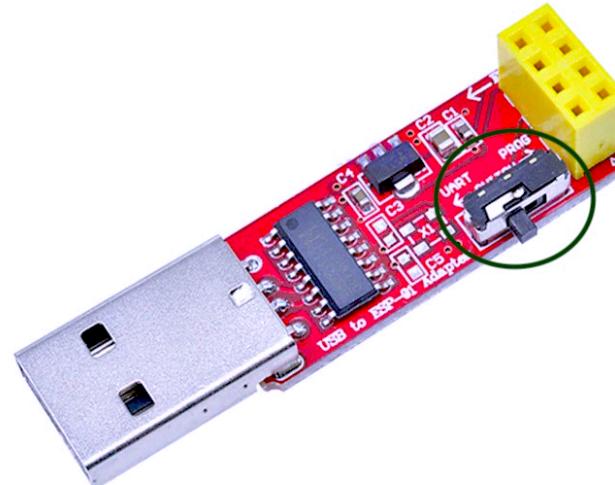
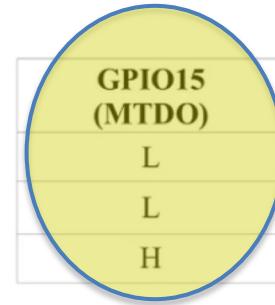




# ESP-01 – Programming module

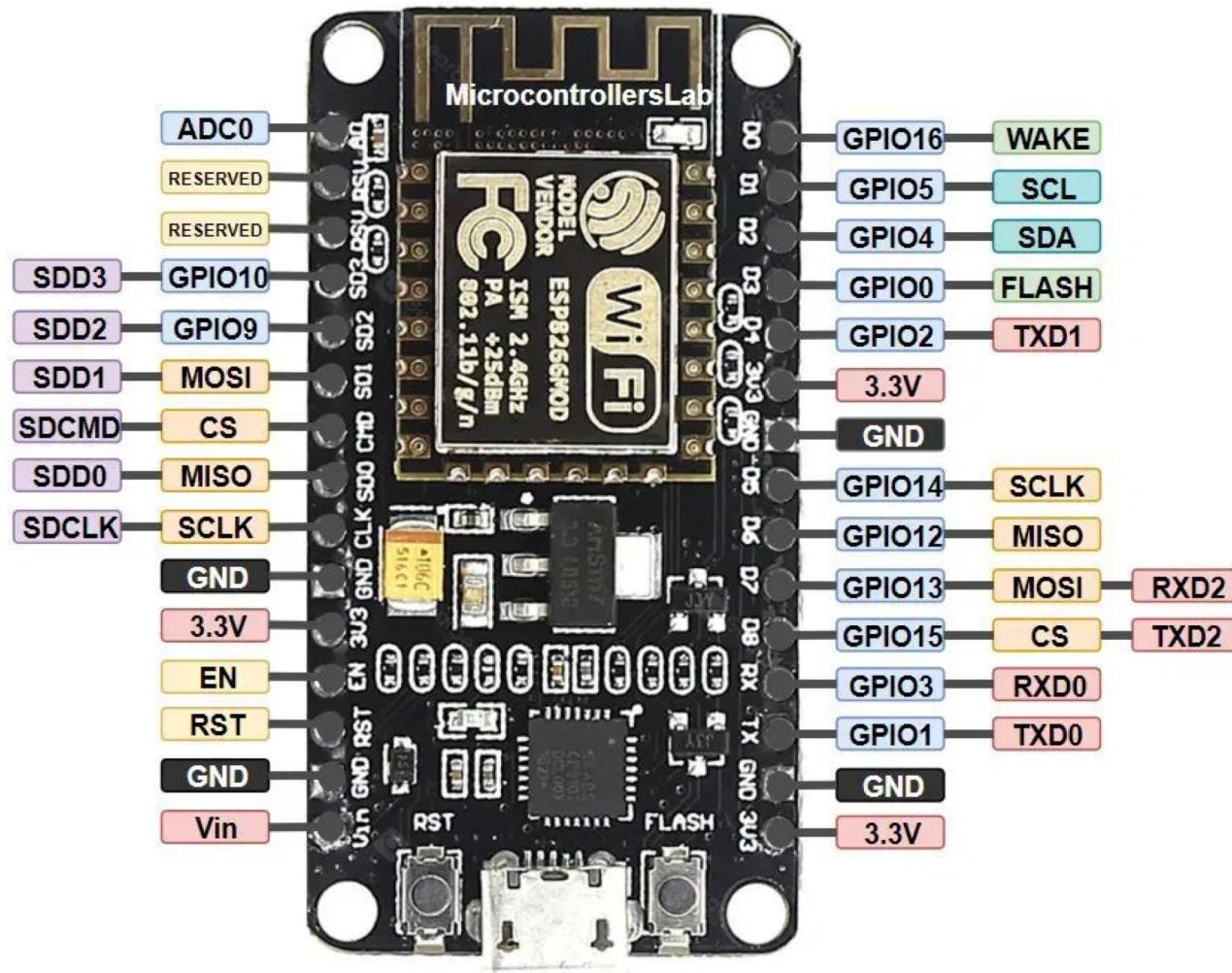


Not Accessible – ESP-01



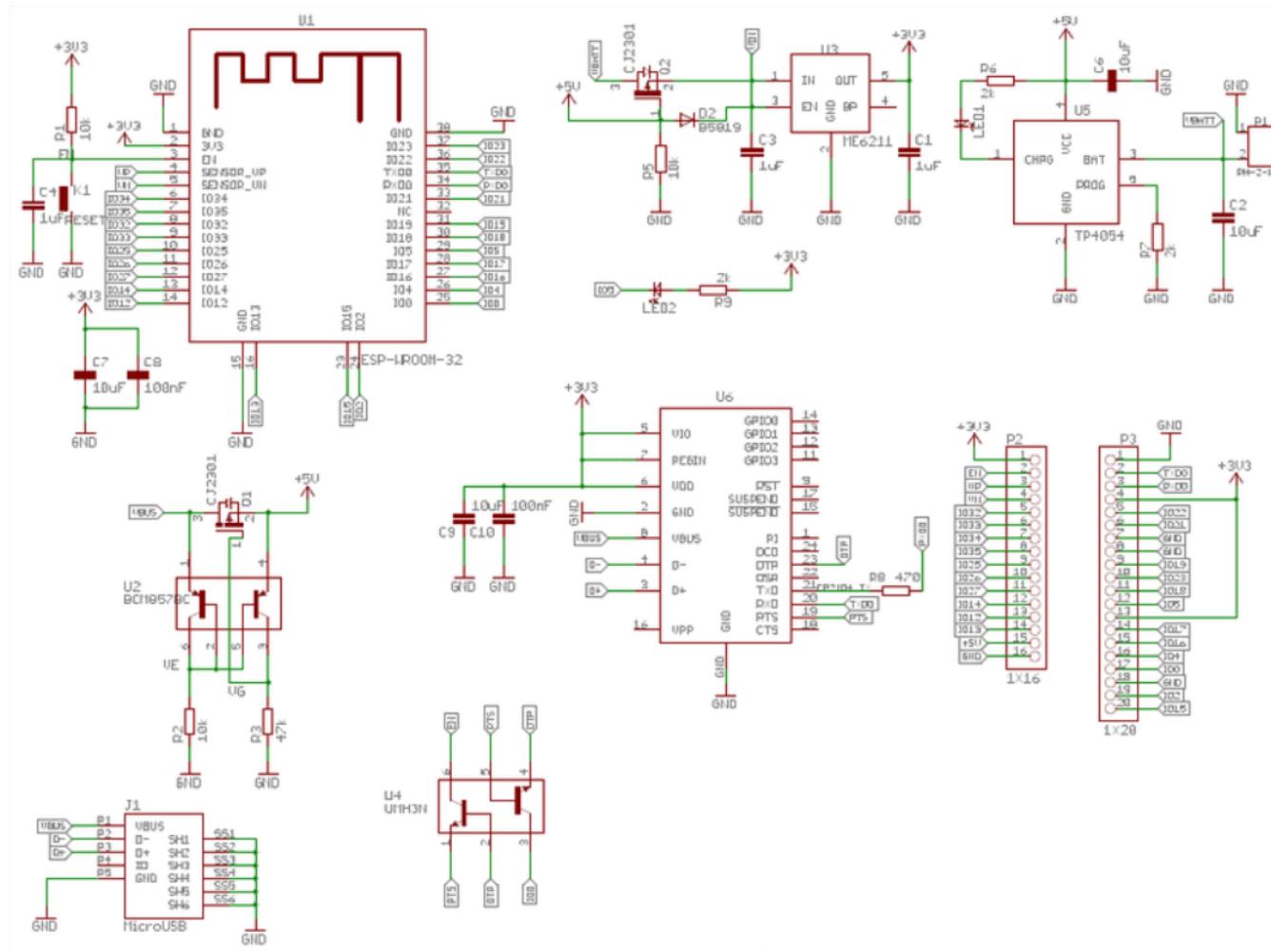


# NodeMCU 12E



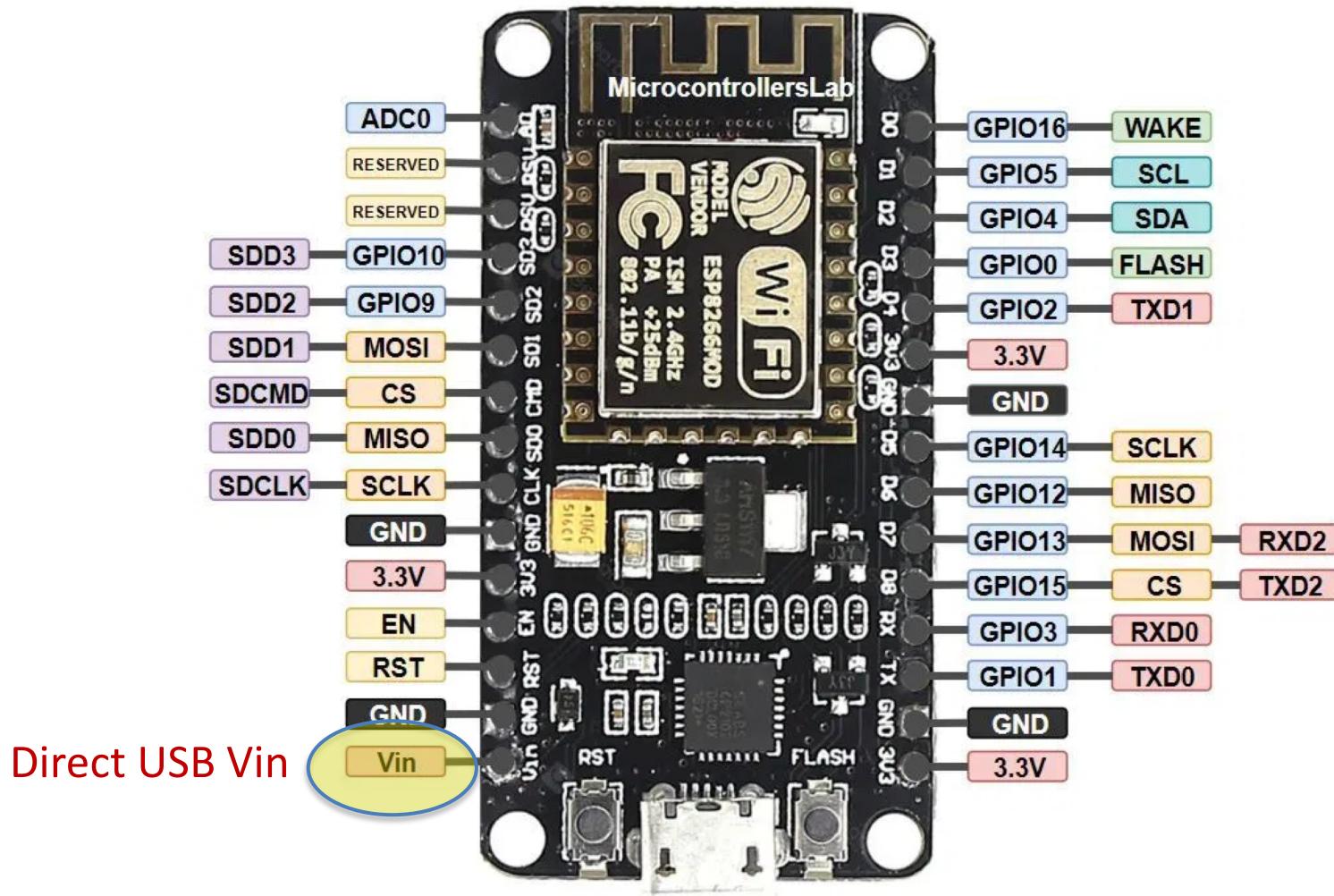


# NodeMCU 12E





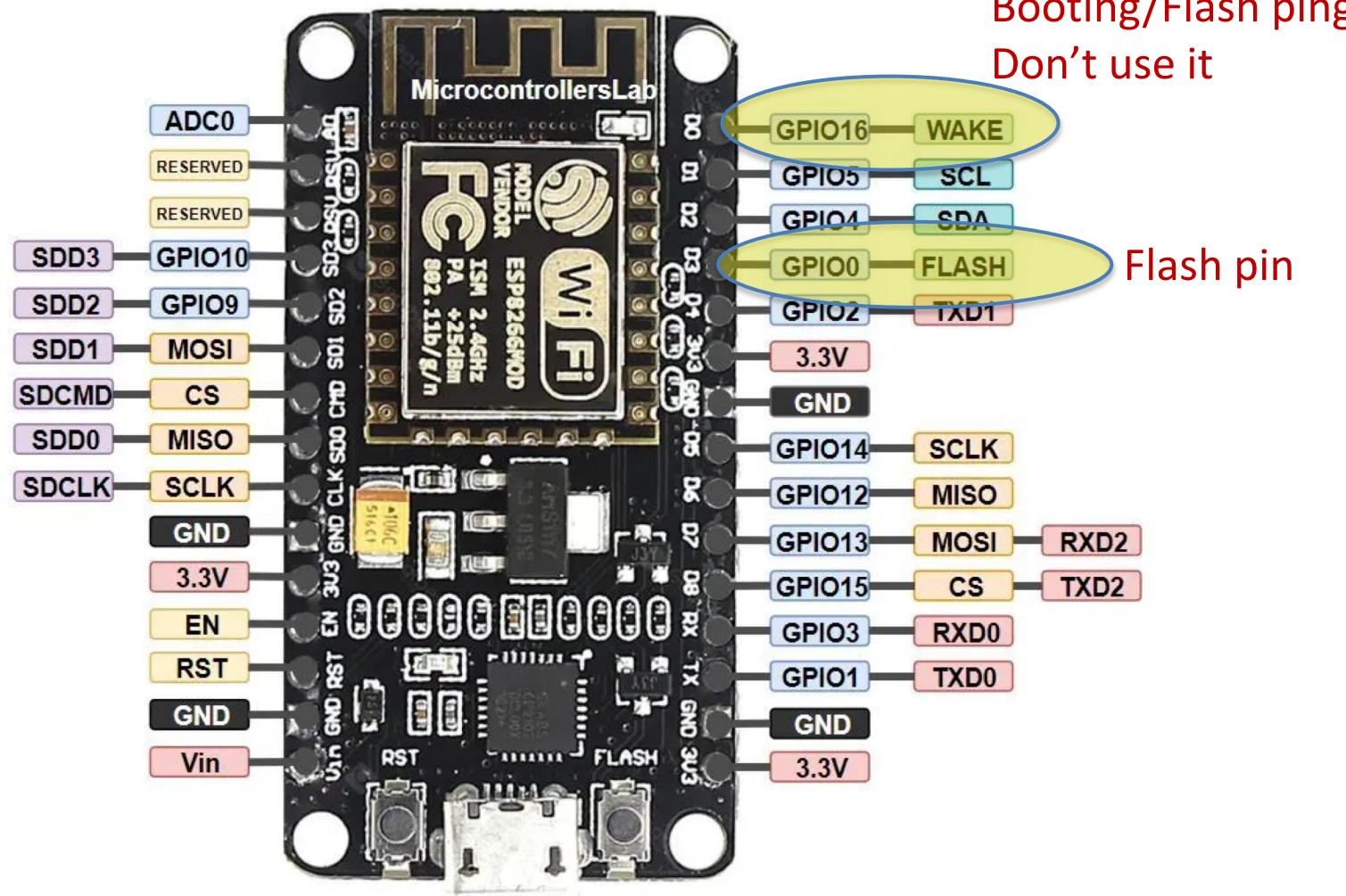
# NodeMCU 12E



Direct USB Vin

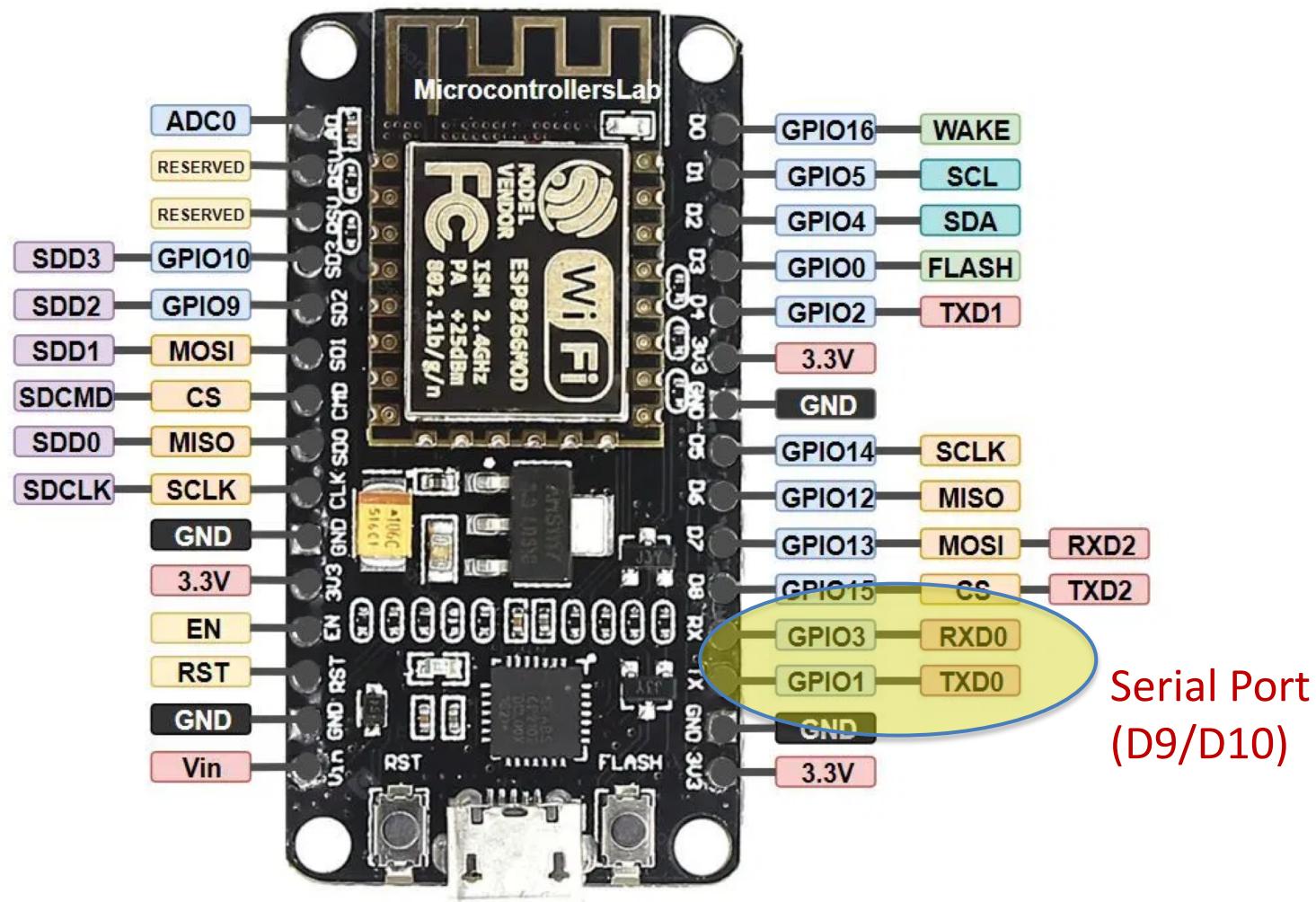


# NodeMCU 12E



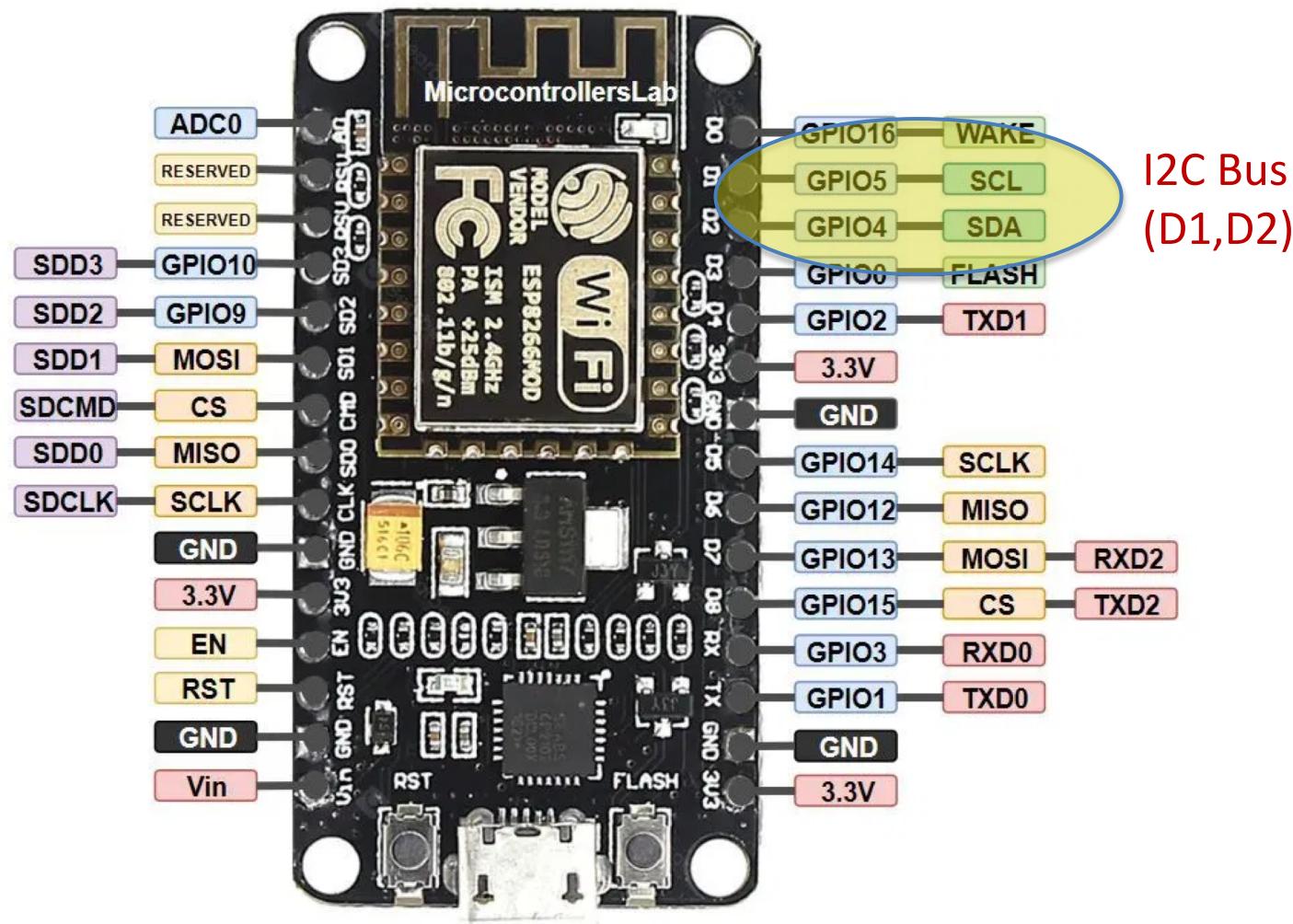


# NodeMCU 12E





# NodeMCU 12E





# Programming ESP8266

## Native Expressig SDK

[ESP8266 - SDK-Getting\\_started\\_guide\\_en.pdf](#)

### 1.5. ESP8266 Toolkit

#### 1.5.1. Compiler

Linux OS is required to compile the ESP8266 SDK. When using Windows OS, we recommend VirtualBox as the virtual machine for ESP8266. In order to simplify the compilation procedure, we have installed the compiling tools on the virtual machine. Users can directly compile the ESP8266 SDK by importing the ESP8266 compiler (OVA image) into the virtual machine. [http://downloads.espressif.com/FB/ESP8266\\_GCC.zip](http://downloads.espressif.com/FB/ESP8266_GCC.zip)

#### 1.5.2. Firmware Download Tool

The ESP8266 DOWNLOAD TOOL is the official firmware download tool developed by Espressif. Users can download multiple binaries to the SPI Flash of the ESP8266 mother board (ESP-LAUNCHER or ESP-WROOM-02) at the same time according to the actual compilation mode and flash size.

#### 1.5.3. Serial Port Debug Tool

The serial port debug tool can be used to directly communicate with the ESP8266 module over a standard RS-232 port. For PCs that do not have a physical serial port, a virtual com port (USB-to-serial converter) can be used.



# Programming ESP8266

# Native Expressig SDK

# ESP8266 - SDK-Getting\_started\_guide\_en.pdf

# Xtensa-lx106-elf Toolchain

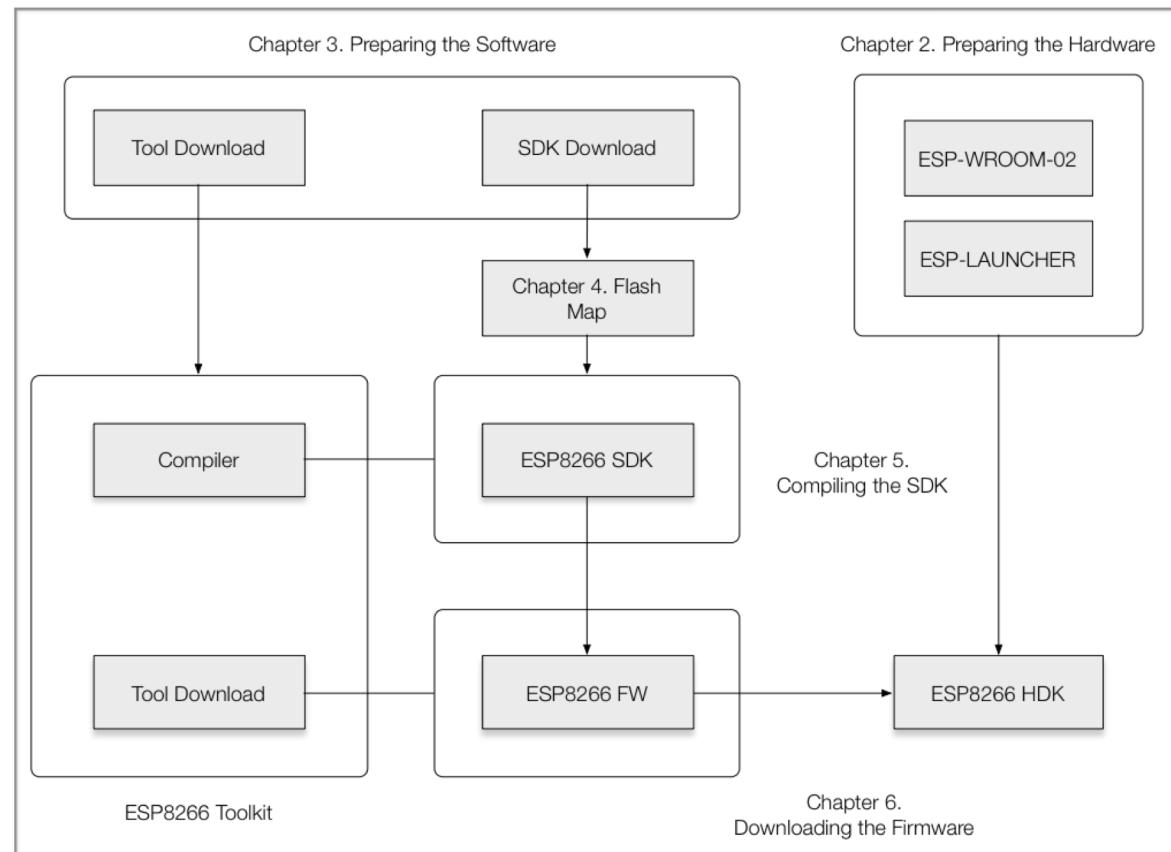
+

# ESP8266 IoT SDK

## Espressif Systems

1

# Standard GNU development tools





# Programming ESP8266

Native Expressig SDK

<https://github.com/pfalcon/esp-open-sdk>

Xtensa-lx106-elf

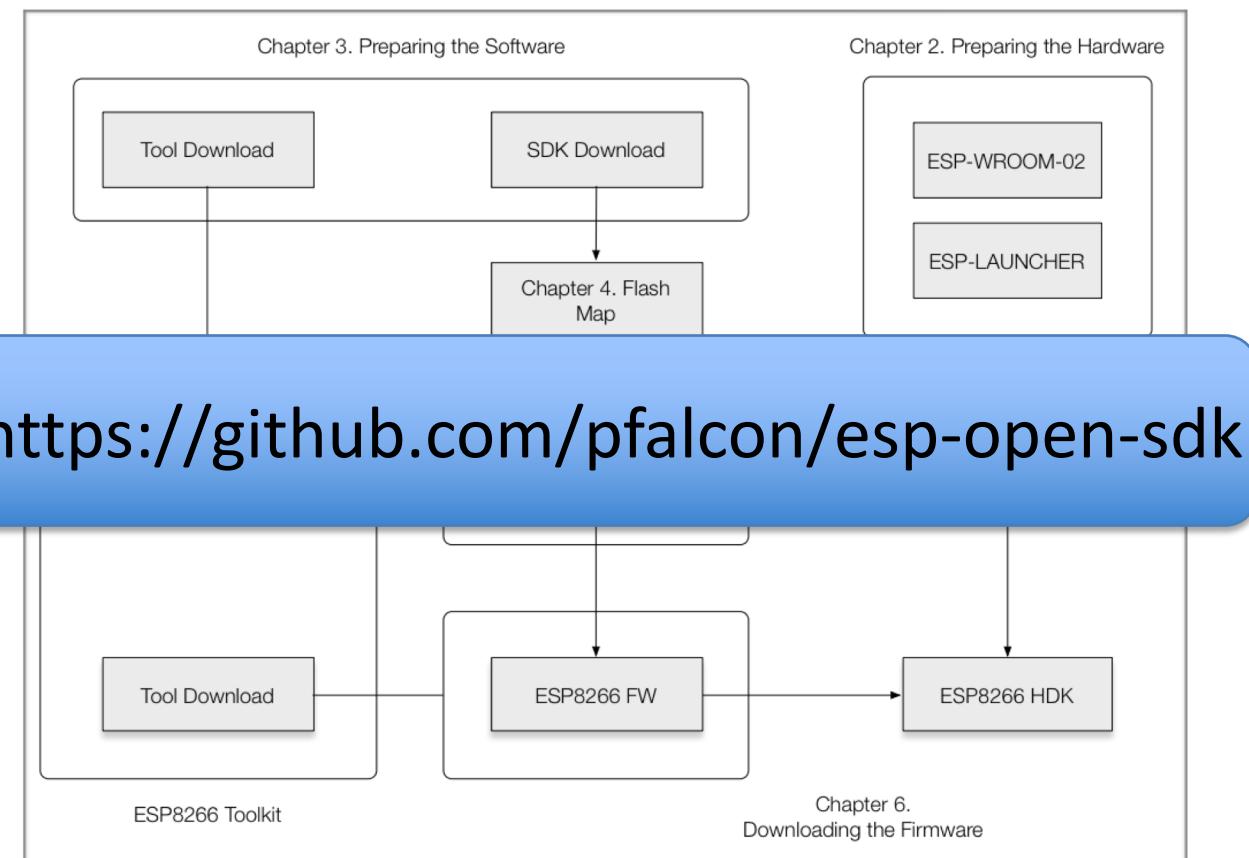
Toolchain



ESP8266 IoT SDK  
Espressif Systems



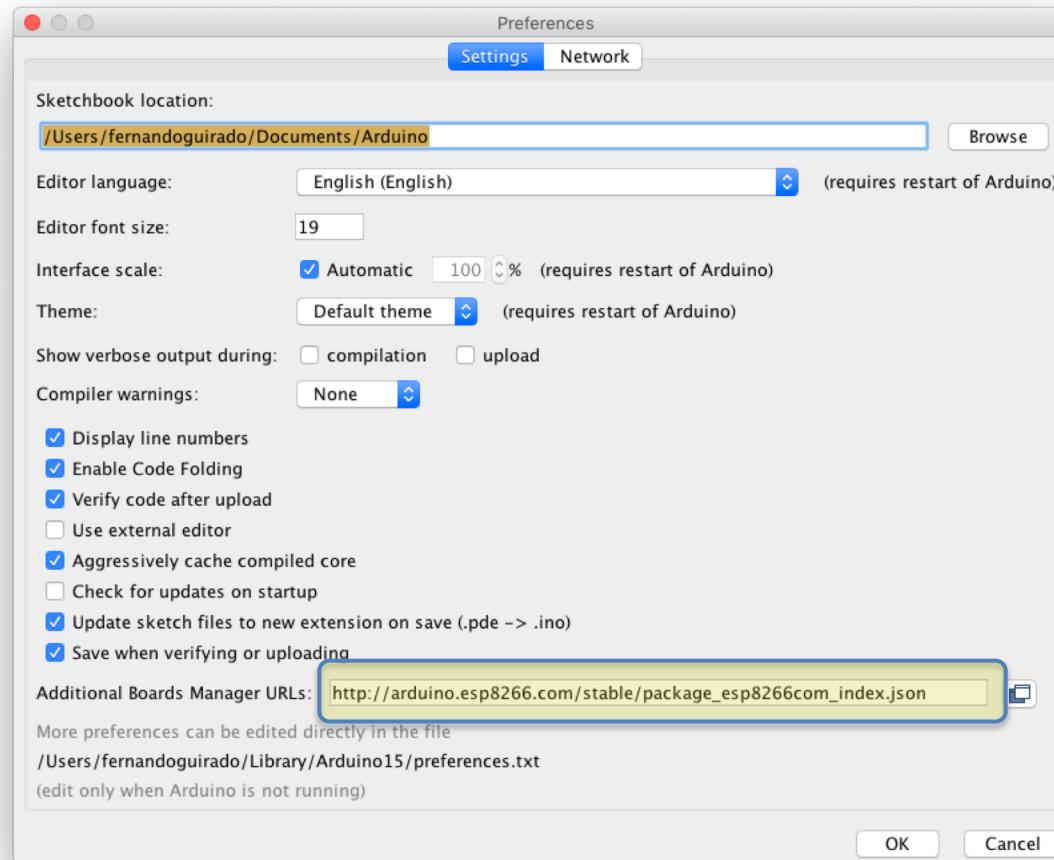
Standard GNU  
development tools





# Arduino IDE – ESP8266

## Arduino - Preferences

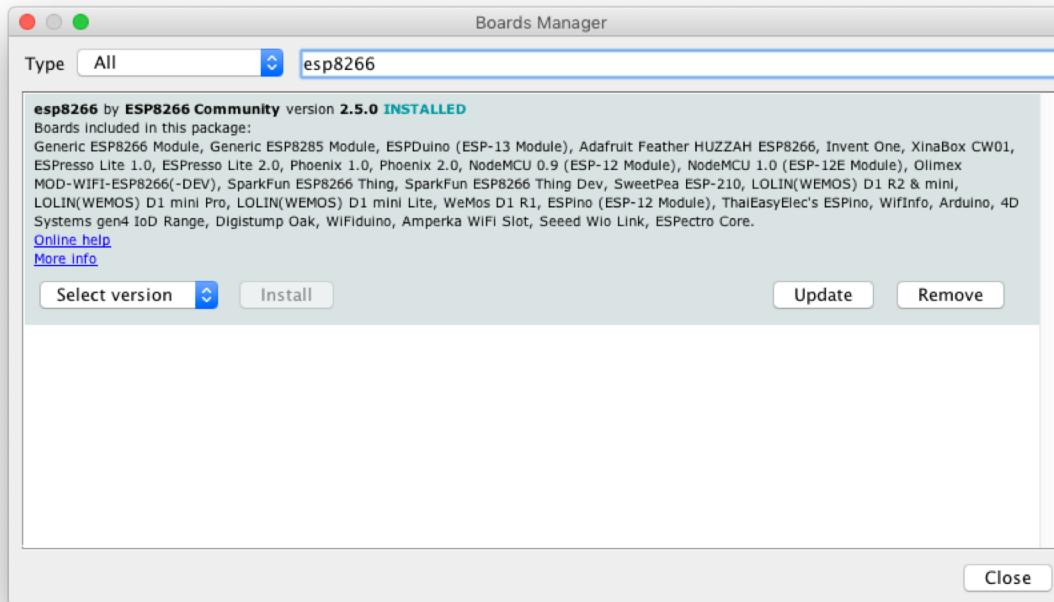


[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)



# Arduino IDE - NodeMCU

Arduino – Tools > Boards > Boards Manager

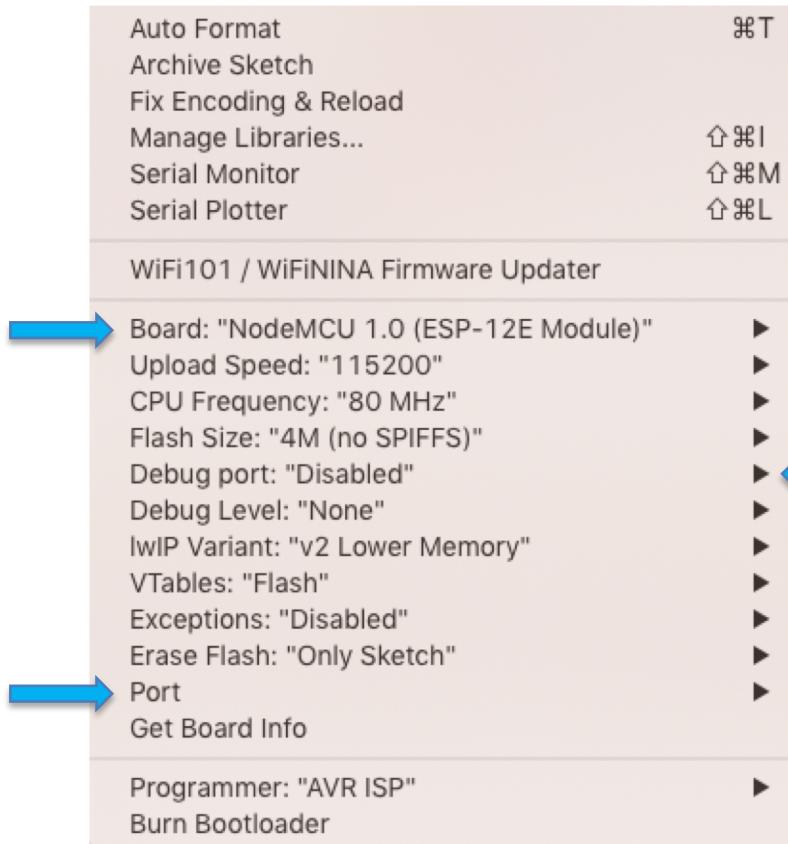


In some cases latest version has problems with older boards!



# Arduino IDE - NodeMCU

## Arduino – Select Board and Serial Port



Debugging feature is controllable over the IDE menu. The new menu points manage the real-time Debug messages.

The Serial port must be initialized by the user at the highest baudrate > `Serial.begin(115200)` and selected in the Debug Port menu

Debugging level is user selected

<https://arduino-esp8266.readthedocs.io/en/latest/>

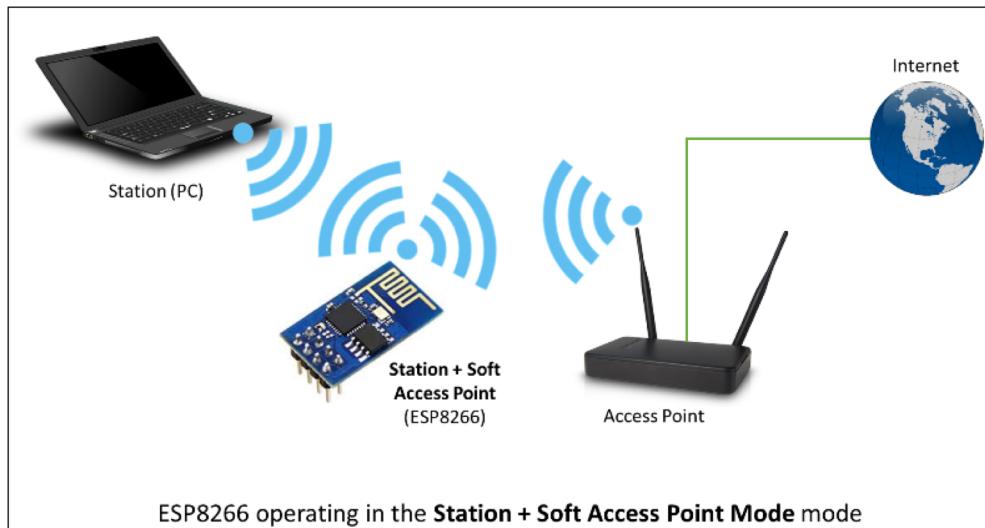


# ESP8266 - WiFi

**Station Mode (WIFI\_STA)** → It can connect to the Wi-Fi network as any other device

**Access Point (WIFI\_AP)** → It establishes its own Wi-Fi network, then any other device can connect to the ESP8266

**Both Station and Access Point (WIFI\_AP\_STA)** → It operates as both a station and a soft access point mode. This provides the possibility of building e.g. mesh networks.





# ESP8266 – WiFi Station

`WiFi.begin(ssid, password, channel, bssid, connect)` / `WiFi.begin(ssid, password)` / `WiFi.begin()`

- **ssid** - a character string containing the SSID of Access Point we would like to connect to, may have up to 32 characters
- **password** to the access point, a character string that should be minimum 8 characters long and not longer than 64 characters
- **channel** of AP, if we like to operate using specific channel, otherwise this parameter may be omitted
- **bssid** - mac address of AP, this parameter is also optional
- **connect** - a boolean parameter that if set to **false**, will instruct module just to save the other parameters without actually establishing connection to the access point

`WiFi.config(local_ip, gateway, subnet, [dns1], [dns2])`

> Disable DHCP client and set the IP configuration of station interface to user defined arbitrary values.

- **local\_ip** - enter here IP address you would like to assign the ESP station's interface
- **gateway** - should contain IP address of gateway (a router) to access external networks
- **subnet** - this is a mask that defines the range of IP addresses of the local network
- **dns1, dns2** - optional parameters that define IP addresses of Domain Name Servers (DNS) that maintain a directory of domain names (like e.g. www.google.co.uk) and translate them for us to IP addresses

> It is possible to configure multiple Access Points using the *ESP8266WiFiMulti* library.



# ESP8266 – WiFi Station

```
#include <ESP8266WiFi.h>

const char* ssid = "*****";
const char* password = "*****";

IPAddress staticIP(192,168,1,22);
IPAddress gateway(192,168,1,9);
IPAddress subnet(255,255,255,0);

void setup(void)
{
    Serial.begin(115200);
    Serial.println();

    Serial.printf("Connecting to %s\n", ssid);
    WiFi.config(staticIP, gateway, subnet);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
    Serial.println();
    Serial.print("Connected, IP address: ");
    Serial.println(WiFi.localIP());
}
```



# ESP8266 – WiFi Access Point

**WiFi.softAP(ssid) / WiFi.softAP(ssid, psk, channel, hidden, [max\_connection])**

- **ssid** - character string containing network SSID (max. 32 characters)
- **psk** - optional character string with a pre-shared key. For WPA2-PSK network it should be minimum 8 characters long and not longer than 64 characters. If not specified, the access point will be open for anybody to connect.
- **channel** - optional parameter to set Wi-Fi channel, from 1 to 13. Default channel = 1.
- **hidden** - optional parameter, if set to true will hide SSID.
- **max\_connection** - optional parameter to set max simultaneous connected stations, from 0 to 8. Defaults to 4. Once the max number has been reached, any other station that wants to connect will be forced to wait until an already connected station disconnects.

\*There are other overloaded methods

**WiFi. softAPConfig (local\_ip, gateway, subnet)**

- **local\_ip** - enter here IP address you would like to assign the ESP station's interface
- **gateway** - should contain IP address of gateway (a router) to access external networks
- **subnet** - this is a mask that defines the range of IP addresses of the local network

>Not using this method the network established by softAP will have default IP address of 192.168.4.1



# ESP8266 – WiFi Access Point

```
#include <ESP8266WiFi.h>
#include <Ticker.h>

Ticker tickerFunction;

void clientsConnected() {
    Serial.printf("Stations connected = %d\n", WiFi.softAPgetStationNum());
}

IPAddress local_IP(192,168,100,1);
IPAddress gateway(192,168,100,255);
IPAddress subnet(255,255,255,0);

void setup()
{
    Serial.begin(115200);
    Serial.println();

    Serial.print("Setting soft-AP configuration ... ");
    Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");

    Serial.print("Setting soft-AP ... ");
    Serial.println(WiFi.softAP("ESPsoftAP_01") ? "Ready" : "Failed!");

    Serial.print("Soft-AP IP address = ");
    Serial.println(WiFi.softAPIP());

    tickerFunction.attach(3, clientsConnected); // 3 seconds
}

void loop()
{
```



# ESP8266 – WiFi Server & Client

It provides functionality to other programs or devices, called clients

Methods come directly from the Arduino library

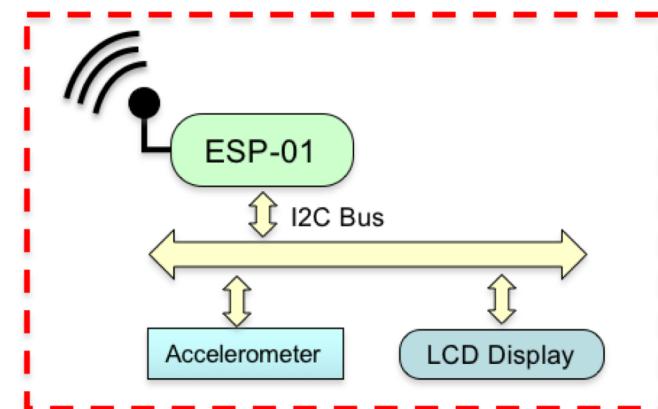
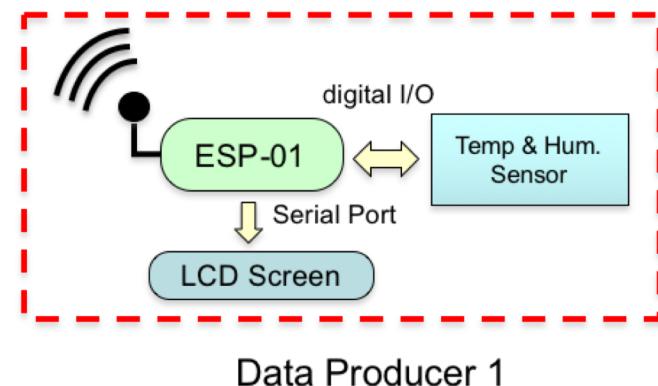
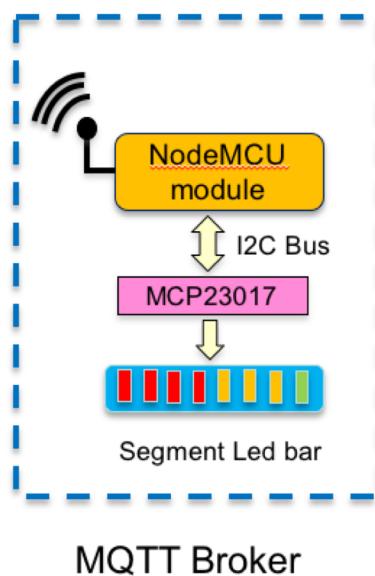
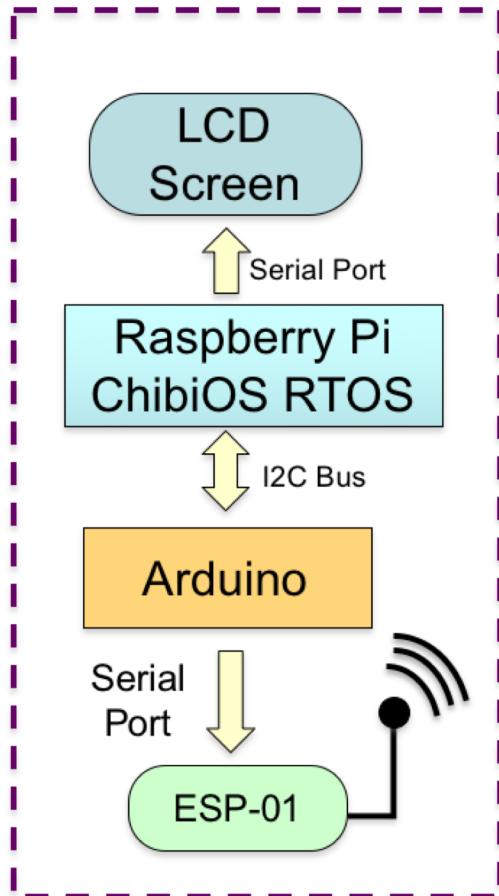
**WiFiServer server(port)** → Creates a new server **port**

- **port** - the port to listen on (int)

**WiFiClient client** → Creates a client able to be connected to any server

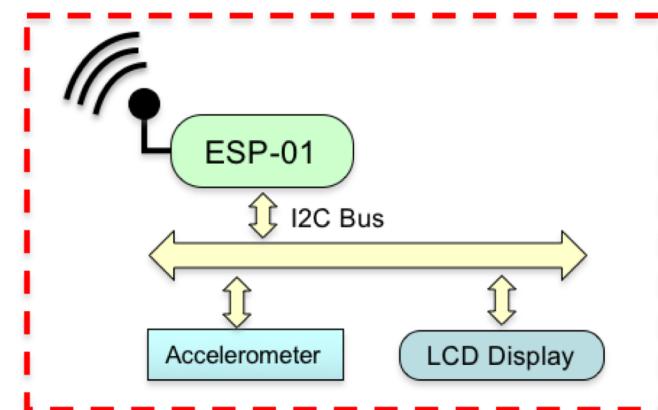
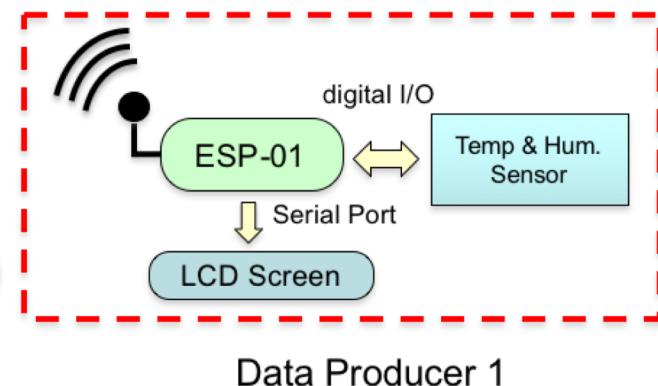
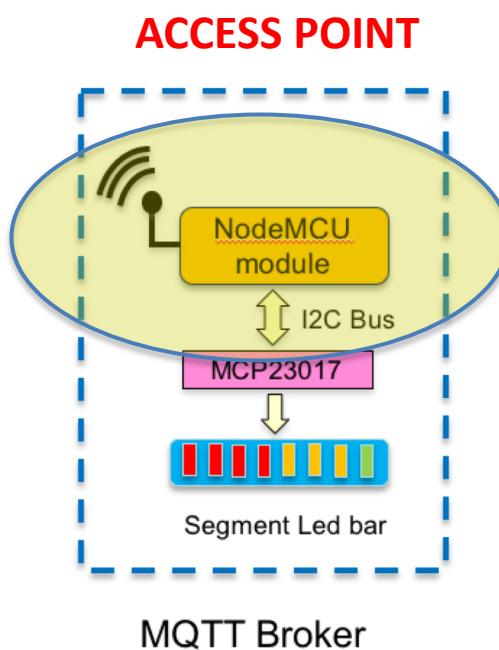
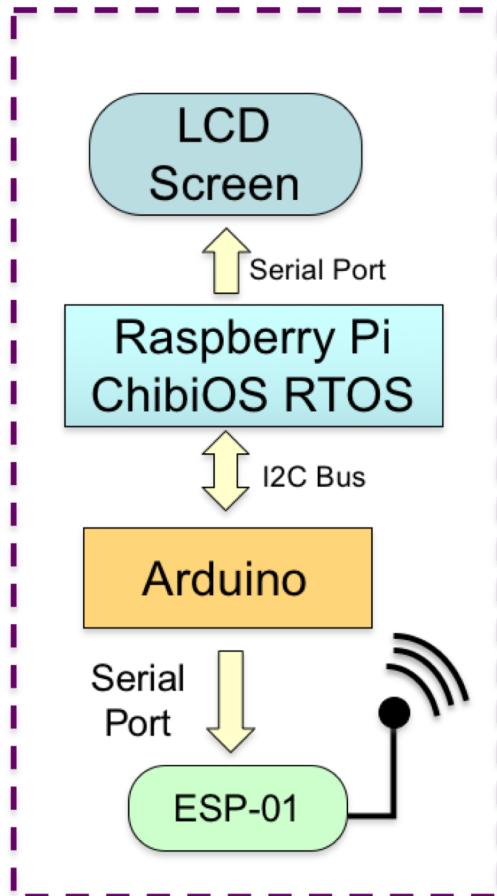


# Project



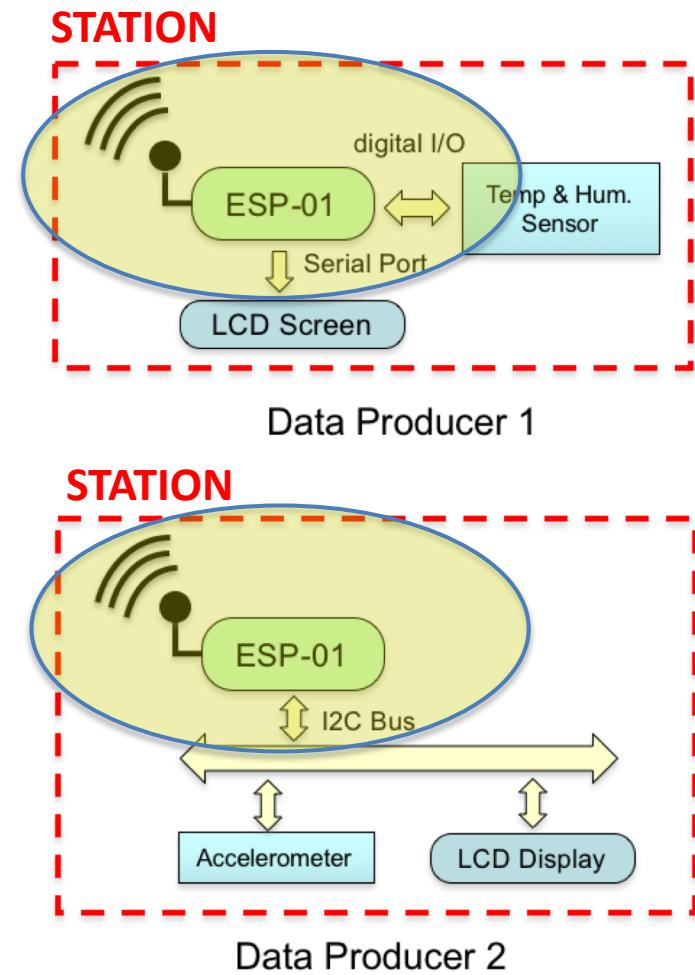
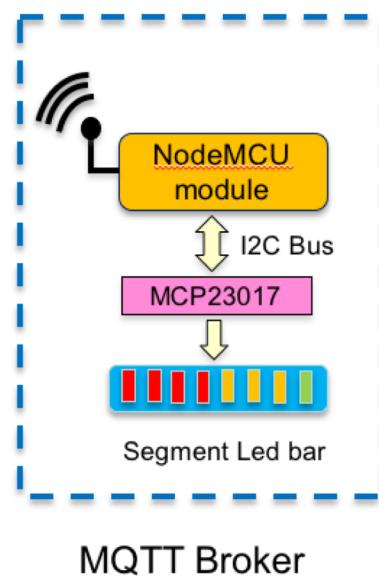
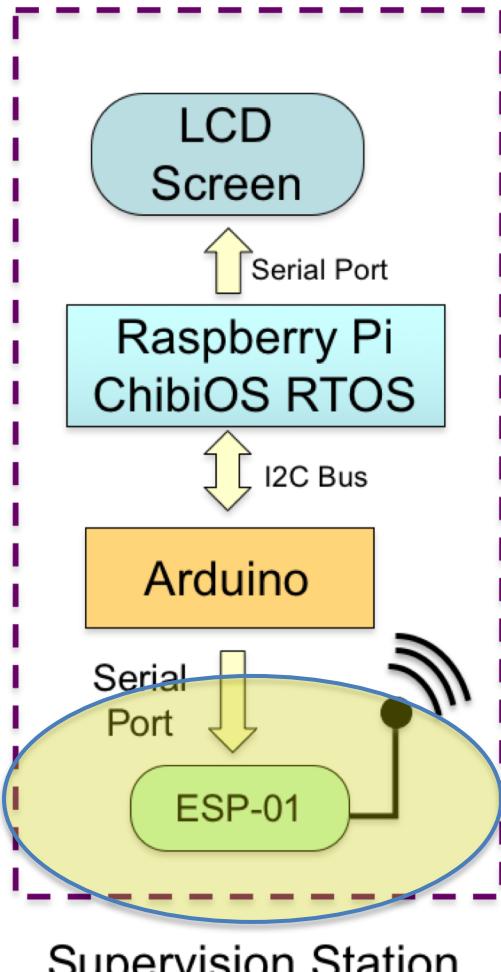


# Project



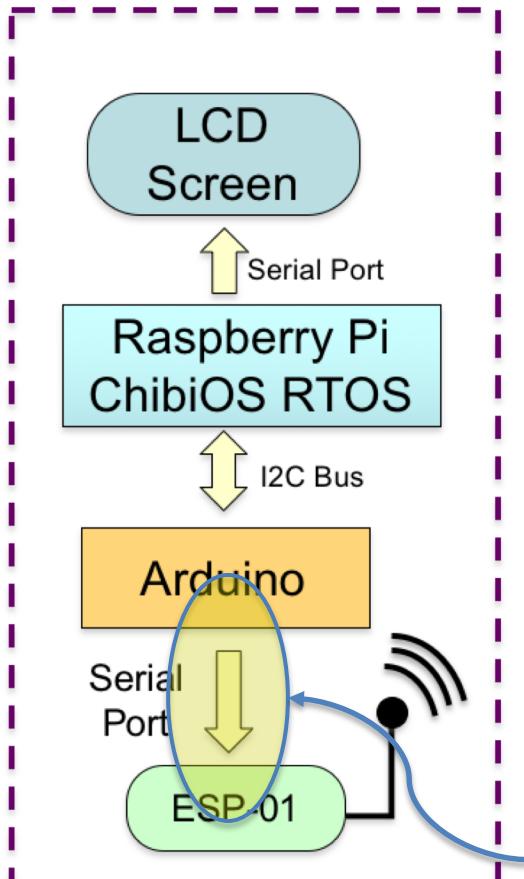


# Project

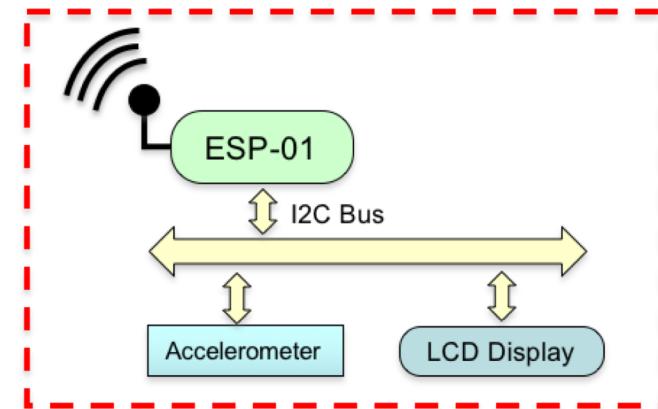
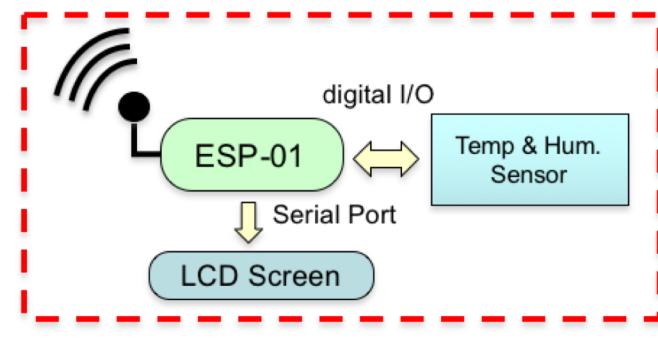
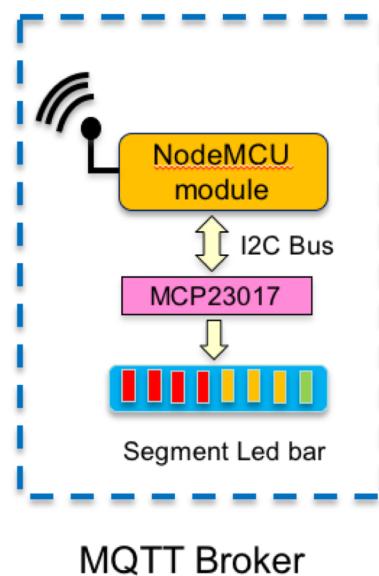




# Project



**COMMUNICATION  
AT COMMANDS**





# AT ESP-01 firmware

1. Download the AT Expressif firmware and flash download tool

> <https://www.espressif.com/en/products/socs/esp8266ex/resources>

Windows – Expressif web

Linux – esptool.py

> <https://github.com/themadinventor/esptool>