

# Elementos de sistemas embebidos

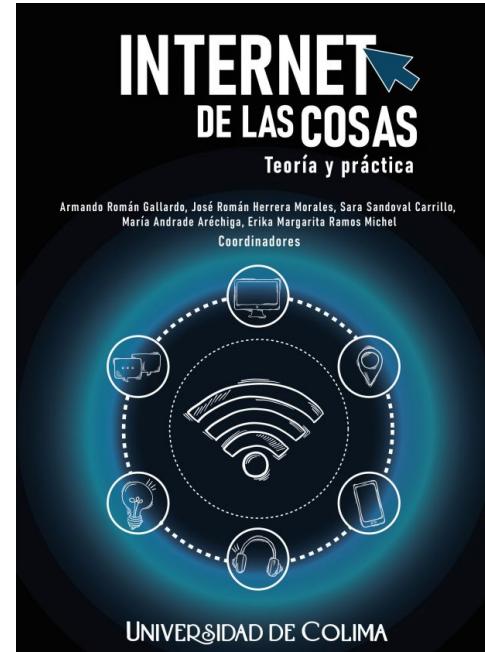
Unidad 6.0  
Introducción a IOT

Versión 1.0.0

Jaír Hnatiuk, Carlos Maidana, Carlos Rodríguez, Edgardo Gho, Martín Ferreyra Biron

# Internet of Things (IOT)

El término surge allá por 2015 con la idea de integrar tecnología RFID en el supply chain de las empresas de forma tal de quitar a las personas como intermediarias en el proceso de generación de información. Ejemplo: Un producto se empaqueta y recibe un número (representado por un código de barras) que lo identifica únicamente. Ese paquete viaja desde el fabricante por barco hasta un depósito, luego cuando se vende se envía mediante courier a su destino final. En su momento este producto era rastreado por su código de barras, pero requería que en cada punto una persona leyera el código con un lector ingresando su información a un sistema (generalmente local). Luego ese dato era exportado a otros sistemas.

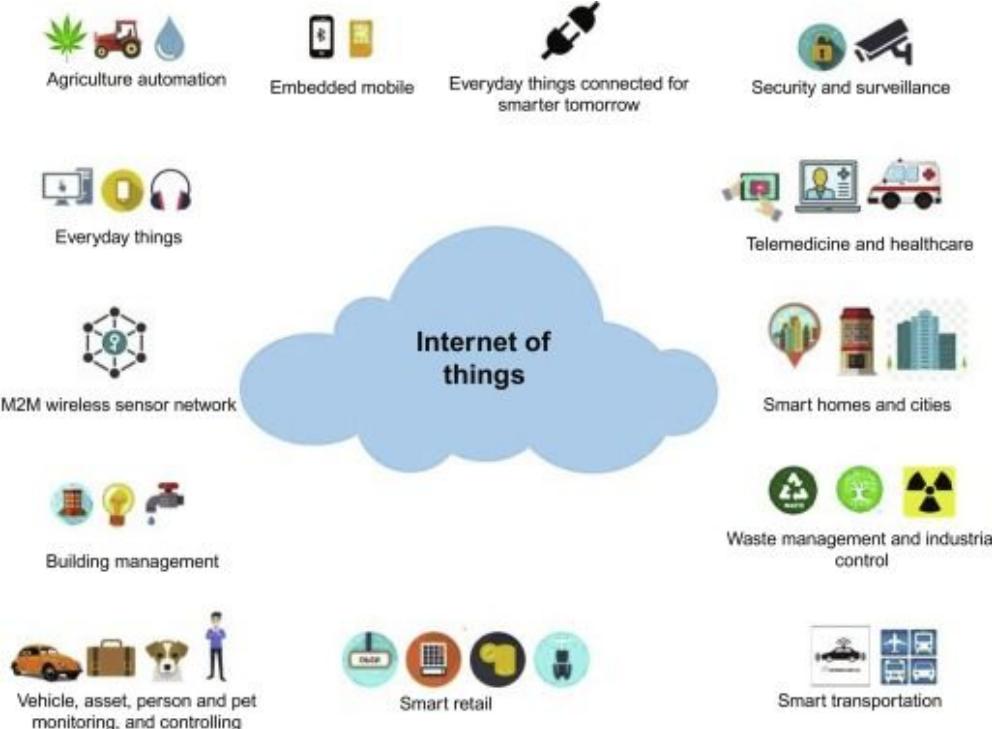


# Internet of Things (IOT) - supply chain

Con IOT, una etiqueta RFID automáticamente reporta al paquete en lectores ubicados en la fábrica, el barco, el depósito y el camión del courier, evitando así que el humano tenga que estar escaneando su ubicación.  
Para lograr esto, se requiere una infraestructura de IOT.



# Internet of Things (IOT) - En todos lados



Hoy en día se busca que cualquier área se vea beneficiada de integrar tecnologías IOT con el fin de buscar la hiperconectividad, es decir, que cualquier persona pueda acceder en cualquier momento y de cualquier lado a la información de un dispositivo o a tener control sobre el mismo.

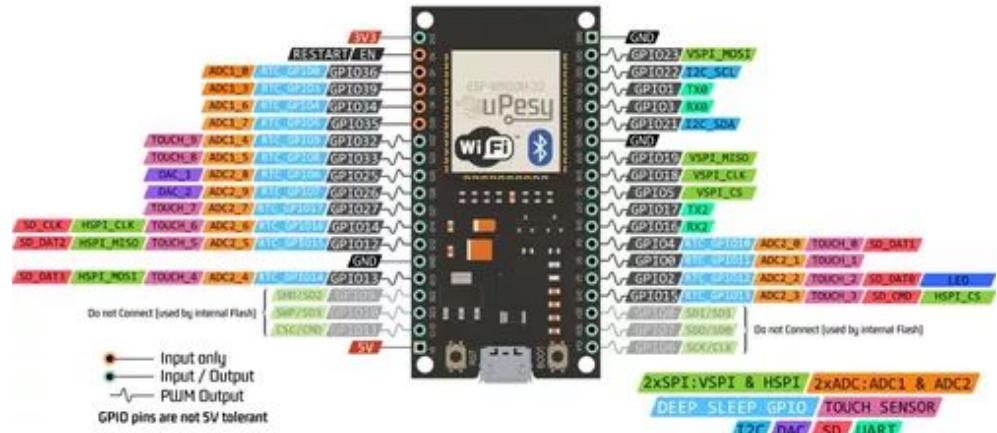
# Tecnología que dan soporte a IOT - Microcontroladores

Los microcontroladores están siempre presente en cualquier stack de tecnologías IOT. Entre los más populares:

- Arduino-ATMega/STM (Zigbee, Bluetooth)
- ESP8266/32 (Wifi,Bluetooth)
- RaspberryPI - BCM2709 (Wifi, Ethernet,Bluetooth)

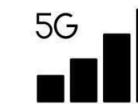
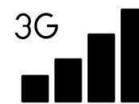


ESP32 Wroom DevKit Full Pinout



# Tecnología que dan soporte a IOT - Conectividad

La capa de comunicaciones suele ser muy variada dependiendo del medio.



# Tecnología que dan soporte a IOT -Protocolos / Infraestructura

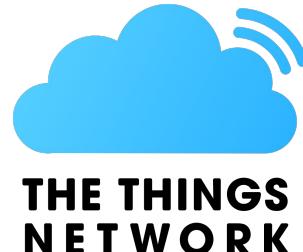
Una vez que los dispositivos están conectados, utilizan algún protocolo.



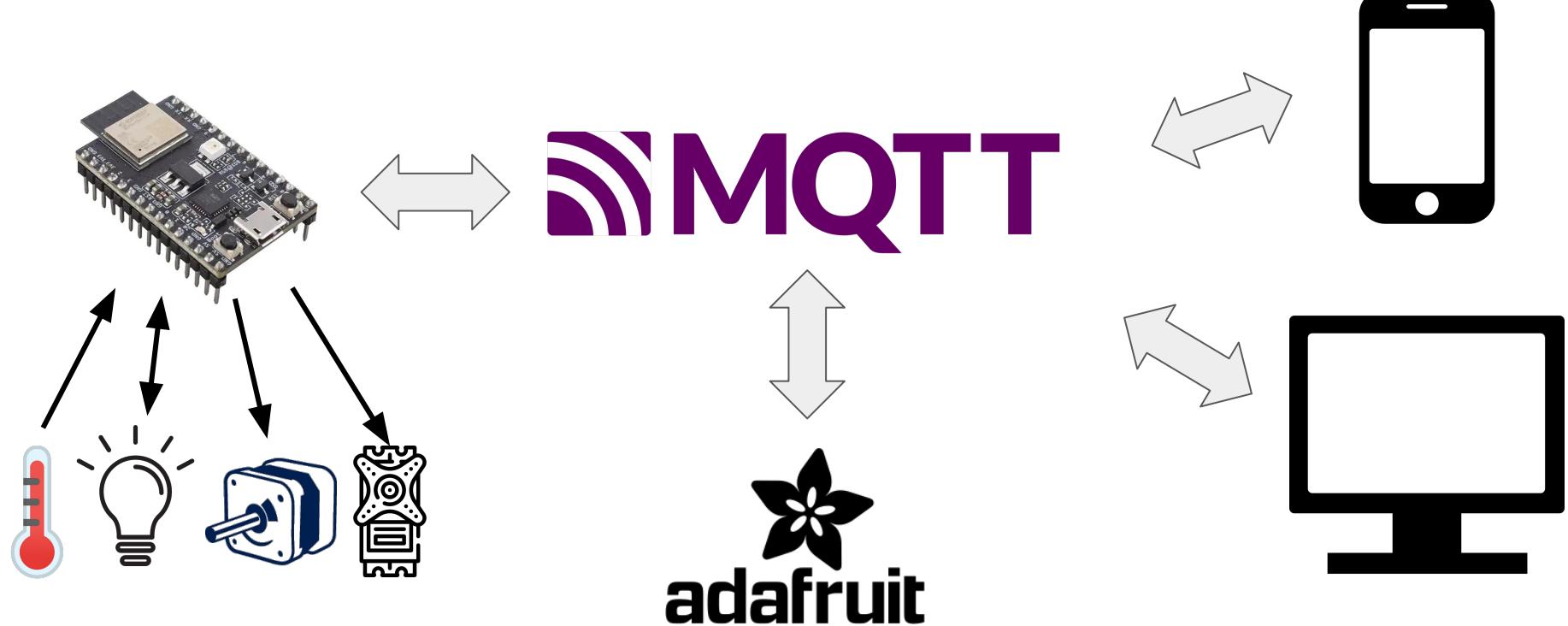
Buscando la hiperconectividad, utilizamos infraestructura cloud



Google Cloud



# Nosotros vamos a elegir un conjunto...



# Elementos de sistemas embebidos

Unidad 6.1  
Sensores y Actuadores

Versión 1.0.0

Jaír Hnatiuk, Carlos Maidana, Carlos Rodríguez, Edgardo Gho, Martín Ferreyra Biron

## 4.4 DC Characteristics (3.3 V, 25 °C)

Table 4-4. DC Characteristics (3.3 V, 25 °C)

# Tensión/Corriente

Siempre debemos tener en cuenta la tensión de alimentación y los valores de tensión y corriente en alto y bajo. **Oficialmente el valor máximo de tensión en los pines es 3,3V.** En la hoja de datos no menciona ser tolerante a 5V en GPIO. **Empíricamente es tolerante a 5V (no recomendado) en GPIO (no en VDD/VCC).**

Parameter	Description	Min	Typ	Max	Unit
$C_{IN}$	Pin capacitance	—	2	—	pF
$V_{IH}$	High-level input voltage	$0.75 \times VDD^1$	—	$VDD + 0.3$	V
$V_{IL}$	Low-level input voltage	-0.3	—	$0.25 \times VDD^1$	V
$I_{IH}$	High-level input current	—	—	50	nA
$I_{IL}$	Low-level input current	—	—	50	nA
$V_{OH}^2$	High-level output voltage	$0.8 \times VDD^1$	—	—	V
$V_{OL}^2$	Low-level output voltage	—	—	$0.1 \times VDD^1$	V
$I_{OH}$	High-level source current ( $VDD^1 = 3.3$ V, $V_{OH} \geq 2.64$ V, PAD_DRIVER = 3)	—	40	—	mA
$I_{OL}$	Low-level sink current ( $VDD^1 = 3.3$ V, $V_{OL} = 0.495$ V, PAD_DRIVER = 3)	—	28	—	mA
$R_{PU}$	Internal weak pull-up resistor	—	45	—	kΩ
$R_{PD}$	Internal weak pull-down resistor	—	45	—	kΩ
$V_{IH\_nRST}$	Chip reset release voltage CHIP_EN voltage is within the specified range)	$0.75 \times VDD^1$	—	$VDD + 0.3$	V
$V_{IL\_nRST}$	Chip reset voltage (CHIP_EN voltage is within the specified range)	-0.3	—	$0.25 \times VDD^1$	V

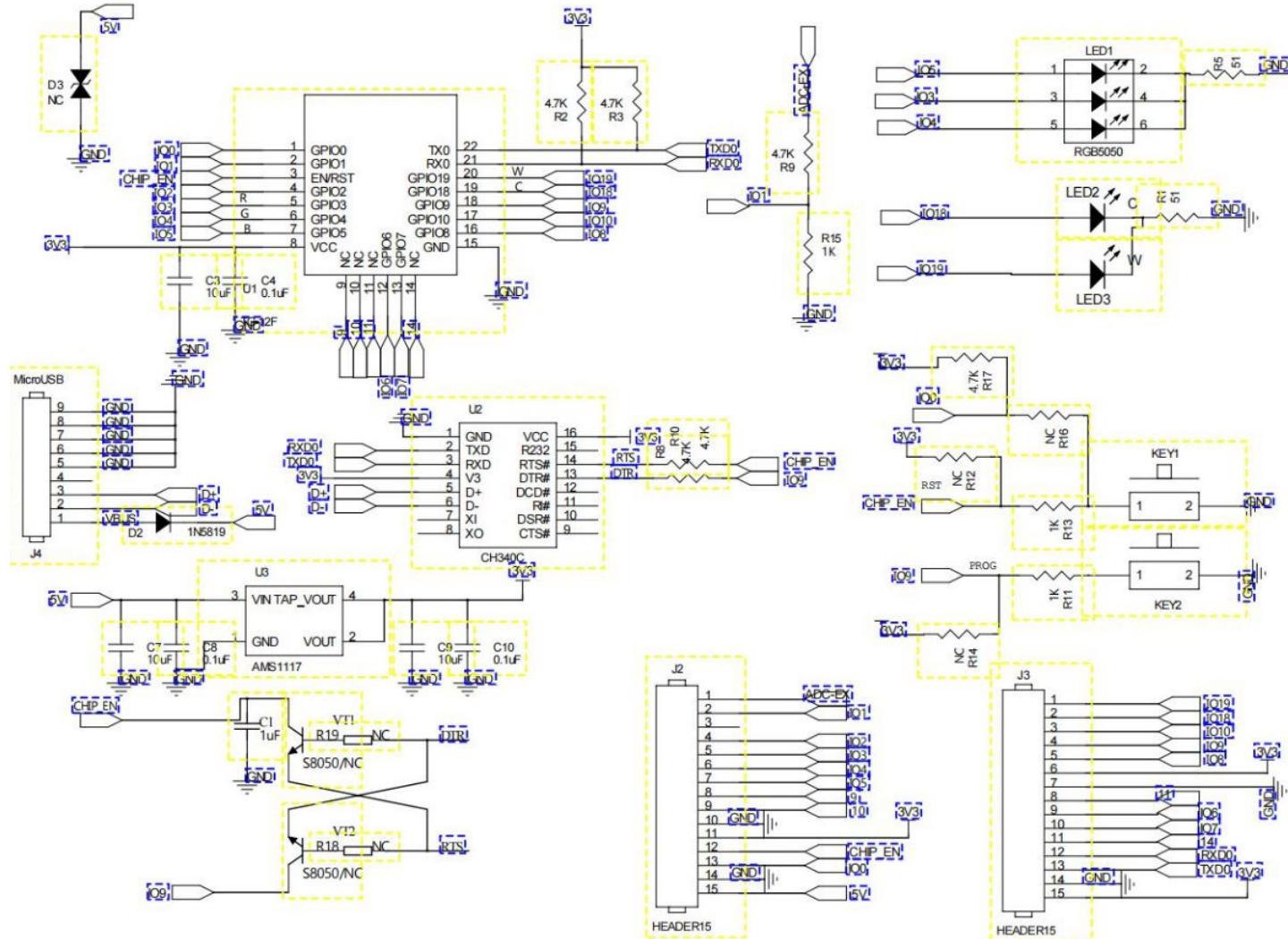
<sup>1</sup> VDD – voltage from a power pin of a respective power domain.

<sup>2</sup>  $V_{OH}$  and  $V_{OL}$  are measured using high-impedance load.

Parameter	Description	Min	Max	Unit
Input power pins <sup>1</sup>	Allowed input voltage	-0.3	3.6	V
$I_{output}^2$	Cumulative IO output current	—	1000	mA
$T_{STORE}$	Storage temperature	-40	150	°C

# Esquemático

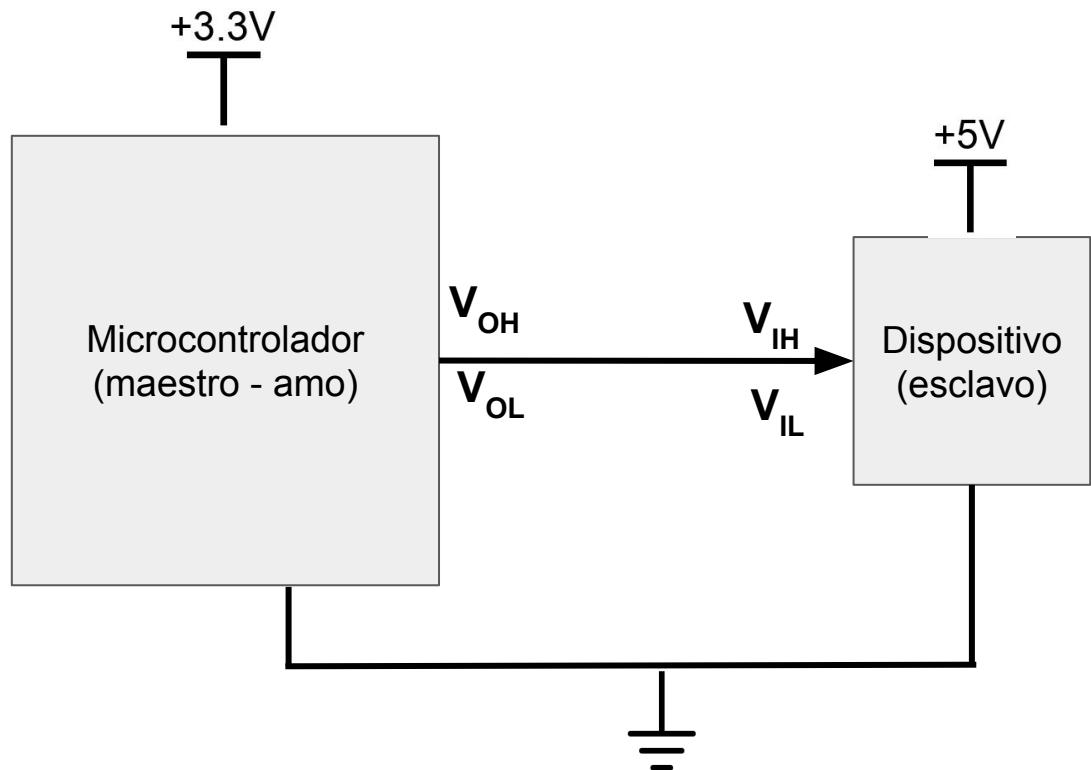
Si alimentamos por el puerto USB, el mismo brinda 5V y 100mA (salvo que configuremos el ch340 para solicitar más corriente). Pasa por el ams1117 que convierte a 3.3V.



# Salidas simples (ON/OFF)

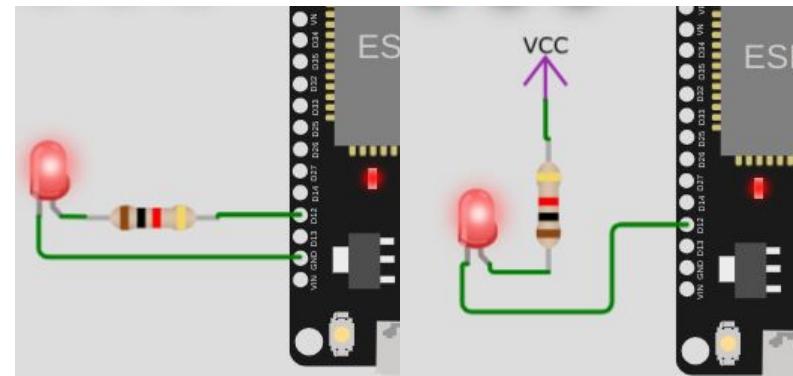
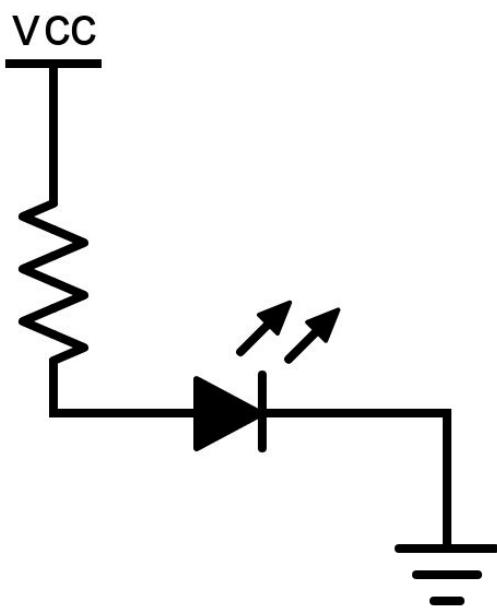
# Señales de Control (Salida)

Vemos ahora esquemas donde el microcontrolador **controla** un dispositivo mediante una señal.



# Circuitos típicos - LED

Sabemos que un LED enciende si el potencial al que se encuentra supera su tensión de umbral, y luego limitamos la corriente con una resistencia. Dependiendo si definimos la salida como Push-Pull o OpenDrain, tenemos que conectar el LED de diversas formas. En cualquier caso debemos verificar que el pin soporte la corriente necesaria!



# Circuitos típicos - LED con PWM

Utilizando PWM podemos variar el porcentaje de tiempo que el LED está encendido. A una frecuencia alta, la persistencia en la retina hace que notemos distintas intensidades en el brillo del LED.

25% Duty Cycle



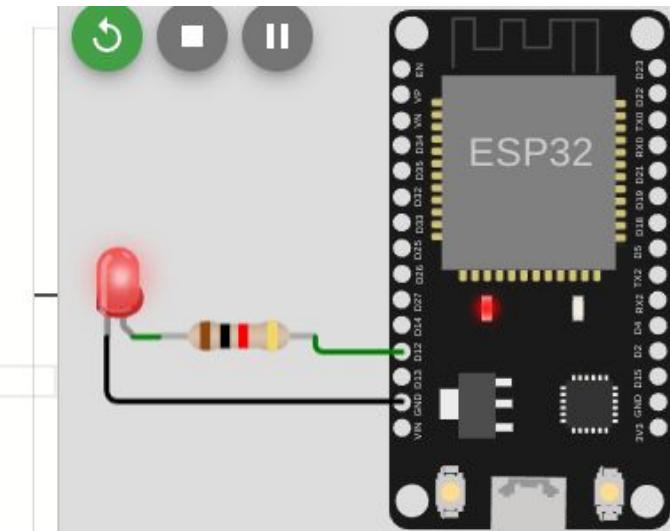
50% Duty Cycle



75% Duty Cycle



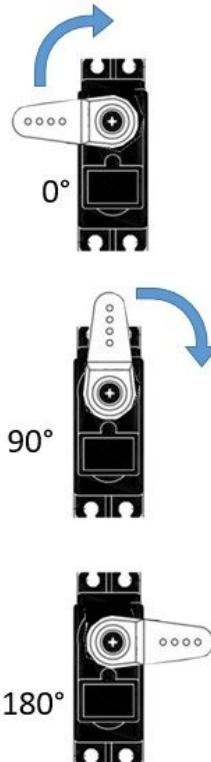
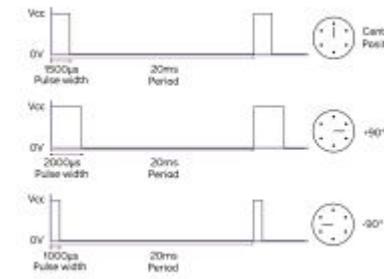
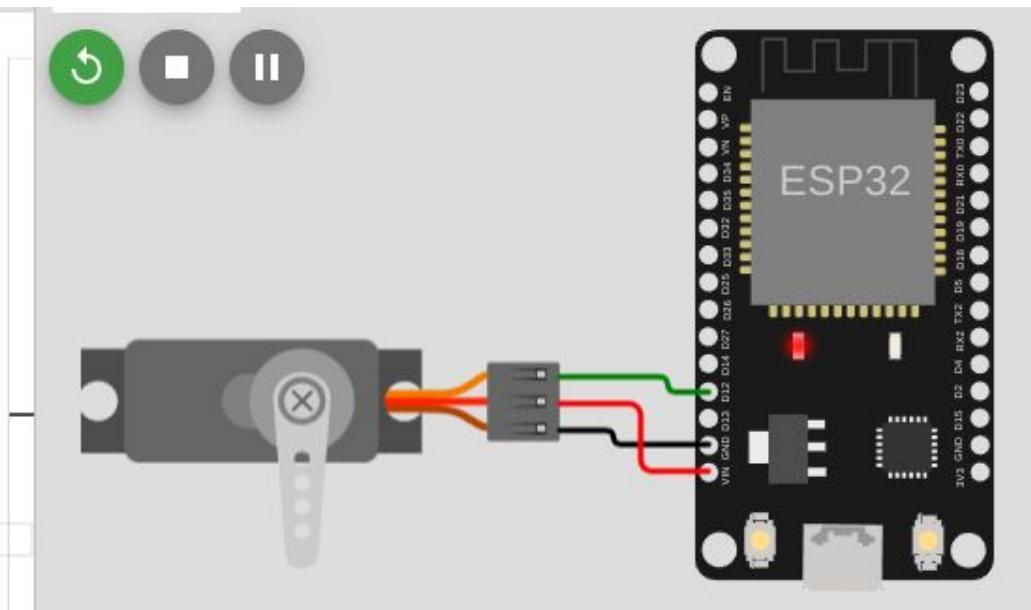
```
1 import machine
2 pin12 = machine.Pin(12)
3
4 pwmPin = machine.PWM(pin12)
5 pwmPin.freq(500)
6 #100% de intensidad
7 pwmPin.duty(1023)
8 #50% de intensidad
9 pwmPin.duty(512)
10 #10% de intensidad
11 pwmPin.duty(102)
```



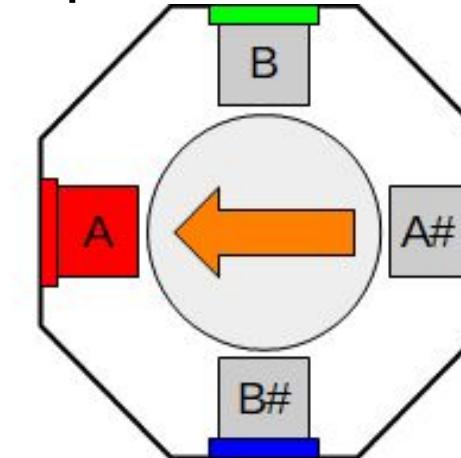
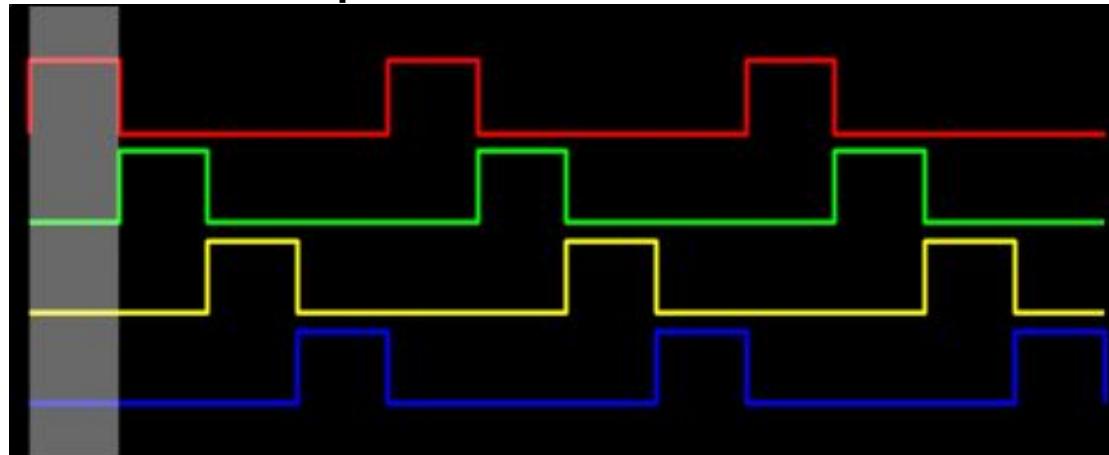
# Circuitos típicos - Servo con PWM

Los servos utilizan PWM para su control. Las especificaciones varían según el modelo, pero en wokwi podemos usar una frecuencia de 50Hz y luego variando el ciclo de actividad definimos la posición.

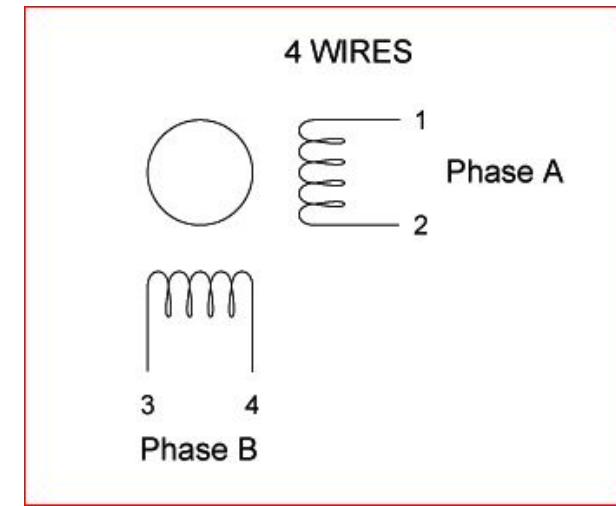
```
1 import machine
2 import time
3 pin12 = machine.Pin(12)
4
5 pwmPin = machine.PWM(pin12)
6 pwmPin.freq(50)
7 while True:
8     #angulo de 0 grados
9     pwmPin.duty(25)
10    time.sleep(3)
11    #angulo de 180 grados
12    pwmPin.duty(125)
13    time.sleep(3)
```



# Circuitos típicos - Motor Paso a Paso Bipolar



Existen varios tipos de motores paso a paso, pero en este tipo tenemos 2 bobinas, y eligiendo cual se encuentra energizada podemos hacer girar el eje el motor en un sentido. En el momento que dejamos de generar la secuencia el motor queda quieto.



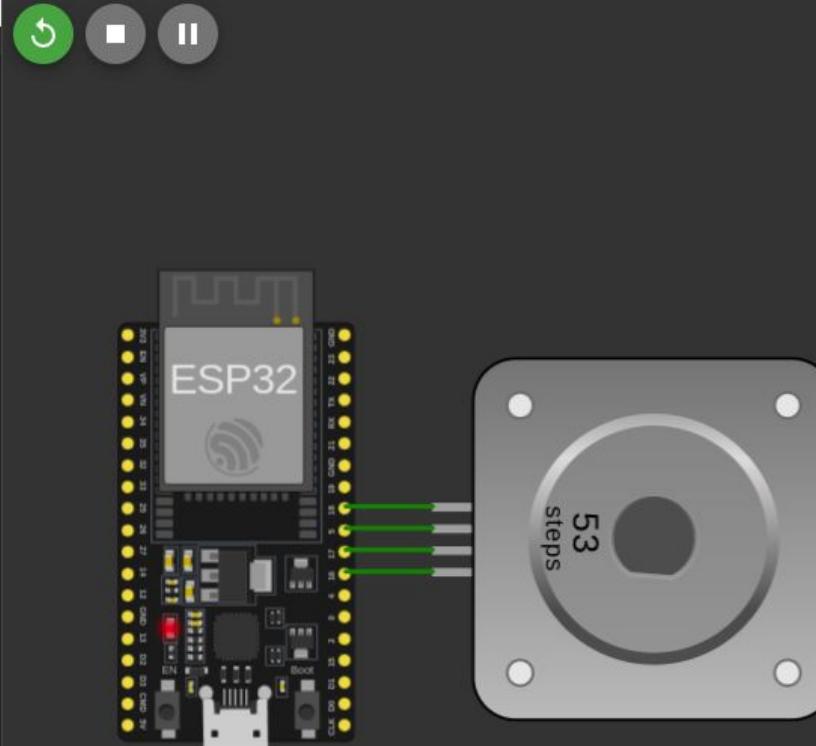
# Circuitos típicos - Motor Paso a Paso Bipolar

main.py

diagram.json

Simulation

```
1  from machine import Pin
2  import time
3  Am = Pin(18,Pin.OUT)
4  AM = Pin(5,Pin.OUT)
5  Bm = Pin(16,Pin.OUT)
6  BM = Pin(17,Pin.OUT)
7
8  while True:
9      AM.value(1)
10     BM.value(0)
11     Am.value(0)
12     Bm.value(0)
13     time.sleep_ms(100)
14
15     AM.value(0)
16     BM.value(1)
17     Am.value(0)
18     Bm.value(0)
19     time.sleep_ms(100)
20
21     AM.value(0)
22     BM.value(0)
23     Am.value(1)
24     Bm.value(0)
25     time.sleep_ms(100)
26
27     AM.value(0)
28     BM.value(0)
29     Am.value(0)
30     Bm.value(1)
31     time.sleep_ms(100)
32
```

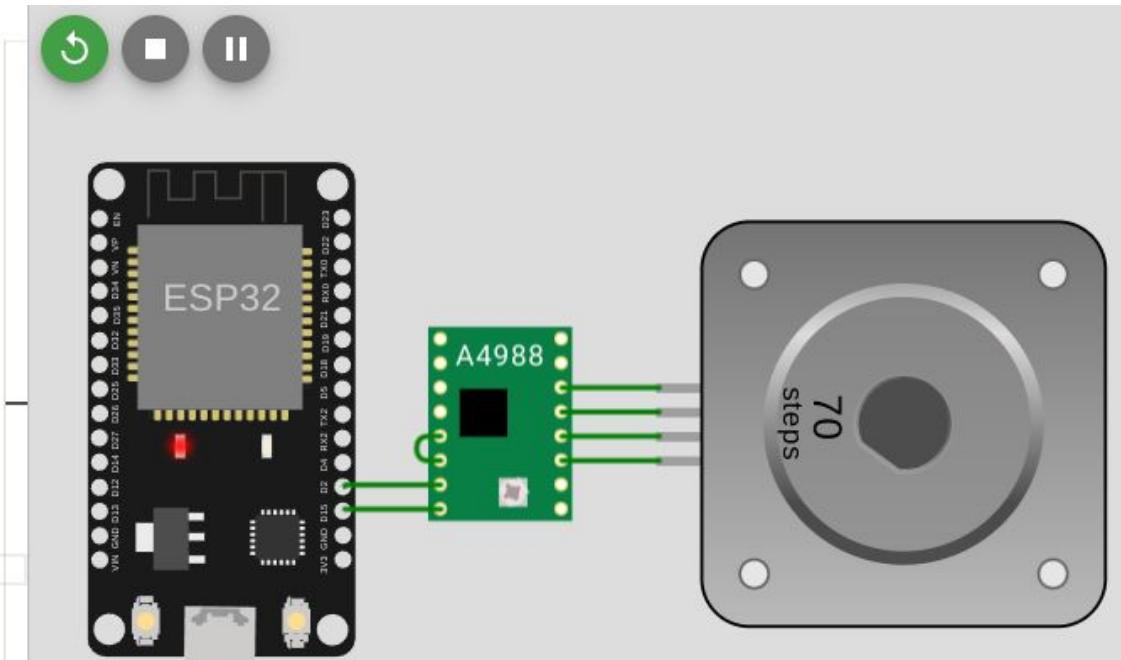


En un simulador esto funciona pero **en la vida real no podríamos alimentar las bobinas directamente del microcontrolador.** Existe como alternativa un Driver para este fin.

# Circuitos típicos - Motor Paso a Paso con Driver

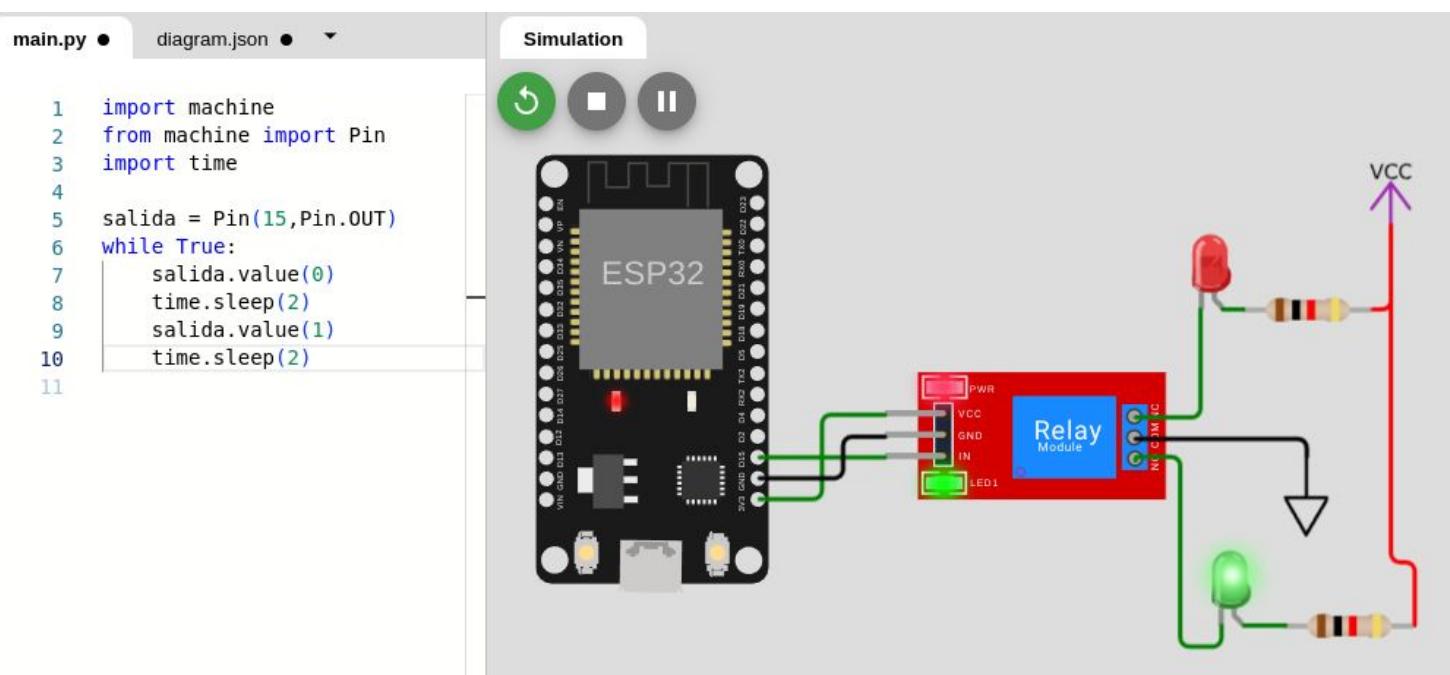
Un driver (A4988) genera la secuencia en las bobinas del motor paso a paso. Utilizando un pin de dirección indicamos horario/antihorario, y luego con un pin de paso le generamos pulsos para que avance un paso. Suelen ser 200 pasos por vuelta.

```
1 import machine
2 from machine import Pin
3 import time
4
5 paso = Pin(2,Pin.OUT)
6 dire = Pin(15,Pin.OUT)
7
8 while True:
9     dire.value(1)
10    paso.value(0)
11    pasos = 10
12    while pasos>0:
13        pasos = pasos -1
14        paso.value(1)
15        time.sleep_ms(5)
16        paso.value(0)
17        time.sleep_ms(5)
18    time.sleep(2)
19
```

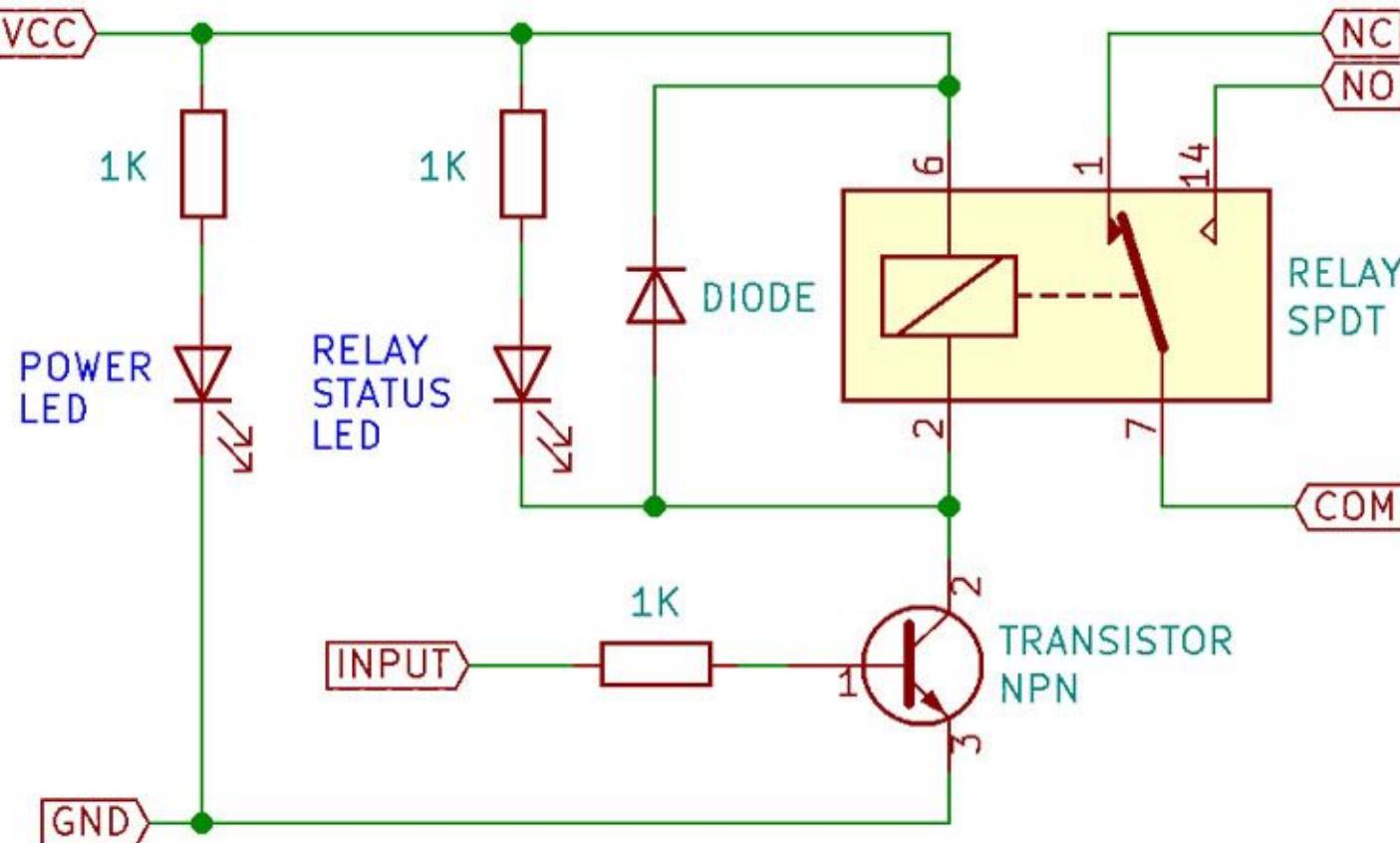


# Circuitos típicos - Relay

Un relay (visto en la unidad 1.2) utiliza una bobina para mover un contacto. Desde una salida digital (microcontrolador) podemos controlar otro circuito de forma aislada. Esto permite controlar elementos a mayor tensión. Su acción es mecánica y sufre desgastes. No son veloces.



# Circuitos típicos - Relay - Transistor NPN emisor común

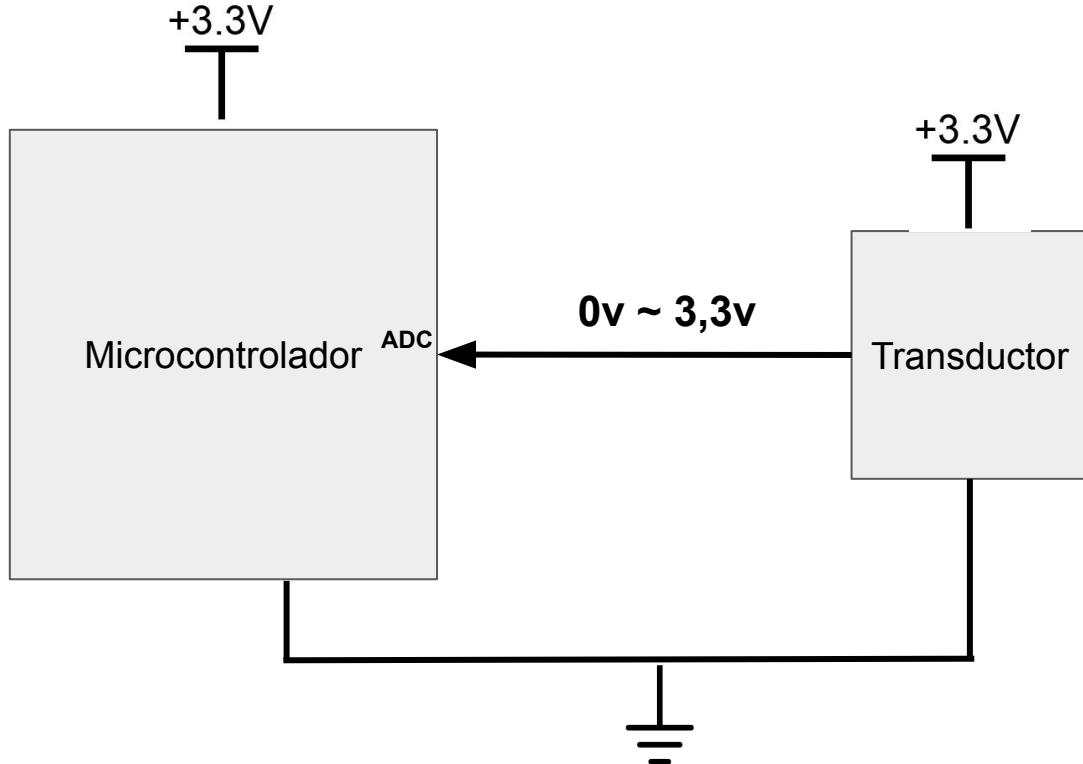


Relay Module Basic Schematic

# Entrada analógica

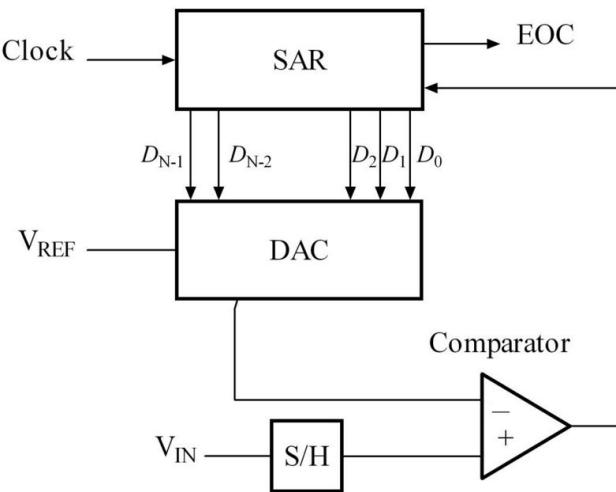
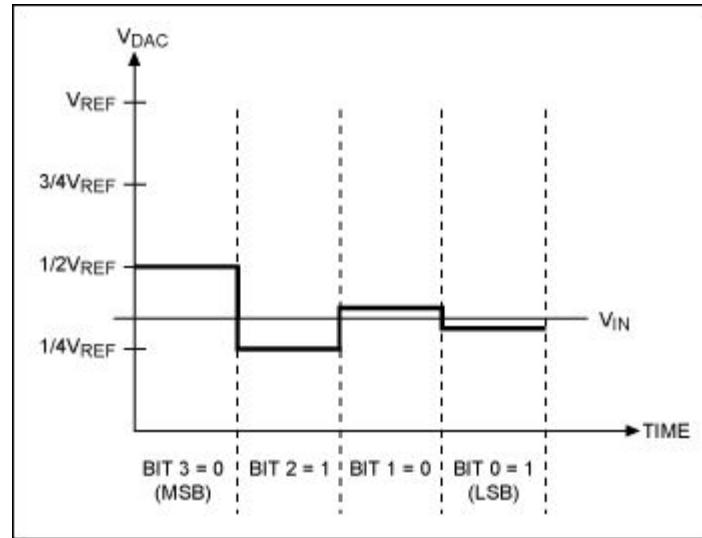
# Señales analógicas (Entrada)

El sensor/transductor convierte un atributo físico de la realidad (temperatura, luz, sonido, etc) a un equivalente en tensión. Medir esa tensión nos permite medir el fenómeno físico.

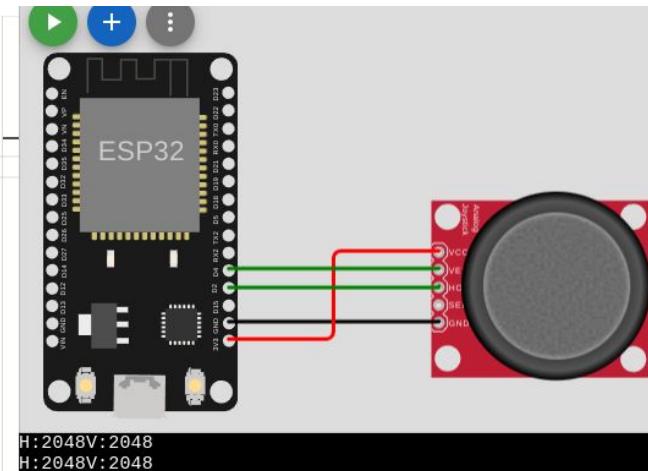


# Conversor A/D

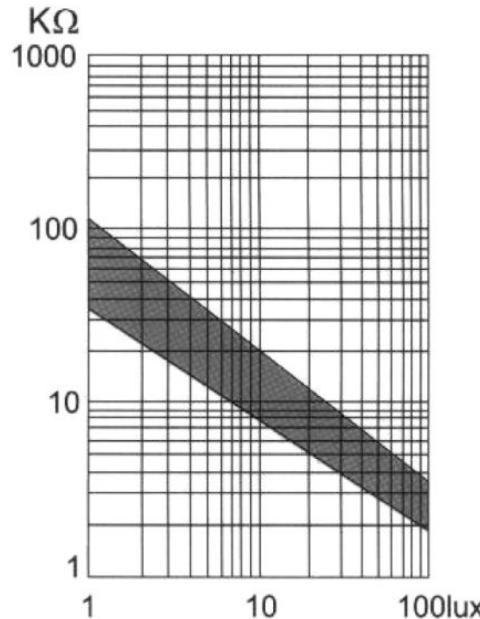
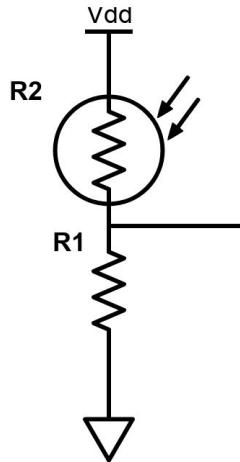
El conversor Analogico Digital convierte un valor de tensión (entre GND y VCC) a un número binario (comúnmente de 12 bits). La forma de generar el valor es por aproximaciones sucesivas. En 12 pulsos de reloj encuentra el valor más cercano que represente la tensión.



```
1 from machine import ADC
2 horizontal = ADC(2)
3 vertical = ADC(4)
4 while True:
5     valHor = horizontal.read()
6     valVer = vertical.read()
7     print("H:"+str(valHor)+"V:"+str(valVer))
```



# Sensor de Luz (LDR - Light Dependent Resistor)



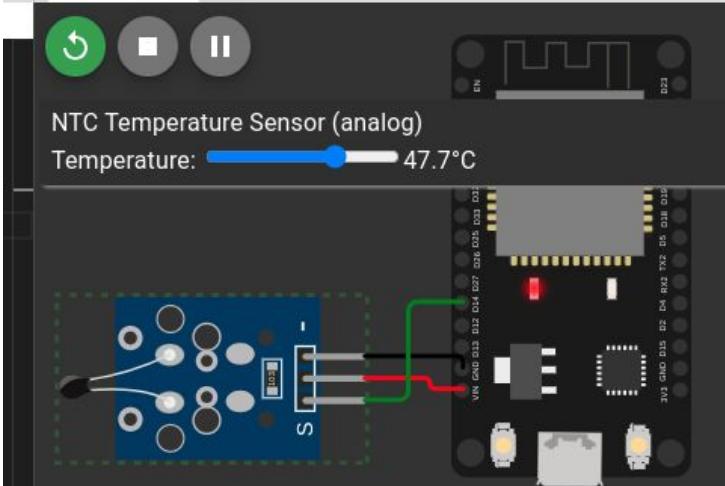
En el caso de Wokwi existe un módulo que tiene ya la resistencia y genera una salida de tensión analógica.

El elemento LDR es una resistencia que cambia su valor en ohms dependiendo de cuánta luz reciba. Es por ello que armamos un circuito con otra resistencia serie de forma tal que la tensión en la unión indique el nivel de intensidad de la luz.

```
1  from machine import ADC
2
3  luz = ADC(14)
4  while True:
5      valor = luz.read()
6      print("Luz="+str(valor))
7
```

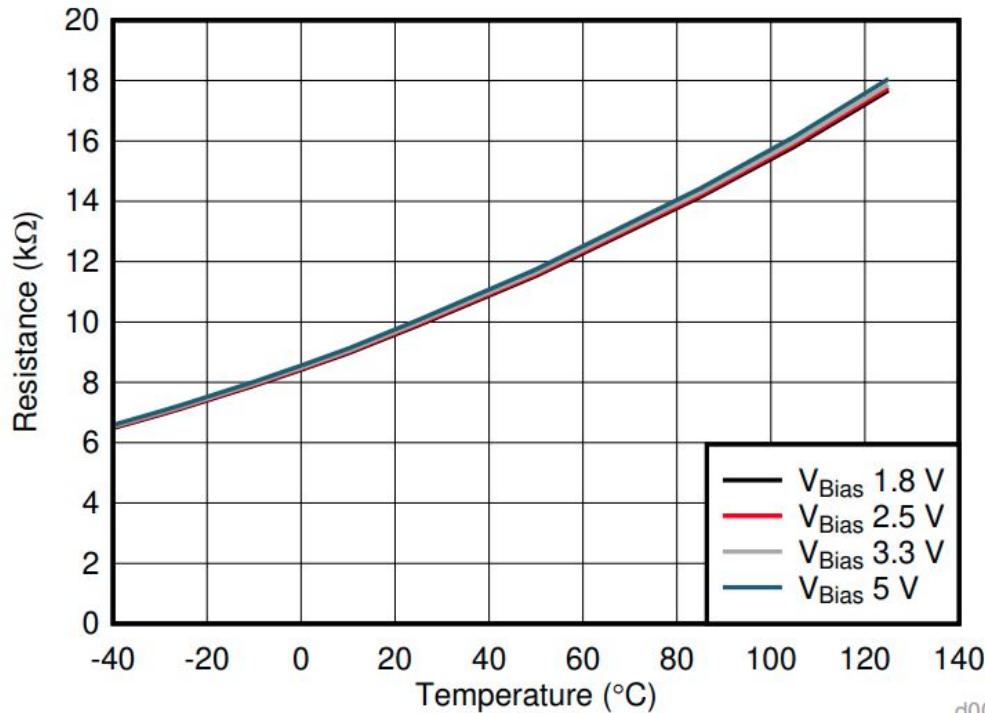
Photoresistor (LDR)  
ILLUMINATION (LUX)  
302 lux

# Sensor de Temperatura (NTC - Analógico)



Wokwi nuevamente provee una placa que ya tiene el circuito con la otra resistencia necesaria para el partidor resistivo.

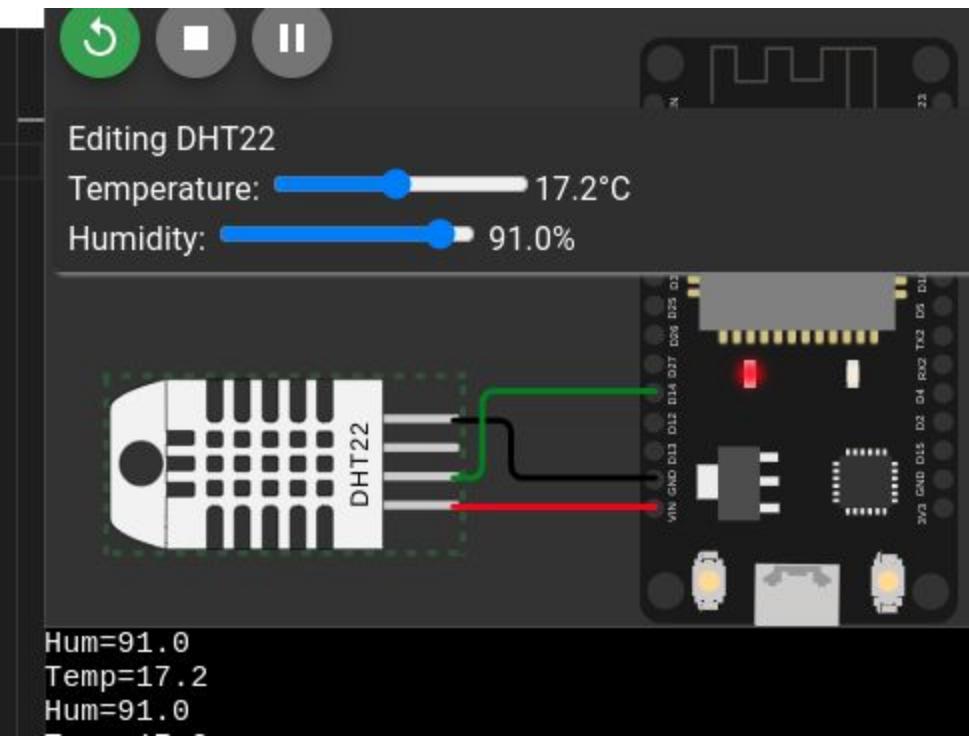
El elemento NTC es similar al LDR, pero mide el coeficiente de temperatura.



# Sensor de Temperatura (DHT22 - Digital)

El DHT22 es un sensor digital soportado por Micropython. Utilizando la biblioteca dht podemos medir, luego leer la temperatura y la humedad.

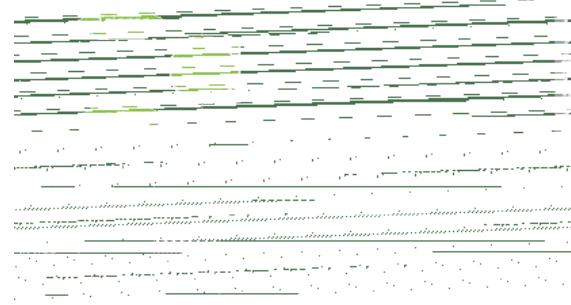
```
1 import dht  
2 import machine  
3 from machine import Pin  
4 sensor = dht.DHT22(Pin(14))  
5 while True:  
6     sensor.measure()  
7     temp = sensor.temperature()  
8     hum = sensor.humidity()  
9     print("Temp="+str(temp))  
10    print("Hum="+str(hum))  
11
```



# Sensor de Distancia por ultrasonido

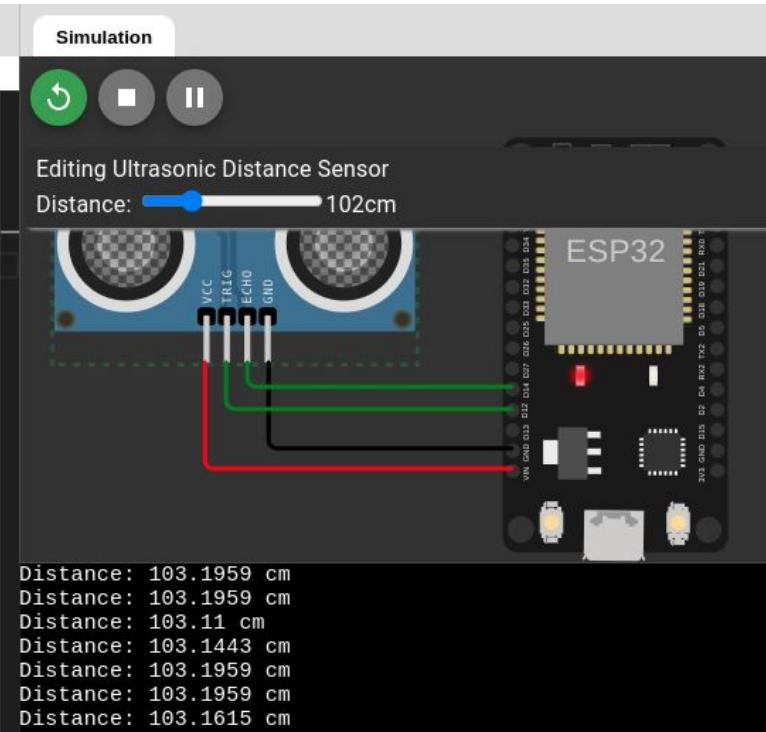
Utilizamos la biblioteca HCSR04.py

<https://github.com/rsc1975/micropython-hcsr04>



main.py ● diagram.json ● hcsr04.py ▾

```
1  from hcsr04 import HCSR04
2
3  sensor = HCSR04(trigger_pin=12, echo_pin=14)
4
5  while True:
6      distance = sensor.distance_cm()
7      print('Distance:', distance, 'cm')
```

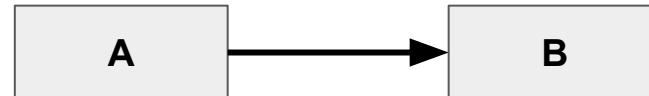


# Salidas de datos (I<sup>2</sup>C,SPI)

# Esquemas de comunicación

Cuando hablamos de comunicación nos referimos a dos (o más) dispositivos que intercambian **datos/información**.

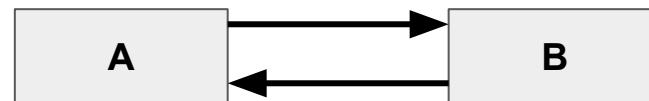
- Simplex



- Half Duplex

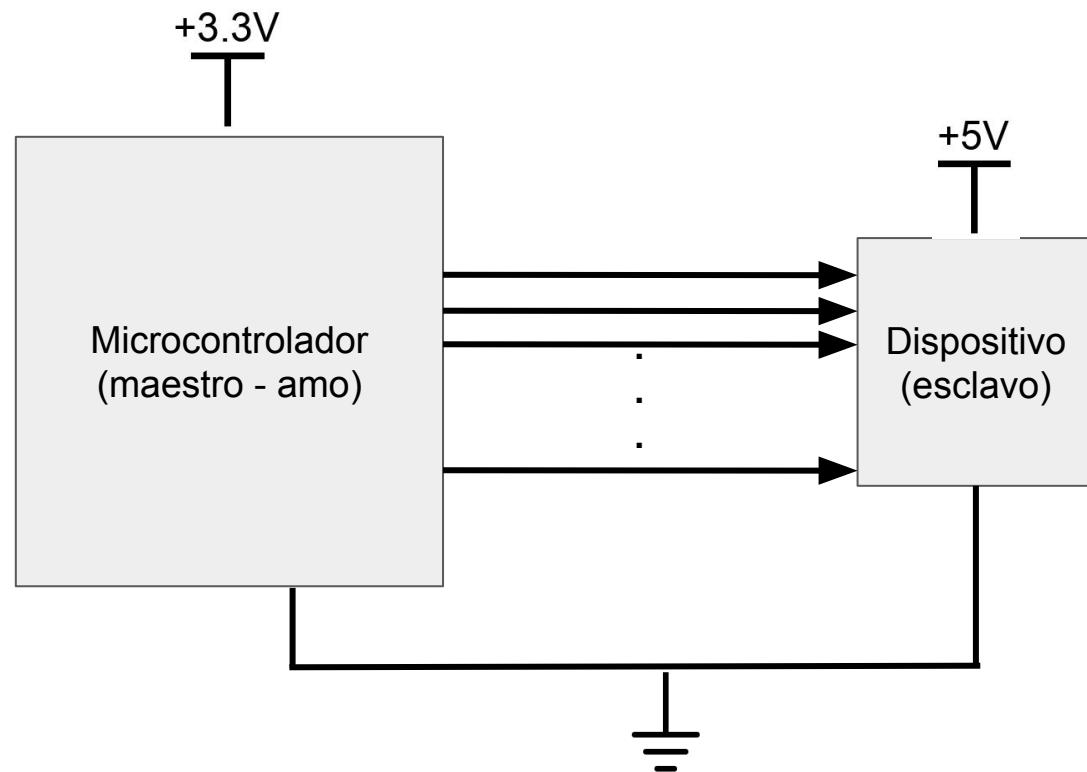


- Full Duplex



# Señales de datos en paralelo

Si enviamos datos en paralelo necesitamos una forma con la cual el maestro le indica al esclavo cuando recibir.

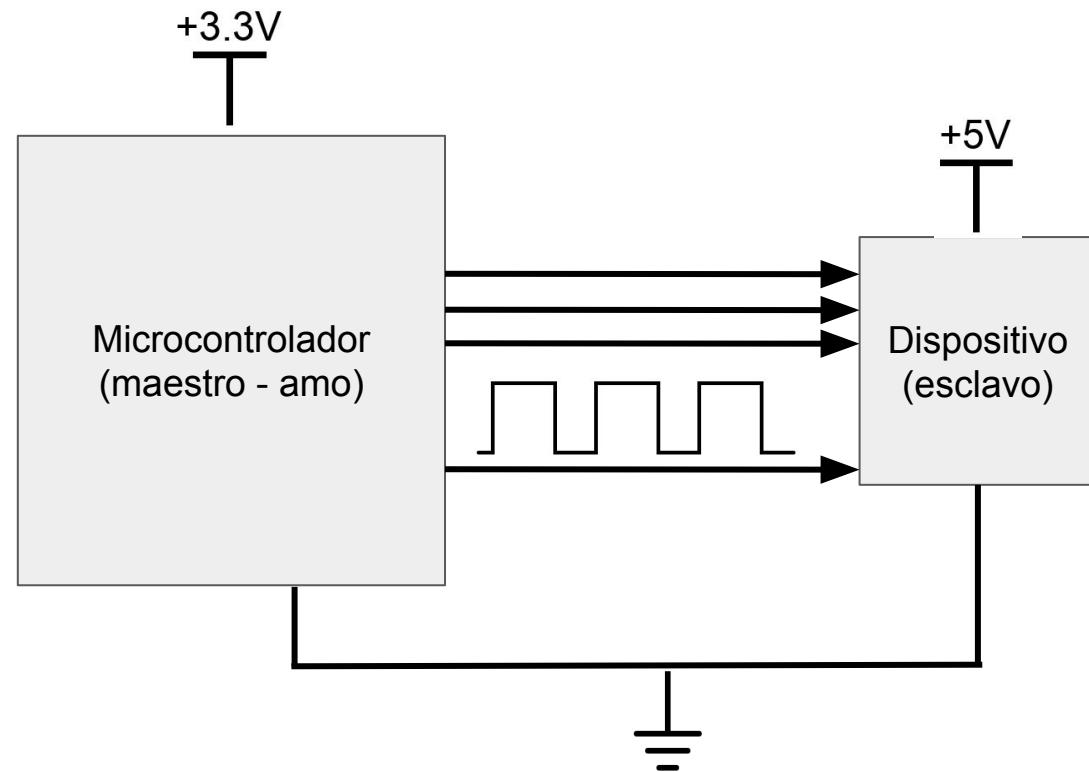


# Señales de datos en paralelo

Agregamos una señal que sincroniza (clock) y le indica al dispositivo cuando leer los datos.

Vemos que cada cable de datos se mide contra la referencia común (GND).

Estos cables en el mundo real son sometidos a ruidos y perturbaciones. Podemos incrementar la tolerancia al ruido en detrimento de la velocidad.

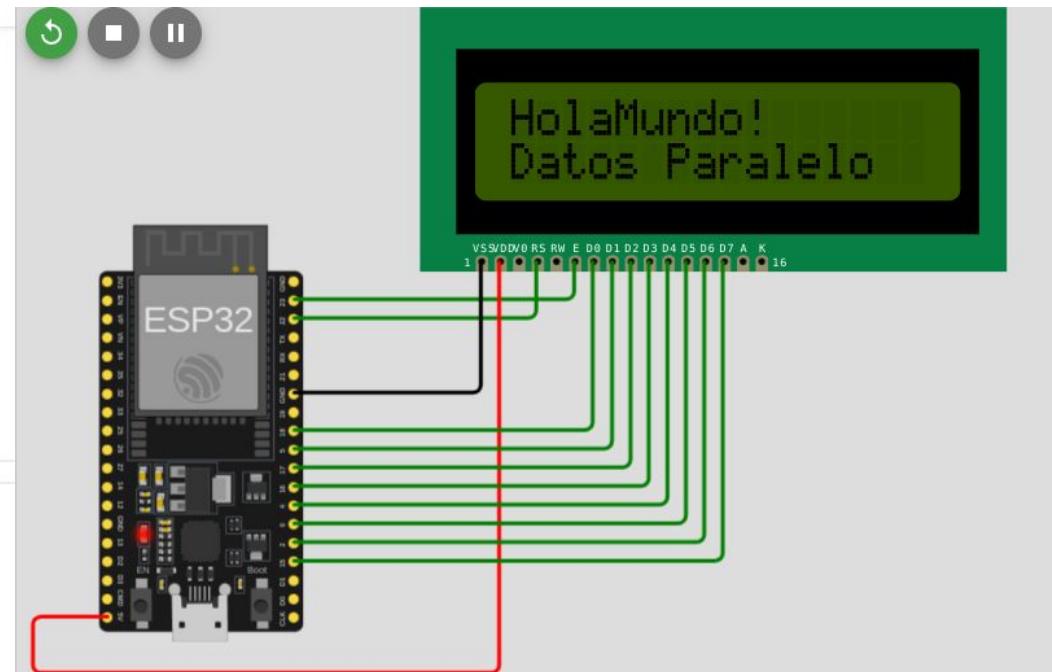


# Señales de datos en paralelo

Los displays alfanuméricicos basados en el HD44780 reciben datos en paralelo (4 u 8 bits) y utiliza una línea de clock (Enable) y una de tipo (RS).

<https://wokwi.com/projects/379745586218313729>

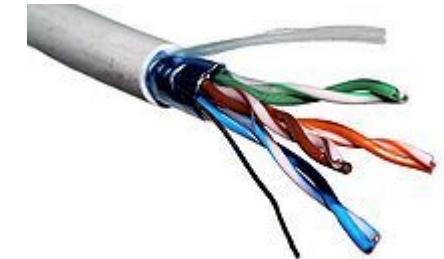
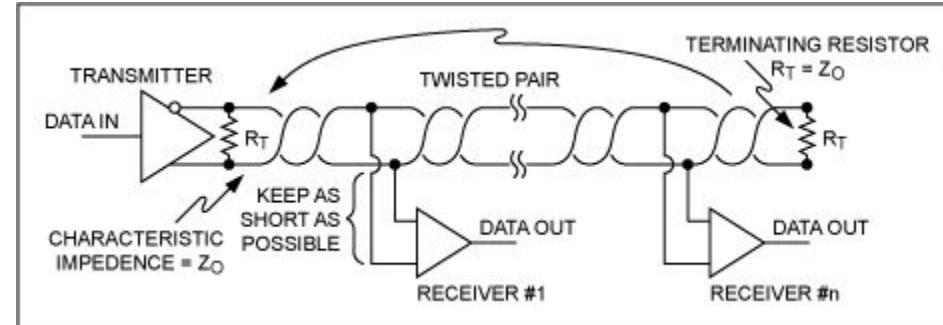
```
115
116     def udelay(self, us):
117         # Delay by us microseconds, set as function for portability
118         utime.sleep_ms(us)
119
120     def pin_action(self, pin, high):
121         # Pin high/low functions, set as function for portability
122         if high:
123             self.pins[pin].value(1)
124         else:
125             self.pins[pin].value(0)
126
127 display = LCD()
128 display.init()
129
130 display.set_line(0)
131 display.set_string("HolaMundo!")
132 display.set_line(1)
133 display.set_string("Datos Paralelo")
```



# Par trenzado

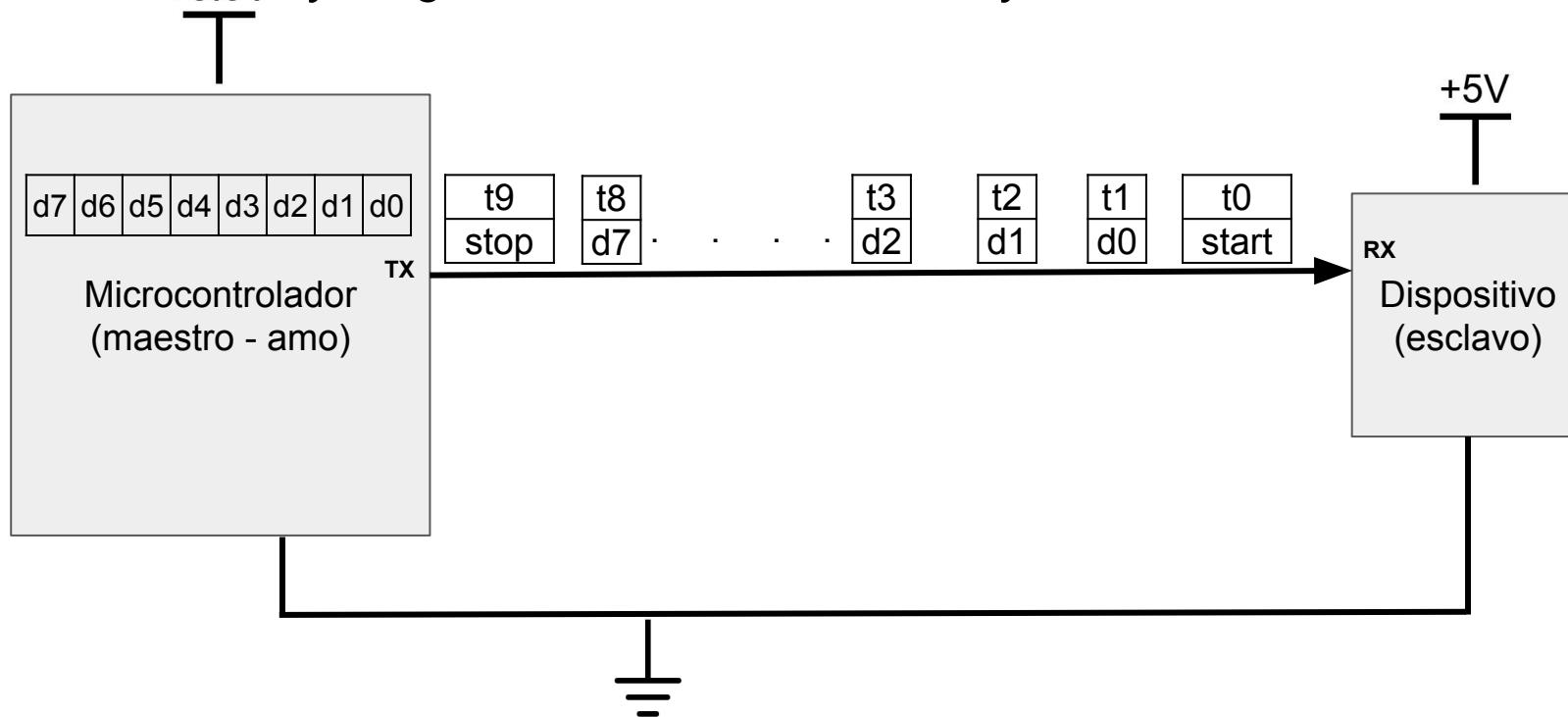
Utilizando cables con par trenzado, evitamos en gran medida el ruido. Esto se logra ya que medimos la diferencia de potencial entre cada par. El ruido va a afectar en igual medida a ambos cables del par por ende la diferencia de potencial entre ambos se mantiene.

Esto obviamente implica utilizar el doble de cables!. Un ejemplo de este tipo de conexión es el estándar LVDS utilizado en pantallas LCD donde se necesita velocidad pero en corta distancia.



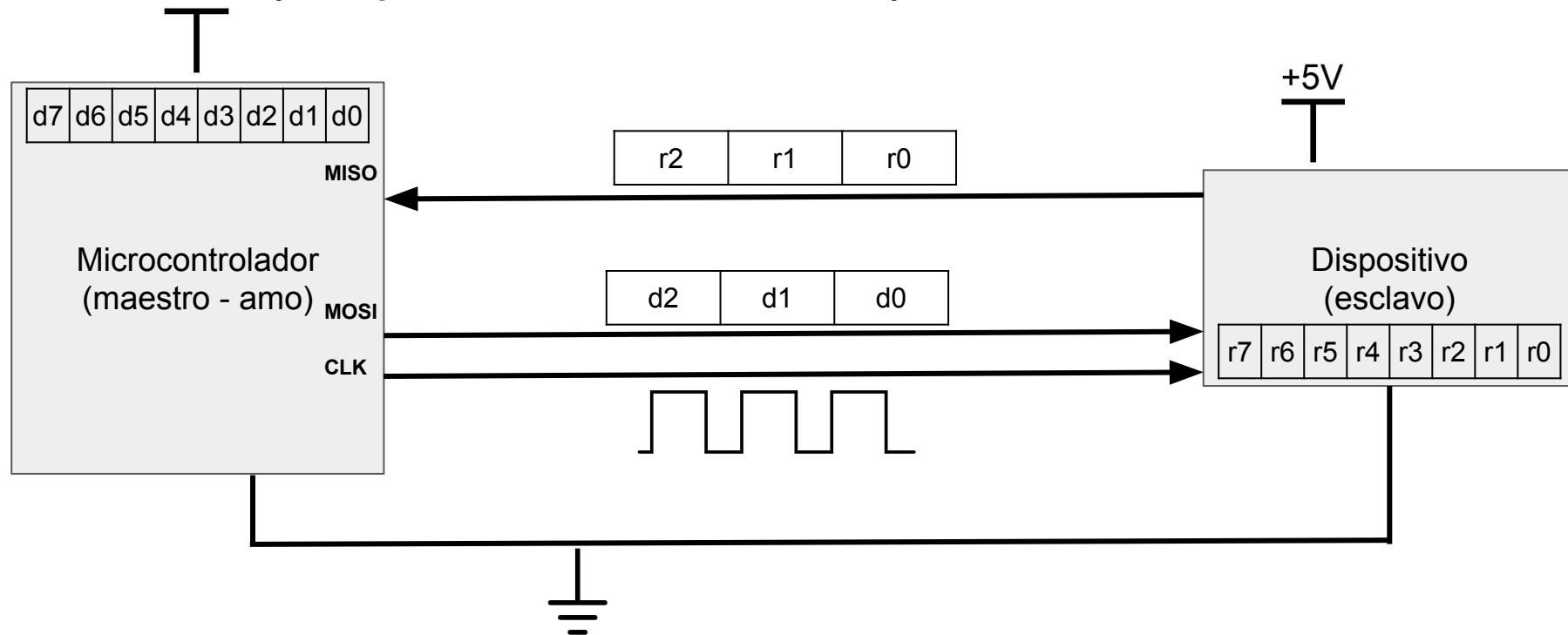
# Señales de datos en serie (asincrónico)

En vez de enviar todos los datos en paralelo, utilizamos más tiempo y enviamos los datos en serie (bit a bit). No hay clock sino un acuerdo de velocidad +3.3V y luego se indica el comienzo y el fin de la transmisión.



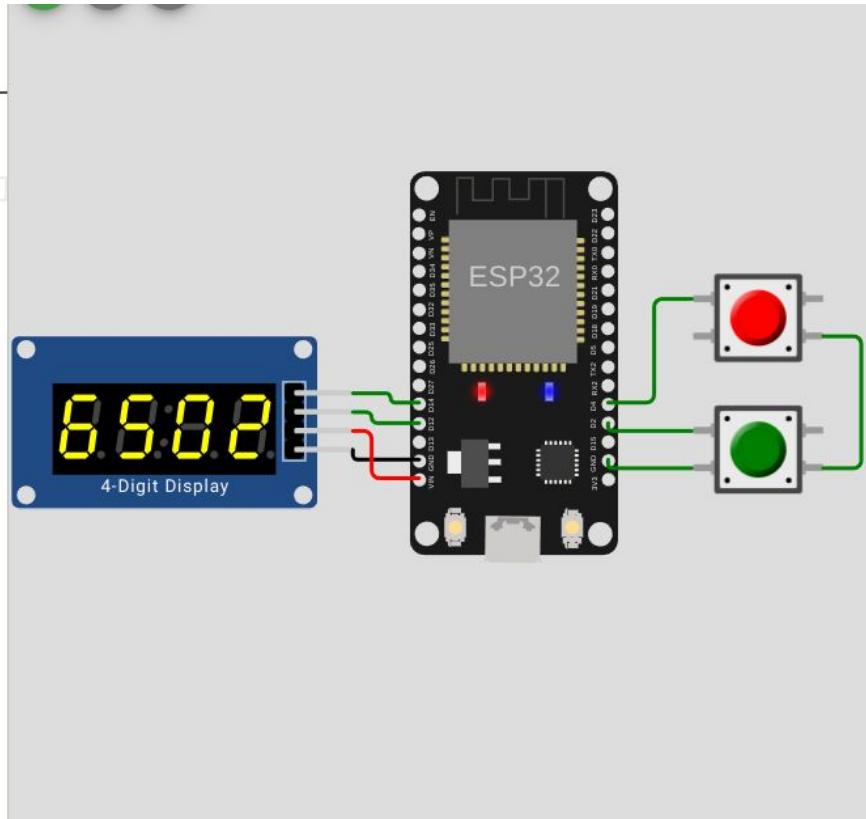
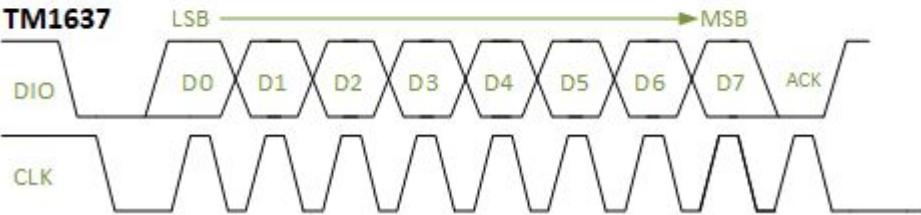
# Señales de datos en serie (sincrónico - SPI - Full Duplex)

En vez de enviar todos los datos en paralelo, utilizamos más tiempo y enviamos los datos en serie (bit a bit). No hay clock sino un acuerdo de velocidad +3.3V y luego se indica el comienzo y el fin de la transmisión.



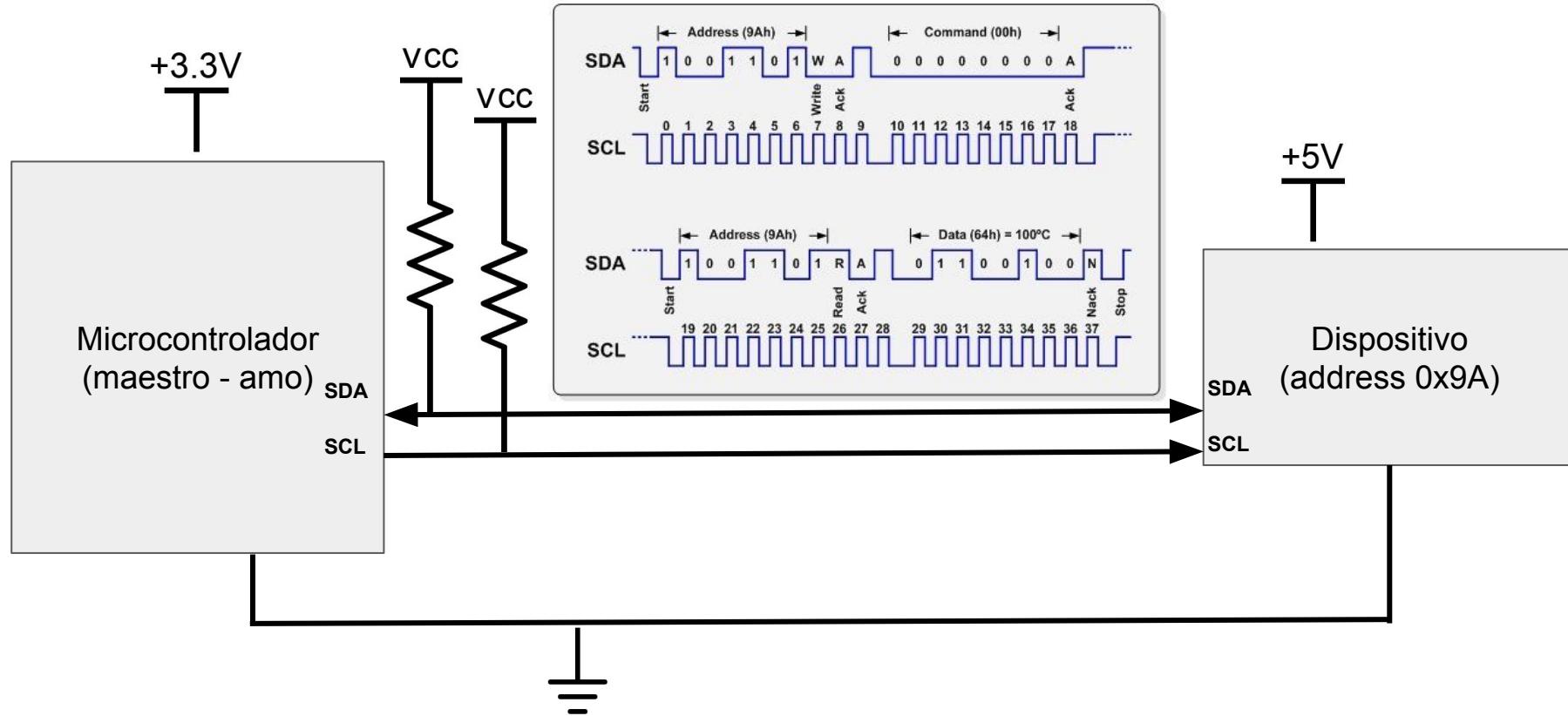
# TM1637 - Serie Síncrono

```
2 from machine import Pin
3 import time
4 import tm1637
5 from tm1637 import TM1637
6 display = TM1637(clk=Pin(14), dio=Pin(12))
7 pulsadorRojo = Pin(4,Pin.IN,Pin.PULL_UP)
8 pulsadorVerde = Pin(2,Pin.IN,Pin.PULL_UP)
9 contador=6502
10 display.number(contador)
11 estadoRojo=0 #iniciamos en estado 0
12 estadoVerde=0 #iniciamos en estado 0
13 while True:
14     time.sleep_ms(3) #forzamos esperas de 3ms
15     if not pulsadorRojo.value() and estadoRojo==0:
16         #Si el estado es 0 y el boton se aprieta...
17         #cambiamos de estado y actualizamos el contador
18         estadoRojo=1
19         contador=contador+1
20         display.number(contador)
21     if pulsadorRojo.value() and estadoRojo==1:
22         #Si el estado es 1 y el boton se suelta..
23         estadoRojo=0
24     #Ahora el verde
25     if not pulsadorVerde.value() and estadoVerde==0:
26         #Si el estado es 0 y el boton se aprieta...
27         #cambiamos de estado y actualizamos el contador
28         estadoVerde=1
29         contador=contador-1
30         display.number(contador)
31     if pulsadorVerde.value() and estadoVerde==1:
32         #Si el estado es 1 y el boton se suelta..
33         estadoVerde=0
34
```



# Señales de datos en serie (sincrónico - i<sup>2</sup>c - Half Duplex)

En i<sup>2</sup>c la línea de datos (SDA) es bidireccional.



# Display LCD alfanumérico usando I2C GPIO

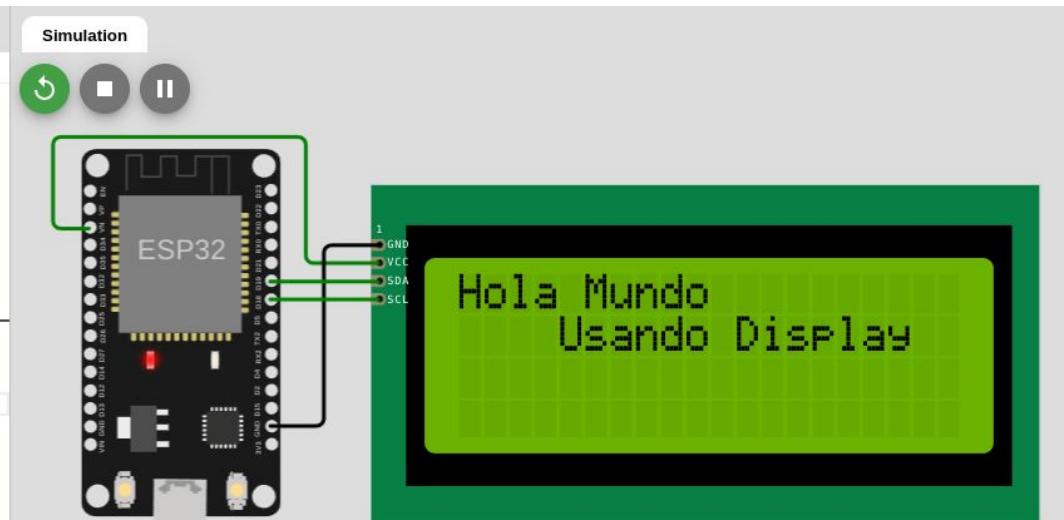
Para utilizar este tipo de display incluimos la biblioteca lcd\_i2c.py.

Recomendamos copiar los archivos de:

<https://wokwi.com/projects/379493618567996417>

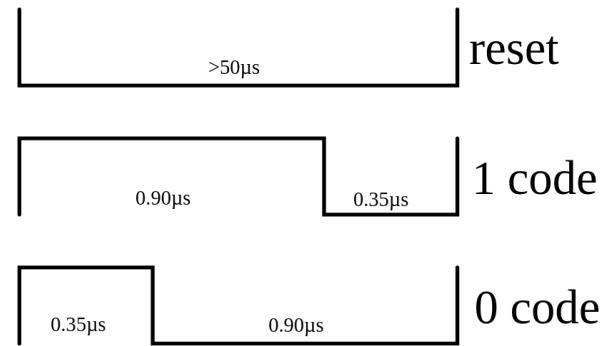
Estos displays utilizan el chip Hitachi HD44780 el cual requiere 6 pines de control, pero utilizando un conversor I2C a GPIO podemos utilizar solo 2 (SCL y SDA).

```
main.py • diagram.json • version.py • typing.py • lcd_i2c.py • const.py •  
1 from lcd_i2c import LCD  
2 from machine import I2C, Pin  
3  
4 I2C_ADDR = 0x27      # DEC 39, HEX 0x27  
5 i2c = I2C(0, scl=Pin(18), sda=Pin(19), freq=800000)  
6 lcd = LCD(addr=I2C_ADDR, cols=20, rows=4, i2c=i2c)  
7  
8 lcd.begin()  
9 lcd.print("Hola Mundo")  
10 #Pongo el cursor en fila 1 columna 4  
11 lcd.set_cursor(col=4, row=1)  
12 lcd.print("Usando Display")  
13  
14
```



# Tira de LEDs RGB tipo WS2812

Los LEDs RGB digitales tipo WS2812 reciben 24 bits (8 rojo, 8 verde, 8 azul) indicando la intensidad de cada color (entre 0 y 255). La biblioteca neopixel permite definir la cantidad de LEDs en la tira y de forma vectorial cargar cada color y con el comando write hace la transmisión.



A screenshot of a software interface showing a simulation of an ESP32 microcontroller connected to a NeoPixel ring. The code in the main.py file defines a NeoPixel object with pin 12 and 16 pixels, sets the first three pixels to red, green, and blue respectively, and then writes the configuration. The simulation window shows the ring with three colored segments and the ESP32 with its pins labeled. A diagram on the right shows the physical connection between the ESP32 pins and the NeoPixel ring.

```
main.py • diagram.json • ▾  
1 import machine, neopixel  
2 np = neopixel.NeoPixel(machine.Pin(12), 16)  
3 np[0] = (255, 0, 0) #Rojo  
4 np[1] = (0, 255, 0) #Verde  
5 np[2] = (0, 0, 255) #Azul  
6 np.write()  
7
```

# Elementos de sistemas embebidos

Unidad 6.2  
WIFI

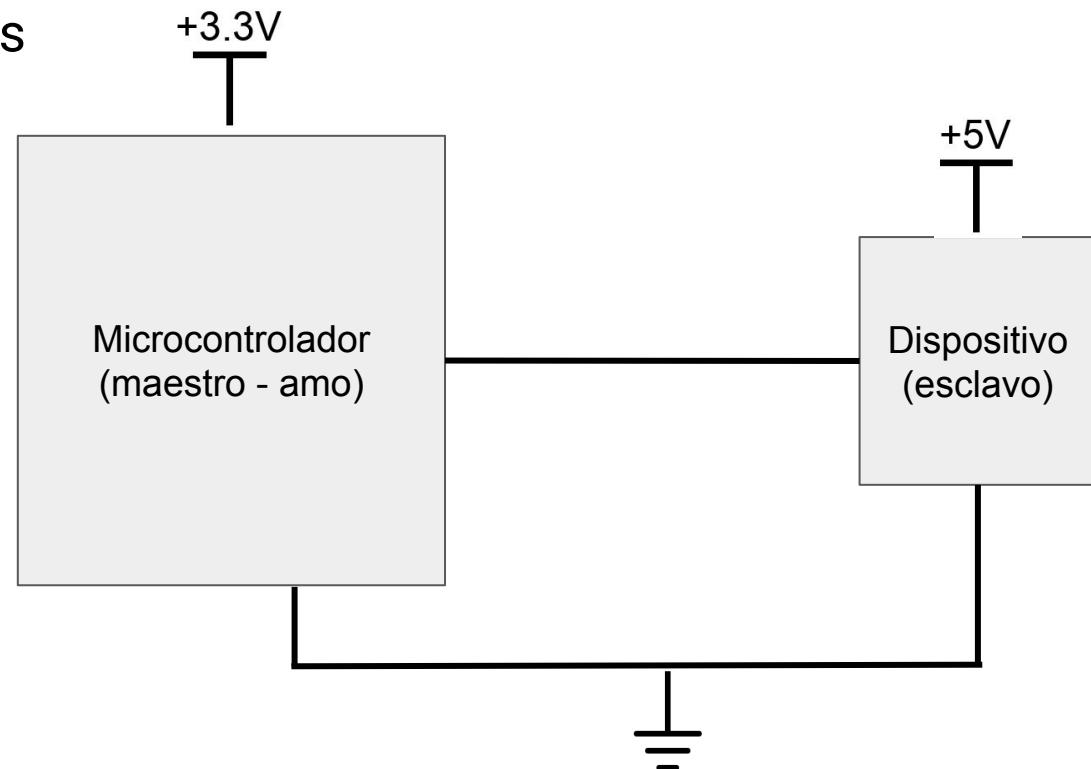
Versión 1.0.0

Jaír Hnatiuk, Carlos Maidana, Carlos Rodríguez, Edgardo Gho, Martín Ferreyra Biron

# Comunicación cableada

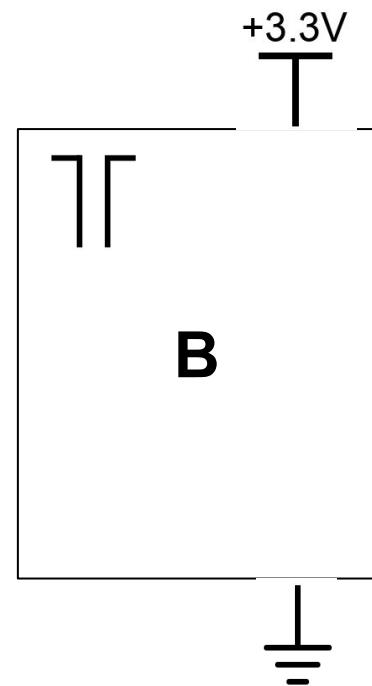
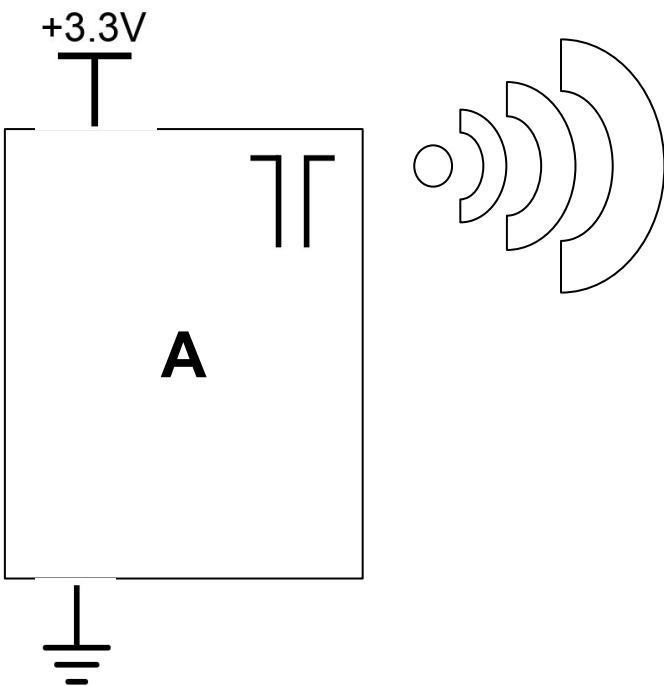
Hasta ahora hemos comunicado dos (o más) dispositivos señalizando con tensión en un circuito cerrado.

Vimos que hay varias formas de lograr esto, pero en cualquiera de las formas debe existir un vínculo mediante un conductor que permita la comunicación.

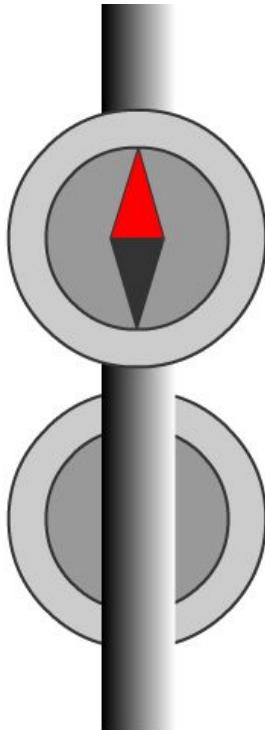


# Comunicación inalámbrica

Mediante dos antenas podemos comunicar mediante un vínculo electromagnético

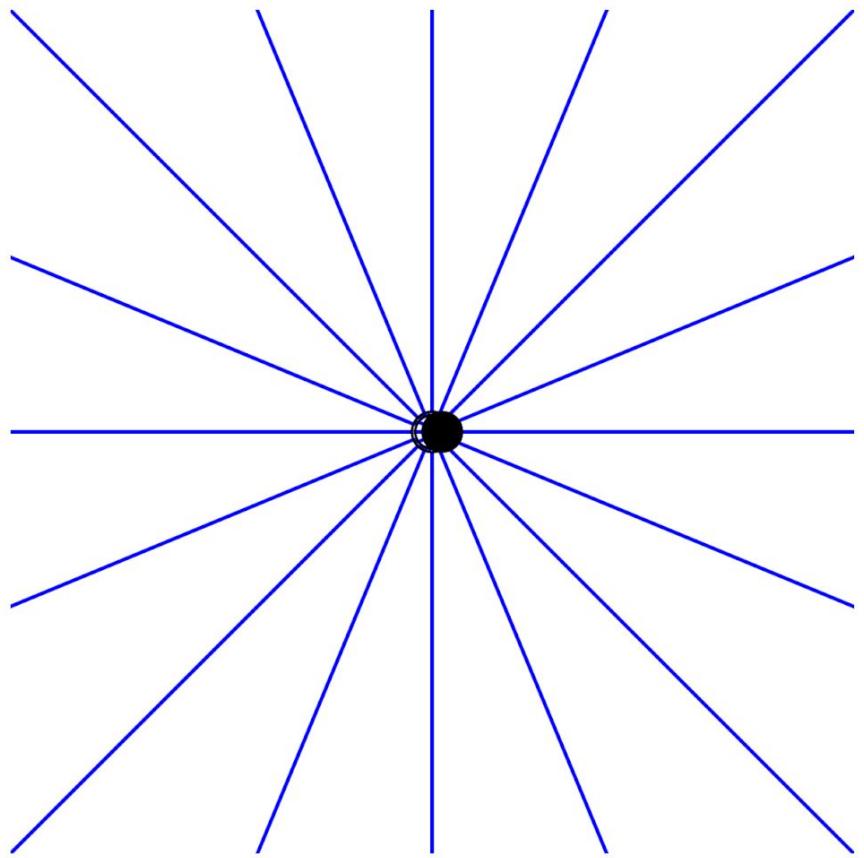


La corriente en un cable genera un campo electromagnético



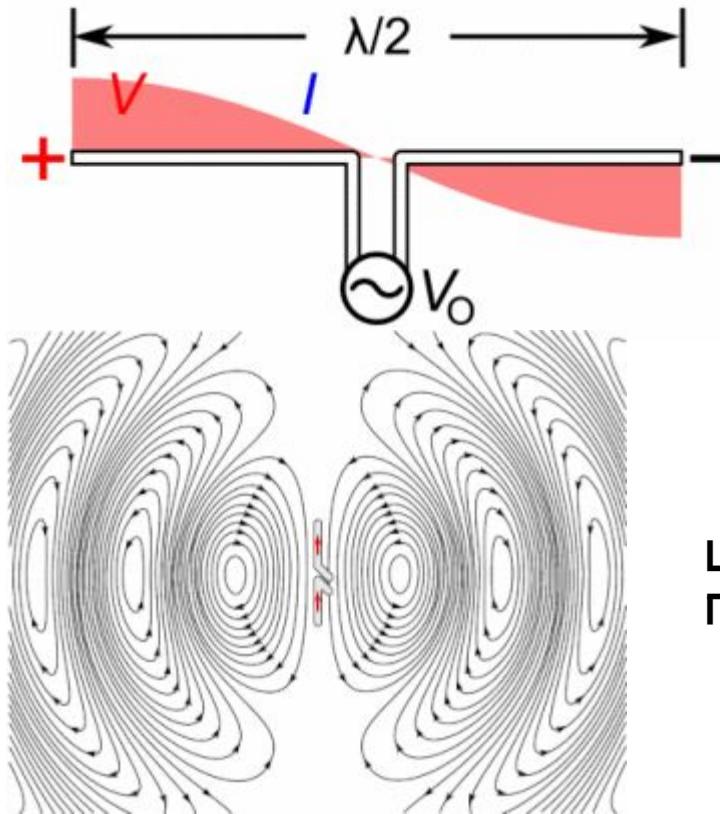
No  
Current

Si aceleramos una carga, el campo emitido se corrige



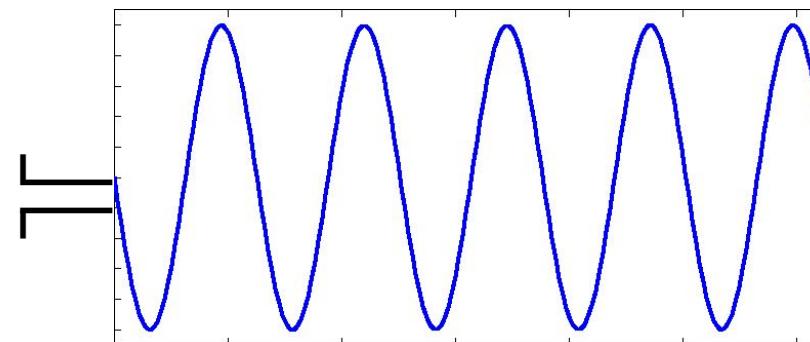
# Antena Dipolo

Si en la antena dipolo aplicamos una frecuencia, se irradia una onda cuya intensidad depende de la potencia aplicada.



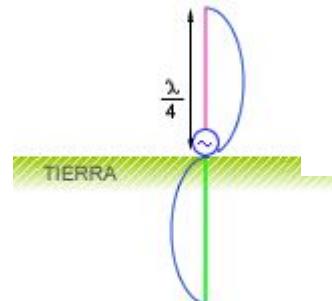
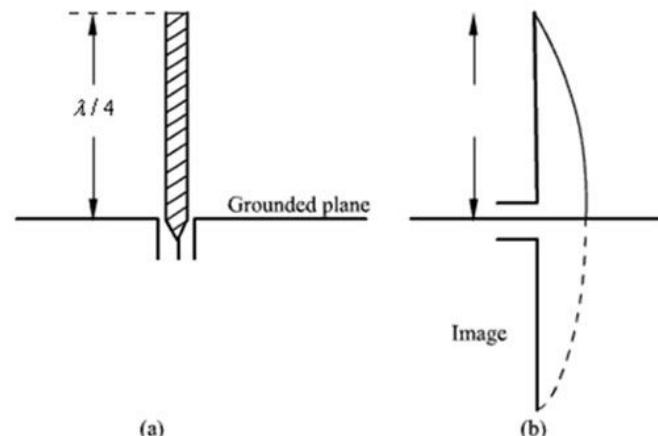
La longitud de onda (metros) se calcula como la velocidad de la luz sobre la frecuencia. Esto define el tamaño de la antena.

Luego otra antena (idéntica) que recibe esa onda puede regenerar la señal eléctrica.



# Antena Monopolio (Marconi)

Podemos diseñar una antena de un cuarto de longitud de onda utilizando un plano de tierra.

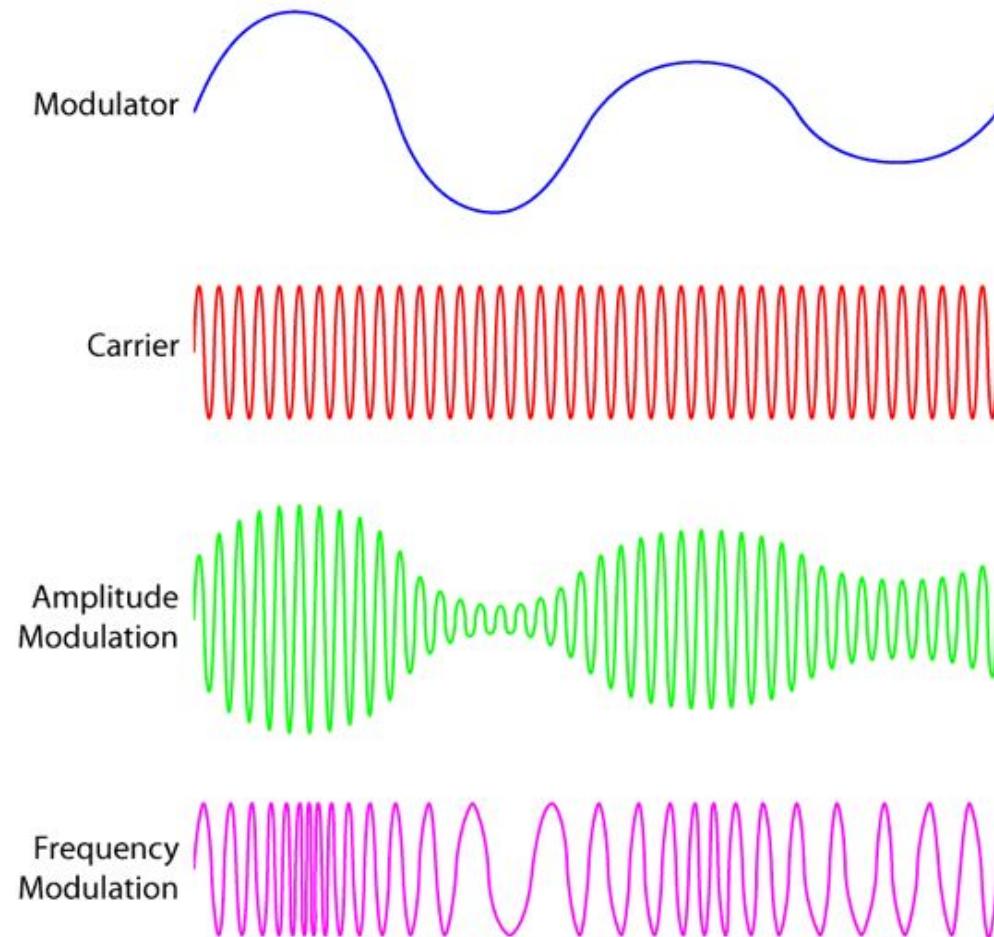


# Antena chip o PCB

Podemos encontrar diseños de antenas que a muy alta frecuencia pueden irradiar utilizando como elemento un diseño plano sobre el PCB (placa de circuito impreso). Estas son las más comunes en dispositivos embebidos como kits de desarrollo, celulares, computadoras, etc.

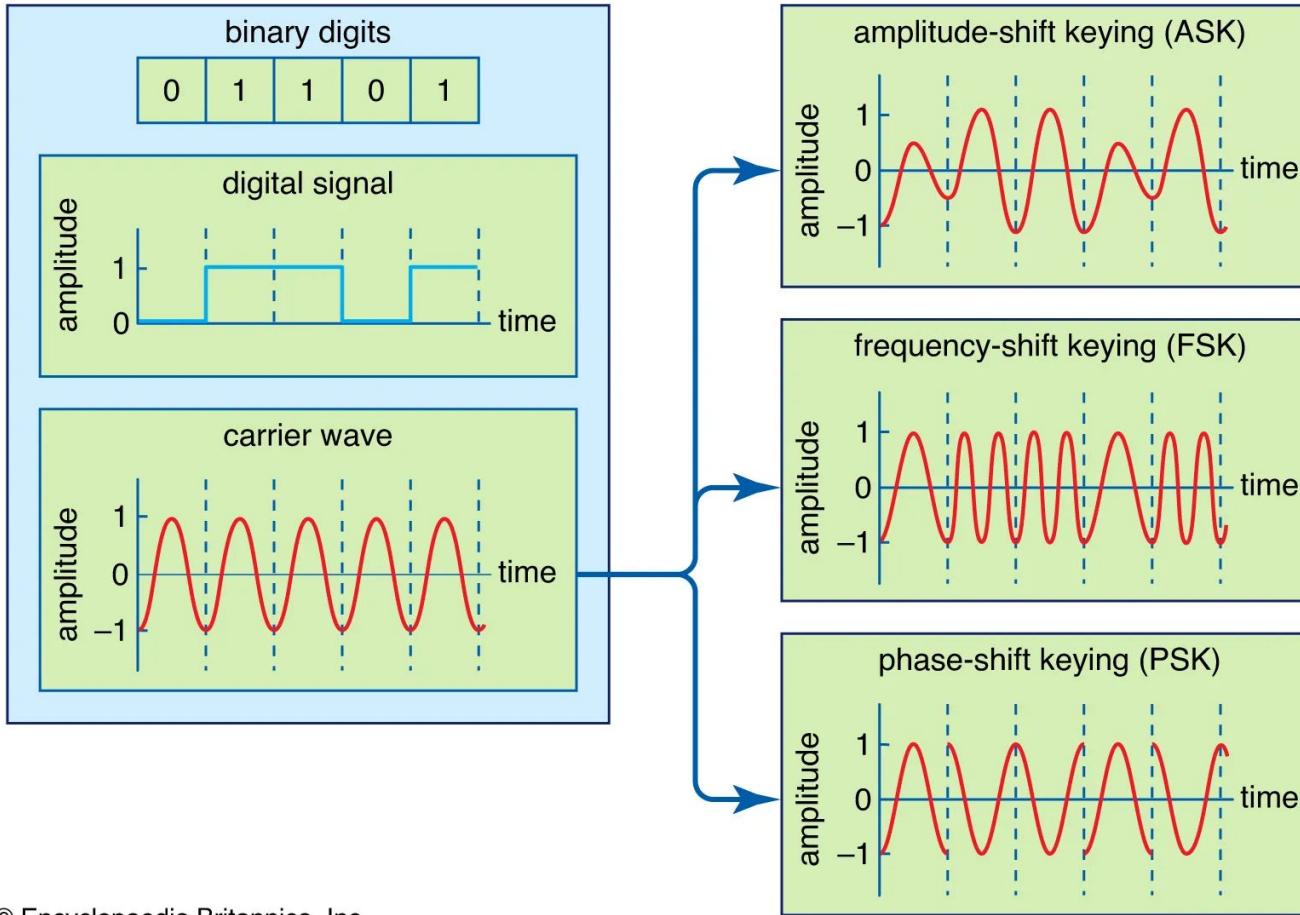


# Amplitud modulada / frecuencia modulada

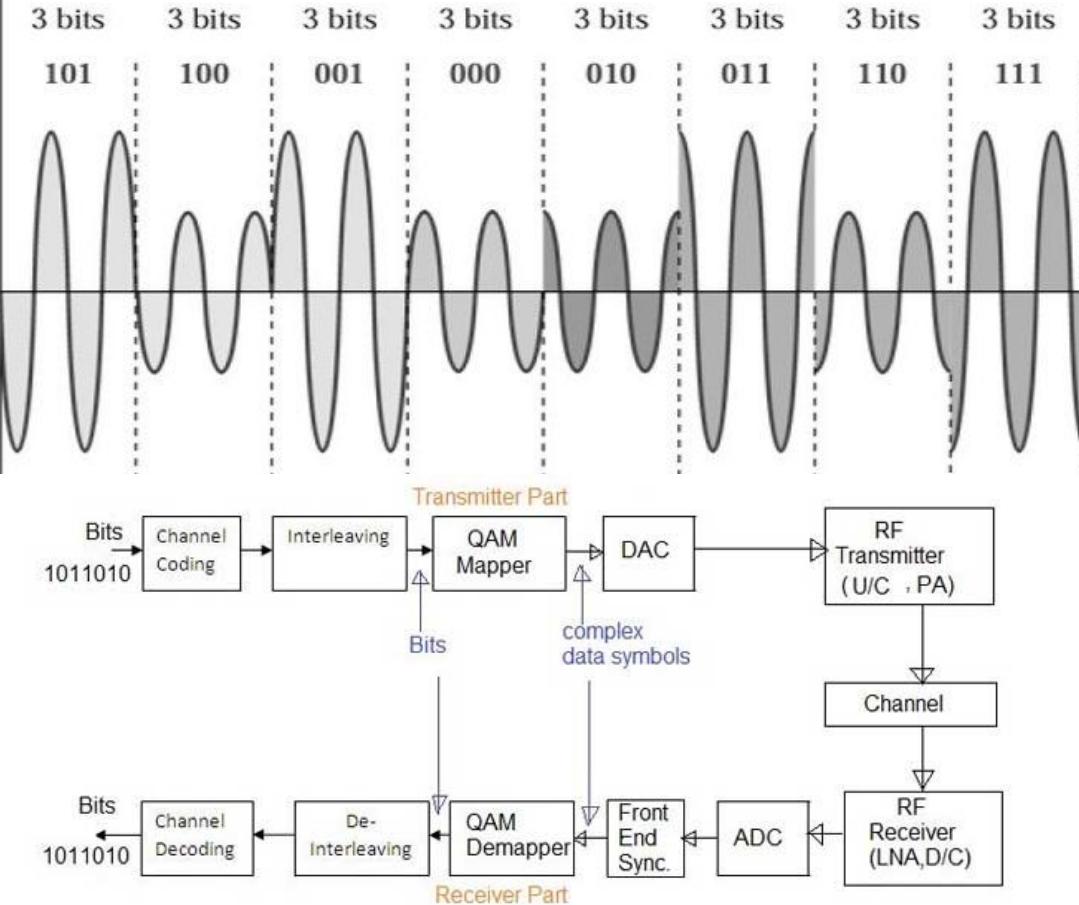


Para enviar información (ej: audio), utilizamos una frecuencia portadora (ej: AM 630 indica que la frecuencia es 630KHz). Combinando la portadora y la señal de audio generamos una nueva señal modulada. Si combinamos de forma que varíe la amplitud, obtenemos AM. Podemos también modular en frecuencia , obtenemos FM. También podríamos modular en cambio de fase (PSK).

# Datos digitales



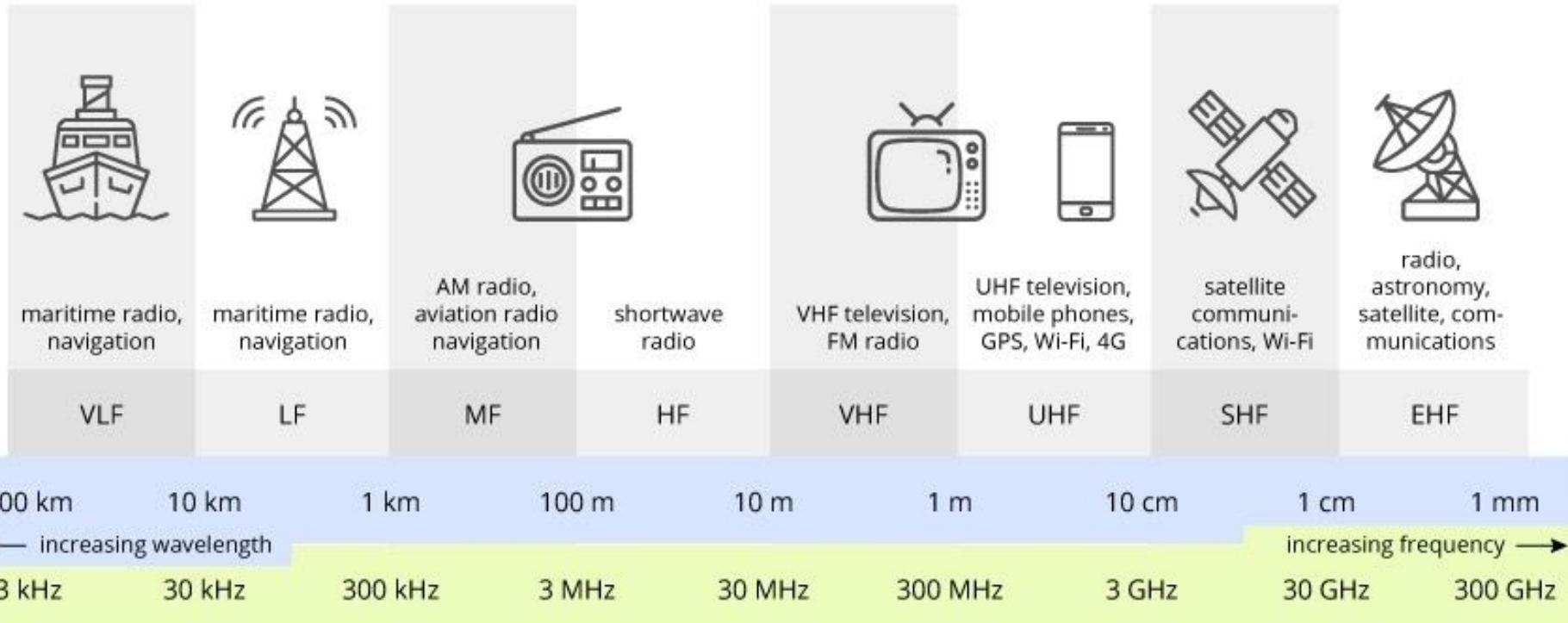
# QAM (Quadrature Amplitude Modulation)



Si modulamos en amplitud, frecuencia y fase a la vez, representamos en cada instante más información, por ende aumentamos la tasa de datos.

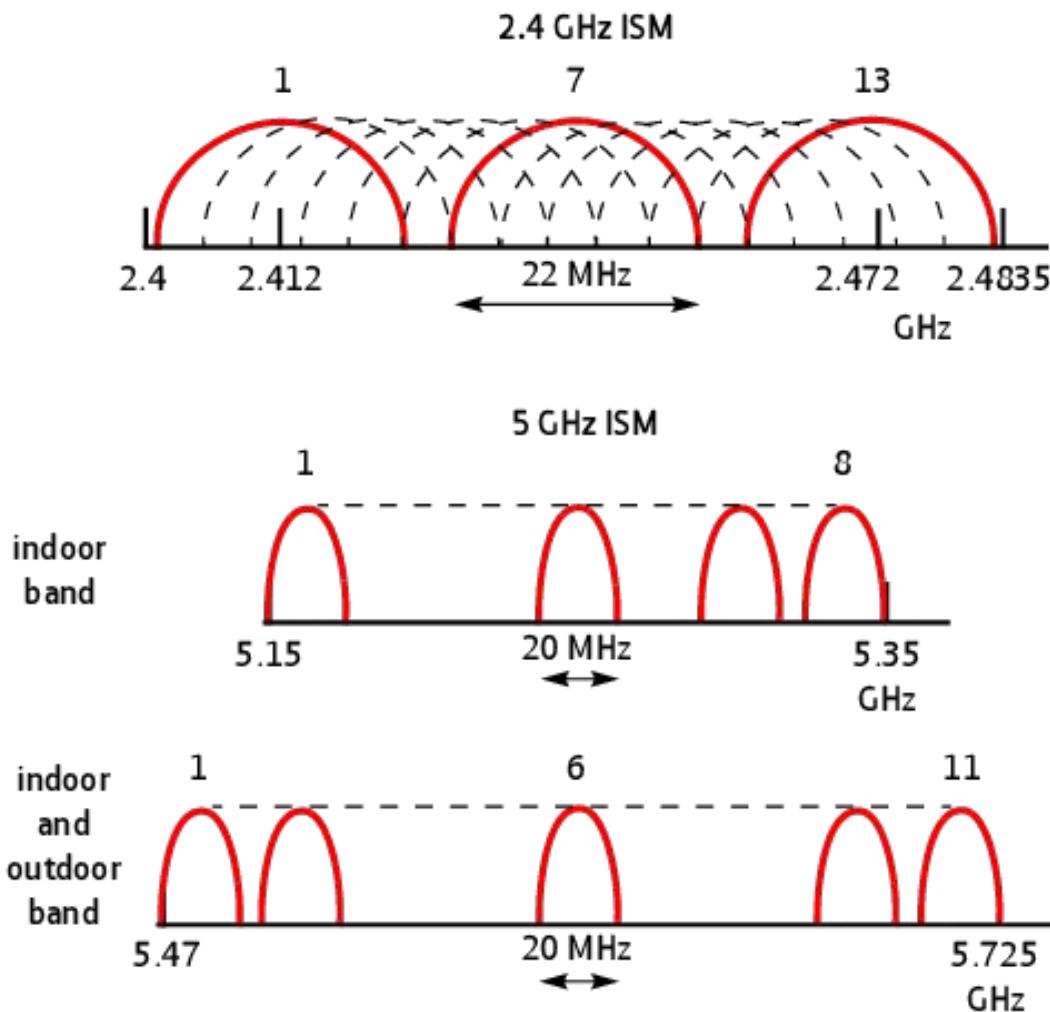
Nuestro sistema ahora consiste de un transmisor, que toma bits, los codifica, modula y transmite. Luego el receptor hace los pasos inversos y devuelve los bits originales.

# El espectro se encuentra regulado y sectorizado



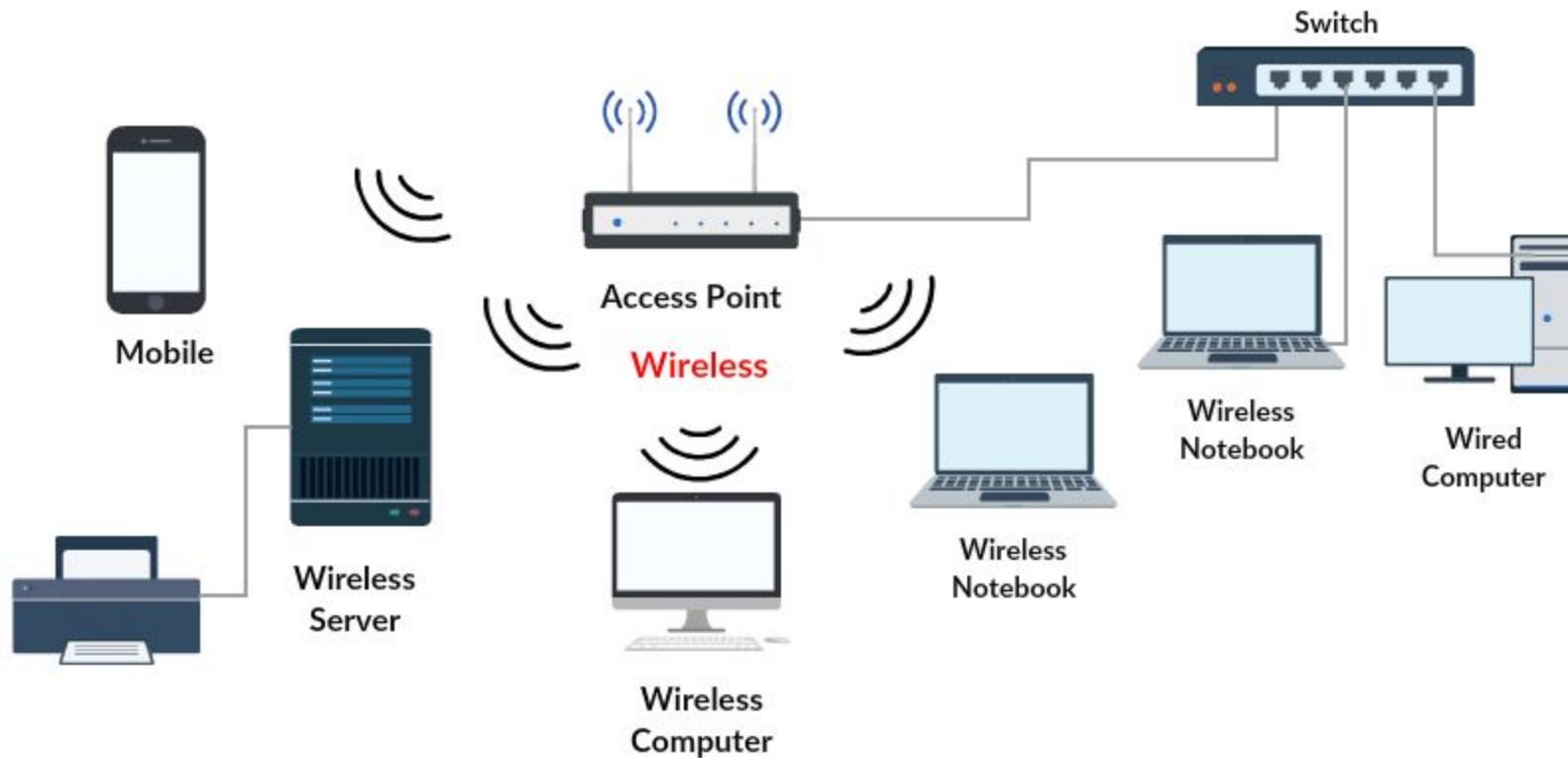
# WIFI

En el caso de WIFI, hay dos rangos de frecuencias autorizados, el de 2,4GHz y el de 5GHz. En cada caso se definen canales con un ancho de banda. El ancho de banda define a qué frecuencia puede cambiar la información montada sobre la portadora. Existe solapamiento entre algunos canales lo cual genera interferencia.



# WIFI - Infraestructura (IEEE 802.11)

El estándar 802.11 define como funciona la comunicación por wifi. Existen varias versiones de este estándar (b/g/n/ac..etc).



# WIFI - Términos definidos por la norma

**3.3 access point (AP):** Any entity that has station (STA) functionality and provides access to the distribution services, via the wireless medium (WM) for associated STAs.

**3.8 association:** The service used to establish access point/station (AP/STA) mapping and enable STA invocation of the distribution system services (DSSs).

**3.9 authentication:** The service used to establish the identity of one station (STA) as a member of the set of STAs authorized to associate with another STA.

**3.136 station (STA):** Any device that contains an IEEE 802.11-conformant medium access control (MAC) and physical layer (PHY) interface to the wireless medium (WM).

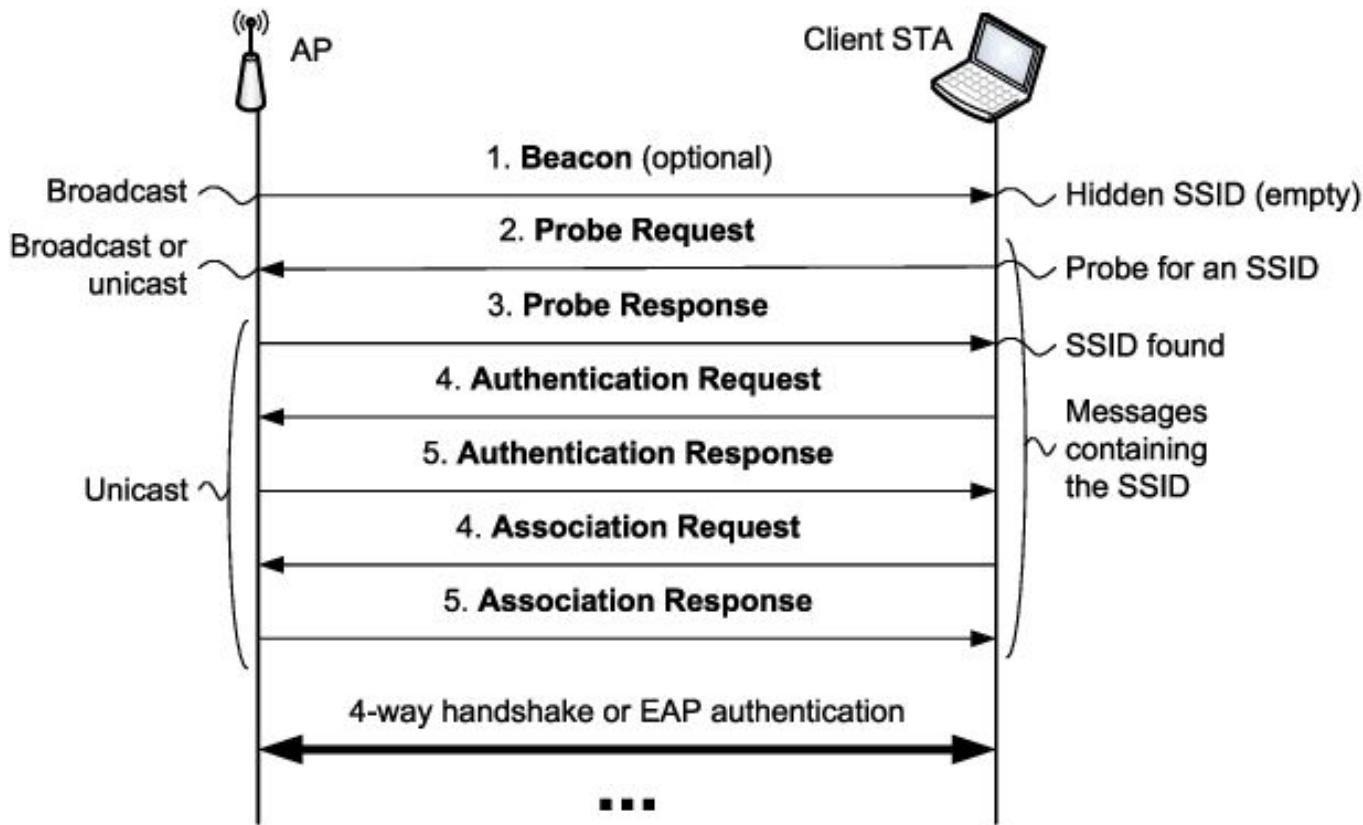
SSID    service set identifier

Table 7-8—Beacon frame body

Order	Information	Notes
1	Timestamp	
2	Beacon interval	
3	Capability	
4	Service Set Identifier (SSID)	

Una trama de beacon (faro) contiene la información necesaria para acceder a un access point. El AP transmite beacons de forma periódica. También el dispositivo puede activamente solicitar beacons (probe).

# WIFI - Conectándose



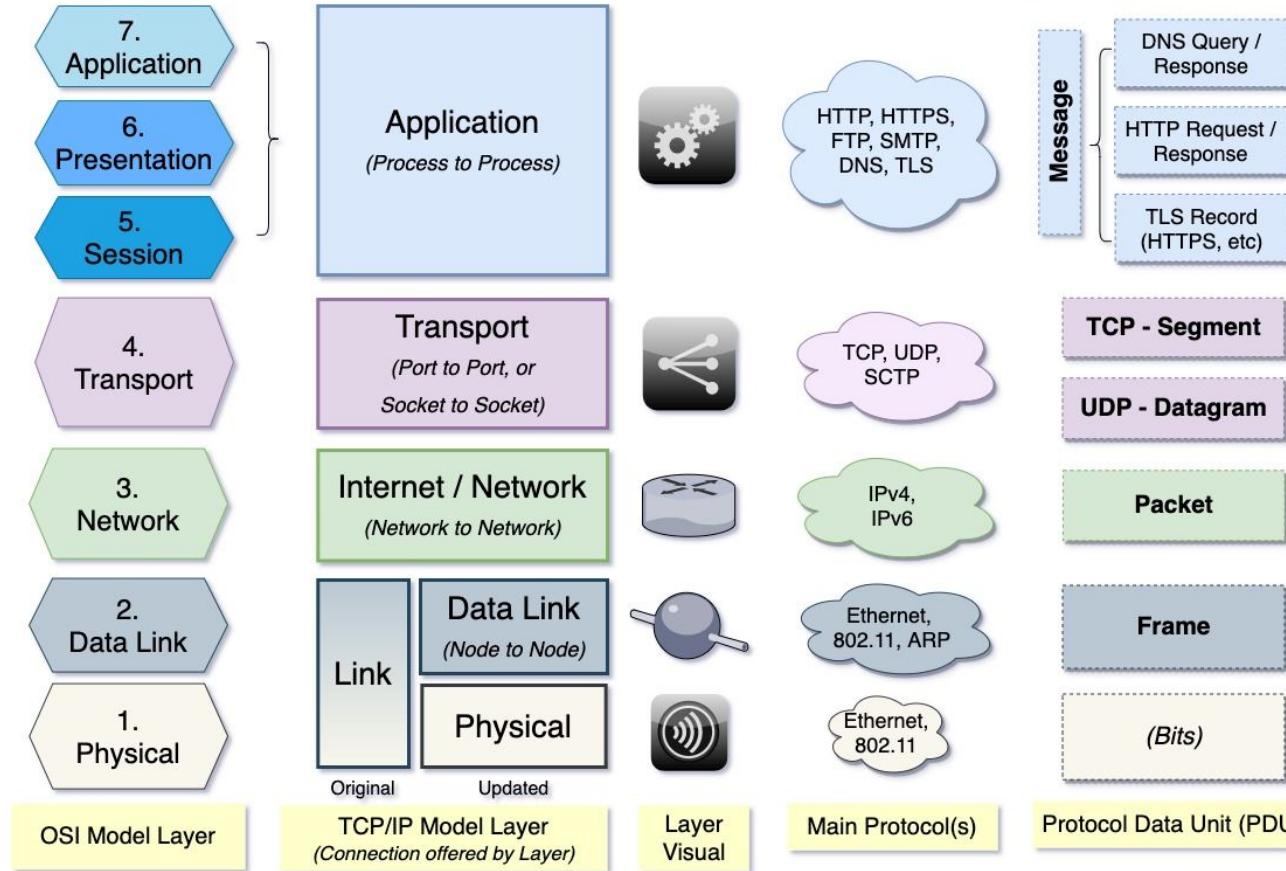
En la autenticación, existen redes abiertas (sin clave) y redes con una clave predefinida. Los dispositivos deben conocer la clave para autenticarse. Existen varios sistemas:

- WEP
- WPA
- WPA2

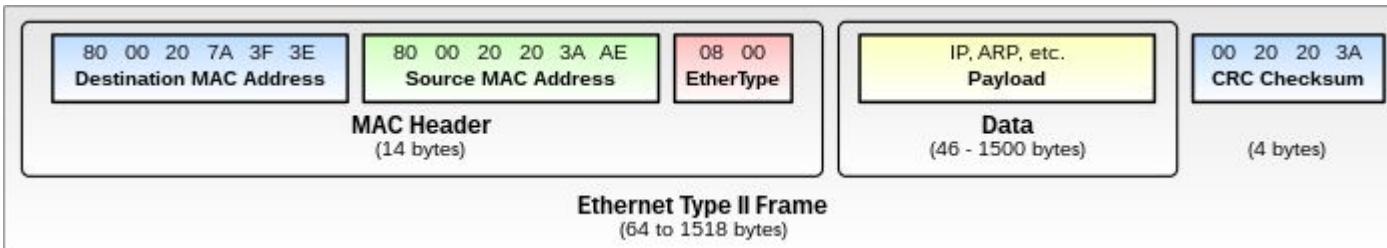
# Modelo OSI vs lo que ocurre en realidad

The Internet Layers: *Overview of the OSI and TCP/IP Models*

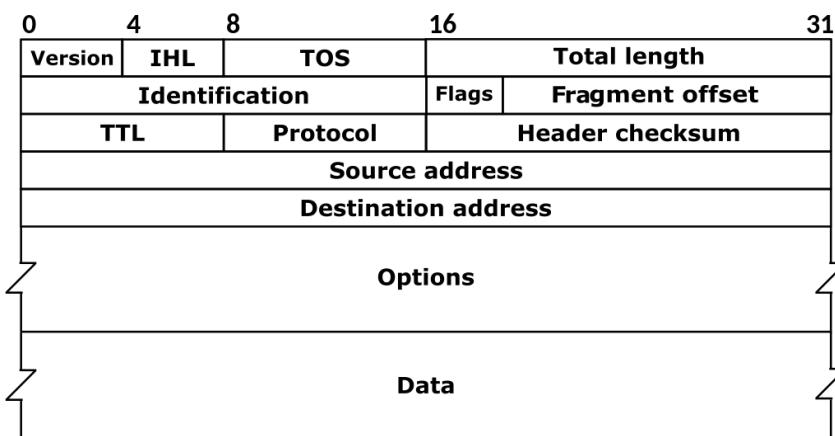
v. 2



# Tramas (frames)



## IP Frame



## TCP Segment Header Format

## UDP Datagram Header Format

Bit #	0	7	8	15	16	23	24	31
0	Source Port				Destination Port			
32	Length				Header and Data Checksum			

# Resumen

Cada dispositivo se enlaza con una red (sea cableada ethernet o inalámbrica wifi). Según el medio usado tiene un identificador único (MAC address). Dentro de esa red puede comunicarse con cualquier dispositivo usando MAC, pero generalmente necesita comunicarse fuera de su red. Aquí entra el router que conecta dos redes entre sí y re-envía mensajes a otras redes. Es aquí donde cada dispositivo obtiene una dirección IP. Esa dirección lo identifica únicamente. Obtiene esta dirección de forma estática (asignada por el usuario) o dinámica DHCP (asignada por un servidor DHCP usualmente en el router). Entre dos dispositivos IP se envían mensajes usando el transporte (UDP o TCP). Este transporte tiene asociado un número de puerto (0 ~ 64K). Hay un puerto de origen y un puerto de destino. Los dispositivos suelen reservar un puerto específico para una aplicación/servicio (ej: HTTP usa 80). Para lograr comprender todo este camino existen otras materias, esto es solo un pequeño resumen.

# Elementos de sistemas embebidos

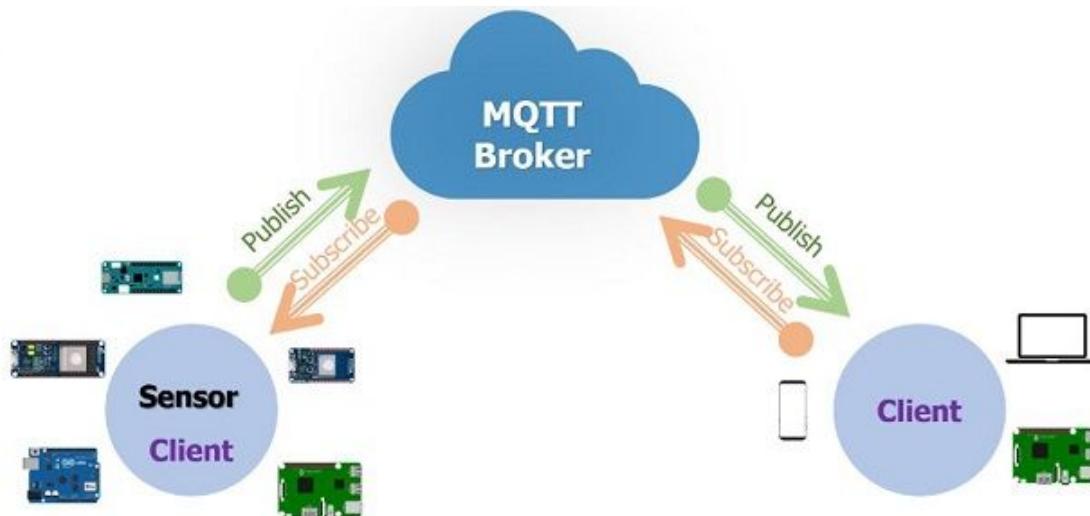
Unidad 6.3  
MQTT

Versión 1.0.0

Jaír Hnatiuk, Carlos Maidana, Carlos Rodríguez, Edgardo Gho, Martín Ferreyra Biron

# Vista del sistema - Partes que intervienen

En MQTT existe un broker que centraliza todas las comunicaciones. Luego los clientes envían y/o reciben mensajes contra el broker. Si bien algunos clientes tendrán sensores o actuadores y otros servirán como visualizadores o controles (celulares, computadoras), en la infraestructura MQTT hay solo Broker y clientes (indistintos). Cada cliente elige un tema (topic) y puede publicar (enviar) o suscribirse (recibir) de ese topic.



# Formato de los mensajes

MQTT-Packet:

**PUBLISH**



contains:

packetId (always 0 for qos 0)

Example

4314

topicName

"topic/1"

qos

1

retainFlag

false

payload

"temperature:32.5"

dupFlag

false

MQTT-Packet:

**SUBSCRIBE**



contains:

packetId

Example

4312

qos1 } (list of topic + qos)

1

topic1

"topic/1"

qos2 }

0

topic2

"topic/2"

...

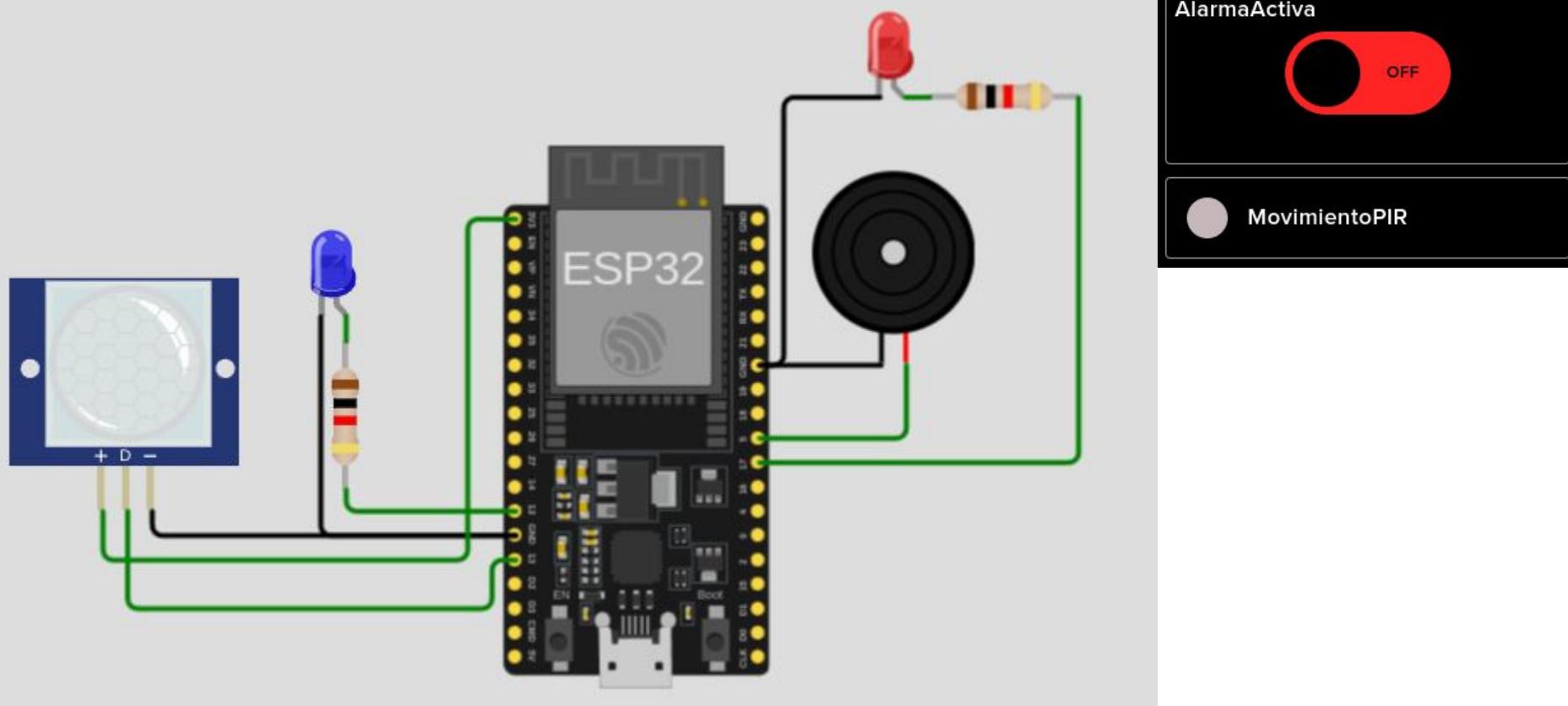
...

bit	7	6	5	4	3	2	1	0			
byte 1	Message Type				DUP Flag	QoS Level					
byte 2	Remaining Length				Variable Header						
Payload											

Name	Value	Description	Variable Header
Reserved	0	Reserved	-
CONNECT	1	Client connect request to server or broker	Present
CONNACK	2	Connect request acknowledgment	Present
PUBLISH	3	Publish message	Present
PUBACK	4	Publish acknowledgment	Present
PUBREC	5	Publish receive	Present
PUBREL	6	Publish release	Present
PUBCOMP	7	Publish complete	Present
SUBSCRIBE	8	Client subscribe request	Present
SUBACK	9	Subscribe request acknowledgment	Present
UNSUBSCRIBE	10	Unsubscribe request	Present
UNSUBACK	11	Unsubscribe acknowledgment	Present
PINGREQ	12	PING request	None
PINGRESP	13	PING response	None
DISCONNECT	14	Client is disconnecting	None
Reserved	15	Reserved	-

Tópico Ejemplo: casa/dormitorio/1/aireacondicionado/temperatura

# Cliente - Sensor/Actuador - Dashboard



# Secuencia

```
from umqtt.simple import MQTTClient
```

Para conectarnos a un broker MQTT, usamos una biblioteca que resuelve el envío de mensajes. Tenemos que tener acceso a internet, lo cual logramos utilizando el módulo wifi integrado. A tal fin, micro python provee bibliotecas y funciones.

```
import network

ssid = 'Wokwi-GUEST'
wifipassword = ''

#Definimos modo Station (conectarse a Access Point remoto)
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)

#Conectamos al wifi
sta_if.connect(ssid, wifipassword)
print("Conectando")

while not sta_if.isconnected():
    print(".", end="")
    time.sleep(0.1)

print("Conectado a Wifi!")
#Vemos cuáles son las IP
print(sta_if.ifconfig())
```

Una vez conectados a wifi, podemos conectarnos al broker MQTT. Podemos configurar nuestro propio Broker usando MosQuiTTo, o utilizar uno gratuito como io.adafruit.com.

```
mqtt_server = 'io.adafruit.com'
port = 1883
user = 'usuario' #definido en adafruit
password = 'aio_f.....' #key adafruit

client_id = 'Identificador' #debe ser único!
topic_1 = 'path/al/topic1'
topic_2 = 'path/otro/topic2'
```

# Conectando al broker MQTT

```
def funcion_callback(topic, msg):
    #completamos con la lógica que debe ejecutarse cuando se reciben mensajes del broker
    #OJO que los argumentos topic y msg vienen codificados en UTF-8, por ende hay que convertirlos a string.
    dato = msg.decode('utf-8')
    topicrec = topic.decode('utf-8')
    print("Mensaje en tópico "+topicrec+":"+dato)
    if topicrec == topic_1 and "OFF" in dato: #Hacemos algo util.

#Intentamos conectarnos al broker MQTT
try:
    conexionMQTT = MQTTClient(client_id, mqtt_server,user=user,password=password,port=int(port))
    conexionMQTT.set_callback(funcion_callback) #Funcion Callback para recibir del broker mensajes
    conexionMQTT.connect() #Hacemos la conexión.
    conexionMQTT.subscribe(topic_1) #Nos suscribimos a un tópico luego del connect
    print("Conectado con Broker MQTT")
except OSError as e:
    #Si falló la conexión, reiniciamos todo
    print("Fallo la conexion al Broker, reiniciando...")
    time.sleep(5)
    machine.reset()
```

# Una vez conectado...

```
while True:  
    #Si se produce una excepción, por ejemplo se corta el wifi  
    #o perdemos la conexión MQTT, simplemente vamos a reiniciar  
    #el micro para que comience la secuencia nuevamente, así que  
    #usamos un bloque Try+Except  
    try:  
        #Tenemos que verificar si hay mensajes nuevos publicados por el broker  
        conexionMQTT.check_msg()  
        time.sleep_ms(500)  
        #Si queremos publicar algo podemos usar la función publish...  
        conexionMQTT.publish(topic_2,"Mensaje a enviar al broker...")  
    except OSError as e:  
        print("Error ",e)  
        time.sleep(5)  
        machine.reset()
```