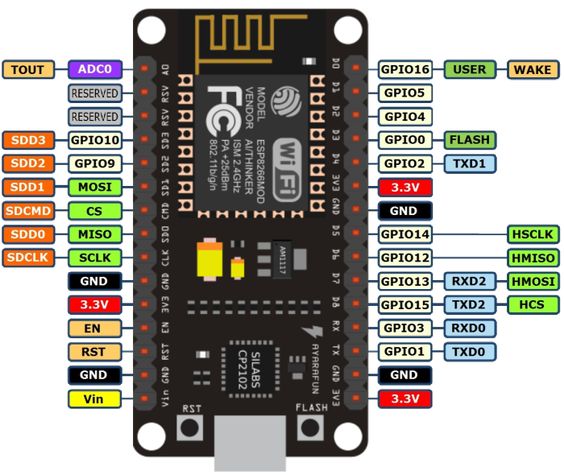
# ESP8266 Pinout



In the ESP8266 firmware for the Arduino IDE pin numbers are defined as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Pin Name on the Board** | **Function** | **Pin Number in Arduino IDE** | **Alias Name in Arduino IDE** |
| D3 | GPIO 0 | 0 | D3 |
| TX | GPIO 1 | 1 | D10 |
| D4 | GPIO 2 | 2 | D4 |
| RX | GPIO 3 | 3 | D9 |
| D2 | GPIO 4 | 4 | D2 |
| D1 | GPIO 5 | 5 | D1 |
| D6 | GPIO 12 | 12 | D6 |
| D7 | GPIO 13 | 13 | D7 |
| D5 | GPIO 14 | 14 | D5 |
| D8 | GPIO 15 | 15 | D8 |
| D0 | GPIO 16 | 16 | D0, LED\_BUILTIN |
| A0 | ADC0 | A0 | analog\_ip |

# Pin Functions

Pin numbers in the Arduino IDE correspond directly to the ESP8266 GPIO pin numbers. **pinMode**,**digitalRead**, and **digitalWrite** functions work as usual, so to read GPIO2, call **digitalRead(2)** or its alias name **digitalRead(D10)**.

At startup, pins are configured as **INPUT**. Digital pins 0—15 can be **INPUT**, **OUTPUT**, or **INPUT\_PULLUP.**

Pin 16 can be **INPUT**, **OUTPUT**or **INPUT\_PULLDOWN\_16** and is connected to the build-in LED. It can be addressed with **digitalRead(D0)**, **digitalRead(16)** or **digitalRead(LED\_BUILDIN)**.

Pins may also serve other functions, like Serial, I2C, SPI. These functions are normally activated by the corresponding library. The diagram above shows the pin mapping for the popular ESP8266 NodeMcu module.

Pin interrupts are supported through **attachInterrupt**, functions. Interrupts may be attached to any GPIO pin, except GPIO16. Standard Arduino interrupt types are supported: **CHANGE**, **RISING**, **FALLING**.

# Reserved Pins

GPIO pins 6—11 are not shown on this diagram because they are used to connect flash memory chip on most modules. Trying to use these pins as IOs will likely cause the program to crash.

Note that some boards and modules (ESP-12ED, NodeMCU 1.0) also break out pins 9 and 11. These may be used as IO if flash chip works in DIO mode (as opposed to QIO, which is the default one).

# Vin, 3V3, GND

**Vin** is the NodeMcu's voltage input that is connected to its internal voltage regulator allowing an input voltage range of 4.75V to10V. It will be regulated to 3.3V. Alternatively an external voltage source of 3.3V can be directly connected to the NodeMcu's 3V3 pins. The **3V3** pin can be also a voltage source to other components such as LEDs. **GND** is the common ground of the board.

# Analog Input

ESP8266 has a single ADC channel available to users. It may be used either to read voltage at ADC pin, or to read module supply voltage (VCC).

To read external voltage applied to ADC pin, use **analogRead(A0)**. Input voltage range is 0 — 1.0V.

To read VCC voltage, use **ESP.getVcc()** while the ADC pin must be kept unconnected. Additionally, the following line has to be added to the sketch:

**ADC\_MODE(ADC\_VCC);**

This line has to appear outside of any functions, for instance right after the **#include** lines of your sketch.

# Analog Output

**analogWrite(pin,value)** enables software PWM on the given pin. PWM may be used on pins 0 to 16. Call **analogWrite(pin,0)** to disable PWM on the pin. **value** may be in range from 0 to **PWMRANGE**, which is equal to 1023 by default. A value of 0, 512 and 1023 sets the PWM duty cycle to 0%, 50% and 100%, respectively. Optionally, the PWM range may be changed by calling **analogWriteRange(new\_range)**.

PWM frequency is 1kHz by default. Call **analogWriteFreq(new\_frequency)** to change the frequency. The unit representation is in [Hz].  Example:

**analogWriteFreq(32500);** //sets PWM frequency to 32.5 kHz