Micropython con Nodemcu

Flashear:

```
$ esptool.py --port /dev/ttyUSB0 erase_flash

$ esptool.py --port /dev/ttyUSB0 --baud 460800 write_flash --flash_size=detect 0
esp8266-20170526-v1.9.bin
```

Ampy comandos:

```
$ ampy -p /dev/ttyUSB0 run -n name_script.py
$ ampy -p /dev/ttyUSB0 put name_script.py /main.py
$ ampy -p /dev/ttyUSB0 get main.py
$ ampy -p /dev/ttyUSB0 ls
```

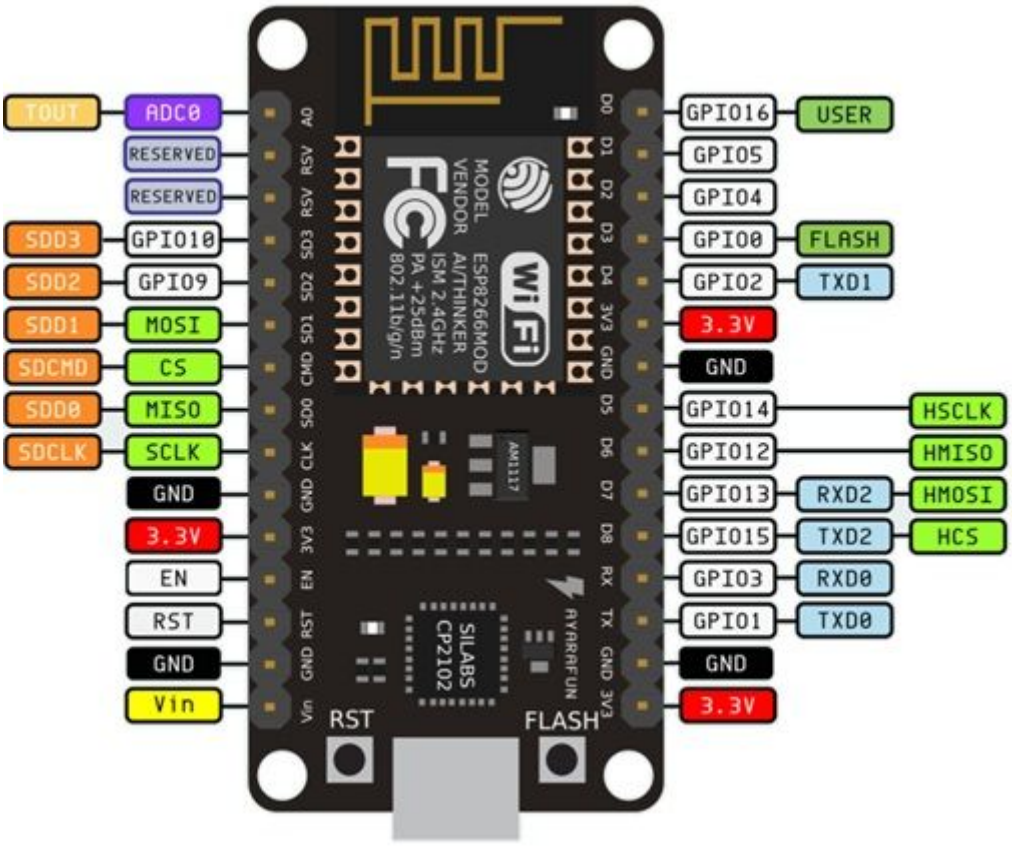
Secuencias de inicio

Hay dos archivos importantes que MicroPython busca en la raíz de su sistema de archivos. Estos archivos contienen código MicroPython que se ejecutará cada vez que la placa se enciende o restablece (es decir, se inicia). Estos archivos son:

- /boot.py - Este archivo se ejecuta primero al encender / restablecer y debe contener código de bajo nivel que configura la placa para finalizar el arranque. Por lo general, no necesita modificar boot.py a menos que esté personalizando o modificando MicroPython. Sin embargo, es interesante ver el contenido del archivo para ver qué sucede cuando la placa se inicia. Recuerde que puede usar el comando ampy get para leer esto y cualquier otro archivo!
- /main.py - Si este archivo existe, se ejecuta después de boot.py y debe contener cualquier script principal que desee ejecutar cuando la placa se encienda o restablezca.

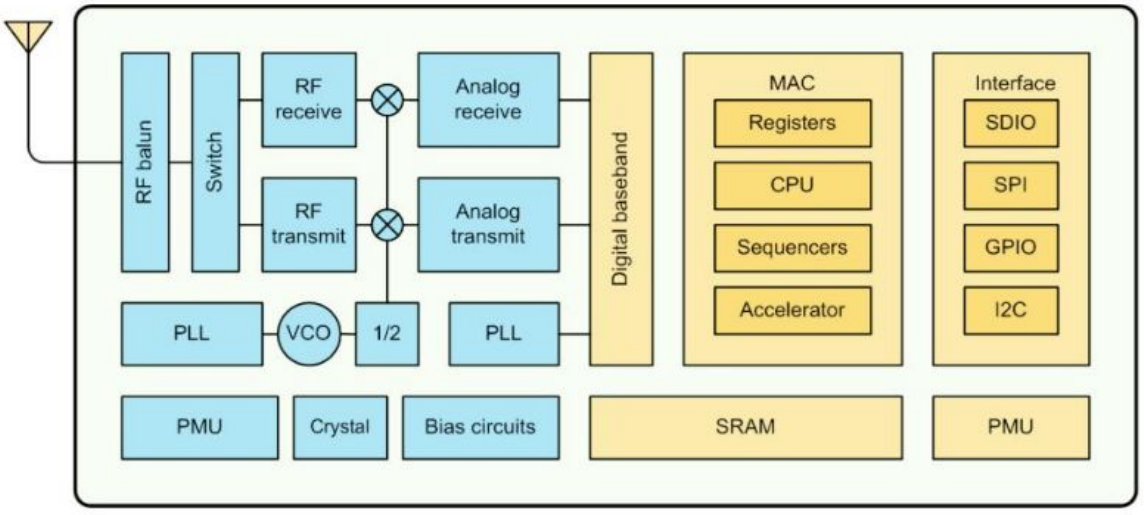
El script main.py es lo que puede utilizar para que su propio código se ejecute cada vez que una placa MicroPython se enciende. Al igual que la forma en que se ejecuta un boceto de Arduino cuando la placa Arduino tiene alimentación, escribir un main.py en una tarjeta MicroPython ejecutará ese código siempre que la tarjeta MicroPython tenga alimentación. Puede crear y editar el archivo main.py en una placa utilizando las operaciones de archivo en amp .

GPIO del nodemcu:



GPIO16: LED(user) ; **GPIO2**: LED(boot)

Arquitectura interna del ESP8266-12E:



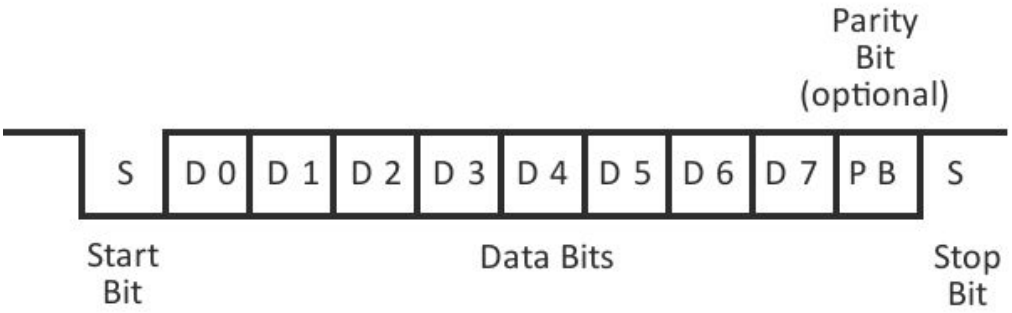
Funcionalidad de los pines:

NO.	Pin Name	Function
1	RST	Reset the module
2	ADC	A/D Conversion result.Input voltage range 0-1v,scope:0-1024
3	EN	Chip enable pin.Active high
4	IO16	GPIO16; can be used to wake up the chipset from deep sleep mode.
5	IO14	GPIO14; HSPI_CLK
6	IO12	GPIO12; HSPI_MISO
7	IO13	GPIO13; HSPI_MOSI; UART0_CTS
8	VCC	3.3V power supply (VDD)
9	CS0	Chip selection
10	MISO	Salve output Main input
11	IO9	GPIO9
12	IO10	GBIO10
13	MOSI	Main output slave input
14	SCLK	Clock
15	GND	GND
16	IO15	GPIO15; MTDO; HSPICS; UART0_RTS
17	IO2	GPIO2; UART1_TXD
18	IO0	GPIO0
19	IO4	GPIO4
20	IO5	GPIO5
21	RXD	UART0_RXD; GPIO3
22	TXD	UART0_TXD; GPIO1

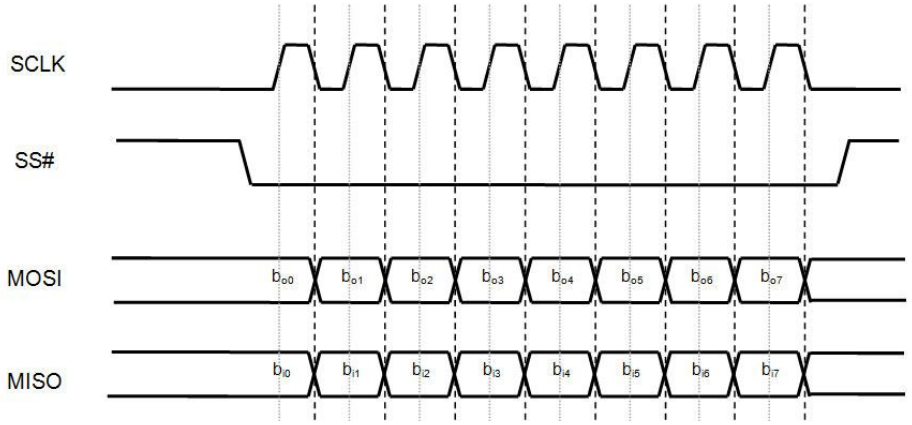
GPIO4: SDA , **GPIO5**: SCL

Protocolos del ESP:

- **UART** (recepción-transmisión asíncrona universal) es uno de los protocolos serie más utilizados. La mayoría de los microcontroladores disponen de hardware UART. Usa una línea de datos simple para transmitir y otra para recibir datos. Comúnmente, 8 bits de datos son transmitidos de la siguiente forma: un bit de inicio, a nivel bajo, 8 bits de datos y un bit de parada a nivel alto. UART se diferencia de SPI y I2C en que es asíncrono y los otros están sincronizados con señal de reloj. La velocidad de datos UART está limitado a 2Mbps



- **SPI** es otro protocolo serie muy simple. Un maestro envía la señal de reloj, y tras cada pulso de reloj envía un bit al esclavo y recibe un bit de éste. Los nombres de las señales son por tanto SCK para el reloj, MOSI para el Maestro Out Esclavo In, y MISO para Maestro In Esclavo Out. Para controlar más de un esclavo es preciso utilizar SS (selección de esclavo).



- **I2C** es un protocolo síncrono. I2C usa solo 2 cables, uno para el reloj (SCL) y otro para el dato (SDA). Esto significa que el maestro y el esclavo envían datos por el mismo cable, el cual es controlado por el maestro, que crea la señal de reloj. I2C no utiliza selección de esclavo, sino direccionamiento.