
Electrónica
con
MicroPython
Daniel Schmidt

Realizado y editado por:

FIRTEC ARGENTINA

e-mail: consultas@firtec.com.ar

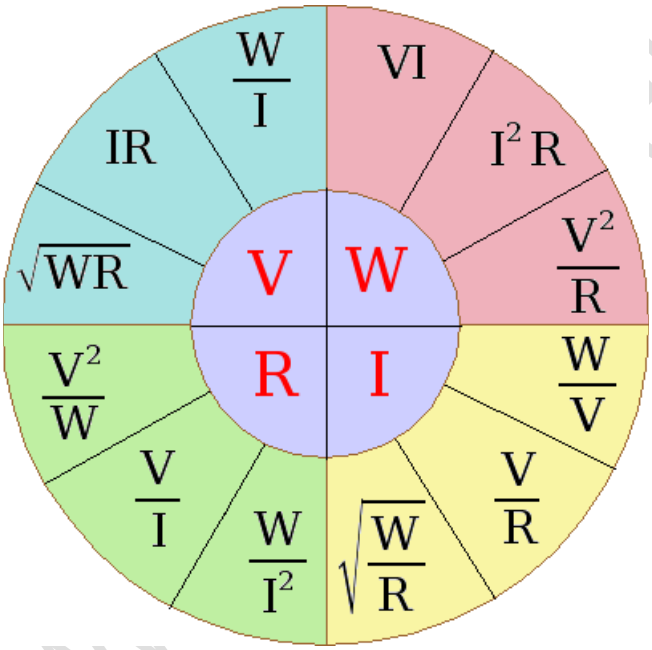
URL: https://www.firtec.com.ar

Tercera Edición (2024) .

Sumario

cuando se diseñe una interfaz de potencia por ejemplo para actuar sobre un motor o sistemas de iluminación de alto consumo.
Con la ley de Ohm se pueden calcular todas las variables eléctricas sin embargo como se verá mas adelante, solo nos vemos en la necesidad de calcular consumos en las etapas con transistores para activar/desactivar contactores o relevadores vinculados a las cargas de potencia.

Colección de Fórmulas para la Ley de Ohm.



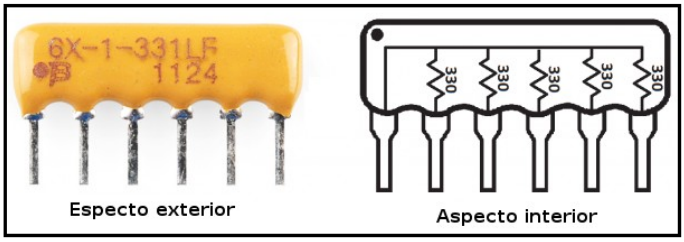
Con los datos de la imagen anterior puede calcular tanto potencia, voltaje, resistencia o intensidad según sea necesario.

Las Resistencias.

Las resistencias son el componente electrónico más omnipresente. Son una pieza crítica en casi todos los circuitos y juegan un rol muy importante en la ley de Ohm. En la siguiente imagen se puede ver el aspecto físico de una resistencia.

Paquetes de Resistencia Especiales

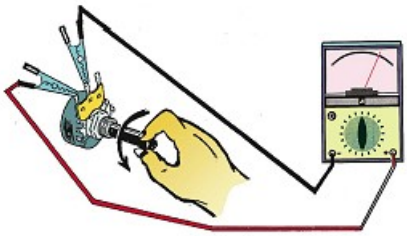
Hay una gran variedad de otras resistencias de propósitos especiales. Las resistencias pueden venir en paquetes pre-conectados de cinco o más resistencias.



Las resistencias en estos conjuntos pueden tener un pin común o estar puestas como divisor de voltaje. Las resistencias no tienen que ser estáticas. Las resistencias variables conocidas como reóstato**,** son resistencias que se pueden ajustar dentro de un rango específico de valores.
Parecido al reóstato es el potenciómetro, los potenciómetros son una resistencia con un patín o contacto que recorre toda esta resistencia creando un divisor de voltaje ajustable entre el pin central y cualquiera de sus extremos.

En la imagen anterior se puede observar a la izquierda el aspecto de un potenciómetro deslizante, al centro un potenciómetro común y a la derecha un Pre-Set, este último es usado para ajustes o calibración de los equipos, son potenciómetros pero sin eje, generalmente fuera del alcance de los usuarios.
Para medir la resistencia total de un potenciómetro o el estado general de la pista podemos hacer la siguiente prueba.

Mide la resistencia total del potenciómetro.



Mide el recorrido del patín de contacto en la pista del potenciómetro.

Para medir un potenciómetro con un multímetro colocamos el instrumento entre punta y punta del potenciómetro y medimos su resistencia total luego colocando el multímetro en su punto medio podemos medir la resistencia entre el cursor y los extremos.

Decodificar Resistencias de Montaje en Superficie.

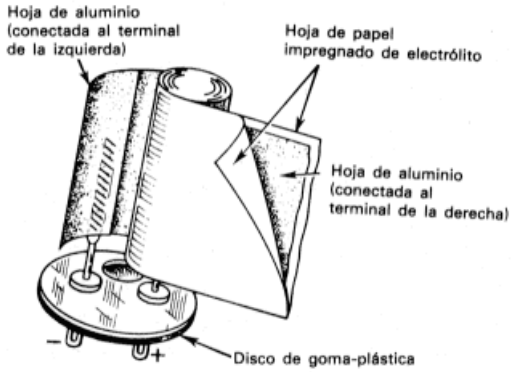
Las resistencias de montaje en superficie como las 0603 o las 0805, tienen su propia forma de mostrar su valor. Hay algunos métodos comunes para marcas estas resistencias. Generalmente tienen tres o cuatro caracteres, ya sean números o letras, impresas en la parte superior de la carcasa.
Si los tres caracteres que ves son todos números, probablemente está viendo una resistencia con marcas **E24.** Estas marcas comparten similitudes con el sistema de color que se usa en las resistencias de agujeros pasantes. Los primeros dos números representan los dígitos más significativos del valor, el número final representa la magnitud.

En el ejemplo superior, las resistencias son marcadas *104*, *105*, *205*, *751*, y *754*. La resistencia marcada con *104* debería ser de 100kΩ (10×10^4),*105* seria 1MΩ (10×10^5), y *205* es de 2MΩ (20×10^5). *751* es de 750Ω (75×10^1), y *754* es de 750kΩ (75×10^4).

Otro sistema de codificación común es el **E96**, y es el más críptico de todos. Las resistencias E96 son marcadas con 3 caracteres, primero dos números y al final una letra.

Condensadores (Capacitores)

- **Corriente de fugas (I_f):** pequeña corriente que hace que el condensador se descargue a lo largo del tiempo.
- **Factor de perdidas (tgΦ):** teóricamente cuando se aplica una tensión alterna a un condensador se produce un desfase de la corriente respecto a la tensión de 90º de adelanto, pero en la práctica esto no es así. La diferencia entre estos 90º y el desfase real se denomina ángulo de pérdidas. En la imagen siguiente podemos ver la estructura interna de un capacitor electrolítico.



En electrónica la palabra condensador o capacitor hacen referencia al mismo componente.
Existen dos formas comunes para dibujar los capacitores en los esquemáticos. Siempre tienen dos terminales, los cuales se conectan con el resto del circuito. El símbolo del capacitor consiste en dos líneas paralelas, que son planas o curvas. Las dos líneas deberían estas paralelas la una a la otra y cerca, pero no tocándose (*esto es representativo de la forma en que se construyen los capacitores*). Es difícil de describir, es más fácil mostrarlo:

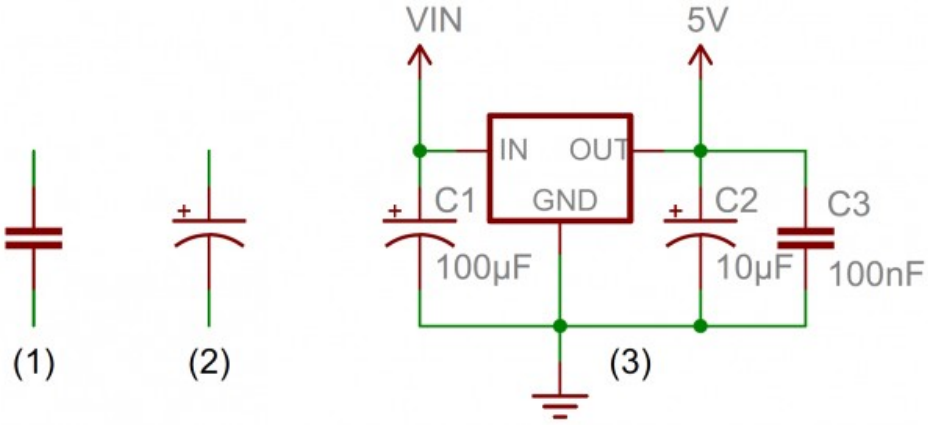
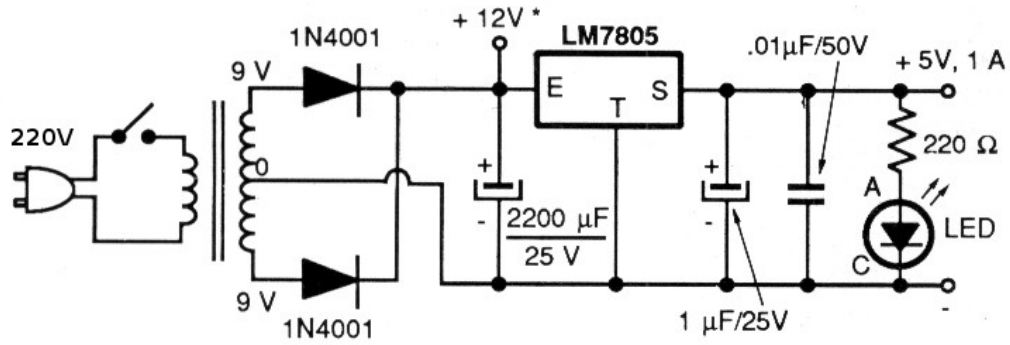
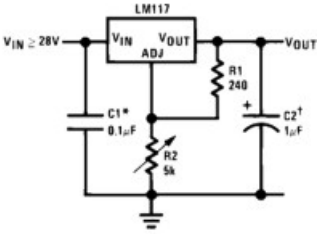


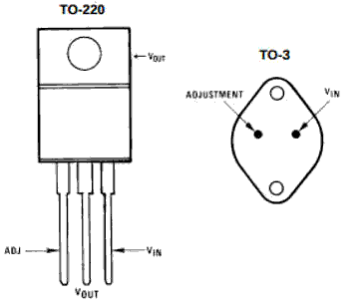
Diagrama esquemático



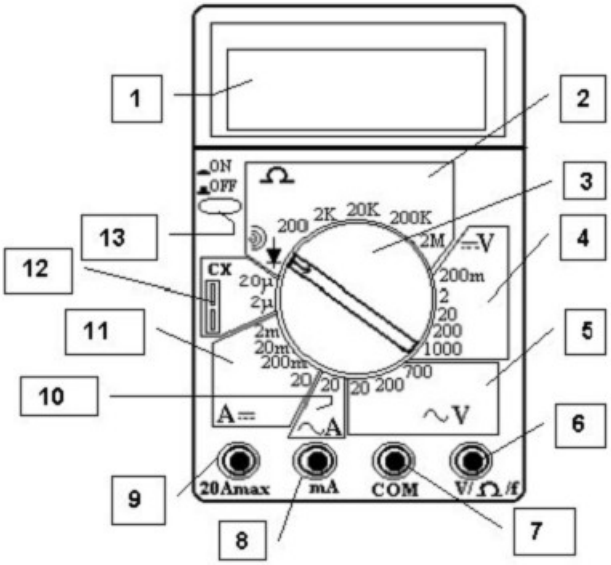
Alguna fuente en desuso de algún módem o router de internet en desuso pueden ser una excelente opción, ya vienen preparados para alimentar circuitos electrónicos con microcontroladores y procesos digitales.
Si contamos con una placa que funciona con 5 y/o 3 Voltios, necesitamos alimentar la placa con una fuente que entregue un voltaje superior a 5 Voltios. Es común leer en las especificaciones que se pueden usar fuentes con rangos de 7 a 15 Voltios, si necesita obtener solo 5 voltios el excedente se disipa en forma de temperatura, alimentar todo con una fuente de 9 Voltios es lo ideal y los reguladores de voltaje trabajaran sin sobrecargas.
Si necesitamos una fuente que además de estabilizada sea variable, el LM317 resuelve el problema. Observe el siguiente circuito electrónico.



Este regulador tiene el mismo encapsulado que el LM7805 (TO-220) solo que uno de esos terminales es el pin de ajuste para el voltaje de salida.



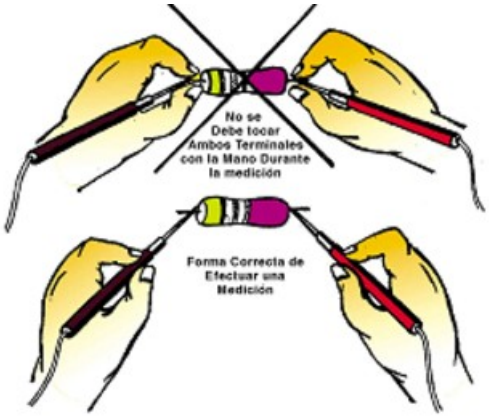
Solo verifique que está conectado los pines correctamente, si los confunde es muy posible que destruya el regulador.
Normalmente tendremos una salida con un rango de 1,5 Voltios hasta casi el voltaje de entrada.
En Internet hay abundante información de este regulador y las posibles configuraciones que se pueden conseguir para el armado de fuentes de alimentación estabilizadas.



Recordar que la corrriente alterna o AC (*Alternal Corrent*), es aquella que se produce mediante generadores electromagnéticos, de tal forma que en el caso de nuestro país, fluye cambiando el polo positivo (polo vivo) a negativo (polo neutro), 50 veces por segundo. Por esto la corriente domiciliaria se dice que tiene un voltaje de 220 V a una
frecuencia de 50 HZ (Hertz), (tener en cuenta que un Hertz es un cambio de polo vivo a polo neutro en un segundo). La razón para que la tensión en el uso domiciliario sea alterna, es que resulta menos costosa que la continua, ya que se la puede suministrar más directamente desde la planta generadora sin rectificarla a corriente continua.
Las baterías y pilas proveen una corriente continua o DC (*Direct Current*), es decir que en todo instante la corriente fluye de positivo a negativo. Para el caso del automóviles es más simple proveerse de un alternador o generador que rectifica la corriente alterna en continua mediante los diodos rectificadores que posee en su interior.

Es muy importante leer el manual de operación de cada multímetro en particular, pues en él, el fabricante fija los valores máximos de corriente y tensión que puede soportar y el modo más seguro de manejo, tanto para evitar el deterioro del instrumento como para evitar accidentes al operario. El mutímetro que se da como ejemplo en esta explicación, es genérico, es decir que no se trata de una marca en particular, por lo tanto existe la posibilidad que existan otros con posibilidad de medir más magnitudes.

hace con resistencia apreciable. El instrumento fija una corriente de prueba de 1mA.
Cuando buscamos un valor de la resistencia, tenemos para elegir escalas o rangos entre los siguientes valores de ejemplo:
200 Ohms, 2K (2 kiloOhms o 2000 Ohms) 20K (20.000 Ohms) y 2M (2 MegOhms o 2 millones de Ohms) y en algunos testers figura hasta 20M.
Cuando se miden resistencias se debe tener cuidado de no tomar con las manos ambas puntas de la resistencia ya que si lo hacemos no solo medimos el valor de resistencia del componente sino también la propia resistencia del cuerpo.



Si el valor a medir supera el máximo de la escala elegida, el display indicará "1"a su izquierda. Por lo tanto habrá que ir subiendo de rango hasta encontrar el correcta.

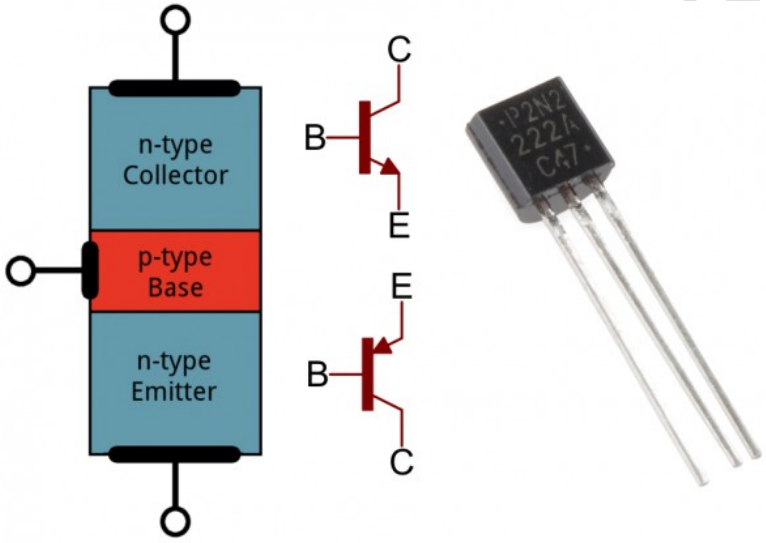
Tensión en DC.

Sabemos que como voltímetro se conecta en paralelo con el componente a medir, de tal manera que indique la diferencia de potencial entre las puntas.
Donde indica 200m el máximo es 200 milivolts (0,2 V), el resto se comprende tal cual están expresados por sus cifras. Por lo tanto para medir tensiones de batería del automóvil debemos elegir la de 20V.
Si se está buscando caídas de tensión en terminales o conductores, podemos elegir una escala con un máximo más pequeño, luego de arrancar con un rango más elevado y así tener una lectura aproximada. Siempre hay que empezar por un rango alto, para ir bajando y así obtener mayor precisión. Cuando el valor a medir supere el máximo elegido, también indicará "1"en el lado izquierdo del display.

la fuente puede suministrar la suficiente corriente, el LED se quemará, destruyendose. Por esto, los LED se ocupan casi siempre junto a un resistor limitador de corriente. El resistor disminuye la cantidad de corriente fluyendo por el LED, manteniéndola a un nivel que no lo destruya.

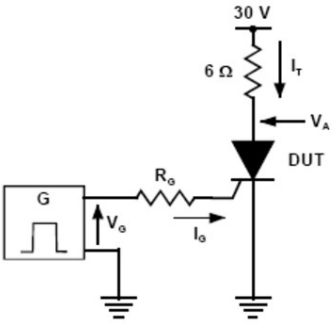
Diodo Led Infrarrojos.
Emiten o reciben rayos infrarrojos al ser recorridos por una corriente eléctrica. Se los utiliza en sistemas de control a distancia de aparatos eléctricos o electrónicos, como TV a color, equipos musicales, video, alarmas, etc.

Funcionamiento teórico de un Transistor.



El transistor convencional o bipolar se denomina así porque en su funcionamiento intervienen corrientes de huecos, o de carga positiva, y de electrones, o de carga negativa. Otros dispositivos como los FET se denominan monopolares porque sólo hay corrientes de un tipo.
Los terminales del transistor reciben el nombre de **emisor**, **colector** y **base.** La base es el terminal que está unido a la zona intermedia del transistor. Las tres partes del transistor se diferencian por el distinto nivel de dopaje del material semiconductor, la zona de menor dopaje es la base, a continuación se encuentra el colector y por último el emisor.

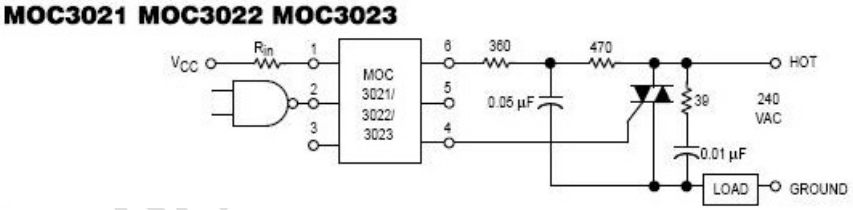
Tiristores.



Un tiristor es uno de los tipos más importantes de los dispositivos semiconductores de potencia y control.
Se puede ver el tiristor como un simple diodo controlado, cuando aparece un pulso de disparo en la compuerta, el tiristor conduce y como diodo su funcionamiento primario esta asignado a corrientes continuas, para muchas aplicaciones se puede suponer que los Tiristores son interruptores o conmutadores ideales con limitaciones.

Triac (Triodo de Corriente Alterna).

Se puede definir como un tiristor que conduce en ambos sentidos en forma controlada, lo que permite funcionar en circuitos de corriente alterna y controlar dispositivos en alterna dejando pasar ambos semiciclos de la fase alterna. Si bien la compuerta del triac se podría conectar directamente a los pines del controlador, es prudente no conectar estos dispositivos directamente a los pines de los microcontroladores, para esto existen los opto-acopladores (*MOC30xx*) que sirven de aislantes o separadores entre la lógica de control y la etapa de potencia.

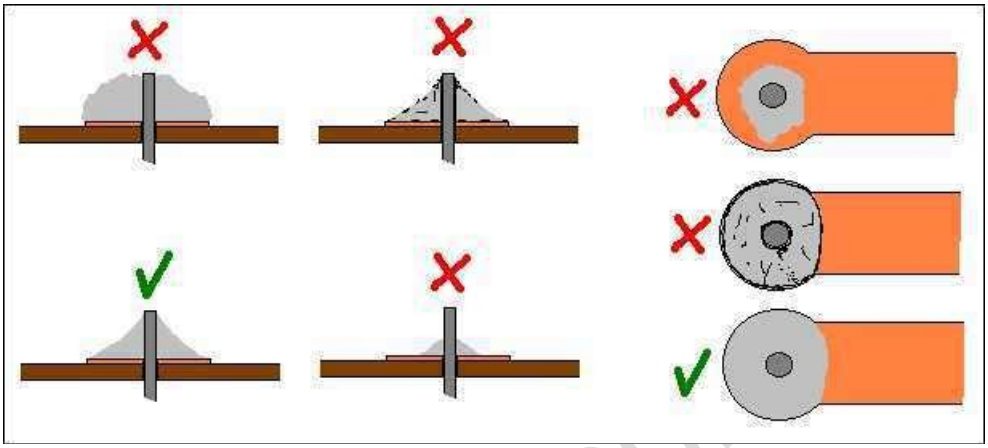


En la imagen anterior se puede ver un opto-acoplador conectado a una lógica de control que transfiere señal a un dispositivo vinculado a la red eléctrica.
El uso de opto-acopladores para separar la etapa de potencia con la zona de control es casi de uso obligado para evitar riesgos eléctricos.

Conclusiones sobre Semiconductores.

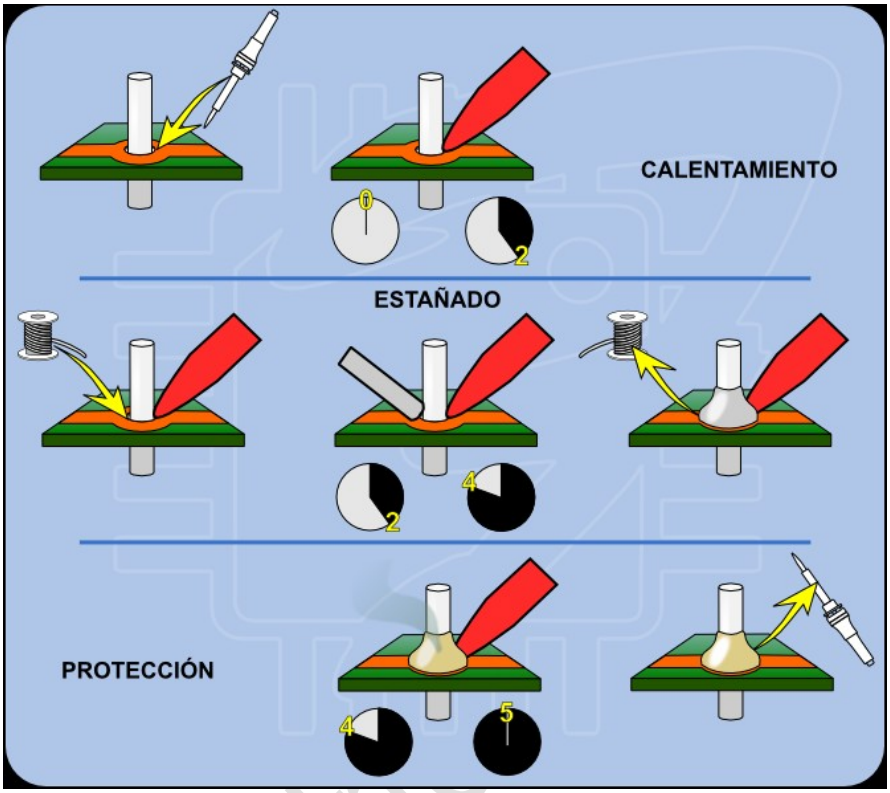
Los semiconductores en general son un gran aporte al campo del desarrollo científico y tecnológico, dado su amplia versatilidad y su gran aplicabilidad en la investigación y desarrollo.
Sus fundamentos físicos son fundamentados en las bases de la mecánica cuántica, aprovechando sus concepciones y formulaciones teóricas es posible

eventualmente agregar un poco mas de estaño para asegurar la correcta conexión y el problema se resuelve.
En la siguiente imagen se pueden ver algunas soldaduras defectuosas.



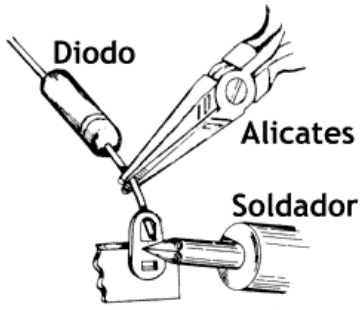
Vemos aquí algunas soldaduras deficientes y la forma correcta de un punto de soldadura.
La técnica para realizar una correcta soldadura se resume en tres pasos:

1. *Calentar con el soldador los elementos a soldar.*
2. *Aplicar estaño en su justa medida.*
3. *No retirar el soldador hasta que el estaño se extienda.*



Técnica de soldado.

Acercar los elementos a unir hasta que se toquen, si es necesario, utilizar unos alicates para sujetar bien las partes.



Aplicar el soldador a las partes a soldar, de forma que se calienten *ambas* partes.
Tener en cuenta que los alicates o pinzas absorben parte del calor del soldador.
Las piezas empiezan a calentarse hasta que alcanzan la temperatura del soldador. Si la punta está limpia, esto suele tardar menos de 3 segundos. Este tiempo dependerá de si se usan alicates y de la masa de las piezas a calentar.

Sin quitar el soldador, aplicar el estaño (unos pocos milímetros) a la zona de la soldadura, evitando tocar directamente la punta.
Cuando la zona a soldar es grande, se puede mover el punto de aplicación del estaño por la zona para ayudar a distribuirlo.

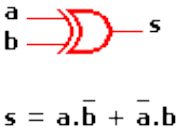
$s = a + b$

a	b	s
0	0	0
0	1	1
1	0	1
1	1	1

Es decir, basta que una de ellas sea 1 para que su salida sea también 1.

Compuerta OR-EX o XOR.

Es OR EXclusiva en este caso con dos entradas (puede tener mas, claro...!) y lo que hará con ellas será una suma lógica entre a por b invertida y a invertida por b.
Al ser O Exclusiva su salida será 1 si una y sólo una de sus entradas es 1.



$s = a.\bar{b} + \bar{a}.b$

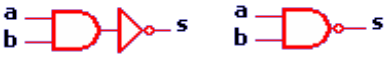
a	b	s
0	0	0
0	1	1
1	0	1
1	1	0

Compuertas Lógicas Combinadas.

Al agregar una compuerta NOT a cada una de las compuertas anteriores, los resultados de sus respectivas tablas de verdad se invierten, y dan origen a tres nuevas compuertas llamadas NAND, NOR y NOR-EX... Veamos ahora como son y cual es el símbolo que las representa.

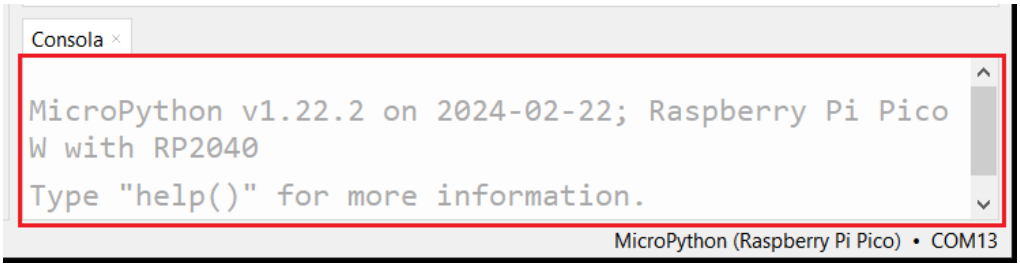
Compuerta NAND.

Responde a la inversión del producto lógico de sus entradas, en su representación simbólica se reemplaza la compuerta NOT por un círculo a la salida de la compuerta AND.



$s = \overline{a \,.\, b}$

a	b	s
0	0	1
0	1	1
1	0	1
1	1	0



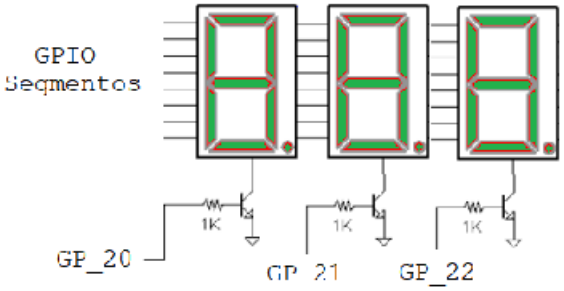
Cuando la carga del interprete termina su identificación aparece en la consola.

Para todo esto estamos suponiendo que usted tiene una placa recién comprada que no tiene ningún programa cargado en ella pero si ya tiene un interprete cargado y quiere actualizarlo (siempre conviene usar las versiones mas actualizadas) entonces el proceso es ligeramente distinto.
Para instalar el interprete primero debemos colocar la placa en modo cargador de arranque (BOOTSEL).
Este programa cargador esta dentro el controlador RP2040 y no hay forma de borrarlo o dañarlo y para activar este modo solo tenemos que iniciar la placa manteniendo presionado el botón bootsel.


Botón Bootsel en la placa Raspberry PI Pico.

Y luego procedemos repitiendo los pasos comentados anteriormente.
El interprete MicroPython es un archivo UF2 que también podemos descargar en su versión mas nueva desde internet.
Este archivo UF2 simplemente se copiará al dispositivo de almacenamiento masivo USB que aparece una vez que pico es puesto en modo Bootsel y una vez que se completa la descarga del nuevo firmware, el dispositivo se restablecerá automáticamente y estará listo para su uso.
Cada vez que quisieramos cargar una nueva versión de Micropython o probar otro firmware, desconectamos el cable USB y manteniendo presionado el

importar cuantos tenemos montados en el circuito electrónico.



Multiplexor usado en el ejemplo con transistores BC337.

Para controlar el encendido de cada dígito usamos un transistor NPN (*para un display cátodo común*), como todos los pines de los segmentos están unidos entre si, cuando se envía un dato a los dígitos este dato se presenta en todos los dígitos pero solo se hace visible en aquel cuyo transistor está activado.
Por ejemplo si vamos a mostrar el número 658, primero enviamos el "8" y activamos el GP_22, se espera un determinado tiempo hasta que la imagen del "8" quede grabado en la retina del ojo, se apaga ese transistor y se envía número "5", se activa GP_21 y el número "5" se hace visible sin embargo el ojo del observador seguirá viendo también el número "8", se repite el proceso con el número "6" y el pin GP_20, la imperfección del ojo humano para procesar imágenes de alta velocidad genera la ilusión de que los dígitos muestran el número 658 en un mismo momento cuando en realidad solo un dígito está encendido en cada instante.
El módulo para el control del display está construido en base a una clase que hemos llamado *Display*.
Las clases son plantillas con las cuales se pueden crear objetos (*Programación Orientada a Objetos*), en el campo del software un objeto es un elemento no físico pero que se comporta como si existiera realmente y todos los objetos se crean a partir de una clase y cuando hablamos de *instancia*, esto es crear un objeto a partir de otro.
El módulo para el control del display tiene el nombre de ***display7seg.py*** y su código es el siguiente:

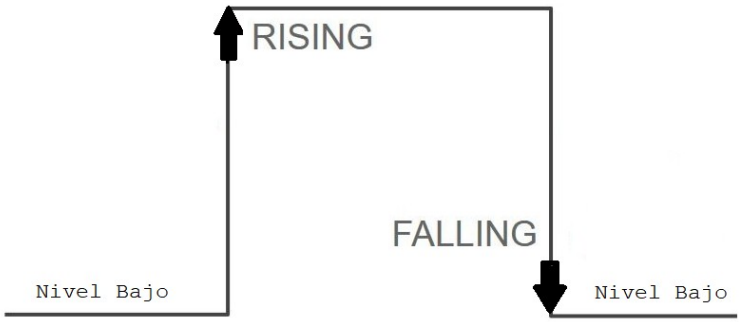
```
from machine import Pin
# Importa las funciones para el control del
# hardware
import utime
# Importa las funciones para el control de tiempos
```

```
            bandera = 1 # Poner bandera en "1"
        if (boton.value()):
# El usuario soltó el botón?
            bandera = 0
# OK, entonces limpiar bandera!!
if __name__ == '__main__':
    main()
```

Observe que la condición *if (boton.value() == 0 and bandera == 0)* es necesaria para que nuestro botón mueva el contador de uno en uno, *bandera* se usa para saber que el usuario a apretado el botón por lo tanto aunque el programa detecte el botón activado *bandera* le dice que esa situación ya fue evaluada y solo será válida nuevamente cuando *bandera = 0* y esto ocurre cuando el usuario suelta el botón.

Interrupciones en GPIO.

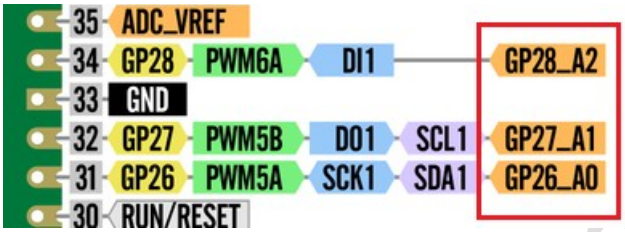
De la misma forma que podemos activar una interrupción en un Timer podemos hacer lo mismo en los pines *GPIO*. Podemos activar interrupciones en el flanco de subida (*Rising)* o en el flanco de bajada (*Falling*) de la señal aplicada a un pin *GPIO*.



Niveles eléctricos aplicados a los pines.

Esto resulta muy útil porque fácilmente podemos medir tiempos entre que la señal sube y baja, podemos claramente detectar eventos que ocurren en los pines.
En el siguiente ejemplo toda la electrónica es igual que en los ejemplos anteriores, también tenemos un botón de cuenta solo que ahora el botón desencadena una ISR que es la encargada de mover el contador.
Podrá notar que todo el programa es igual salvo las configuraciones del botón y su interrupción.

El ADC_0 conectado al pin GP_26, ADC_1 conectado al pin GP_27 y el ADC_2 en el pin GP_28, un cuarto canal está conectado a un sensor de temperatura integrado en el chip, en la placa no existe el GP_29. El quinto canal se puede usar para medir el voltaje VSYS de la propia placa.

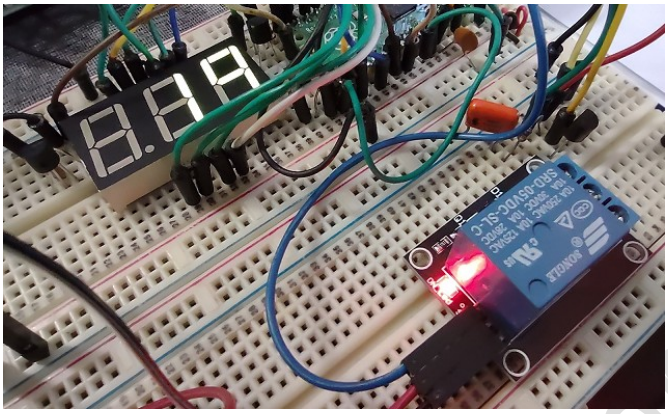


Canales analógicos en Raspberry PI Pico

Ejemplo simple con potenciómetro para el conversor A/D.

Para el siguiente ejemplo tenemos conectado el punto medio de un potenciómetro al GP_26 y los extremos del potenciómetro a +3V y GND.
Si movemos el potenciómetro de un extremo al otro veremos en la terminal de Thonny el voltaje medido en el punto medio.
El código para verificar el funcionamiento del conversor es el siguiente:

```
import time
from machine import ADC
# Importa lo necesario para poder manejar el
# conversor A/D
def main():
    potenciometro = ADC(0)
# Canal ADC en pin 26
    escala = 3.3 / (65535)
# Escala los 16 bits
    while True:
# Bucle infinito
        bits_16 = potenciometro.read_u16();
# Lee el canal analógico
        volts_16 = bits_16 * escala
# Escala la medición
        datos = "%.03f" % (volts_16)
# Da formato al dato para mostrarlo en la terminal
        print('Voltios: {}'.format(datos))
# Muestra del dato
        time.sleep(1)
# Espera un segundo
if __name__ == '__main__':
    main()
```



Relay conectado a Raspberry Pico para el control del termostato con LM35.

Uno de los problemas que encontramos al controlar la temperatura es la histéresis que se define como la tendencia de un material a conservar una de sus propiedades, esto en ausencia del estímulo que la ha generado.
Supongamos que tenemos fijado el ajuste en 20°C y el sensor acaba de indicar que estamos en 19°C por lo tanto el calefactor enciende pero casi de inmediata alcanza los 20°C y apaga nuevamente, esta condición puede repetirse por varios segundos o minutos con el resultado de una degradación de los contactos del relay y generando mucho ruido eléctrico en cada conexión/desconexión.
Para minimizar esta situación se ha dispuesto que el relay se active un grado abajo del Set-Point y se desactive un grado por encima del valor de ajuste. De esta forma las dos acciones conectar/desconectar están separadas lo suficiente como para que el sistema no se confunda.
Obviamente estamos proponiendo un sistema On-Off y en la actualidad esto ya es un anacronismo, un sistema de control de temperatura usaría un sistema con ajuste proporcional de potencia, un PID o cualquier sistema más inteligente que un simple On-Off.
El código del termostato es el siguiente.

```
from machine import Pin, Timer
from machine import ADC
import time
import display7seg
def ISR(p):
    global bandera
    bandera = 1
def main():
    led= machine.Pin('LED', machine.Pin.OUT)
```

potenciometro = ADC(0)
factor_16 = 3.3 / (65535)

De acuerdo a lo visto también necesitamos escalar lo que nos llega del conversor, recuerde que Micropython siempre lee el conversor en 16 bits sin importar cuantos bits reales tiene este conversor.

En el programa principal existe un bucle infinito donde continuamente leemos el canal analógico sin embargo para poder mostrar el dato en la pantalla LCD debemos convertir el dato a una cadena ASCII.

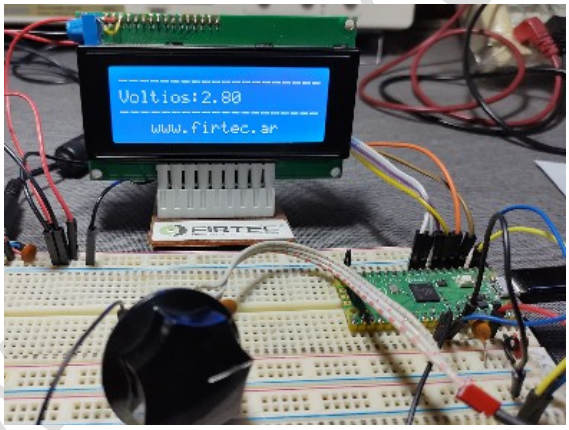
En las siguientes líneas se puede ver como se realiza la lectura del canal analógico.

bits_16 = potenciometro.read_u16();
volts_16 = bits_16 * factor_16

Para la conversión del dato binario a un valor ASCII tenemos la siguiente línea.

datos = "%.02f" % (volts_16)

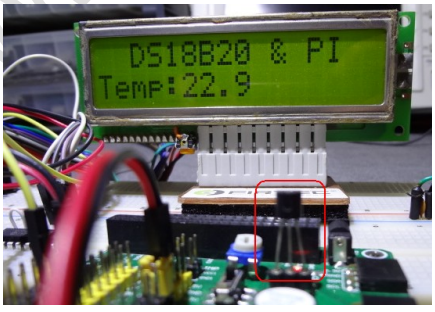
En *datos* se almacenará el valor del voltaje en un formato con dos decimales.



Potenciómetro conectado al pin GP_26 y pantalla LCD.

```
# Descripción : Lee el canal analógico 0 (GP_26) y
# muestra los datos en pantalla LCD
# Target :      Raspberry Pi PICO + Hitachi 44780
# ToolChain :   MicroPython
from machine import Pin, ADC
from time import sleep
import utime
import lcd_44780
# Importa el driver para la pantalla
def main():
    # Pines usados por la pantalla LCD
```

precisión, en el rango comprendido entre -10 y 85 grados es de ±0.5 grados. Su precio es económico, su interfaz de funcionamiento es sencilla y su uso es muy provechoso para proyectos que requieran mediciones precisas y confiables.
Una particularidad interesante de estos sensores es que poseen un número *ID*, algo así como un número *MAC* único grabado por el fabricante en cada sensor lo que posibilita tener muchos sensores y direccionarlos por su *ID* para conocer la temperatura en ese punto en particular. Recuerde que el protocolo 1-Wire exige una resistencia de 4,7K colocada entre el pin de datos y 3,3V, estos sensores son en realidad un poderoso microcontrolador mas un sensor de temperatura más un puerto de comunicaciones, todo contenido en un pequeño encapsulado que parece ser un simple transistor. Existe una comunicación entre el sensor y el master (*Pico o también ESP32*) donde se envían comandos a los cuales el sensor responde. Pico y ESP32 son microcontroladores y el DS18B20 es otro microcontrolador, ambos deben dialogar para un funcionamiento correcto.
El esquema de funcionamiento es bastante complejo por eso es de gran ayuda que existan módulos de software que transparentan toda la complejidad inherente al protocolo como así también los aspectos electrónicos del sensor. Para poder usar estos sensores solo tenemos que importar los correspondientes módulos, *import machine, onewire, ds18x20.*
Estos sensores tienen distintos modelos como el *DS1820, DS18S20 y DS18B20* la diferencia en ellos es la resolución y la velocidad de conversión siendo el primero que apareció el DS1820, como casi no hay diferencia en el costo es recomendable usar el mas nuevo, en este momento el DS18B20.



El sensor DS18x20 tiene un encapsulado TO92, parece un transistor **pero no lo es**.
El siguiente es un ejemplo para verificar el funcionamiento de estos sensores, se leen los ID de cada sensor y la temperatura. Estos datos son pasados a la

otro es evaluado hasta el siguiente bucle es por esto que aunque se incremente la variable *bandera* y el siguiente *elif()* sea verdadero, este no será evaluado sino hasta el siguiente lazo *for*.

Pantallas LCD con solo dos conexiones.

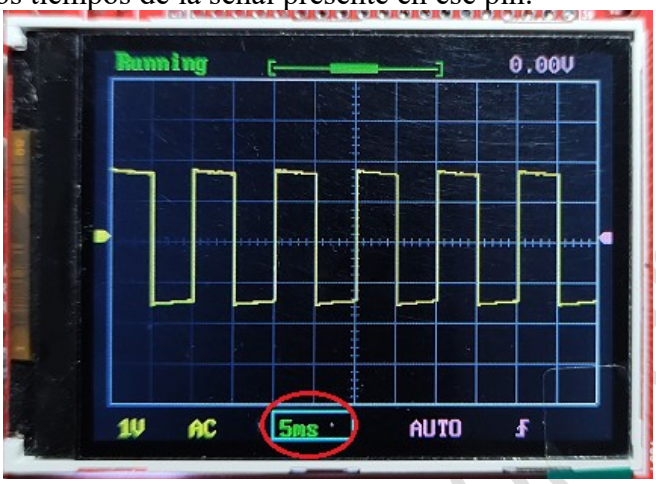
Como seguramente ha notado el uso de pantallas LCD otorgan a los desarrollos un aspecto profesional y son sencillas de implementar pero como seguramente también notó, requieren de varias conexiones que obliga a que varios pines estén dedicados al control de la pantalla.
Para solucionar esto podemos usar el mismo enfoque usado en el mundo Arduino, los controladores I2C para pantallas LCD.
Estas pequeñas placas tienen todo lo necesario para manejar la pantalla incluso el pre-set para fijar el nivel de contraste.

Podemos incluso conseguir la pantalla completa ya con la placa soldada y lista para usar.
Estos LCD solo requieren cuatro conexiones de las cuales dos son para alimentación y un pin de datos llamado SDA que se conecta al GPIO 1 y otro pin de reloj llamado SCL que se conecta al pin GPIO 2.

Placa I2C para el manejo de LCD.

Vamos a necesita un paquete de software para manejar esta placa, observe la imagen siguiente.

Para verificar el funcionamiento colocamos un osciloscopio en *GPIO 16* y poder medir los tiempos de la señal presente en ese pin.



Tiempos obtenidos con el ejemplo.

Como ser puede ver en la imagen anterior el ajuste de tiempos sobre el pin es muy exacto, vamos a ver cómo funciona el programa.
Primero importamos la biblioteca *rp2* que contiene todo lo necesario para trabajar con el ensamblador PIO.
El programa PIO está contenido en el método *LED()*, pero antes de poder usarlo debemos usar el decorador *@rp2.asm_pio*.
En los parámetros del decorador se establecen cosas como la dirección del PIN y la dirección FIFO.

```
@rp2.asm_pio(set_init=rp2.PIO.OUT_LOW)
```

Un decorador es una función que toma otra función y extiende su funcionamiento pero sin modificarla implícitamente.
En la configuración del decorador estamos diciendo que el pin o los pines a usar serán salidas e inician a nivel bajo, cual pin o pines serán afectados se definen en otra línea del programa.
Si por ejemplo necesitamos trabajar con tres pines la sintaxis seria.
@rp2.asm_pio(set_init=rp2.PIO.OUT_LOW, rp2.PIO.OUT_LOW, rp2.PIO.OUT_LOW)
El programa PIO inicia con `wrap_target()` que una forma de iniciar un bucle sin usar la instrucción *JMP*, instrucción clásica de salto, la diferencia entre las dos formas en la cantidad de ciclos de CPU que necesitan una y otra.
La línea de programa que enciende el LED es la siguiente.

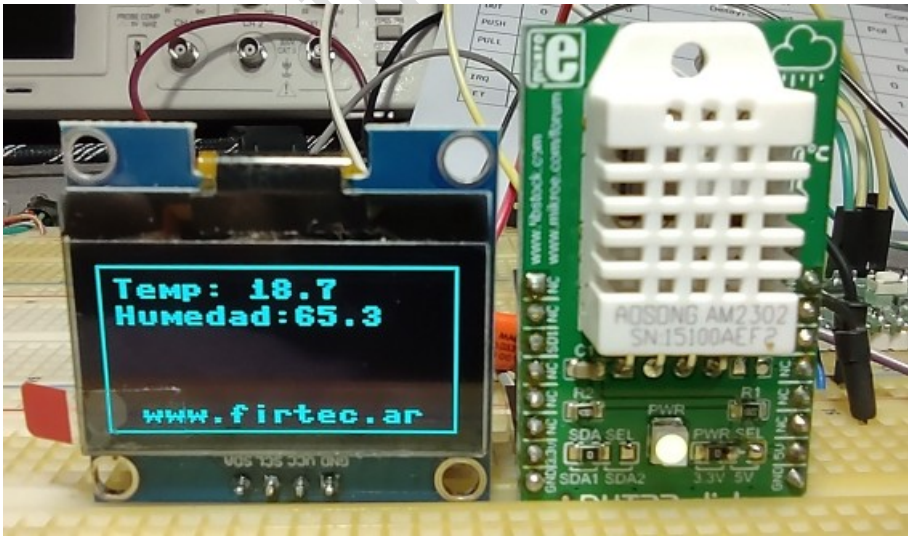
El programa PIO se encarga de evaluar el estado del pin con la función *boton()* que es en si el programa que cargamos en la máquina de estados.
En el programa tenemos la línea *wait(0, pin, 0)[10]* donde esperamos que el pin GPIO 16 pase a nivel bajo, y sumamos a la espera 10 ciclos de máquina. Inmediatamente después hay una instrucción de salto que se ejecutará si GPIO 16 estuviera a nivel alto luego de pasado 10 ciclos de máquina, obviamente esto no puede ocurrir si el nivel bajo se produjo por el accionar del botón y no por una interferencia. Con esta simple línea tenemos un simple y muy efectivo filtro para ruidos o interferencias eléctricas o incluso mecánicas del propio botón pulsador.

```
import time
from machine import Pin
from rp2 import PIO, StateMachine, asm_pio
PIO_boton = Pin(16, Pin.IN, Pin.PULL_UP)
# Pin 16 es puesto como entrada con resistencias
# Pull-Up
conta = 0
# Variable usada como contador
@asm_pio()
# Ninguna configuración aquí, el pin ya fue
# configurado con su resistencia Pul-Up.
def boton():
    label("repetir")
    wait(0, pin, 0)  [10]
# Espera por el pin 16 pase a cero luego espera 5mS
    jmp(pin, "repetir")
# Si el pin es "1" repetir
    irq(block, 0)
# Coloca bandera para incrementar el contador
    wait(1, pin, 0)  [10]
# Esperar que suelte el botón y suma 10 ciclos de
# máquina
def contador(sm):
# Código que se ejecuta si la interrupción PIO
# a ocurrido
    global conta
    conta += 1
# Incrementa la variable contador
    print(conta)
# Muestra el valor de contador

sm = StateMachine(0,boton,2000,in_base=PIO_boton)
# Configura la máquina, reloj 2Khz con un tiempo
# de 500uS.
```

Resultado obtenido con el ejemplo.

DHT22 + Pantalla OLED.

En este ejemplo vamos a conectar el sensor DHT22 al GPIO_2 y también la pantalla OLED para poder ver los datos enviados por el sensor directamente en la pantalla.
También hemos agregado unas funciones simples para generar líneas y poder construir un simple marco que contiene los datos del sensor.



Resultado obtenido con el ejemplo propuesto.

Puede ver que el error hace referencia a la biblioteca DHT22.py que en su código tiene mucho ensamblador PIO y al intentar ejecutar el programa una segunda vez consecutiva, como la memoria PIO no se ha borrado la memoria de comandos excede su capacidad y se genera el error.
Para corregir esto se debe agregar la siguiente línea.

```
rp2.PIO(0).remove_program()
```

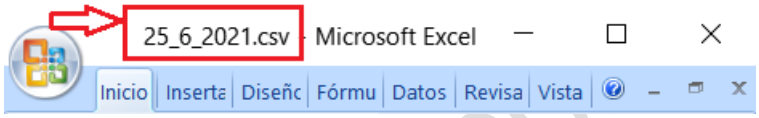
El módulo *rp2* escrito especialmente para el controlador RP2040 contiene una extensa cantidad de funciones y clases para el control y ajuste fino del microcontrolador y desde ahí podemos remover el programa que se ejecuta en la memoria PIO.
Se supone que este problema con la memoria PIO será solucionado con versiones futuras de MicroPython y según se informa el problema esta relacionado con un conflicto con el chip inalámbrico.

Pantallas Nextion con MicroPython.

Nextion es probablemente una de las mejores opciones para crear proyectos con electrónica. Son compatibles con Arduino, Raspberry PI, Atmel, Microchip, ARM, etc, además podemos utilizarlos de forma independiente contando la propia placa con una serie de pines de uso general incluso un RTC para sistemas de adquisición de datos.
Para crear una interfaz en una pantalla Nextion usaremos el editor Nextion, un software provisto por el propio fabricante que hace la tarea del diseño realmente simple.
Una vez que tenemos todo los elementos de la interfaz desplegados en la pantalla se envía la programación a la propia pantalla usando una simple conexión serial.
Este editor de Nextion es la mejor solución, en realidad casi la única para crear proyectos con las pantallas Nextion.
Lamentablemente solo existe en versión para Windows que se descarga a través del enlace oficial de Nextion.
La pantalla dispone solamente de 4 pines. Dos de ellos son de alimentación (cable rojo y negro) y los otros dos son de recepción (RX) y transmisión (TX) de datos a través de su puerto serie.
La pantalla se programa por medio de una conexión serial a 115200 baudios desde el propio editor Nextion, y también se conecta con nuestra placa (electrónica del usuario) por la misma conexión serial pero a 9600 baudios.
Esta claro que no se puede tener la pantalla conectada a la electrónica del usuario y también conectada al editor Nextion.

El reloj *DS3231* lo usaremos para conocer la hora de toma de muestra y también lo usaremos para generar el nombre del archivo, las siguientes líneas de programa le dan nombre al nuevo archivo que tendrá extensión *cvs*. Se lee el calendario y se acondicionan las lecturas para concatenar todos los datos que darán nombre al archivo.

```
ext = ".csv"
b = ds.Date()
dia = "%.d" % (b[2])
mes = "%.d" % (b[1])
year = "%.d" % (b[0])
fecha = "_".join((dia,mes,year))
archivo = "/fc/" + fecha + ext
```



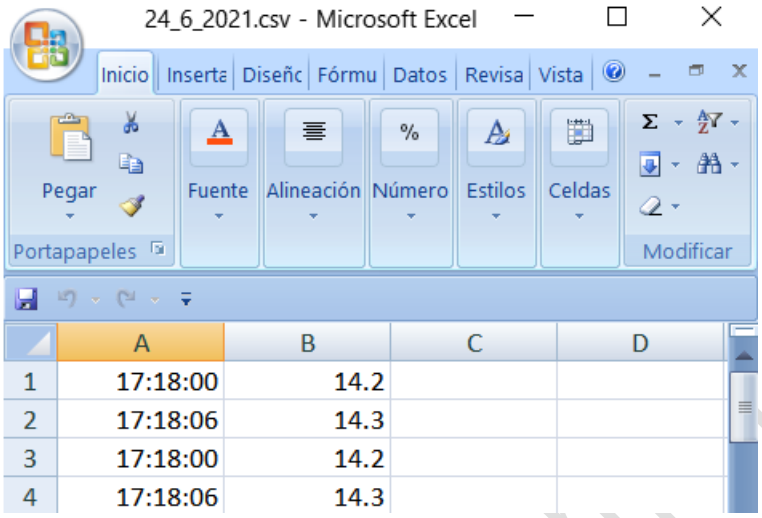
El archivo lleva por nombre la fecha en que se toman los datos.

En "*archivo*" tendremos una cadena con todo lo necesario para darle nombre al archivo que contendrá los datos del sensor.
El archivo se abre con el modificador "a", si no existe lo crea y si existe agrega datos al final de la última entrada.

```
with open(archivo, "a") as f:
                    n = f.write(historial)
```

El siguiente código forma la cadena que se escribirá en el archivo.
Se lee la hora, minutos y segundos. Se procesa los datos y se concatena todos los datos.

```
a = ds.Time()
hora = "%.d" % (a[0])
minutos = "%.d" % (a[1])
segundos = "%.d" % (a[2])
hora = ":".join((hora,minutos,segundos))
historial = ",".join((hora,temperatura))
historial = historial + "\n"
```



Contenido del archivo con los datos de temperatura.

El separador "," es necesario para que la hoja de calculo "entienda" que debe cambiar de celda y el salto de línea para pasar a la línea siguiente con sus respectivas celdas.
En nuestro ejemplo vamos a conectar la salida de pulsos del *DS3231* ajustado a un pulso por segundo para generar una interrupción en el pin GP_17.
Esta interrupción la usaremos para incrementar una variable que llamaremos "*retardo*" y llevar la cuenta del tiempo transcurrido entre cada lectura del sensor. Recuerde que aunque la lectura del sensor se realiza en una determinada secuencia de tiempo, si el valor leído es igual al anterior el dato se considera irrelevante y no se escribe en memoria.
El código del servicio de interrupción es el siguiente.

```
def ISR_rtc(p):
    global retardo
    retardo += 1
```

La variable "*retardo*" se ha definido como global porque será accedida desde afuera de la rutina de interrupción.
El código completo del ejemplo es el siguiente.

```
from machine import I2C, Pin
import DS3231
import time
from machine import SPI
```

6 cables es porque cada par de bobinas tienen un común separado, si tiene 5 cables es porque las cuatro bobinas tienen un polo común; un motor unipolar de 6 cables puede ser usado como un motor bipolar si se deja las líneas del común al aire.

Motores paso a paso Bipolares.

Estos tienen generalmente 4 cables de salida. Son un poco mas complejos para ser controlados debido a que requieren del cambio de dirección de flujo de corriente a través de las bobinas en la secuencia apropiada para realizar un movimiento.
Para simplificar el control de los motores paso a paso existen muchos tipos distintos de drivers que simplifican la activación y control de las bobinas de estos motores, uno de estos drivers es el A4988 fabricado por la empresa Pololu que veremos en el próximo ejemplo.

Pololu A4988.

Es bastante conocido dentro del mundo del CNC en general y de las impresoras 3D en particular, por ser un controlador de pasos con una buena resolución (de hasta 1/16) y un precio razonable.
La idea es usar el A4988 como interfaz de control entre Raspberry PI y el propio motor PAP.

Este tipo de controlador de pasos opera desde 8 a 35 V, por lo que lo que es apto para los motores de 12 V. También según las especificaciones, el A4988 puede proporcionar hasta 2 A de intensidad, 1 A si se usa sin disipador térmico, y que serán regulables por medio de un potenciómetro, para ajustarlo según las necesidades.

Programa Python para lee los datos enviados por Micropython.

El código para la recepción de los datos es el siguiente.

```
#============================================================
# Conectando con MicroPython usando Python 3 y
# la biblioteca Pyserial.
# - Firtec Argentina
# - www.firtec.com.ar
# ============================================================
# -*- coding: utf-8 -*-
import serial
from tkinter import Label
from tkinter import *
import time
puerto = serial.Serial('COM9', 9600, timeout=.1)
# Configuración del puerto serial. El número de
# puerto será diferente según su computador.
class Aplicacion:
        def __init__(self):
            global ventana
            ventana = Tk()
            ventana.iconbitmap('11.ICO')
            ventana.title(" Voltimetro UART")
            ventana.config(bg="coral")
# Color del fondo para la ventana
            ventana.geometry("330x150")
# Cambia el tamaño de la ventana
            ventana.resizable(0,0)
# Evita que se le pueda cambiar de tamaño a la
# ventana
            ventana.label_1 = Label(ventana, text="Voltios:
0.0", bg="coral", fg="black", font=("Helvetica", 40))
            ventana.label_1.place(x=14, y=30)
            ventana.label = Label(ventana, text="0",
bg="coral", fg="black", font=("Helvetica", 40))
```

Capítulo IV

Conectividad en redes.

Está claro que la característica mas relevante de esta placa Pico W es su capacidad de conexión a redes Wi-Fi, si bien el chip GYW43439 tiene el hardware para conexiones bluetooth de alta velocidad al momento de escribir estas líneas, el bluetooth no se encuentra activo en la placa Pico W por no contar con el firmware correspondiente. Se supone que en versiones posteriores esta placa también contara con con bluetooth (BLE).

Conceptos protocolos de RED.

El modelo de referencia OSI (**O**pen **S**ystem **I**nterconnection) fue creado en el año 1984 por la ISO (*Organización Internacional para la Estandarización*). El objetivo era solucionar los problemas de compatibilidad entre los sistemas provenientes de distintos fabricantes.

Este modelo define una estructura de siete capas, en donde cada una es completamente independiente del resto y soluciona un aspecto particular del sistema de comunicación en red.

Modelo teórico OSI para redes.

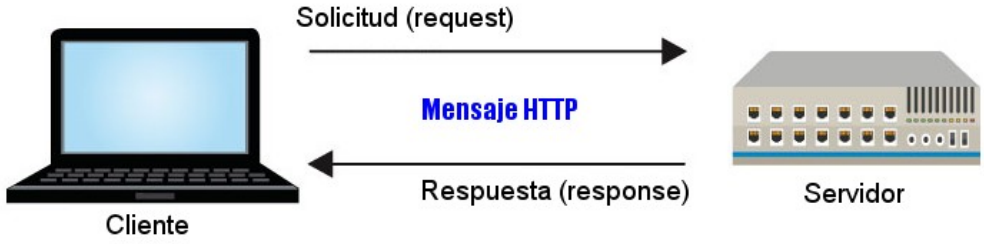
Capa 1: FÍSICA

La capa física es la que se encarga de transmitir los bytes por el medio que sea. En esta capa se define el hardware, la electrónica de conexión, los niveles de

Resultado obtenido con el ejemplo.

GET( ) y POST( ).

Cuando un usuario interactúa en un botón o formulario en una página web, los datos hay que enviarlos de alguna manera al servidor. Las dos formas de envío de datos posibles son el método POST o el método GET.



Normalmente POST está "escondido" y GET es mas visible. Cuando hacemos clic en una URL eso es GET y cuando se envía un formulario es POST.
Tanto el método GET como POST son protocolo HTPP el cual envía al servidor una petición (*request*) y recibe una respuesta a dicha solicitud (*response*).
El concepto GET es **obtener información** del servidor. Traer datos que están en el servidor, por ejemplo la temperatura de un sensor. Independientemente

Resultado obtenido con el ejemplo propuesto.

Recuerde, para conectar con el servidor solo escribimos en la barra de direcciones del navegador la dirección IP que tiene asignada nuestra placa Pico W por ejemplo 192.168.1.200 (esto dependerá de su configuración de red), el navegador interroga si hay "algo" en esa dirección y de ser así se inicia el dialogo con el servidor.
Los mensajes en ambos sentidos se leen carácter a carácter (byte a byte) y no pueden ser procesados hasta que se termine de recibir en su totalidad.
Básicamente una página HTML no es otra cosa que un texto con una sintaxis particular donde se mezcla tanto la información que se quiere mostrar como las órdenes que decodifica el navegador para darle un determinado formato al texto o información que se muestra en la página web.
El texto completo del ejemplo propuesto es el siguiente:

```
from machine import Pin
import time
from time import sleep
import utime
import network
import socket
import utime
```

```
    except OSError as e:
        w.close()
        print('Conexión cerrada')
```

Control web de un LED.

Para hacer las cosas un poco mas interesantes vamos a controlar el estado del led de placa Pico W desde una página web.

Sitio web propuesto para el ejemplo.

El sitio web que vamos a construir tendrá dos enlaces para tanto para encender como para apagar el led y un cartel indicador del estado eléctrico del led.
Observe que en este ejemplo la información intercambiada con el sitio web será bidireccional.
Por un lado enviamos mediante los enlaces comandos para encender o apagar el led y también vamos a recibir el estado que nos reporta la electrónica sobre el estado real del led.
El sitio web está desarrollado con el siguiente código:

```
html = """<!DOCTYPE html>
<html>
<head> <title>Raspberry Pico W</title> </head>
<body>
<meta charset="UTF-8">
<center>
```

```
    return ((p + var1 + var2) >> 8) + (self.dig_P7 << 4)
  def read_humidity(self):
    adc = self.read_raw_humidity()
    # print 'Raw humidity = {0:d}'.format (adc)
    h = self.t_fine - 76800
    h = (((((adc << 14) - (self.dig_H4 << 20) - (self.dig_H5 *
h)) +
          16384) >> 15) * (((((((h * self.dig_H6) >> 10) * (((h
*
                              self.dig_H3) >> 11) + 32768)) >> 10)
+ 2097152) *
                              self.dig_H2 + 8192) >> 14))
    h = h - (((((h >> 15) * (h >> 15)) >> 7) * self.dig_H1) >>
4)
    h = 0 if h < 0 else h
    h = 419430400 if h > 419430400 else h
    return h >> 12
  @property
  def temperature(self):
    "Return the temperature in degrees."
    t = self.read_temperature()
    ti = t // 100
    td = t - ti * 100
    return "{}.{:02d}C".format(ti, td)
  @property
  def pressure(self):
    "Return the temperature in hPa."
    p = self.read_pressure() // 256
    pi = p // 100
    pd = p - pi * 100
    return "{}.{:02d}hPa".format(pi, pd)
  @property
  def humidity(self):
    "Return the humidity in percent."
    h = self.read_humidity()
    hi = h // 1024
    hd = h * 100 // 1024 - hi * 100
    return "{}.{:02d}%".format(hi, hd)
```

IMPORTANTE:
Recuerde que siempre las bibliotecas o módulos contenidos en archivos independientes que no están integrados en las bibliotecas propias de MicroPython deben estar cargadas en memoria interna de Pico W.

Vamos a crear una simple pagina web para poder leer los datos de temperatura, presión y humedad del sensor.

```
        conn, addr = server.accept()
        conn.settimeout(3.0)
        print('Cliente conectado desde ', addr)
        request = conn.recv(1024)
        conn.settimeout(None)
        response = Sitio_Web()
        conn.send('HTTP/1.0 200 OK\r\nContent-type: text/html\
r\n\r\n')
        conn.sendall(response)
        conn.close()
    except OSError as e:
        conn.close()
        print('Conexión cerrada')
```

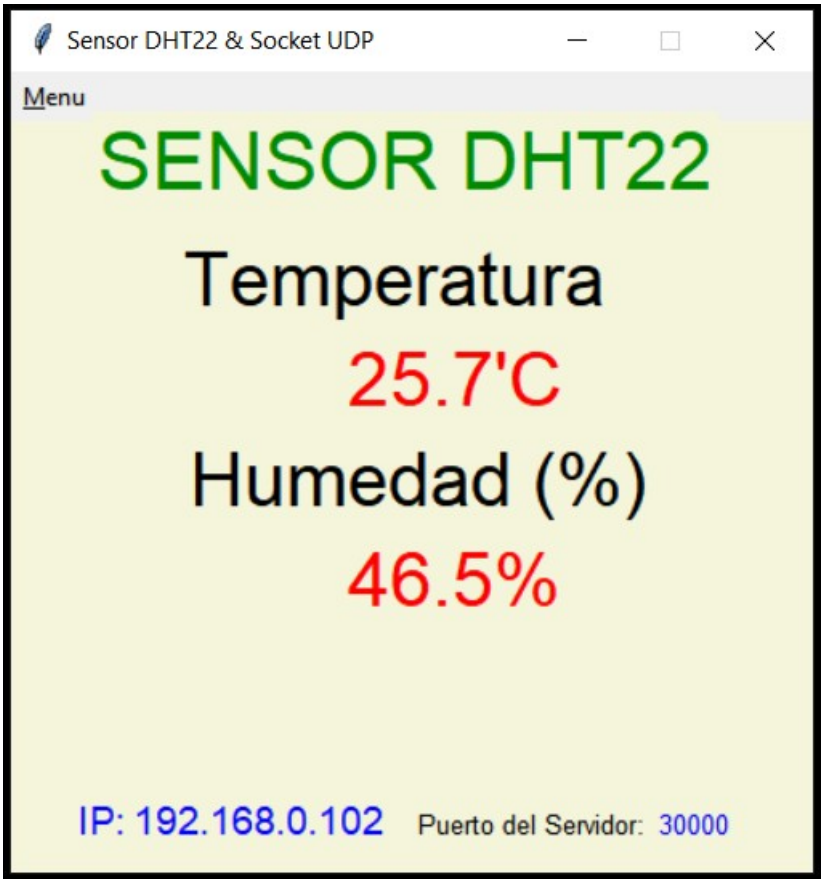
En el código luego que el socket *TCP* se ha creado y antes del *bind()* puede ver la linea.

```
server.setsockopt(socket.SOL_SOCKET, socket.
        SO_REUSEADDR, 1)
```

Esta línea permite re-utilizar el puerto 80 si una de las puntas del socket se desconecta.

Lectura del sensor BME280 mediante una web embebida.

En este código se puede ver claramente donde los datos son obtenidos desde el sensor y donde son enviados por el socket previamente acondicionados con el separador de campo.
También puede ver que la conexión se establece la misma dirección IP que tiene el servidor y el puerto asignado.
La ventana obtenida para el servidor será la siguiente.



Servidor UDP para leer el sensor DHT22.

El código MicroPython completo es el siguiente.

```
from machine import Pin
import time
from time import sleep
import network
import socket
from PicoDHT22 import PicoDHT22
import utime
```

http://firtec.net configura el acceso al sistema de control.

Una vez que los datos son salvados la red firadmin desaparece y el sistema pasa al modo usuario intentando validar las credenciales de acceso a la red Wi-Fi y cuando sea necesario las credenciales de Telegram.
Para entrar al modo programación se ha colocado un pequeño pulsador en el GPIO_16 que si se oprime el sistema se desconecta de la red Wi-Fi, borra todas las credenciales y pasa a modo programación.
Para detectar la actividad de las puertas se usaron sensores magnéticos que activan una interrupción por cambio de estado en el GPIO_11.

Sensor magnético para puertas o ventanas.

Observe como se ha escrito la rutina de interrupción con un funcionamiento algo particular para poder detectar cuando la puerta se abre y cierra usando

```
content="0;url=http://{{domain}}">
    </head>
    <body>
        <p>Redirecting...</p>
    </body>
</html>
```

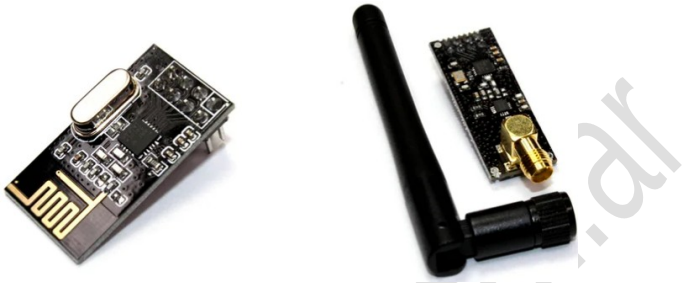
Pico W con enlace NRF24L01.

En mi caso una de las puertas a controlar estaba alejada de la señal del Wi-Fi y para solucionar esto coloqué una Pico estándar (sin Wi-Fi) conectada por el puerto SPI a un enlace NRF24L01.
Esta placa verifica el estado de la puerta y mediante el enlace de radio le comunica los estados a la otra Pico W que tiene otro NRF24L01 como receptor y conectividad Wi-Fi para enviar los respectivos mensajes.
El siguiente es el código completo de la Pico estándar .

```
import ustruct as struct
import utime
from machine import Pin, SPI
from nrf24l01 import NRF24L01
from micropython import const
from machine import Pin
from machine import WDT
bandera = 0
enviado = 0

def ISR_11(p):
         global bandera
         global enviado
         if (sensor.value() == 0):
             led_rojo.value(0)
             bandera = 2
             #led_rojo.value(1)
             #bandera = 1
             #enviado = 0
         if (sensor.value() == 1):
             led_rojo.value(1)
             bandera = 1
             enviado = 0

             #led_rojo.value(0)
             #bandera = 2

sensor = Pin(11, Pin.IN, Pin.PULL_UP)
sensor.irq(trigger=Pin.IRQ_RISING | Pin.IRQ_FALLING, handler=
```

En la pico w que se conecta a internet simplemente se han agregado las líneas de código para manejar el enlace de radio, prácticamente lo mismo que el transmisor salvo que los nodos están cruzados para que el receptor reciba la información del transmisor.

IMPORTANTE:
Los módulos de radio NRF24L01 se consiguen en dos modelos básicos.



Distintos modelos del NRF24L01.

Uno es simplemente el chip con sus conexiones y una antena estampada en la propia placa, este modelo funciona sin problemas montado junto a la placa pico w.
Pero el modelo que trae un amplificador lineal y una antena **NO FUNCIONA** si se colocan junto a la placa pico w debido a que las frecuencias del Wi-Fi y del emisor de radio se molestan entre si.
Otro detalle importante es que estos enlaces de radio solo operan con 3V, deberá alimentarlos desde la salida de 3V de la placa pico o colocar una fuente aparte para alimentarlos.

Que es MQTT.

Cuando los enfoques informáticos clásicos (servidores web, socket de red, etc) resultan ser demasiado "pesados" por la cantidad de recursos necesarios para sostenerlos o resultan ser soluciones exageradas para resolver una simple comunicación de algunos Bytes, entran en juego protocolos como **MQTT** (Message Queue Telemetry Transport), ideado por IBM es un protocolo usado para la comunicación máquina a máquina (M2M).
Es un protocolo específico para Internet de las cosas y orientado a la comunicación de sensores y dispositivos con una tasa de transferencia de datos baja, necesita muy poco ancho de banda y puede ser utilizado en la mayoría de los dispositivos con escasos recursos (CPU, RAM, etc).

La arquitectura de MQTT sigue una topología de estrella, con un nodo central que hace de servidor o "*broker*" normalmente con una capacidad teórica de hasta 10000 clientes.
El broker es el encargado de gestionar la red y de transmitir los mensajes, para mantener activo el canal, los clientes mandan periódicamente un paquete de datos y según el caso pueden esperar una confirmación del broker.
La comunicación se basa en "*topics*" o temas, y para que un cliente tenga acceso a la información debe estar subscrito al tema sin importar cuantos clientes estén siguiendo el tema.
Un cliente (cualquiera) puede publicar mensajes y los nodos, que deseen recibirlo deben estar subscrito a él.
La comunicación puede ser de uno a uno, o de uno a muchos. Un "*topic*" se representa mediante una cadena y tiene una estructura jerárquica separada con '/'.
Por ejemplo, "firtec/sensor_1/temperatura" o "firtec/sensor_2/ruido". De esta forma se pueden crear jerarquías de clientes que publican y reciben datos.

Por qué MQTT.

MQTT es un protocolo abierto, sencillo, ligero y fácil de implantar.
Es ideal para responder a las siguientes necesidades:

- *Está especialmente adaptado para utilizar un ancho de banda mínimo.*
- *Es ideal para utilizar redes inalámbricas.*
- *Consume muy poca energía.*
- *Es muy rápido y posibilita un tiempo de respuesta superior al resto de protocolos web actuales.*
- *Permite una gran fiabilidad si es necesario .*
- *Requiere pocos recursos procesadores y memorias lo que lo convierte en el candidato ideal para ser usado con microcontroladores.*

El MQTT no es el único protocolo que intenta imponerse, hay otros como XMPP, REST API y CoAp que también tienen ciertas ventajas pero sin dudas MQTT es uno de los mas populares.

Como funciona MQTT?

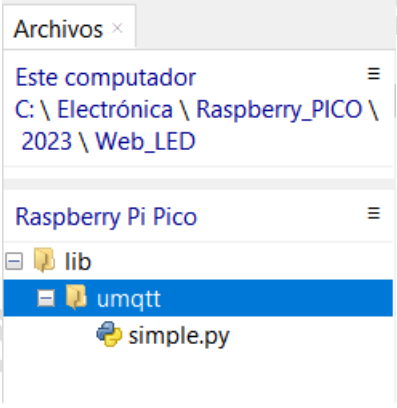
Como se dijo, es un servicio de publicación/suscripción basado en TCP/IP, es sencillo y sumamente ágil. El broker, recopila los datos que los *publishers*, los clientes o nodos que publican los tópicos.
Determinados datos recopilados por el broker se enviarán a determinados nodos que previamente así se lo hayan solicitado al broker.

Necesitamos tener instalado el paquete *umqtt.simple* y para esto hacemos lo siguiente desde la consola de Thonny.

```
>>> import upip
>>> upip.install("umqtt.simple")
  Installing to: /lib/
  Warning: micropython.org SSL certificate is not validated
  Installing umqtt.simple 1.3.4 from https://micropython.org/pi/umqtt.simple/umqtt.simple-1.3.4.tar.gz
```

Escribimos en la consola de Thonny .

Con esto estamos instalando el paquete que necesitamos para poder usar MQTT desde MicroPython.
Observe la imagen anterior con atención, debe escribir en la consola el texto mostrado tal cual se ve en la imagen.
Es importante que este conectado a la placa Pico W porque parte de la biblioteca se instala directamente en la memoria de Pico W como vimos en ejemplos anteriores .

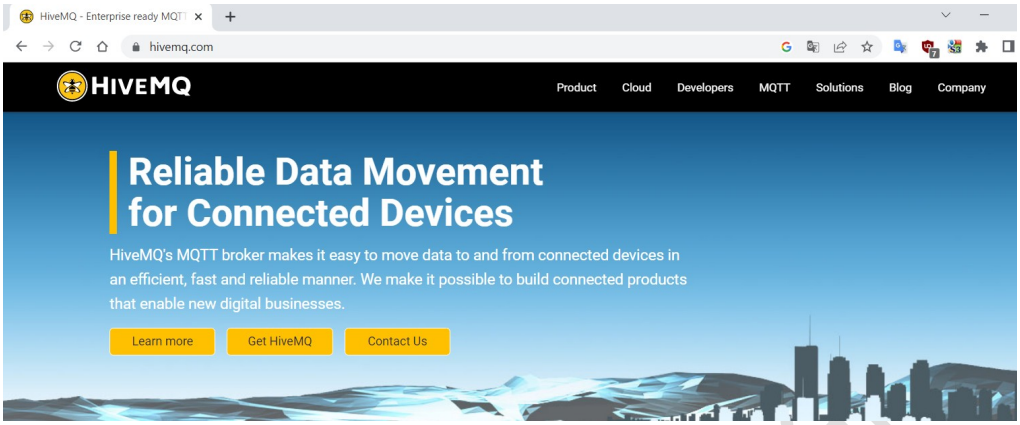


La biblioteca se instala en la memoria de Pico W.

En la lengüeta Archivos de Thonny deberá ver una carpeta *lib* ya cargada en la memoria de Pico W.
Luego de esto estará en condiciones de usar MQTT con MicroPython y Pico W.

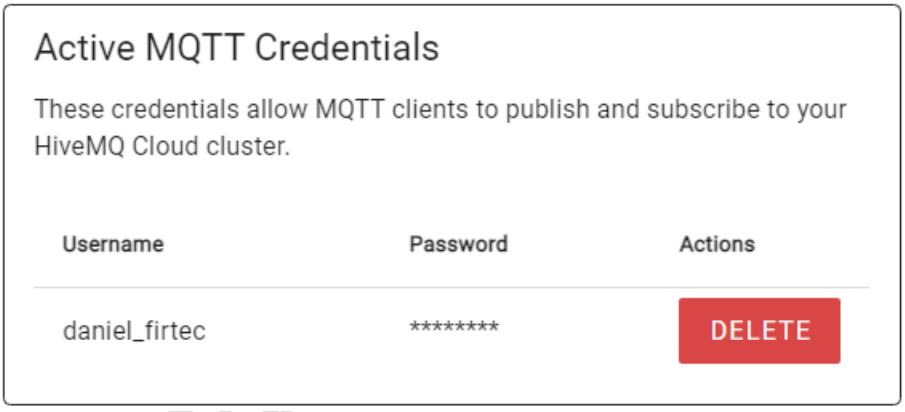
Configurando un Broker en la nube.

Vamos a usar ***HiveMQ*** que tiene una opción libre con prestaciones mas que suficientes para desarrollos simples o que no involucren muchos sensores o datos en red. Lo primero que necesitamos es crear una cuenta dentro de la plataforma.



Sitio web de HiveMQ donde se crea una cuenta para usar MQTT.

Una vez que tenemos un perfil dentro del sitio tendremos un nombre de usuario y una clave de acceso.

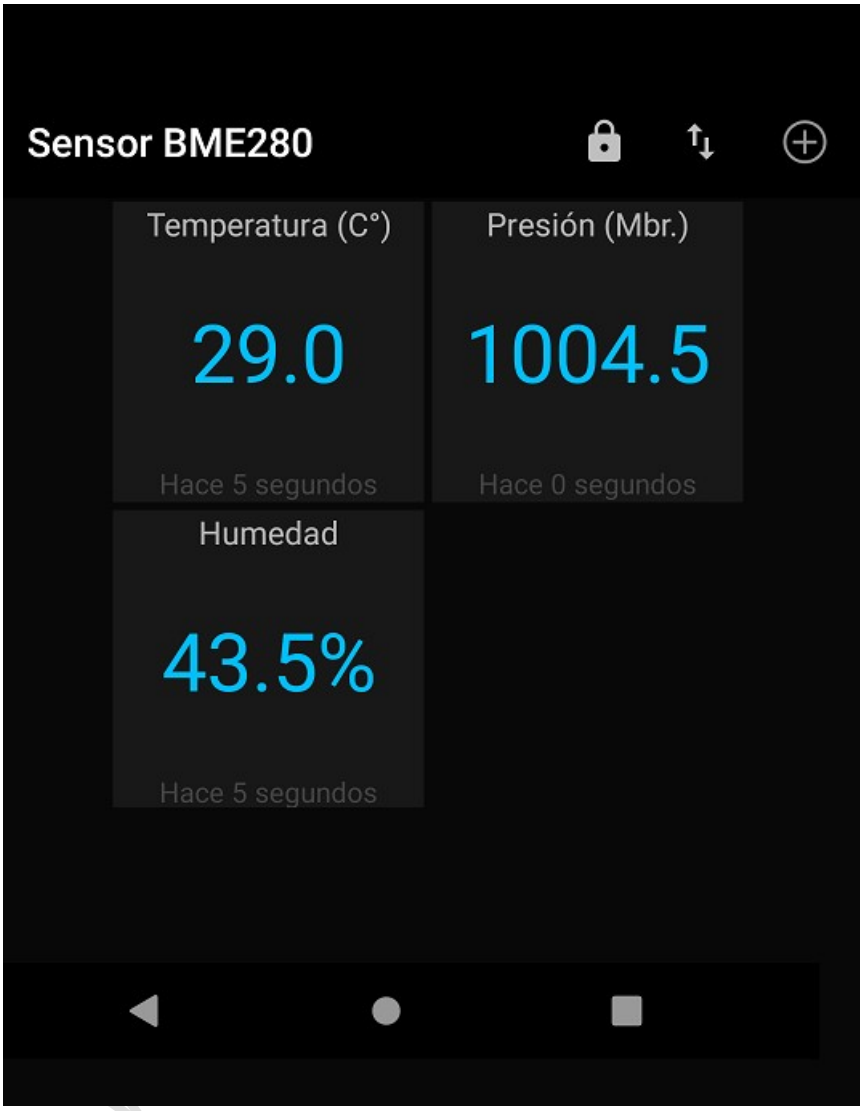


Credenciales de acceso para MQTT.

Estas credenciales serán usadas en nuestro programa MicroPython para acceder al espacio asignado en el broker.
Recuerde que un broker no hace nada mas que recibir las publicaciones y distribuirlas a los clientes que están subscriptos a las mismas por lo tanto no hay mucho mas que hacer cuando creamos el perfil y se asigna la clave y el usuario.
Este perfil en la nube es único, algo así como la dirección IP en una red informática.
Esta dirección es una larga lista de caracteres y números y la usaremos para poder publicar los tópicos.

Configuraciones para el acceso al broker MQTT

Puede ver que se encuentran los datos de usuario y clave de acceso lo mismo que la URL de donde se encuentra el broker.
Una vez que ha logrado conectar con el broker verá un signo “+” que al hacer click sobre el se abre una ventana como se ve en la imagen siguiente.



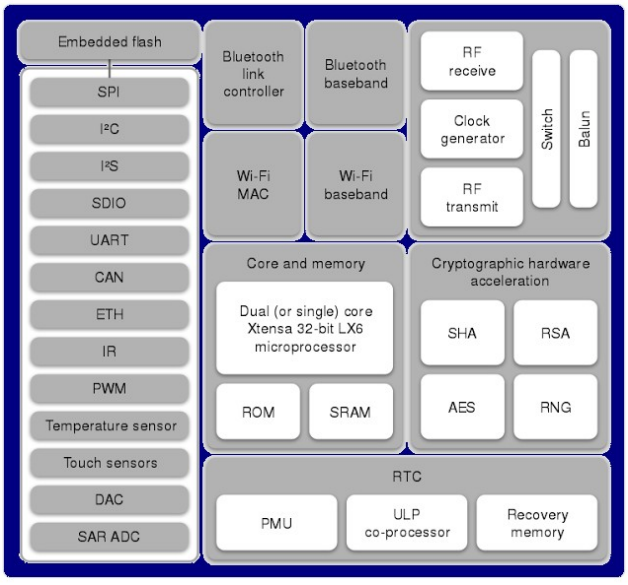
Lectura de datos desde el broker MQTT para el sensor BME280.

Esta claro que en este ejemplo solo estamos haciendo la lectura de los datos de un sensor ambiental que han sido publicados en el broker y la aplicación que estamos usando se ajusta para ver estos datos pero de la misma forma podríamos estar viendo cámaras o evaluando el estado de sensores de movimiento o haciendo mediciones mas complejas y entonces seguramente vamos a necesitar otro tipo de aplicación para el móvil.

Con este simple cambio obtenemos una gran mejora en la respuesta a los comandos enviados para encender o apagar el led.

ESP32 con MicroPython.

ESP32 es una serie de microcontroladores de bajo costo y bajo consumo con Wi-Fi integrado y Bluetooth.
La serie ESP32 emplea un microprocesador Tensilica Xtensa LX6 con doble núcleo, existiendo una versión de un solo núcleo. Está desarrollado de Espressif Systems, una compañía china con sede en Shanghai.
Es un controlador con un grado de complejidad muy alto como se puede ver en el diagrama de bloques internos en la imagen siguiente.
Este controlador está presente en muchos sistemas de alarmas y controladores para domótica debido al alto nivel de conectividad que ofrece lo mismo que su eficiencia en cuanto a conexiónes.
ESP32 es sin dudas el sucesor del popular ESP8266.



Estructura en bloques del ESP32.

Algunas características del ESP32 son las siguientes.

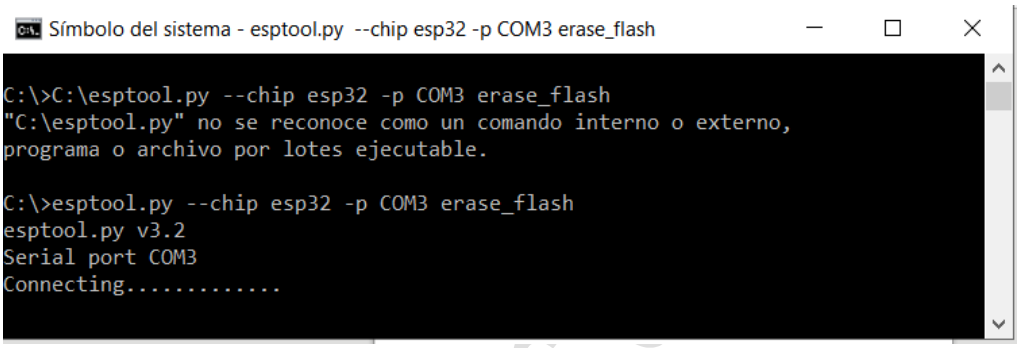
- *CPU: microprocesador Xtensa de doble núcleo y 32 bits LX6, que funciona a 160 o 240 MHz y funciona con hasta 600 DMIPS .*
- *Memoria: 520 KiB SRAM.*
- *Wi-Fi: 802.11 b / g / n / e / i .*
- *Bluetooth: v4.2 BR / EDR y BLE.*

instalado!!) abrimos una ventana de sistema escribiendo CMD y una vez que tenemos la clásica ventana abierta escribimos.

```
pip3 install esptool
```

Luego limpiamos el contenido de memoria del ESP32.

```
C:\esptool.py --chip esp32 -p COM9 erase_flash
```



Borrando la memoria interna de ESP32.

En la imagen anterior se observa como el sistema intenta conectar con la placa ESP32. **Es necesario que durante este proceso el usuario oprima el botón marcado como "O" para permitir que el nuevo software se descargue en la memoria Flash de la placa ESP32 o el borrado ocurra.**

Si no actúa sobre este botón el software no se descargará.
Obviamente deberá reemplazar el puerto COM por el puerto asignado en su sistema.
Una vez la memoria a sido borrada se podrá grabar el firmware MicroPython.

Puesto que con 10 bits tendríamos una secuencia de número en el conversor que irían de 0 a 1023, en total 1024 números.
Un detalle interesante es la línea *adc.atten(adc.ATTN_11DB)* con el atenuador podemos ajustar el nivel voltaje de entrada para obtener rangos mas acotados con mayo precisión según los siguientes valores.

- ADC.ATTN_0DB: 0dB atenuación para un voltaje máximo de 1.0 voltio configuración por defecto.
- ADC.ATTN_2_5DB: 2.5dB atenuación para un voltaje aproximado de 1.34 voltios.
- ADC.ATTN_6DB: 6dB atenuación para un voltaje aproximado de 2.00 voltios.
- ADC.ATTN_11DB: 11dB atenuación para un voltaje aproximado de 3.6 voltios.

No olvide ajustar el atenuador ya que la configuración por defecto solo permite medir voltajes de 1 voltio como máximo.

Pantalla OLED con MicoPython y ESP32.

Para usar la pantalla OLED necesitamos contar con la bliblioteca *SH1106.py.*
La pantalla se conectará en el puerto I2C_0 en los pines *GPIO_21* para el SDA y *GPIO_22* para SCL con una frecuencia de reloj de 400 Khz.
Un programa básico que solo muestre un cartel sería el siguiente.

```
from machine import Pin, I2C
# Importa la biblioteca para el puerto I2C
import sh1106
# Importa la biblioteca para la pantalla
import utime
# Importa la biblioteca para el control de tiempos
i2c = I2C(0, sda=Pin(21), scl=Pin(22), freq=400000)
#  Configura el puerto I2C
display = sh1106.SH1106_I2C(128,64,i2c,Pin(4),0x3c)
# Configura el driver de pantalla
display.sleep(False)
display.fill(0)   # Limpia la pantalla
display.text('Firtec', 0, 0, 1)
# Muestra un único mensaje
display.show() # El mensaje se hace visible
print('Terminado')# Muestra cartel en la consola
```

Conectado ESP32 a la red WiFi con MicroPython.

Una de las características destacadas del ESP32 al igual que Pico W es la facilidad con que tenemos conectividad sobre una red WiFi, el ejemplo siguiente muestra la estructura mínima necesaria de un programa para acceder a una red WiFi.

```
import network
# Biblioteca para el manejo de redes
import utime
ap_if = network.WLAN(network.AP_IF)
# Crea el objeto para interactuar con el router
sta_if = network.WLAN(network.STA_IF)
# Crea el objeto para interactuar con el punto de acceso
if not sta_if.isconnected():
# Verifica si se conecto a una red
        print('Conectado con la red WiFi...')
# Cartel para saber lo que esta pasando
        sta_if.active(True)
# Interfaz activa
        sta_if.connect('xxxxxxx','xxxxxxx')
# Credenciales de la red
        while not sta_if.isconnected():
# Bucle para intentar re-conectar
            pass
#print('network config:', sta_if.ifconfig())
print(sta_if.ifconfig()[0])
# Muestra la dirección IP asignada al ESP32
while True:
    pass
```

Servidor Web con MicroPython y ESP32.

Vamos a desarrollar un pequeño código que controlará mediante una página web el estado del *GPIO_5* donde tenemos conectado un led.
Para hacerlo funcionar vamos a crear un socket TCP que "*escucha*" en el puerto 80. Este puerto es el asignado para recibir las solicitudes que vienen desde los navegadores.

Página web en ESP32 para controlar el estado de un pin.

Cuando el socket recibe una solicitud desde el navegador este contesta con un encabezado *HTTP* para indicar que se enviará un contenido con formato *HTML* y seguidamente envía la página web.

Todo el sitio web no es mas que un larga cadena de texto ya formateado para que sea entendido por el navegador, incluso todo el sitio se pude desarrollar en algún programa especifico para el diseño de sitios web y luego pasarlo a la cadena que será enviada por el socket.

La cadena que contiene el sitio web es la siguiente:

```
html = """<html><head><title>Micropython & ESP32</title><body
style=background:#F0FFFF>
<h1>Control de un pin con ESP32</h1>
<meta name="viewport" content="width=device-width,
initial-scale=1">
<link rel="icon" href="data:,"> <style>html{font-family:
Helvetica; display:inline-block; margin: 0px auto;
text-align: center;}
h1{color: #0F3376; padding: 2vh;}p{font-size:
1.5rem;}.button{display: inline-block; background-color:
#e7bd3b; border: none;
border-radius: 4px; color: white; padding: 16px 40px;
text-decoration: none; font-size: 30px; margin: 2px;
cursor: pointer;}
button2{background-color: #4286f4;}</style></head><body>
<p>Estado del Pin: <strong>""" + pin_estado + """
</strong></p><p><a href="/?led=si">
<button class="button">Encendido</button></a></p>
<p><a href="/?led=no"><button class="button
button2">Apagado</button></a></p></body></html>
<br>
<hr Size=7 noshade/><H5><font color='black'>by. Firtec
```

Recuerde:

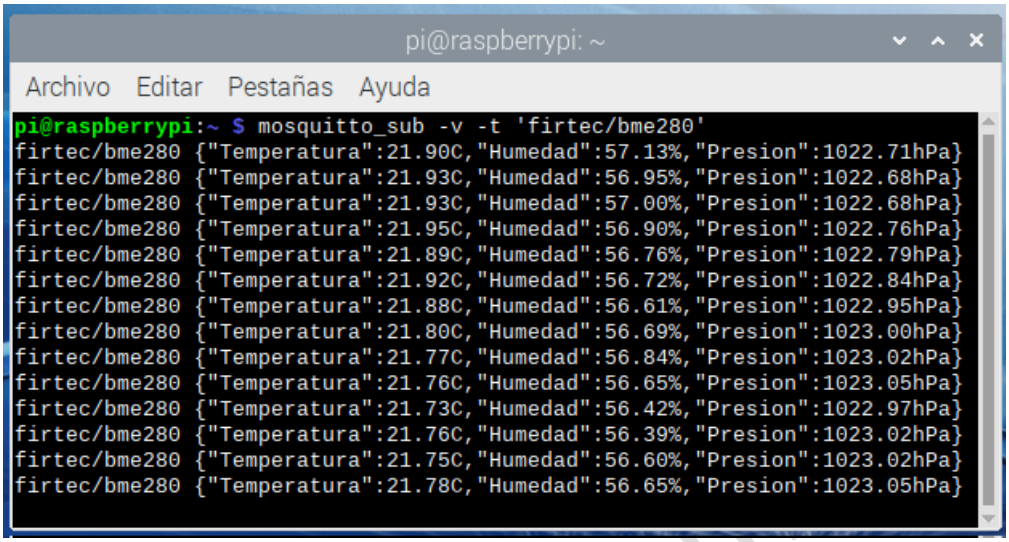
La totalidad del sitio web solo se carga una vez cuando el cliente (navegador) se conecta con el servidor, luego de esto el navegador no envía ningún otro mensaje hasta que la función Ajax envíe una solicitud de datos.
Este funcionamiento lo puede ver en las siguiente líneas.

```
consulta = mensaje.find('/leer_sensor')
# Busca la cadena enviada por Ajax y guarda el
# indice en consulta
    if consulta == 6:
# Si el indice es 6 el mensaje recibido es el
# comando esperado
        temp = bme.temperature
# Mide la temperatura
        hum = bme.humidity
# Mide la humedad
        pres = bme.pressure
# Mide la presión
        respuesta = temp + "|"+ hum + "|"+ pres
# Ensambla la trama de respuesta para Ajax
        led.value(not led.value())
# Cambia de esta do el led testigo
    else:
# Si no es el comando Ajax enviar todo el sitio web
       respuesta = web_page() # Envía el sitio
```

Como se puede ver el funcionamiento con web´s embebidas no es diferente a los visto en los ejemplos con Pico W. Si bien es verdad que el interprete MicroPython para ESP32 tiene algunas diferencias con el usado para Pico W, prácticamente todos los métodos son los mismos incluso cualquier ejemplo para Pico W puede ser usado en ESP32 con ajustes mínimos.

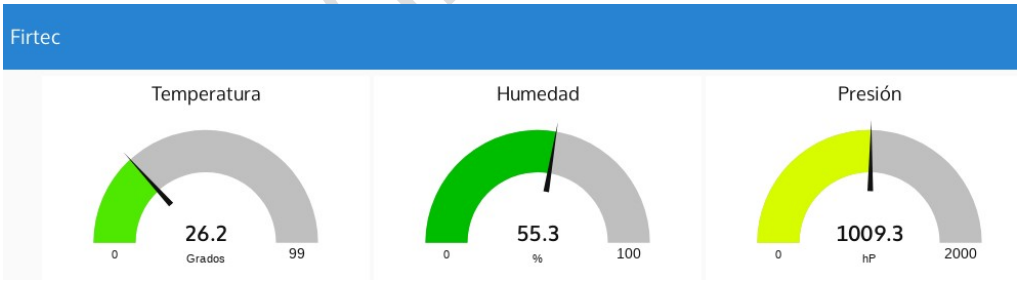
Mosquitto como broker MQTT en Raspberry PI.

Como ya sabemos para trabajar con MQTT necesitamos un broker que es el encargado de distribuir los mensajes a los clientes.
Podemos usar algún broker que están en línea como el que se trató en los ejemplos para Pico W.
Usar algún broker en línea tiene la ventaja de no tener que instalarlo, solo nos damos de alta, configuramos los tópicos y empezamos a procesar los datos.
Sin embargo para esto es requisito que exista una conexión estable y permanente a Internet puesto que el broker esta en la nube.
Para evitar esta situación vemos un ejemplo donde se instala un broker en una



Resultado obtenido con el ejemplo propuesto.

Para verificar su funcionamiento podemos abrir una consola en Raspberry y escribimos *<mosquitto_sub -v -t 'firtec/bme280'>*.
Con esto estamos creando un cliente que suscribe al tópico *firtec/bme280*.
Como se puede ver el cliente recibe los datos del *tópico* enviado por ESP32, también podríamos hacer lo mismo usando *Node-Red* y crear una interfaz mas simpática que reciba los datos del *tópico* como se aprecia en la siguiente imagen.



Resultado obtenido con *Node-Red* incluido en Raspberry PI.
Si necesitamos tener los datos en el móvil desde el *Play Store* descargamos alguna de las tantas aplicaciones para MQTT como la que vimos en Pico W y podremos acceder a los datos desde cualquier lugar.