



Fco. Javier Rodriguez Navarro

De 0 a 100
con Arduino
y ESP32



<https://amzn.eu/d/32OJjdF>

Arduino vs ESP32

Javier Rodriguez (@emb0scad0)

Github: <https://github.com/pinguytaz>

Web: www.pinguytaz.net

22 Noviembre 2025

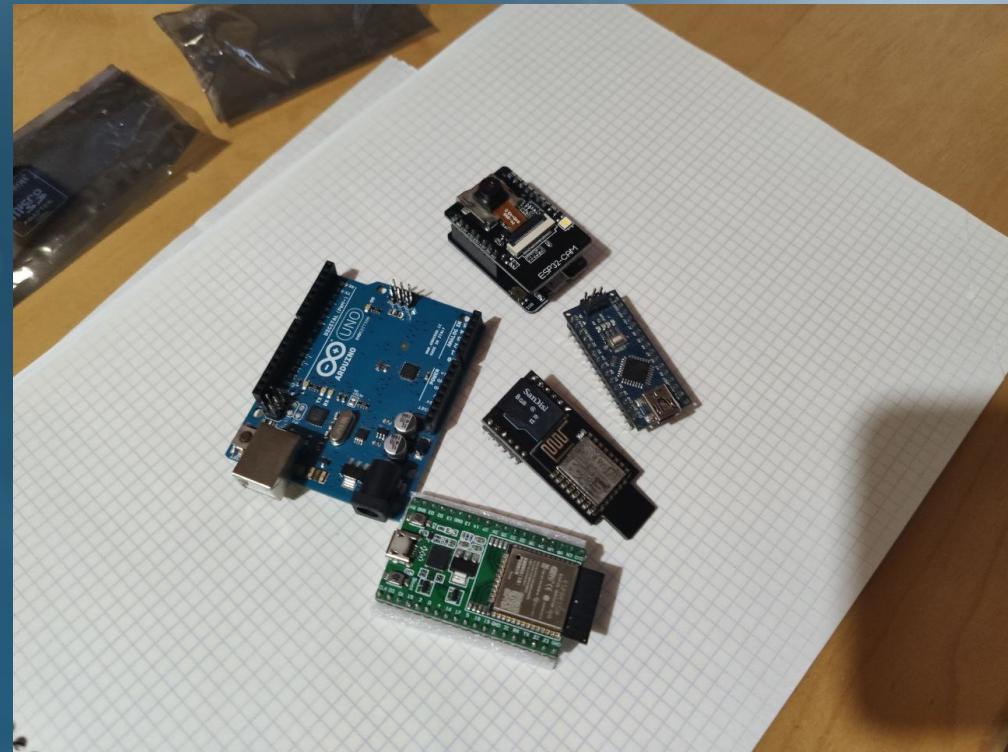
Objetivos

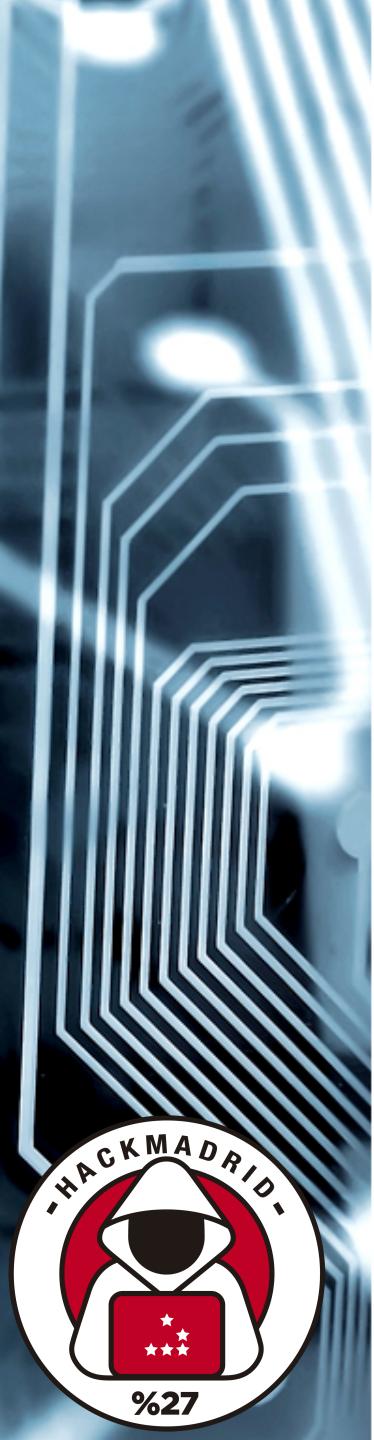
- Que son estos dispositivos
- Para que sirve
- Como iniciarse
- Avanzamos algo
- Ejemplos



Tipos

- Arduino AVR
 - UNO
 - Nano
 - Leonardo
- ESP32
 - DevKitC
 - ESPCAM
 - LilyGo
- ESP-8266, PiZero, Raspberry-Pi, Orange-Pi



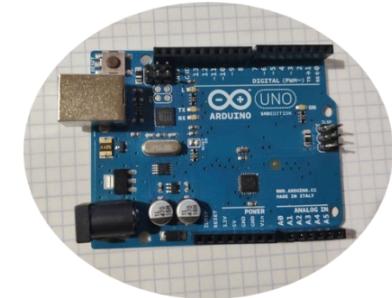


Microcontrolador vs CPU

- CPU
- Memorias integradas:
 - RAM datos temporales
 - Flash/ROM/EEPROM persistente
- Periféricos:
 - GPIO, temporizadores, UART/SPI/I2C, ADC/DAC,
- Una CPU
Es solo el núcleo de cómputo (cerebro) y requiere memoria y periféricos externos para funcionar como sistema completo.



Arduino



- Microcontrolador AVR
- Puerto E/S digitales (5V 20mA)
 - Algunos son salidas PWM
- Puerto E analógicas (precisión 10bits 0-5V)
- Puerto de comunicación
 - I2C
 - UART
 - SPI
- EPROM (Nuestros datos)
- SRAM (Variables)
- Flash (nuestro programa) en ella el Bootloader
- Que nos facilita su desarrollo
 - Entorno de desarrollo Arduino-IDE
 - Bootloader
Cargar sketches por el puerto serie/USB sin un programador externo (protocolo STK500)



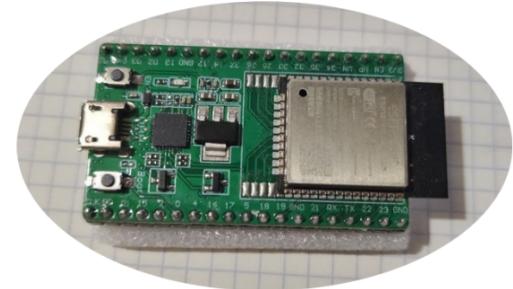


Identificación ESP32

- SoC (System on a Chip) es la CPU
- Modulo es la CPU y elementos que lo convierten en microcontrolador.
- Placas de desarrollo
Nos facilitan el desarrollo y se especializan:
ESPCAM, LilyGo....
Son las que normalmente utilizaremos.



ESP32



- Microcontrolador ESP32
Dual Core
- Puertos E/S digitales, PWM
- ADC Entradas analógicas (precisión 12bits)
- Puertos de comunicación: I2C, UART, SPI
- Memorias: EPROM, SRAM, Flash
- Sensores capacitivos
- SO Multitarea FreeRTOS
- Modos de operación: Activo, Sleep, hibernación
- WIFI
- Bluetooth (con BLE)
- DAC Salida analógicas (8bits)
- Desarrollo
 - Su propio entorno SDK ESP-IDF
 - IDE Arduino



ESP32 Modos ahorro

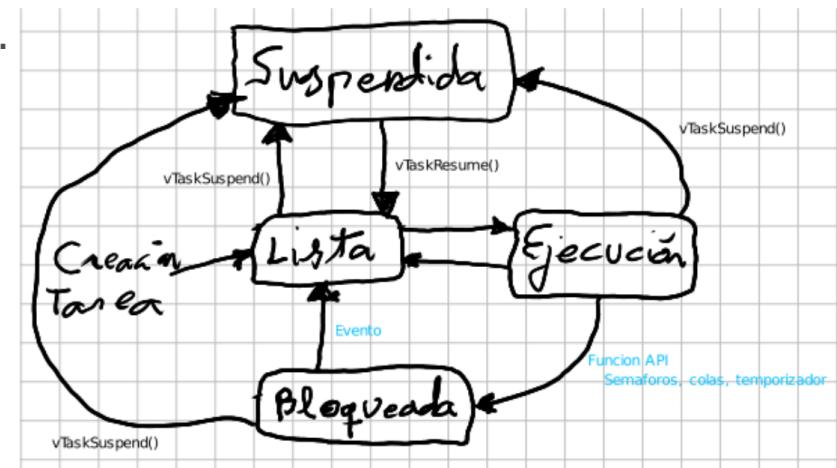
Modo	CPU	WIFI Bluetooth	RTC	ULP	Consumo medio típico
Activo	Encendido	Encendido	Encendido	Encendido	~200 mA
Modem-sleep	Encendido	Apagado	Encendido	Encendido	~50 mA
Light-sleep	Pausa	Apagado	Encendido	Encendido	~1 mA
Deep-sleep	Apagado	Apagado	Encendido	Encendido	150 microA
Hibernación	Apagado	Apagado	Apagado	Apagado	5 microA

- Modo activación `esp_sleep_enable_X_wakeup()`
 - Temporizador (“timer”) maximo 71 minuto
 - “touchPad” Entrada capacitiva.
 - “ext0” entrada RTC_GPIO (17)
 - “ext1” multiples RTC_GPIO
 - 4 Registros de 16 bits.
 - 8K de memoria lenta RTC
 - No dispone de pila, solo saltos y no call
 - Monitorización ADC
 - Leer y escribir en el I2C
 - “gpio” solo en ligera
 - “uart” cuando llegan datos por el puerto serie
 -

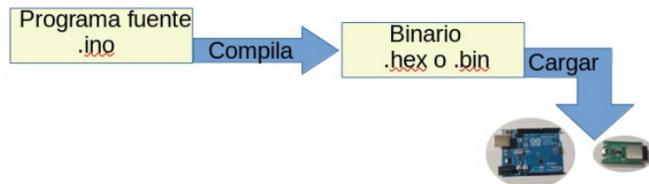


Multitarea FreeRTOS

- Normalmente usamos solo el core 1
- Core 0
 - Bluetooth
 - WIFI
- Arduino-UNO (un solo core) TaskScheduler ¿?
- **FreeRTOS** sistema operativo en tiempo real
 - Incorporado en el ESP32
 - Ejecución de tareas en paralelo.
 - Comunicación entre las tareas.
 - Ejecución basada en prioridades.
 - Compartir recursos.



IDE-Arduino



EjDosLeds.ino

```
23
24 void setup()
25 {
26     // También cambiaremos la velocidad del puerto de consola segun
27     #if defined(PLAT_ARDUINO)
28         Serial.begin(9600);
29     #else
30         Serial.begin(115200);
31     #endif
32     pinMode(P_LED, OUTPUT);
33     pinMode(P_Boton, INPUT);
34 }
```

Salida

```
El Sketch usa 2142 bytes (6%) del espacio de almacenamiento de programa.
Las variables Globales usan 188 bytes (9%) de la memoria dinámica, dejando
```

- Gestión de librería
- Monitor Serie, Serial Plotter
- Gestión de placas
ESP32
["https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json"](https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json)



Programación

- Programa principal .ino
- Cabeceras .h
- Clases (útiles en nuestras librerías) .cpp
- Estructura de un programa
 - Definiciones globales, includes
 - setup()
 - loop()
- Para iniciarnos en el lenguaje Arduino (C) no usemos inicialmente las E/S sino el monitor serie



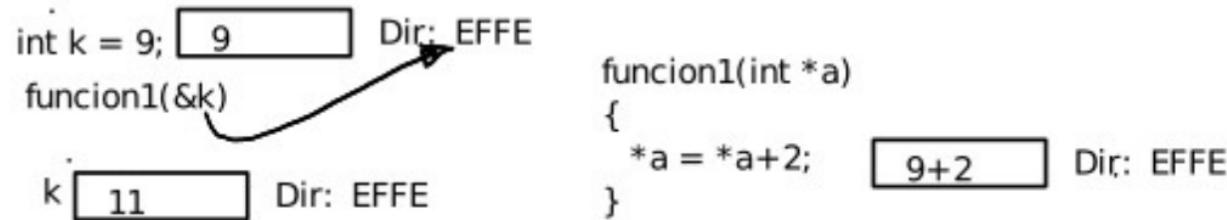
Variables y constantes

- Globales (antes del setup)
- Locales (a los bloques estructuras y funciones)
- <tipo> <nombre>
- Operaciones aritméticas, binarias
- Diferentes tipos: int, long.....
- Calificadores
 - volatile (forzamos que este en memoria)
 - static (retienen su valor)
 - const (solo lectura)
- Constantes
 - HIGH, LOW, true, false, PI
- Arrays o matrices [0-x]
- String (un array que termina en \0)
- Tipos avanzados: enum, struct, union, typedef



Control de flujo

- Bloque { }
- If, switch-case
- while, do for
 - break, continue
- Funciones
 - **Parámetros por referencias**



- Funciones internas



Punteros

- Con cabeza
- Arrays
- Punteros a función “CallBack”

`z = 15` Dir: ZXYW
`a = 3` Dir: ABCD
`*Pc`
`Pc=&a`
`*Pc=7` Dir: ABCD // Por lo tanto a valdra 7
`Pc=9876` // Ya no apuntara a 'a' y sera una dirección que no sabemos cuales

`Pc=&z`
`*Pc = a+5` Dir: ZXYW // z valdra 12 7+5
`a=*Pc+2` Dir: ABCD // a valdra 14 12 (valor de z) + 2

`int a[4] = {11,22,33,44}` Dir: EFA0
 Dir: EFA2
 Dir: EFA4
 Dir: EFA0
`*p`
`p=a`
`*p` Valdra 11=p[0]
`++`
`*p` Valdra 22=p[1]

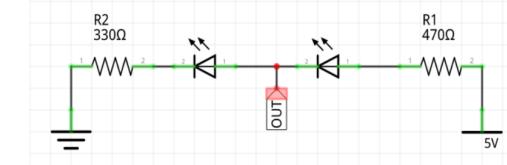


Librerías

- Sketch → incluir biblioteca->añadir biblioteca ZIP
- ZIP
 - \examples
 - keywords.txt
 - library.properties
 - \src

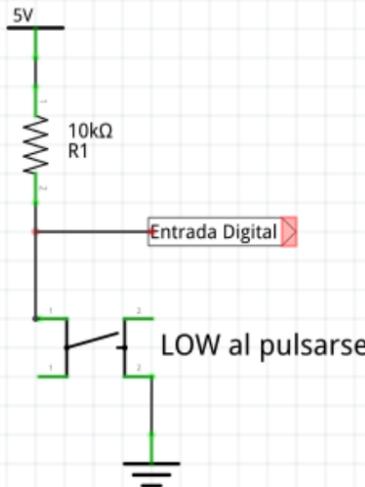


E/S digitales

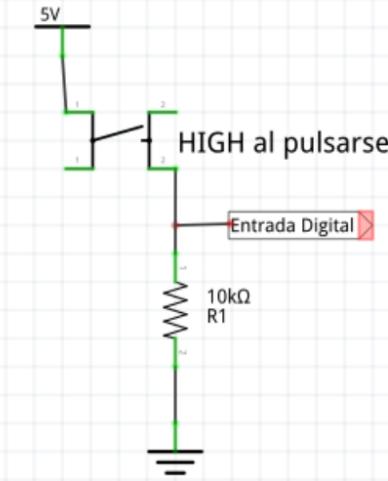


- OJO: Arduino 5V pero ESP32 3.3v
 - Ojo consumo, ejemplo motores.
- LOW=0 HIGH=1
- Primero los definiremos con “pinMode”
- PCF8574 o PCF8575 ampliación de digitales
- Algunas podrán comportarse como PWM

Pushbutton a GND



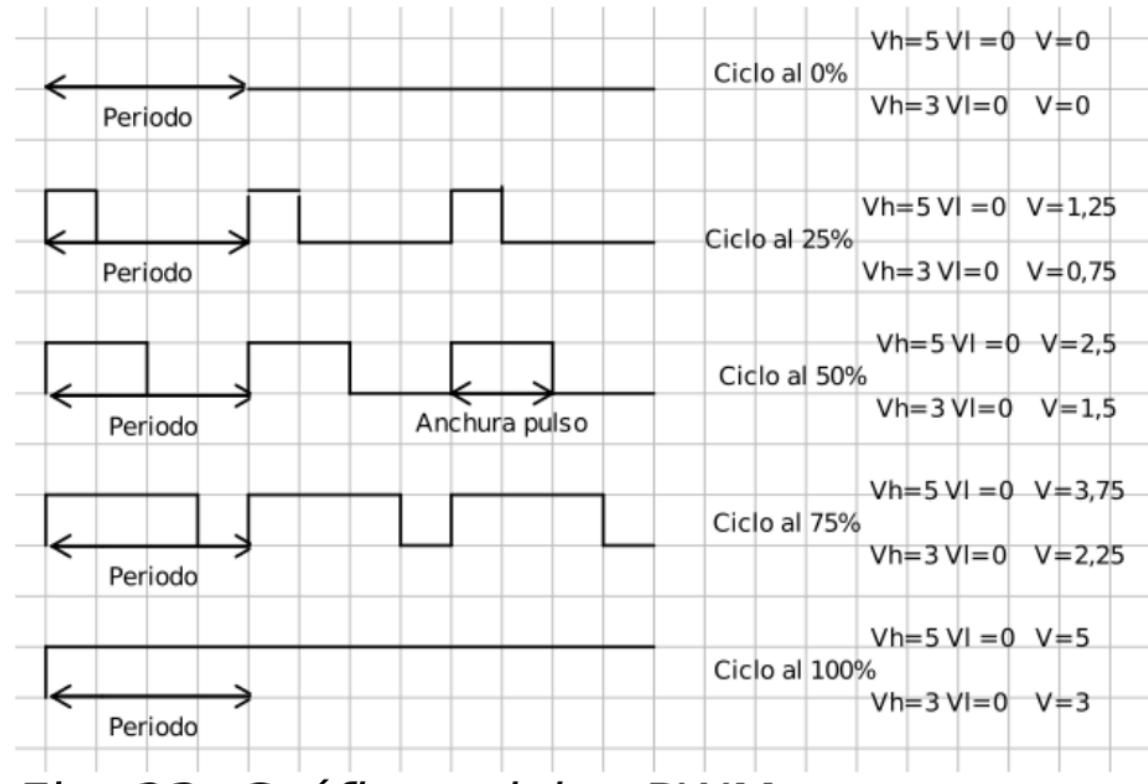
Pushbutton a 5V



PWM

$$V = (Vh - Vl) - \frac{Ciclo}{100}$$

- “Simula” salida analógica
- Los PWM usan los Timer



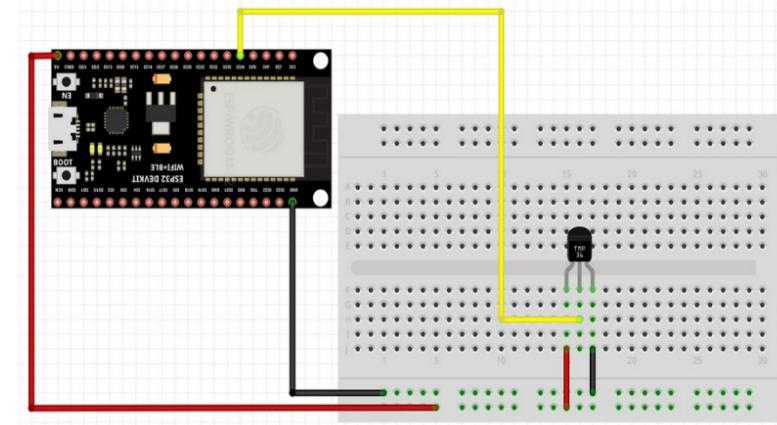
Interrupciones HW

- Tenemos pines que lanzan una interrupción y ejecuta una función.
 - Arduino UNO (pin 2 y 3)
 - En el ESP32 podemos usar cualquier GPIO
-
- Las funciones ISR
 - Cortas y rápidas
 - No usar funciones de tiempo (delay millis)
 - Uso de puerto serie, marcar FLAG y hacerlo fuera.
 - Las funciones que deban ser conocidas dentro y fuera de la función “volatile”



Entradas Analógicas ADC

- En Arduino 0-5V
- Pines A0-A5
- Arduino UNO 10 bits (0-1023)
- En ESP32 12 bits (0-4095) 0-3.3V (18 entradas)
 - 8 en ADC1 y 10 en ADC2 (Ojo no usar con WIFI y Bluetooth)
- Podremos definir la resolución para compatibilizar con programas de Arduino UNO
- Interesante poner condensador de 0,1mF por el ruido.



Salidas analógica DAC solo ESP32

- En arduino podríamos usar PCF8591 o MCP4725
- 8 bits (0-255) 0-3.3V

```
int IO2 = 2;  
void setup()  
{  
    Serial.begin(115200);  
}  
void loop()  
{  
    float valor;  
    for (int V = 0; V < 256; V++)  
    {  
        dacWrite(25, V);  
        valor = analogRead(32);  
        Serial.println(valor);  
        // Serial.println("Pra el valor: "+String(V)+" el voltaje es: " + String(valor));  
        delay(10);  
    }  
}
```





Gestión de memoria

- SRAM, EEPROM, Flash (No modificable en ejecución)
- En ESP32 “Preference.h” permite almacenar clave-valor
- Particiones con multiples sistemas de archivos (116)
 - data/ota indica que aplicación ejecutar.
 - data/nvs clave-valor
 - data/fat
 - data/spiffs (archivos grandes)
 - data/coredump
 - app/factory si no tenemos aplicación OTA

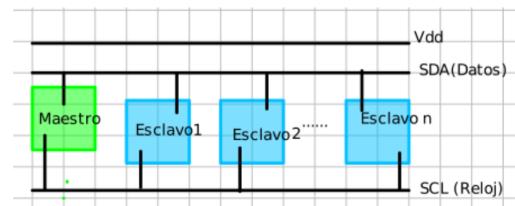
Nombre	Tipo	Subtipo	Desplazamiento	Tamaño	Flags
nvs	DATA 0x01	NVS 0x02	0x9000	0x6000	No Encriptada
phy_init	DATA 0x01	PHY 0x01	0xf000	0x1000	No Encriptada
factory	APP 0x00	Fabrica 0x00	0x10000	0xf0000	No Encriptada
datos	DATA 0x01	Spiffs 0x82	0x100000	0x100000	No Encriptada
datosWEB	DATA 0x01	Spiffs 0x82	0x200000	0x1f0000	No Encriptada
coredump	DATA 0x01	Coredump 0x03	0x3f0000	0x10000	No Encriptada

Comunicaciones

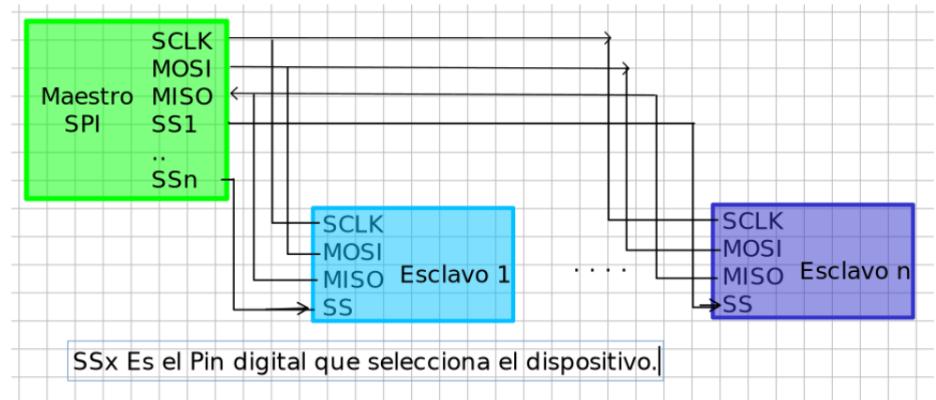
- UART
 - SofwareSerial.h
 - ESP32 2 puertos



- I2C
 - 2 cables
 - Hasta 128 dispositivos



- SPI
 - 3 +1



Bluetooth solo ESP32

- En Arduino tenemos modulos como HC05 o HM-10(BLE)
- Clásico "BluetoothSerial.h"
 - Maestro y esclavos
- BLE
 - Servidor(esclavo) anuncia su existencia
 - Cliente (Maestro) busca servidores





WIFI solo ESP32

- En Arduino ESP-01(usa ESP8266)
-
- Modos
 - Estación conectamos a un AP
 - AP se comporta como punto de acceso.
 - Mixto

