KIT BOX

UNIT DUALMCU

RP2040+ESP32

Development board

Manual de Prácticas

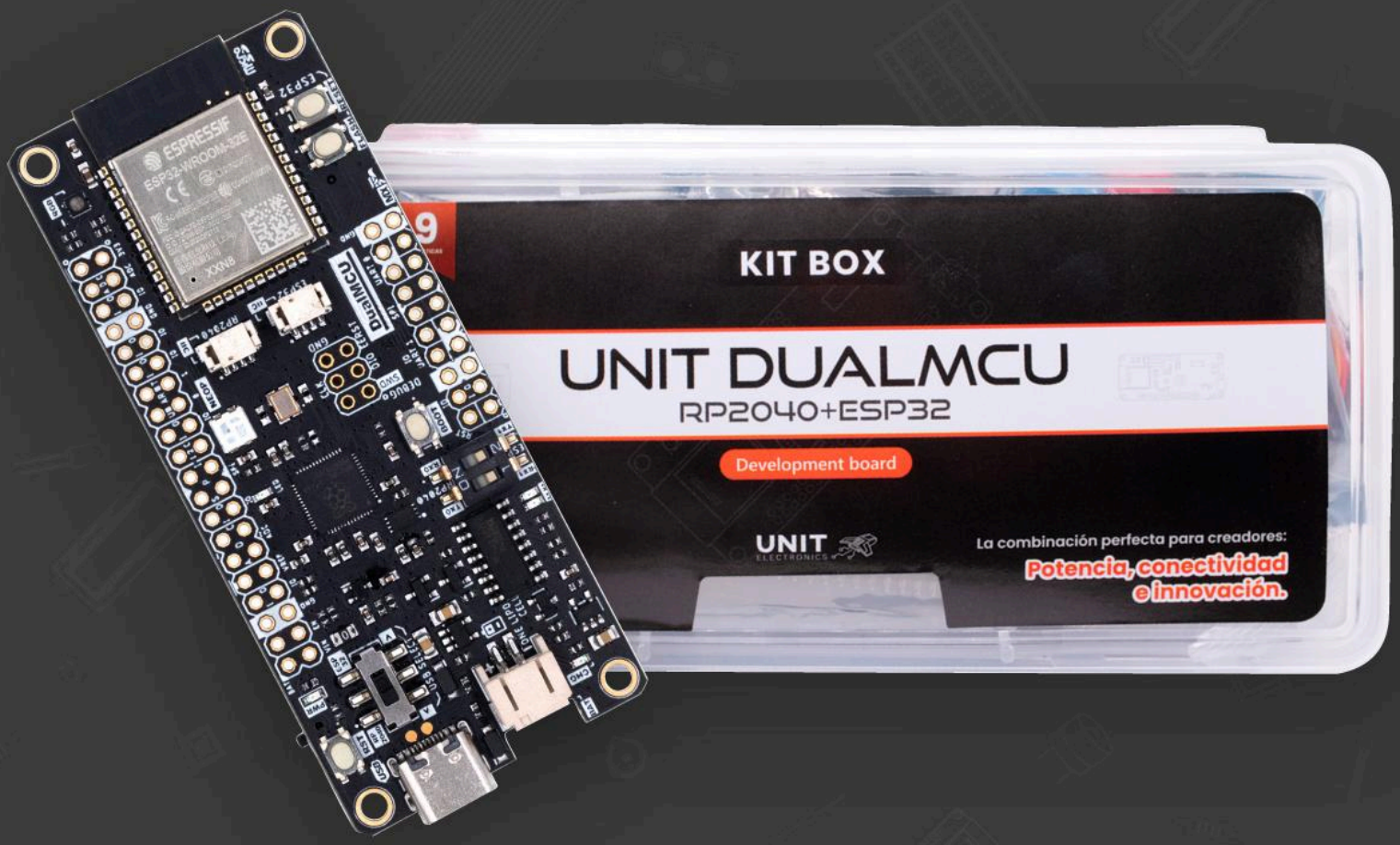


Imagen solo con fines ilustrativos. El producto real puede diferir.

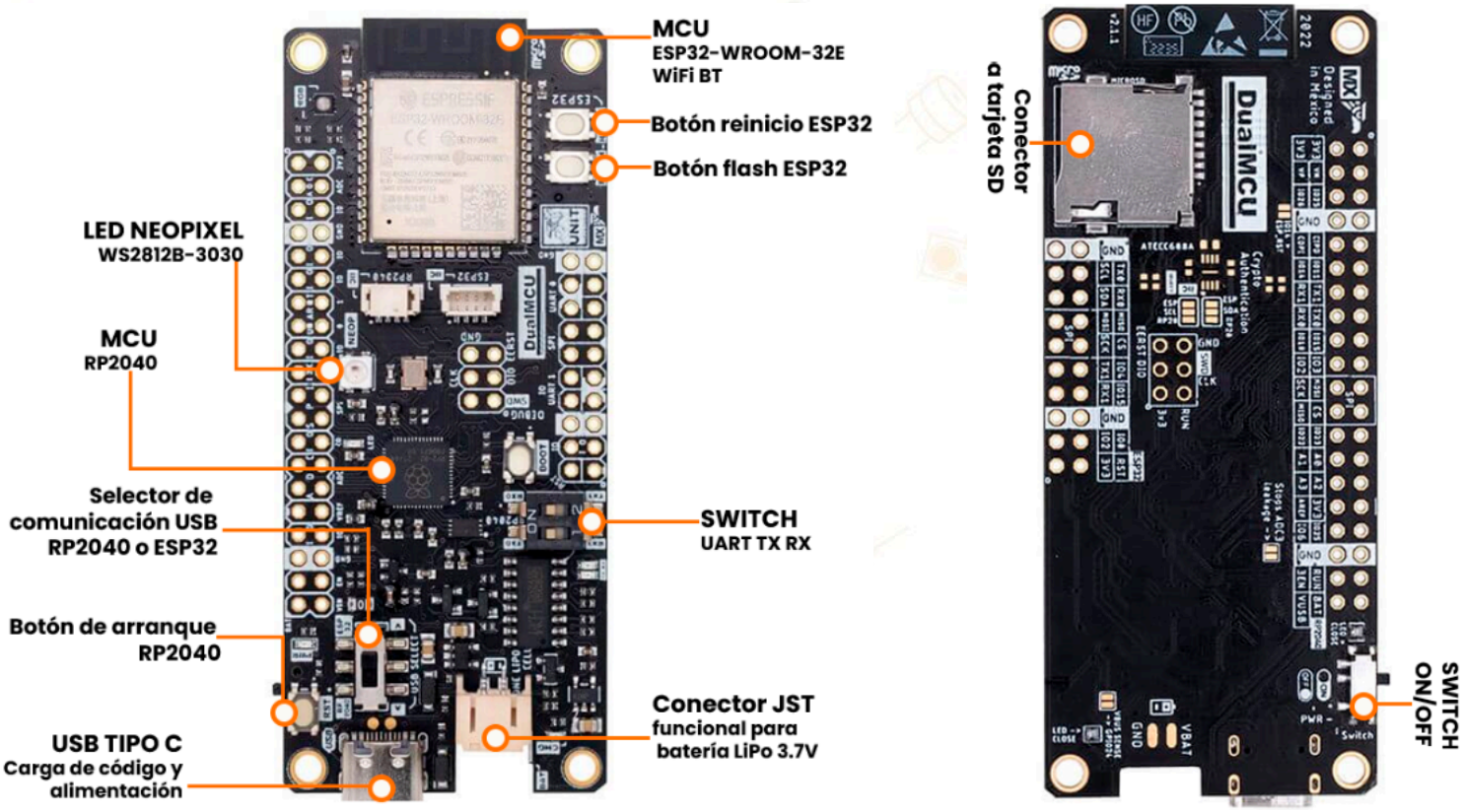
UNIT ELECTRONICS
TIENDA DE COMPONENTES ELECTRÓNICOS

ÍNDICE

Primeros pasos con DualMCU

La DualMCU es una tarjeta de desarrollo que reúne el microcontrolador Raspberry Pi RP2040 y un chip Espressif ESP32 WROOM , aprovechando al máximo el doble núcleo Arm® Cortex®-M0+ de 32 bits para realizar proyectos de Internet de las Cosas con conectividad Bluetooth® y Wi-Fi.

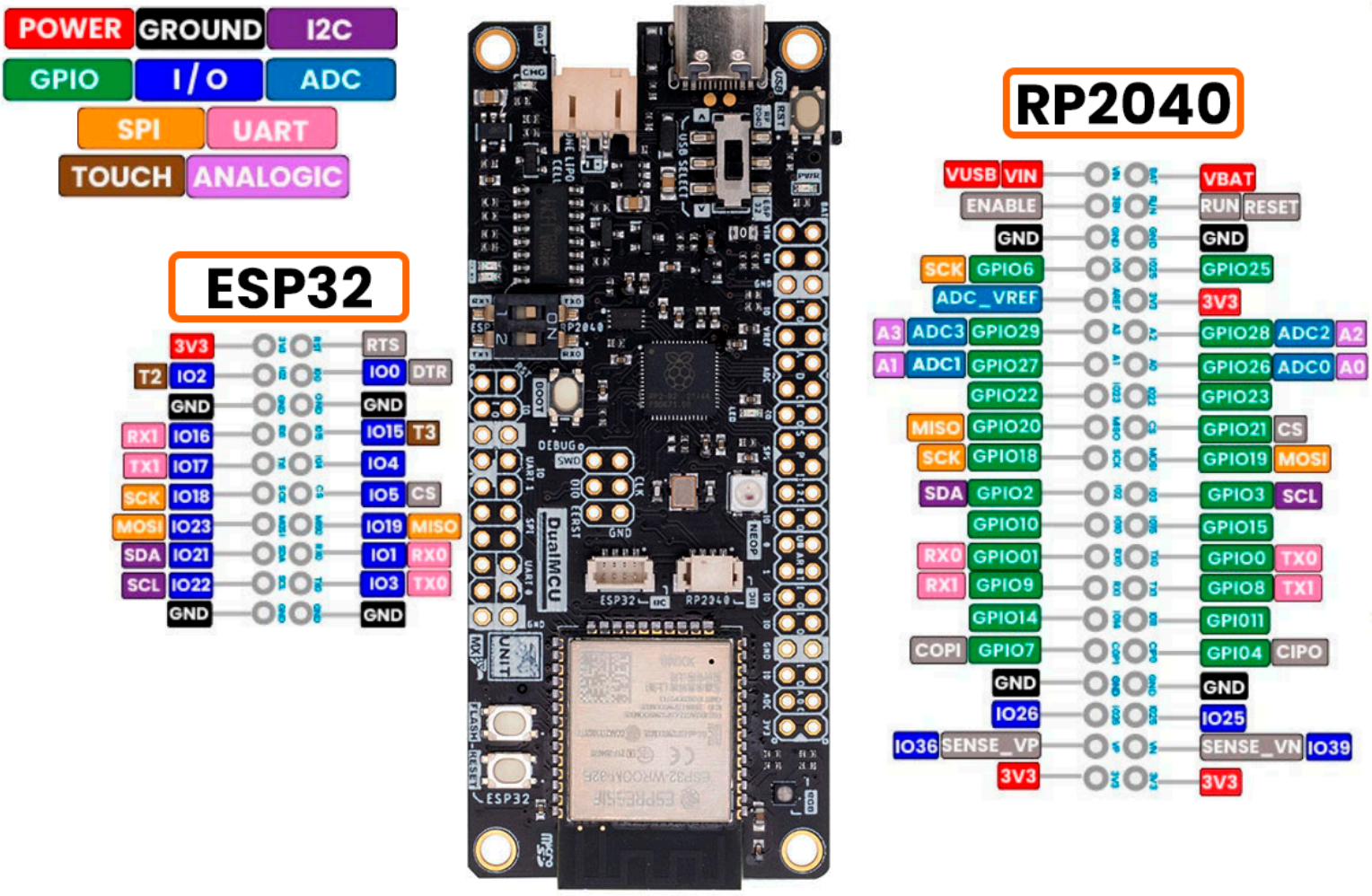
La placa de desarrollo DualMCU es compatible con el entorno de desarrollo integrado (IDE) Thonny (Micropython,) y Arduino(C++). Las prácticas que encontrarás dentro de este manual serán desarrolladas para el entorno de programación MicroPython con la propuesta de que también puedas desarrollar de manera independiente las prácticas en C++.



Pines

Una parte fundamental en el uso de la tarjeta de desarrollo DUALMCU, es el reconocimiento de la ubicación de los pines para cada una de los microprocesadores que incluye: ESP32 y RP2040.

Los pines para cada uno de los MCU están ordenados en el perímetro de la tarjeta de de desarrollo como se muestran a continuación:



Como podrás observar de lado izquierdo se encuentran todos los pines para el uso de la ESP32 y de lado derecho encontrarás los GPIO dedicados al RP2040.

Lo interesante de de este uso es que tienes cerca de 44 GPIO en conjunto , 32 para PWM; contando con interfaz de comunicación para I2C, UART, SPI, Programmable I/O Y Two-Wire Automotive Interface.

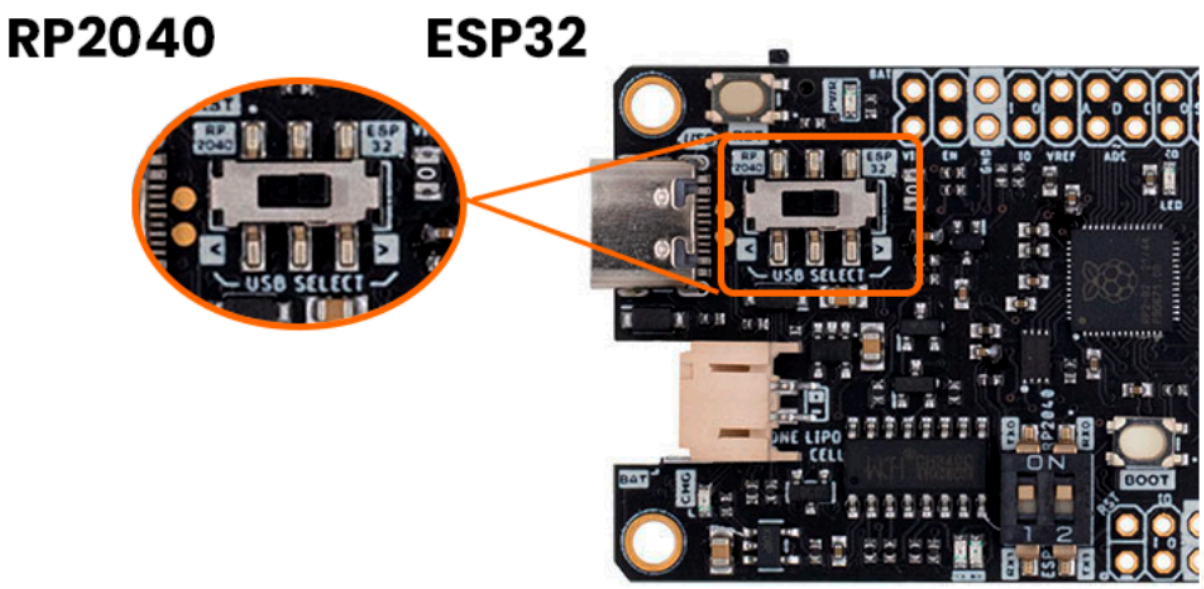
Selector de microcontrolador

Como se ha comentado la tarjeta DUALMCU cuenta con dos microprocesadores , cada uno con sus pines y GPIOS dedicados. Para poder trabajar en uno u otro MCU, será necesario activar el selector incluido en la tarjeta.

En este manual, al inicio de cada práctica se indicará con qué MCU se estará trabajando para que puedas estar atento a estos cambios de switch.

La intención es darte a conocer que puedes usar solo una parte de la tarjeta para una sola práctica o ambos MCU para una misma práctica y sacar el mayor provecho a este tipo de tarjetas de desarrollo; por ejemplo robusto, el ESP32 se puede dedicar a la comunicación vía WiFi y el RP2040 a realizar funciones y detección de datos con sensores y actuadores.

El interruptor con el que constantemente estaremos trabajando es el que se muestra a continuación:



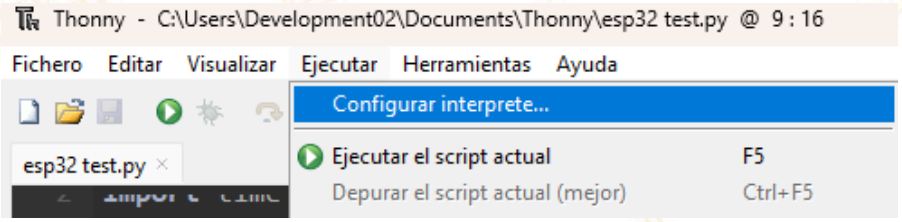
Preparación de Thonny para uso de la DUALMC

Se realizará la siguiente configuración para poder usar la tarjeta DualMCU en el entorno de IDE Thonny.

Instalación de Software

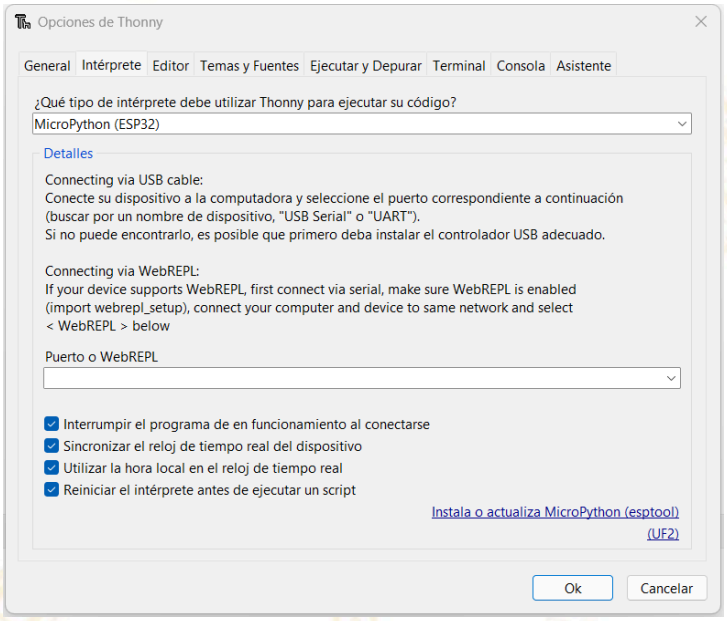
En el siguiente link podrás descargar el firmware para poder desarrollar tus prácticas en Thonny: **Firmware Thonny.** Posteriormente a la instalación pasaremos a la configuración del ambiente.

Dirígete a "Ejecutar" -> "Configurar intérprete" para completar la configuración.



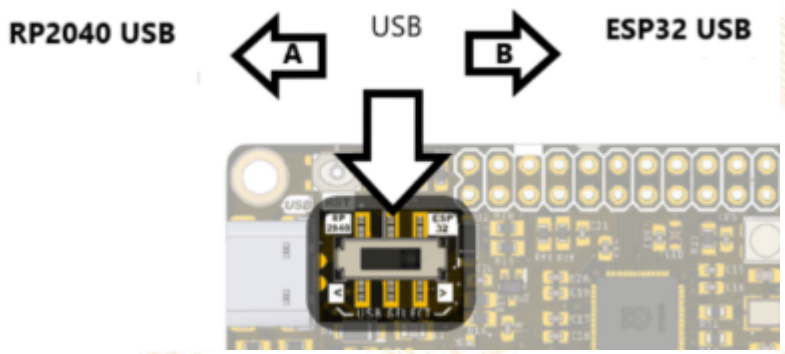
NOTA:En caso de que la UNIT DUALMCU ESP32 no sea reconocida será necesario instalar el controlador CH340.

Al hacerlo, se abrirá la siguiente ventana,

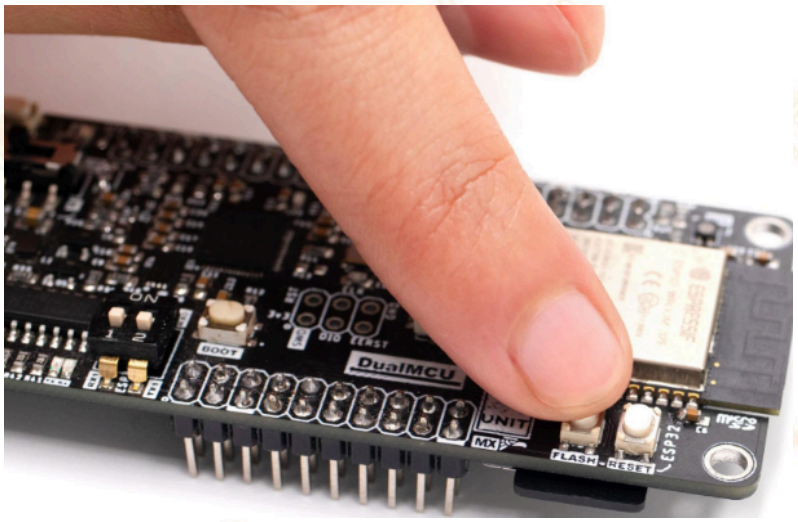


Actualización Firmware ESP32

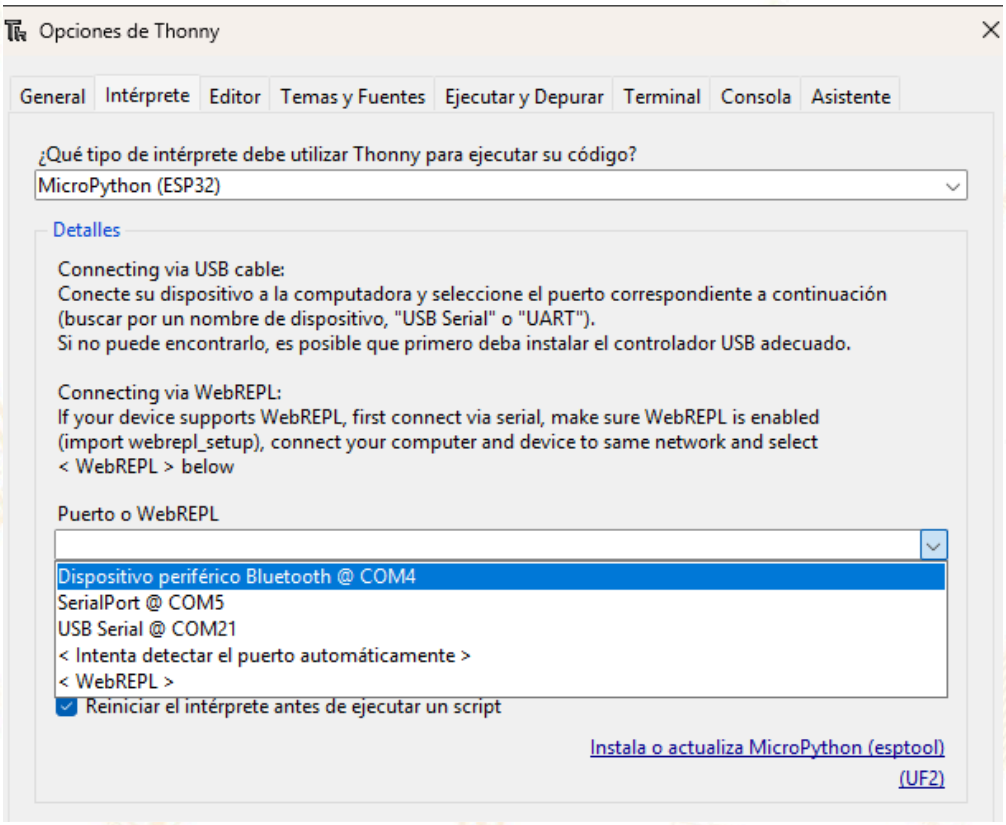
Físicamente, selecciona con el switch a la ESP32



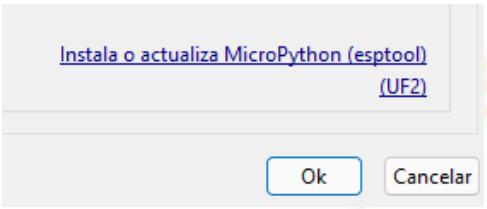
Adicional, presiona el botón de FLASH para ESP32 y sin dejar de presionar conecta el dispositivo a la PC.



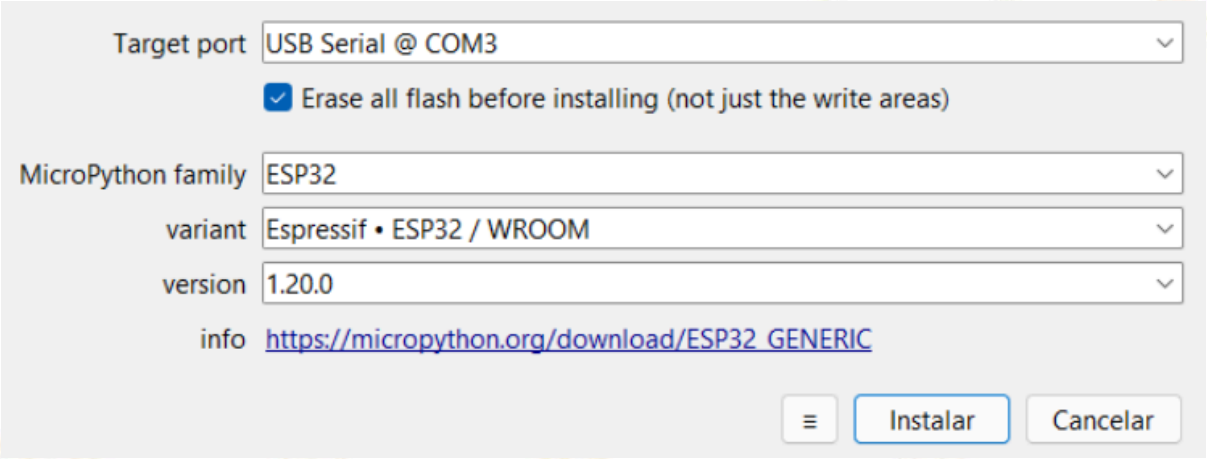
Volviendo a la interfaz de Thonny (Opciones Thonny) y después de que hayas conectado la tarjeta a la PC, podrás visualizar en la sección Puerto acorde a tu tarjeta:



Selecciona el puerto y posteriormente da click en **Instala o Actualiza MicroPython**



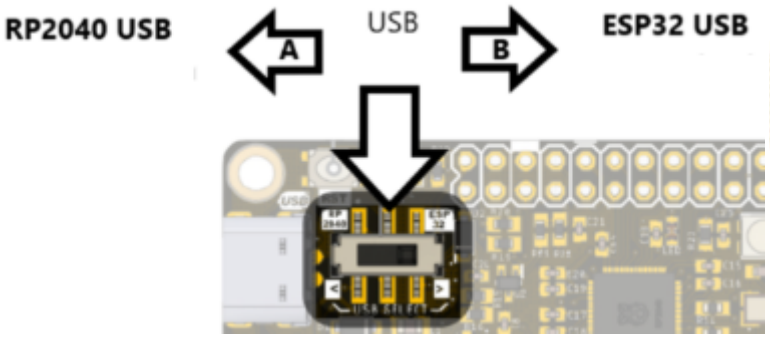
Posteriormente, coloca nuevamente el puerto que es el indicado para el uso de la ESP32 y la siguiente configuración e instala.



Finalmente en la ventana principal de Thonny podrás observar que la tarjeta ESP32 aparece como opción para trabajar. Ten en cuenta que el puerto COM es asignado por la máquina y puede variar.

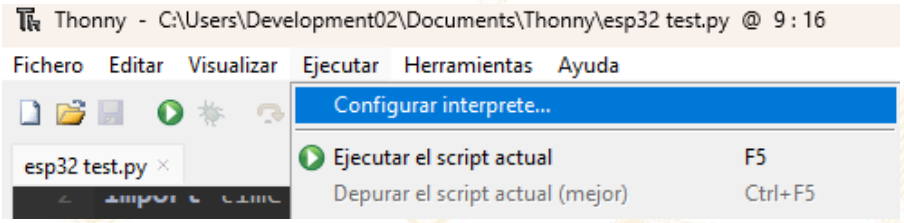
Actualización de firmware RP2040

Ahora realizaremos el cambio del switch en posición A para iniciar la UNIT DualMCU con el microcontrolador RP2040.

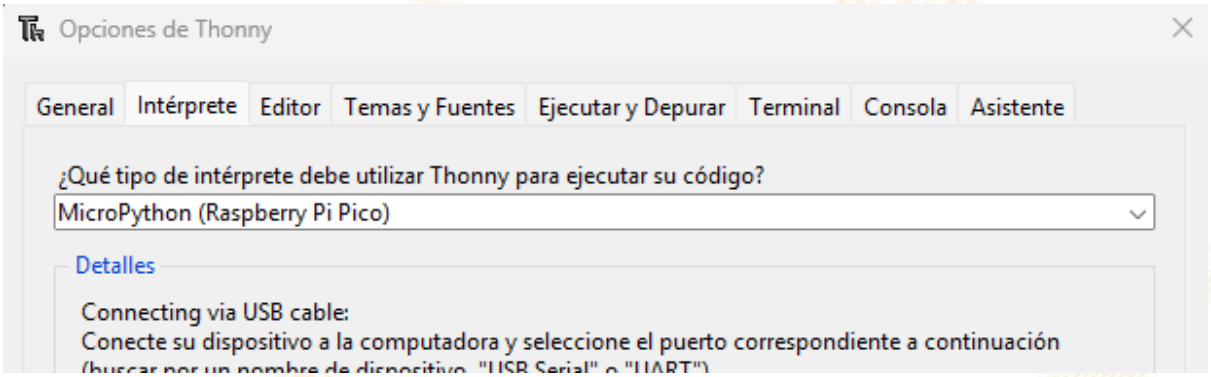


De momento aún no realices la conexión de la tarjeta a tu PC.

De nueva cuenta dentro de la interfaz de Thonny, volveremos a la barra de herramientas: "Ejecutar" -> "Configurar intérprete"

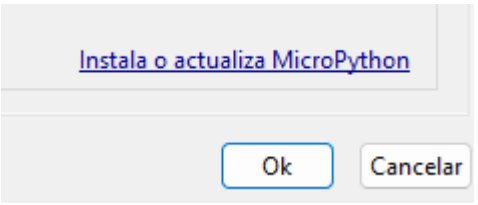


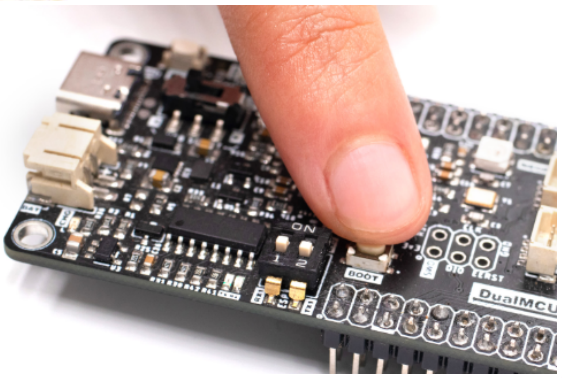
Nuevamente nos encontramos en esta interfaz, pero ahora seleccionaremos a la **Raspberry Pi Pico**



Seleccionamos el puerto en que nuestra PC haya reconocido al RP2040

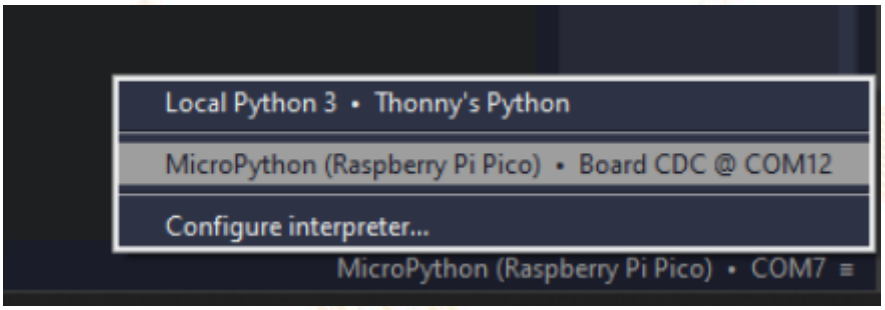
Y ahora , a la par que das click a "**Instalar o actualizar MicroPython**", presionaremos el botón de BOOT para RP2040 y sin dejar de presionar conecta el dispositivo a la PC y da click a "**Instalar o actualizar MicroPython**"





Realiza la siguiente configuración e instala

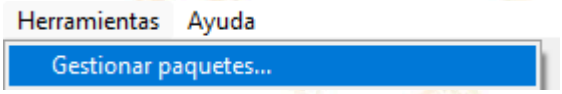
Finalmente en la ventana principal de Thonny podrás observar que el RP2040 aparece como opción para trabajar. Ten en cuenta que el puerto COM es asignado por la máquina y puede variar.



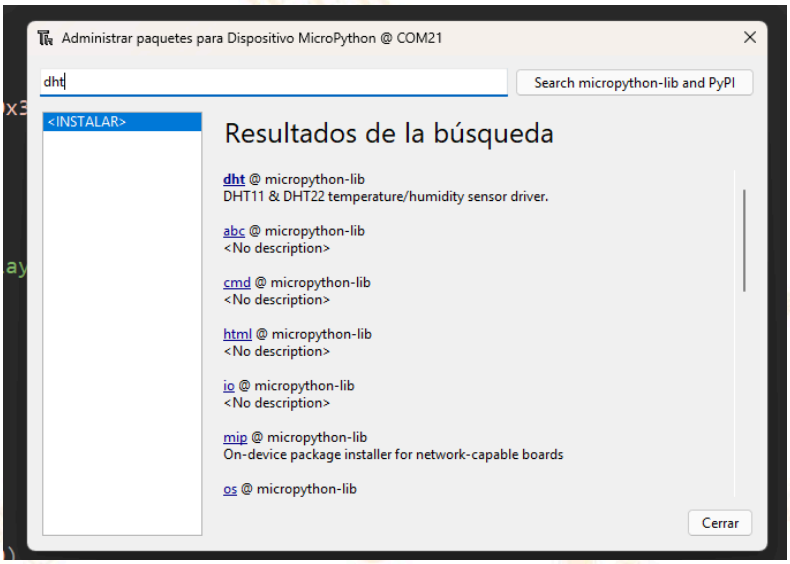
Instalación de librerías

En algunas de las prácticas tendrás que realizar instalación de algunas librerías para poder usar sensores o motores. A continuación una guia de los pasos a seguir:

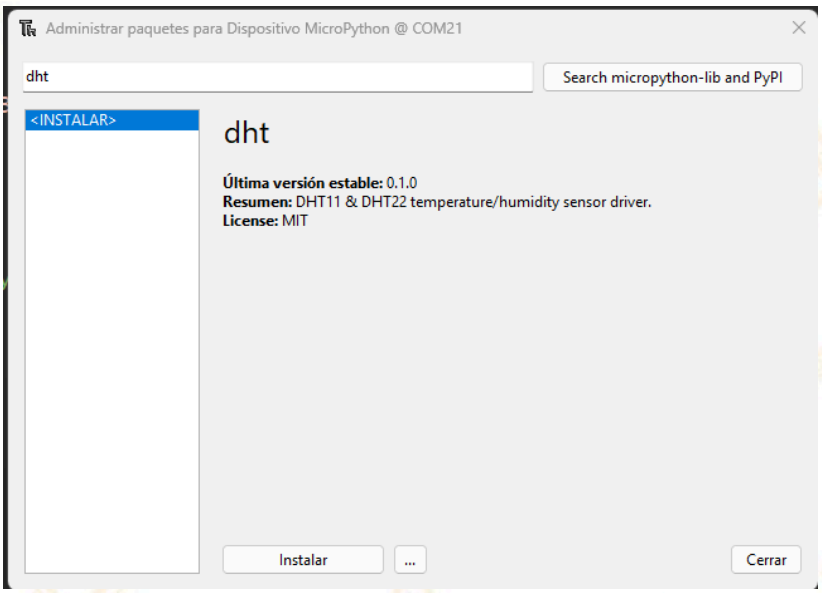
1. En la barra de herramientas de Thonny , seleccionar Gestionar paquetes:



2.Colocar en el buscador el nombre de la librería a instalar:



3. Selecciona la librería e instala.



Lección 1 Entradas y salidas digitales

En esta lección aprenderás a encender un led al presionar un botón utilizaremos ambos MCU: **RP2040 y ESP32**; con la finalidad de que puedas familiarizarte con el uso de la tarjeta de desarrollo.

Materiales

1.- Dual MCU x1
2.- LED 5mm x2
3.- Resistencia 220 Ω ¼ w x2
4.- Push button x1
5.- Cables dupont Macho - Hembra
6.- Protoboard x1

Conocimientos previos

En un microcontrolador, las entradas y salidas digitales son los pines o terminales que permiten la interacción del microcontrolador con el entorno, mediante señales digitales (es decir, señales que pueden tener solo dos estados: ALTO (1) o BAJO (0)).

Entradas digitales

Una entrada digital es un pin que se configura para ***recibir*** señales externas. El microcontrolador lee el valor de este pin para determinar si la señal externa está en un estado de ALTO (1) o BAJO (0).

Por ejemplo:

- Si tienes un botón conectado a un pin, cuando presionas el botón, el pin puede cambiar a ALTO (1), y el microcontrolador puede detectar esa acción como un evento.
- Si no se presiona el botón, el pin puede permanecer en BAJO (0).

Salidas digitales

Una salida digital es un pin que se configura para *enviar* señales a dispositivos externos. El microcontrolador puede cambiar el estado de este pin a ALTO (1) o BAJO (0).

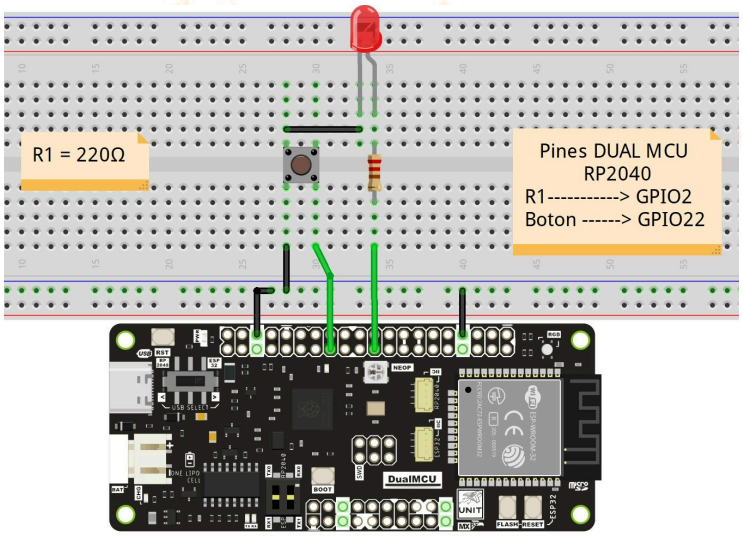
Por ejemplo:

- Si tienes un LED conectado a un pin, el microcontrolador puede enviar un ALTO (1) para encender el LED o un BAJO (0) para apagarlo.
- Si tienes un relé o un motor conectado, el microcontrolador puede activar o desactivar el dispositivo controlando el estado de la salida.

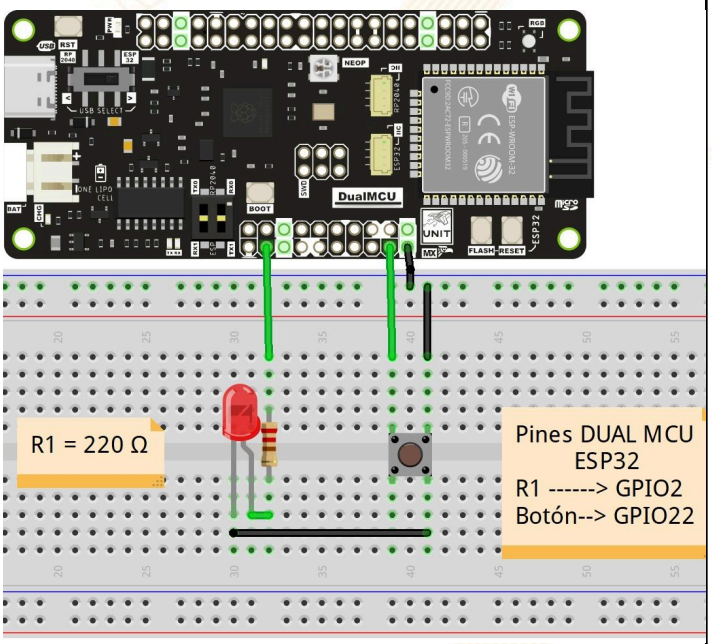
Diagrama de conexiones

A continuación se muestran dos diagramas de conexión para hacer uso del MCU RP2040 y un segundo diagrama para la conexión a los pines del ESP32.

RP2040



ESP32



Se presentan ambas conexiones por separado, pero si lo prefieres puedes hacer ambas conexiones al mismo tiempo.☺

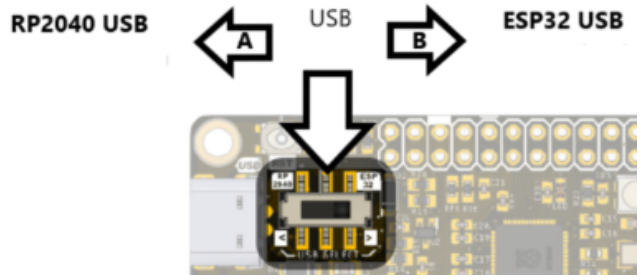
Observa que en ambas conexiones se está ocupando el "mismo" pin para el botón(GPIO 22) y el led (GPIO2). Esta similitud nos ayudará a que un mismo código pueda ser usado para ambas MCU.

Código de funcionamiento

Este programa funciona en ambos microcontroladores; solo recuerda que tendrás que realizar el cambio de switch y dentro del firmware para cada una de las cargas de código al MCU correspondiente.

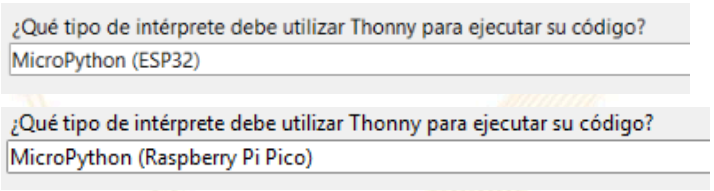
Hardware

Realizando el cambio de switch

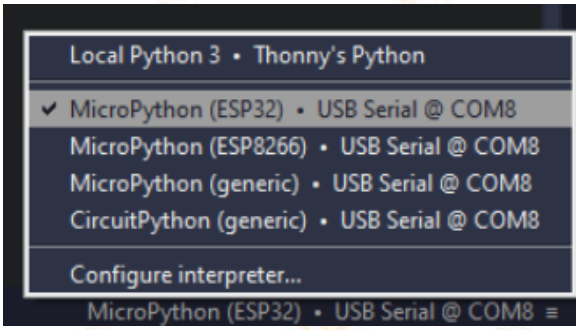


Firmware

1. Barra de herramientas >Opciones de Thonny
2.Selecciona el MCU



3. Observa que se encuentre el MCU seleccionado



Una vez realizada esta selección, indiferentemente con cual MCU quieras empezar a programa (ESP32 o RP2040), ejecuta el siguiente scrip:

Python

```python
#Programa para encender un LED al presionar un botón con UNIT DUAL MCU,este código se puede cargar tanto en la ESP32 como en el RP2040

from machine import Pin
import time

# configura el pin 2 como salida
led = Pin(2, Pin.OUT)

# configura el pin 22 como entrada con una resistencia interna de pull up
boton = Pin(22, Pin.IN, Pin.PULL_UP)

#Comienza el ciclo infinito

while True: #lee el valor del botón
   valor = boton.value()  #imprime el valor del botón
#si botón es 0,se ha presionado el botón, se enciende el led

   if valor == 0:
       led.on()
       print("BOTON PRESIONADO")
   #si no se presiona el botón, el led permanece apagado

   else:
       led.off()
       print("BOTON SIN PRESIONAR")

   #espera 100 mili segundos

   time.sleep(0.1)
```

Este scrip funciona para ambos MCU y la intención es que puedas cargar este programa en ambos. Esto con la intención de familiarizarnos con el uso del switch selector de MCU y con la interfaz de Thonny al momento de seleccionar entre uno u otro.

Lección 2 Entradas analógicas

En esta lección aprenderás a utilizar los convertidores analógicos a digital incorporados en ambos microcontroladores.

Primero tomaremos la lectura arrojada por un potenciómetro por medio del MCU **ESP32**. Por otro lado el sensor LM35 detectara temperatura y los datos se estarán registrando en el **RP2040** mediante sencillas operaciones matemáticas

Materiales

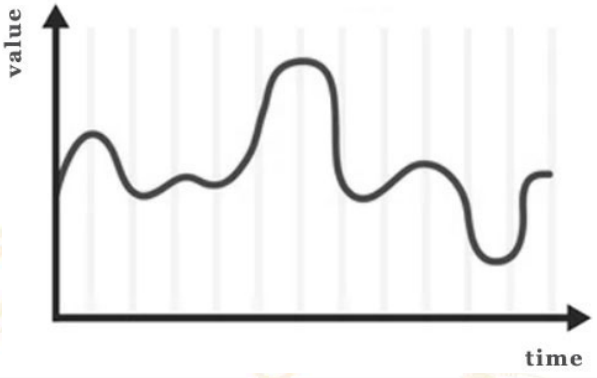
1.- Dual MCU x1
2.- LED 5 mm x2
3.- Resistencia 220 Ω ¼ w x2
4.- Potenciómetro x1
5.- Cables dupont Macho - Hembra
6.- Protoboard x1
7.- Sensor de temperatura LM35 x1

Conocimientos previos

Entradas Analógicas

Las entradas analógicas son señales eléctricas que pueden tomar un rango continuo de valores dentro de un determinado intervalo, en contraste con las señales digitales que sólo pueden tomar valores discretos (generalmente 0 o 1).

Las entradas analógicas se encuentran en dispositivos como sensores de temperatura, humedad, o señales de voltaje que varían de forma continua.



Convertidor Analógico a Digital

Un ADC (Analog-to-Digital Converter o Convertidor Analógico a Digital) es un dispositivo que convierte señales analógicas en datos digitales. El ADC toma una señal analógica de entrada y la convierte a una representación digital (generalmente en formato binario) que puede ser procesada por un microcontrolador o cualquier otro dispositivo digital.

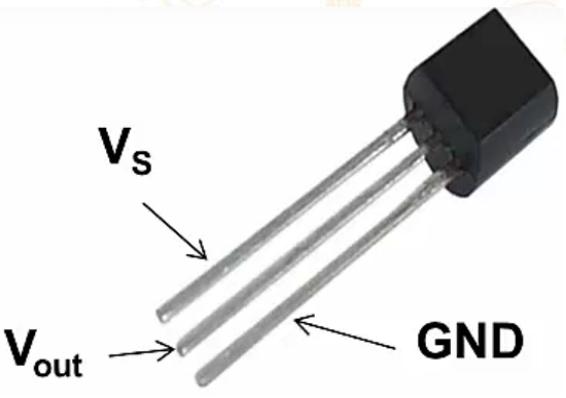
Resolución de un ADC

La resolución de un ADC se refiere a la cantidad de bits con la que puede representar una señal analógica. Cuantos más bits tenga el ADC, más precisa será la conversión de la señal analógica a digital. Por ejemplo, un ADC de 8 bits puede representar 256 valores diferentes (2^8), mientras que uno de 10 bits puede representar 1024 valores diferentes (2^10).

Ejemplo de Resolución: Supongamos que tienes un ADC de 8 bits y un rango de voltaje de entrada de 0 a 5 V. El ADC podrá dividir ese rango en 256 niveles posibles. Esto significa que la diferencia entre cada nivel será de aproximadamente 0.0195 V (5V / 256). Si se utilizara un ADC de 10 bits, la resolución sería mayor, y la diferencia entre niveles sería de 0.0049 V (5V / 1024).

LM35

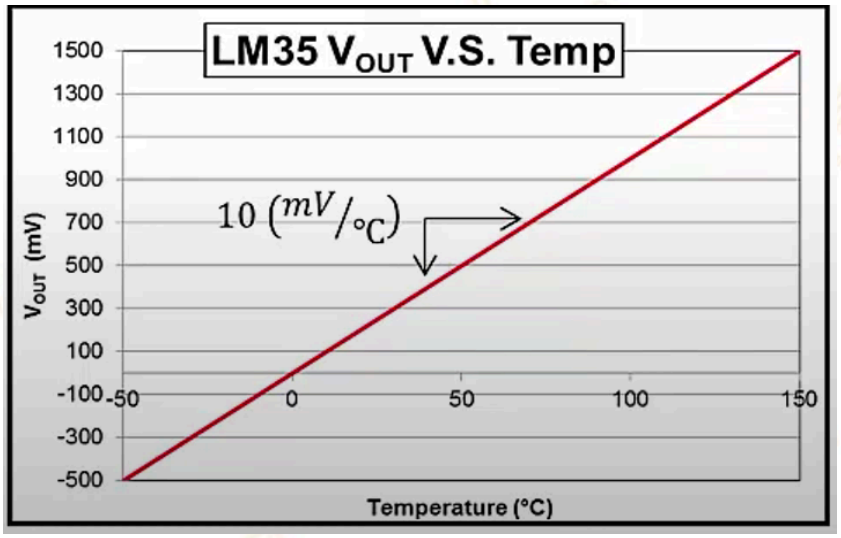
El LM35 es un sensor de temperatura analógico muy común, que convierte la temperatura en una señal de voltaje proporcional. Su salida está linealmente relacionada con la temperatura medida en grados Celsius.



La sensibilidad del LM35 es de 10 mV por grado Celsius. Esto significa que por cada grado Celsius de cambio en la temperatura, la salida de voltaje del LM35 varía en 10 mV.

Por ejemplo:

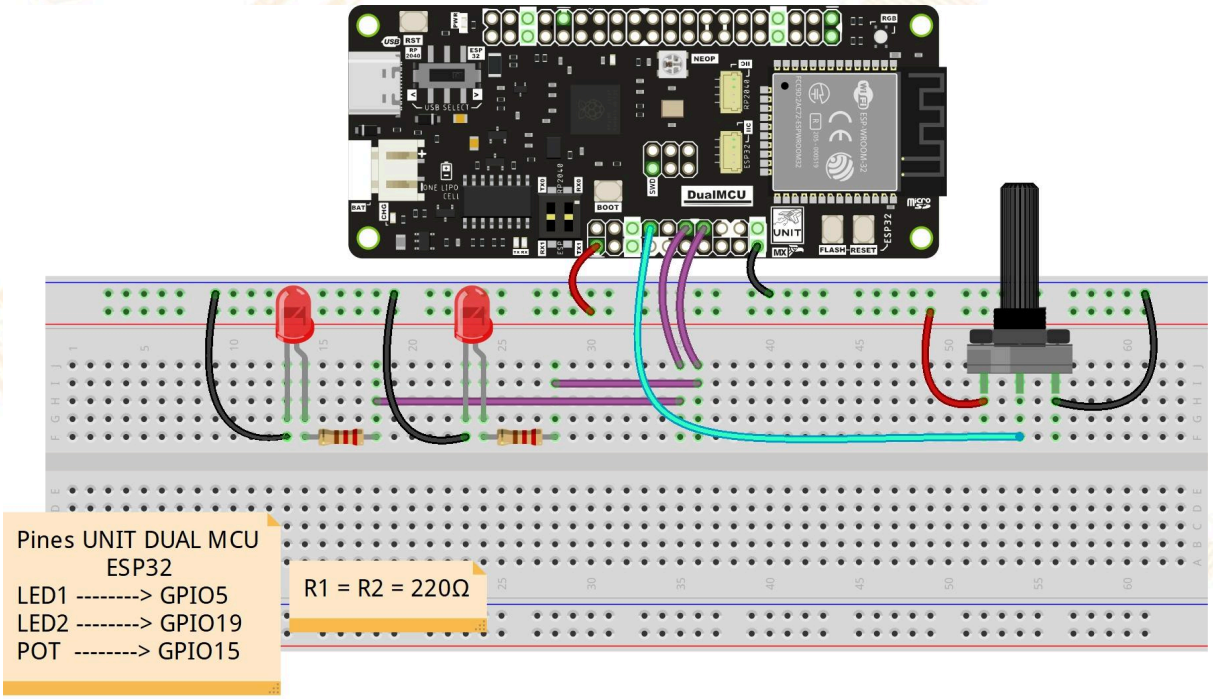
- Si la temperatura es de 25°C, la salida del LM35 será de 250 mV (25°C * 10 mV/°C).
- Si la temperatura sube a 30°C, la salida será de 300 mV (30°C * 10 mV/°C).



La precisión de la medida depende de varios factores, como la resolución del ADC que esté usando para leer la señal, y la calidad del LM35, pero en general, este tipo de sensores es muy útil para aplicaciones que requieren medir la temperatura de manera continua y precisa.

Diagrama de conexión

Para entender cómo se comportan los GPIO de ADC en cada uno de los MCU, comenzaremos con la siguiente conexión hacia la MCU ESP32, en donde interviene el potenciómetro y un par de leds.



Código de funcionamiento en ESP32

Este primer programa permite la lectura en forma de voltios de un potenciómetro conectado al pin analógico número 15 de la ESP32.

Presta especial atención dentro del bucle "while" a la forma en la que se adquieren las lecturas y mediante una sencilla operación matemática, dichos datos se procesan para ser mostrados en forma de voltajes de 0 a 3.3v

Intenta imprimir por tu cuenta el valor llamado **adc.read()** y te darás cuenta de que esos valores van de 0 a 4095, valores de acuerdo con la resolución de 12 bits del ESP32 (2^12 = 4096)

Ahora, cuando el **adc.read()** nos entregue su valor máximo de 4096 cuando el potenciómetro dé toda la vuelta, es decir, que entregue el voltaje máximo de 3.3v debemos realizar una división para que 4096, corresponda con 3.3v y encontramos ese valor de la siguiente forma:

$$\frac{4096}{3.3} \approx 1241.21$$

Si realizamos el despeje

$$\frac{4096}{1241} = 3.3$$

De manera que a cada valor que entregue la función **adc.read()** si lo dividimos entre 1241, nos devuelve un valor de 0 a 3.3v

$$\frac{\text{adc.read()}}{1241} = \text{Valor en el rango de 0 a 3.3 V}$$

Así es como puedes escalar lo valores del ADC de cualquier microcontrolador para que representen cantidades que te sean útiles, como en este ejemplo es el voltaje, pero puede darse el caso de que necesites temperatura, presión o algún otro parámetro, solo necesitas conocer la resolución de microcontrolador en cuestión y tener clara la representación de valores en tu aplicación que en esta lección fue el rango de tensión de 0 a 3.3 V

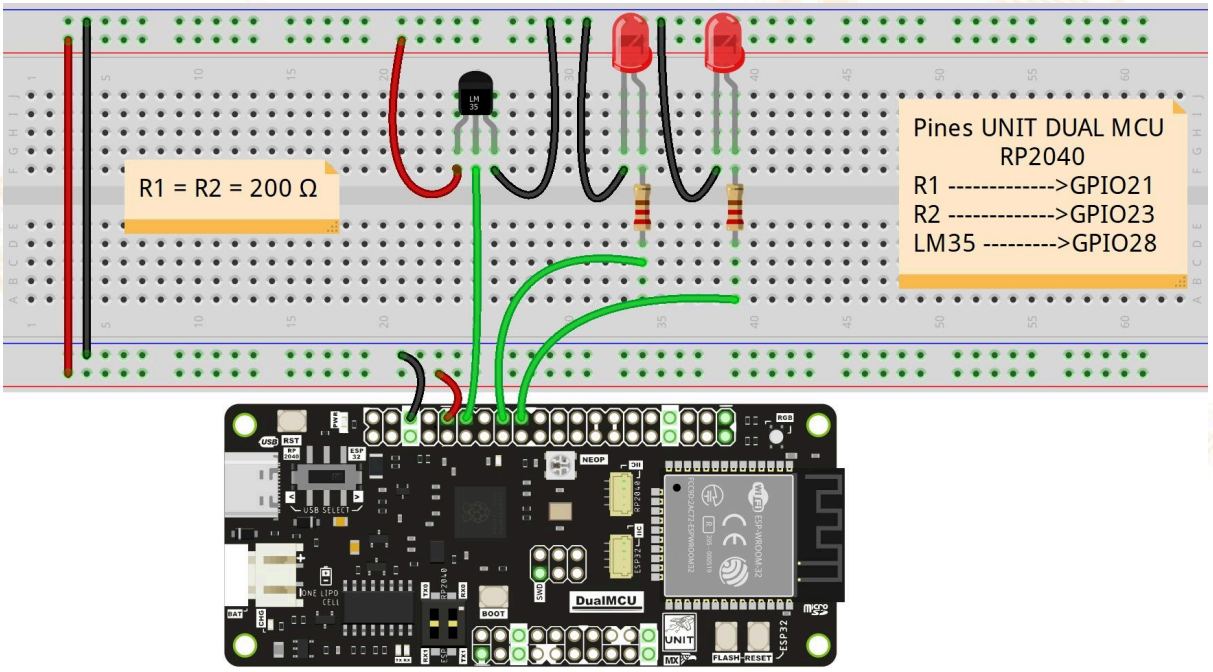
Con estos valores calculados los aplicaremos al scrip de código para poder realizar la carga a la DUAL MCU. Y recuerda, realizar la configuración anteriormente vista de hardware y firmware para poder usar el ESP32.

Python

```python
#Con este código se lee el valor de un potenciómetro se imprime en forma de volts

import machine
import time

# configura entrada de adc en el pin 15 donde se conecta el pin central del potenciómetro
adc = machine.ADC(machine.Pin(15))

# configura el rango de lectura de 0 a 3.3v
adc.atten(machine.ADC.ATTN_11DB)

# configura el rango de lectura de 0 a 4095
adc.width(machine.ADC.WIDTH_12BIT)

# configura el pin 5 como salida
led1 = machine.Pin(5, machine.Pin.OUT)

# configura el pin 19 como salida
led2 = machine.Pin(19, machine.Pin.OUT)

#Comienza el ciclo infinito, lee el valor del adc y mediante una division escala los valores en tension de 0 a 3.3v

while True:
    valor = adc.read() / 1240.91
    print(valor)    #imprime el valor en volts
    #si el valor es mayor a 1.65v enciende el led 1
    if valor > 1.65:
        led1.value(1)
        led2.value(0)
    #si el valor es menor a 1.65 v enciende el led 2
    else:
        led1.value(0)
        led2.value(1)

    time.sleep(1)    #espera 1 segundo
```

Diagrama de conexiones RP2040

En este segundo diagrama hacemos uso del RP2040 para obtener las lecturas de tensión de un sensor LM35 y dentro del programa interpretar esos valores como temperatura.



Código de funcionamiento

Ahora en este programa observamos cómo se procesan las lecturas del ADC que provienen del sensor LM35. Recuerda realizar el cambio de switch de la DUALMCU.

Para una temperatura de 25 °C el LM35 entrega una tensión de 250 mV o 0.25 V mismo valor que vas a poder visualizar si imprimes la variable **lm35_output_mv** pero necesitamos que esos 0.25 V se representan como 25 °C por lo que se multiplica la variable **lm35_output_mv** por **100 y ese valor de - 0.5** solo es una resta para ajustar el valor leído que tú mismo puedes modificar en caso de que el valor desplegado en el monitor serial no sea exacto.

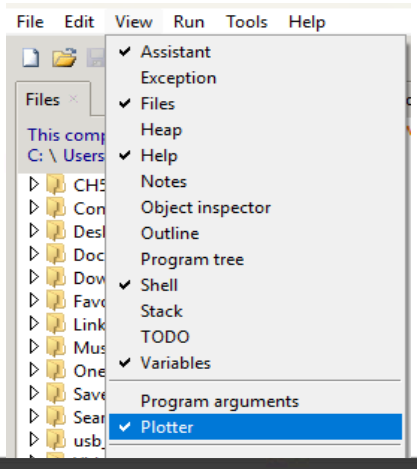
Posteriormente, tenemos una condición por medio de una sentencia **if else** donde enciende un led y se apaga otro si se mide una temperatura mayor 25 °C.

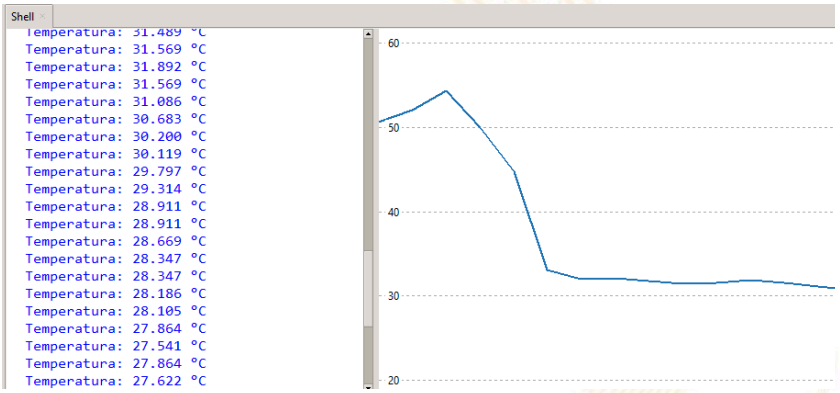
Python

```
# Configura el pin 28 de entrada analógica para leer la salida del LM35
pin_lm35 = machine.Pin(28, machine.Pin.IN)
adc = machine.ADC(pin_lm35)
led1 = machine.Pin(21, machine.Pin.OUT)
led2 = machine.Pin(23, machine.Pin.OUT)

while True:
    # Lee el valor del sensor LM35
    lm35_output_mv = adc.read_u16() / 19860

    # Convierte el valor a grados Celsius
    temperatura_celsius = ((lm35_output_mv* 100)-0.5)

    # Imprime la temperatura en grados Celsius
    print("Temperatura: {:.3f} °C".format(temperatura_celsius))
    # Controla los LEDs dependiendo de la temperatura
    if temperatura_celsius > 30:
        led1.value(1)
        led2.value(0)
    else:
        led1.value(0)
        led2.value(1)

    # Retraso de 3 segundos antes de la próxima lectura
    time.sleep(1)
```

Adicionalmente, si lo deseas, puedes abrir una herramienta para observar las lecturas de tu sensor en un gráfico que se actualiza en tiempo real.

Esto lo consigues dando click en la opción llamada **“View”** ubicada en la parte superior izquierda del Thonny y después da click en la opción llamada **“Plotter”**





Lección 3 PWM

En esta lección nos centraremos en usar el MCU **RP2040** para poder comprender el funcionamiento de la modulación por ancho de pulso y realizar el control del ángulo de giro de un servomotor SG90 con ayuda de push button para seleccionar el sentido de giro.

Materiales

1.- Dual MCU x1
2.- LED 5 mm x2
3.- Resistencia 220 Ω ¼ w x2
4.- Push button x2
5.- Cables dupont Macho - Hembra
6.- Protoboard x1
7.- Servomotor SG90 x1
8.- Módulo fuente para protoboard x1

Conocimientos previos

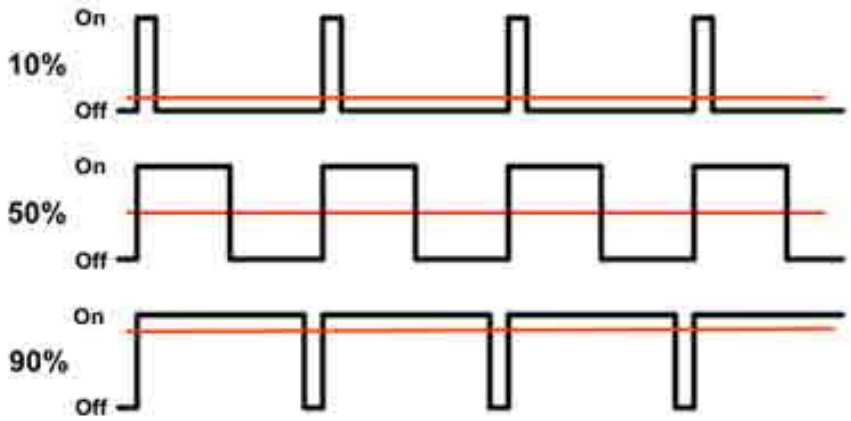
PWM

El PWM (Modulación por Ancho de Pulso, por sus siglas en inglés Pulse Width Modulation) es una técnica utilizada para controlar la potencia entregada a una carga eléctrica mediante una señal digital (encendido/apagado). Se utiliza en diversas aplicaciones, como el control de la intensidad de un LED, la velocidad de un motor o la temperatura de un calefactor.

El PWM no entrega una señal analógica continua, sino una señal digital que oscila entre dos estados: encendido (alto) y apagado (bajo). Sin embargo, la clave es que la duración del encendido y el apagado no son fijas, lo que permite controlar la cantidad de energía entregada a la carga.

La señal PWM tiene dos componentes clave:

1. **Frecuencia:** Es la cantidad de ciclos que la señal completa por segundo. Se mide en Hertz (Hz). Una frecuencia más alta significa que los pulsos se repiten más rápido.
2. **Ciclo de trabajo (Duty Cycle):** Es la proporción del tiempo durante el cual la señal está en estado alto (encendido), en relación con el ciclo total. Se mide en porcentaje.
    - Ciclo de trabajo al 0%: La señal está completamente apagada.
    - Ciclo de trabajo al 100%: La señal está completamente encendida todo el tiempo.
    - Ciclo de trabajo al 50%: La señal está encendida la mitad del tiempo y apagada la otra mitad.

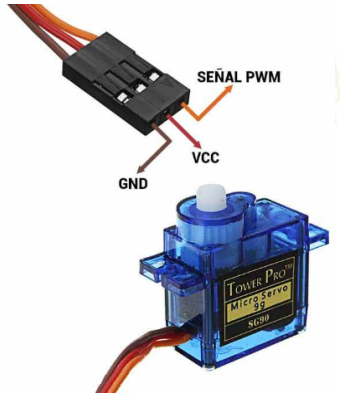


Algunas aplicaciones de la modulación por ancho de pulso son:

- Control de intensidad de luz: Se usa para regular el brillo de un LED, controlando el tiempo durante el cual está encendido.
- Control de velocidad de motores: Al variar el ciclo de trabajo del PWM, se ajusta la cantidad de energía que recibe el motor, lo que permite controlar su velocidad.
- Control de temperatura: En sistemas de calefacción, el PWM puede regular la cantidad de energía entregada al elemento calefactor.

Servomotor

Un **servomotor** es un tipo de motor eléctrico diseñado para moverse con una precisión muy alta a un ángulo específico, controlando tanto la **posición** como la **velocidad**. Estos motores tienen un mecanismo de retroalimentación (feedback), lo que les permite ajustar y corregir su movimiento de acuerdo con una señal de control. Esto los diferencia de los motores comunes, que funcionan de manera continua sin tener en cuenta la retroalimentación de su posición.



Funcionamiento del Servomotor

- **Señal de control**: El servomotor recibe una señal externa, generalmente una señal de **modulación por ancho de pulso (PWM)**. Esta señal indica el ángulo al que el servomotor debe girar. La duración de cada pulso de la señal PWM define el ángulo deseado del motor.
- **Comparación**: El controlador del servomotor compara la posición actual del motor (medida por el potenciómetro) con la posición deseada de la señal de control.
- **Corrección**: Si la posición actual no coincide con la deseada, el controlador ajusta la corriente al motor, lo que hace que el motor gire hasta que la posición deseada sea alcanzada.
- **Retroalimentación**: El potenciómetro ajusta continuamente la señal para asegurarse de que el motor permanezca en la posición deseada, proporcionando retroalimentación al controlador.

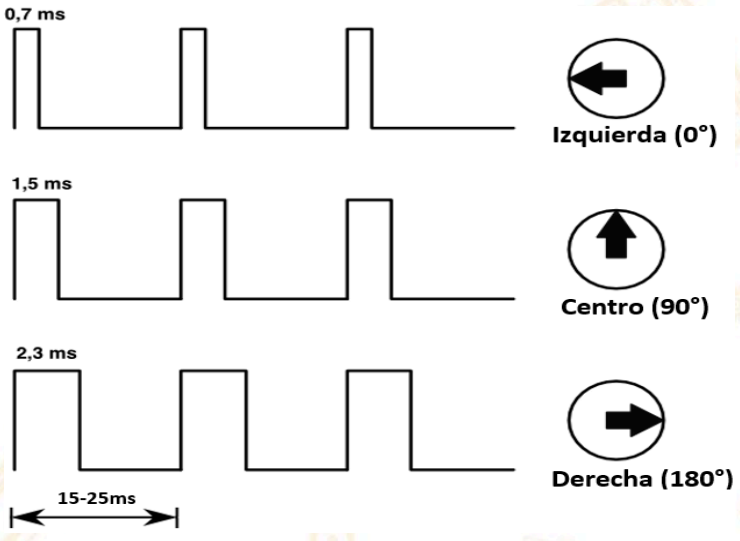
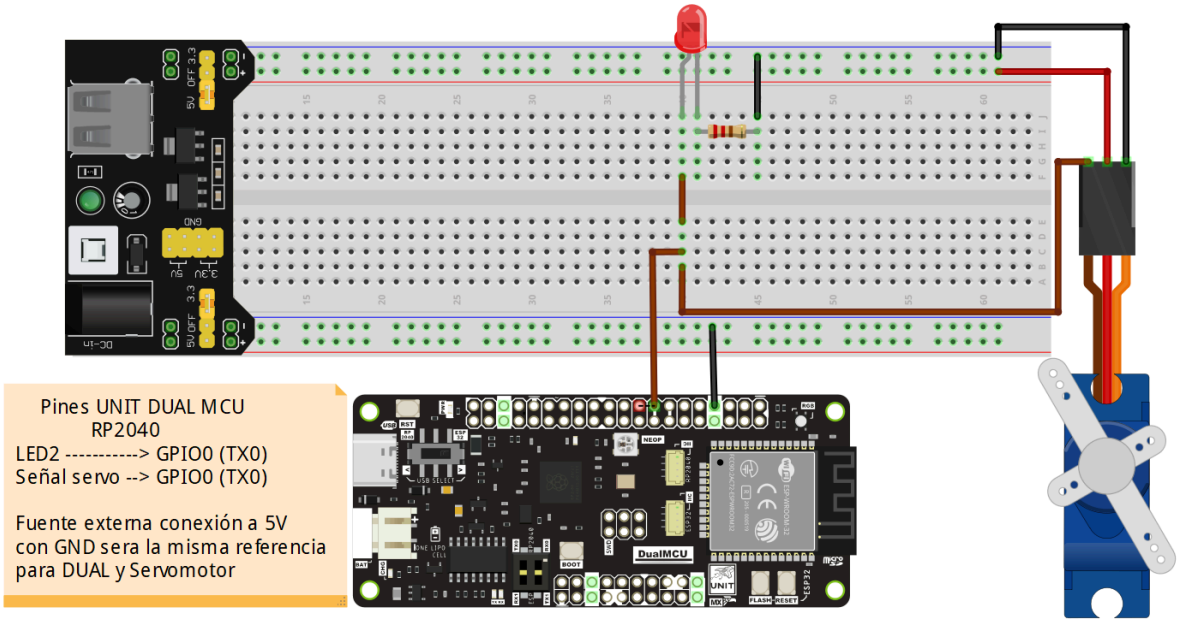


Diagrama de conexión

Primero comprenderemos el funcionamiento del PWM mediante la intensidad luminosa del Led, ya que cuando el motor realice el ciclo de trabajo ; el led tendrá variación en su iluminación.

Adicional usaremos la fuente de alimentación para protoboard(5V), debido a que el servomotor consume una corriente mayor a la tolerada por el MCU (1A). Por su parte, entre el módulo fuente y la tarjeta UNIT DUAL MCU comparten la conexión a GND para estar bajo la misma referencia de tensión.



Código de funcionamiento

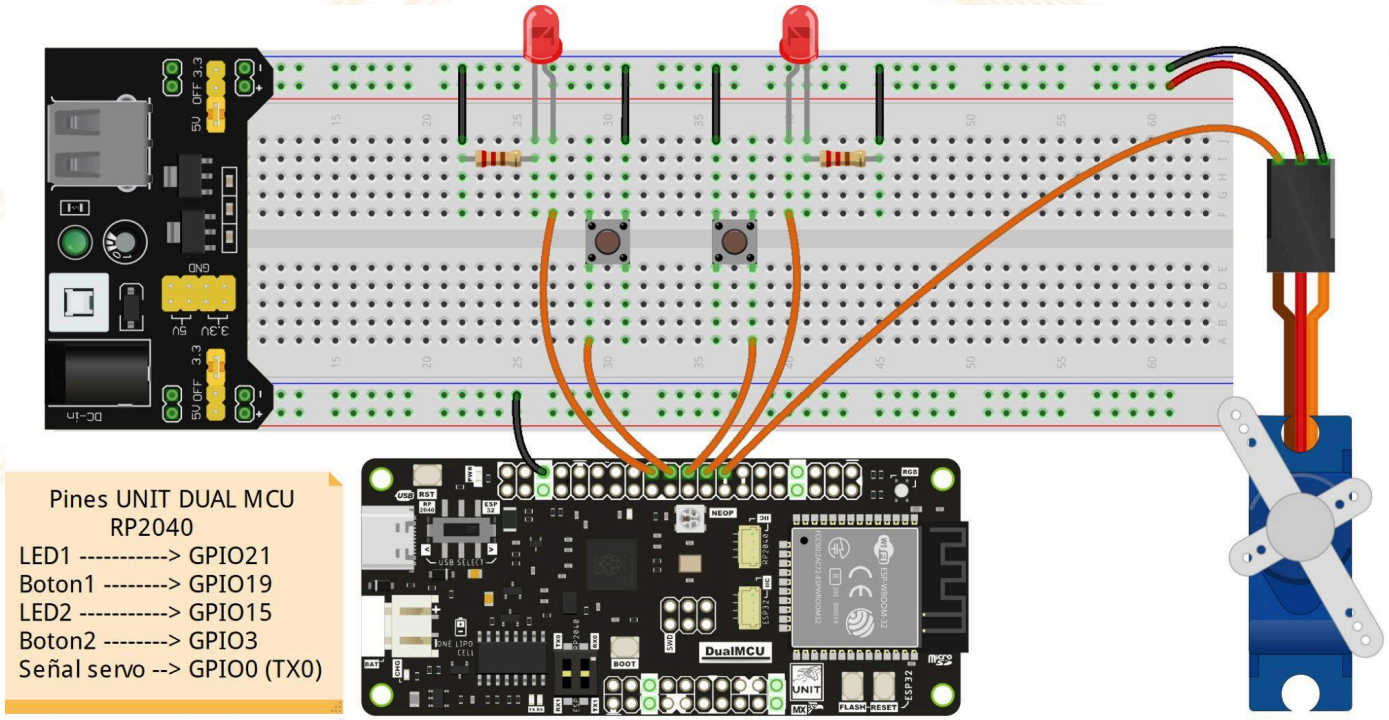
Nuevamente te recordamos configurar el switch y configurar Thonny para poder cargar el programa al **RP2040**.

Python

```python
import machine
import utime

# Configuración del pin de control del servomotor
servo_pin = machine.Pin(0)

# Crea un objeto PWM para controlar el servomotor
pwm_servo = machine.PWM(servo_pin)

# Frecuencia del PWM para el servomotor (50 Hz)
pwm_servo.freq(50)

def set_servo_angle(angle):
#Convierte el ángulo deseado(en grados)a un valor de ciclo de trabajo ten en cuenta que los valores específicos pueden variar según el servo
    duty_cycle = int(1024 + (angle / 180) * 3072)
    pwm_servo.duty_u16(duty_cycle)

try:
    while True:
        # Mueve el servomotor de 0 a 180 grados
        for angle in range(0, 181, 10):
            set_servo_angle(angle)
            utime.sleep(0.05)
        # Mueve el servomotor de 180 a 0 grados
        for angle in range(180, -1, -10):
            set_servo_angle(angle)
            utime.sleep(0.05)
except KeyboardInterrupt:
#Detén el PWM y limpia los recursos al interrumpir el programa con Ctrl+C
    pwm_servo.deinit()
    print("\nPWM detenido. Recursos liberados.")
```

Diagrama de conexión

Ahora que ya se comprende lo que realiza una señal PWM, realizaremos el ejercicio de cambio de giro en el servomotor mediante un par de push button.



Código de funcionamiento

Observar la lógica en la que se está llevando a cabo el giro del motor, ya que, si el botón 1 se presiona el programa hace girar al motor en un sentido y en el opuesto si el botón 2 es presionado , previamente desactivando el botón 1.

Python

```python
import machine
import utime

# Configuración de los botones
boton1 = machine.Pin(19, machine.Pin.IN, machine.Pin.PULL_UP)
boton2 = machine.Pin(3, machine.Pin.IN, machine.Pin.PULL_UP)
# Configuración de los LEDs (opcional)
```

Python

```
led1 = machine.Pin(21, machine.Pin.OUT)
led2 = machine.Pin(15, machine.Pin.OUT)
# Configuración del pin de control del servomotor
servo_pin = machine.Pin(0)
# Crea un objeto PWM para controlar el servomotor
pwm_servo = machine.PWM(servo_pin)
# Frecuencia del PWM para el servomotor (50 Hz)
pwm_servo.freq(50)

# Función para ajustar el ángulo del servomotor
def set_servo_angle(angle):
    duty_cycle = int(1024 + (angle / 180) * 3072)
    pwm_servo.duty_u16(duty_cycle)
try:
    valor = 90  # Ángulo inicial en 90 grados
    while True:
#Si el botón 1 está presionado, mueve el servomotor en una dirección
(aumenta el ángulo)
        if boton1.value() == 0:  # Si el botón 1 está presionado
            if valor < 390:
                valor += 10
                set_servo_angle(valor)
                print(f"Ángulo del servo: {valor} grados")
            led1.on()#Enciende el LED1 cuando botón 1 está presionado
            led2.off()  # Apaga el LED2
#Si el botón 2 está presionado, mueve el servomotor en la dirección
opuesta (disminuye el ángulo)
        elif boton2.value() == 0: #Si el botón 2 está presionado
            if valor > 0:
                valor -= 10
                set_servo_angle(valor)
                print(f"Ángulo del servo: {valor} grados")
            led2.on()#Enciende el LED2 cuando botón 2 está presionado
            led1.off()  # Apaga el LED1
        else:
            led1.off()  # Apaga LED1 cuando no se presiona el botón 1
            led2.off()  # Apaga LED2 cuando no se presiona el botón 2
       utime.sleep(0.02)
except KeyboardInterrupt:
    # Detén el PWM y limpia los recursos al interrumpir el programa
con Ctrl+C
    pwm_servo.deinit()
    print("\nPWM detenido. Recursos liberados.")
```

Hasta aquí esta lección, pero solo hemos utilizado el microcontrolador RP2040 por lo que te invitamos a cargar el código y adaptar las conexiones para el ESP32.

Lección 4 Sensor DHT11

En esta lección aprenderás a utilizar el sensor de temperatura y humedad DHT11 mediante un sencillo programa en Arduino IDE que será cargado en el **ESP32** de la UNIT DUAL MCU ONE.

Materiales

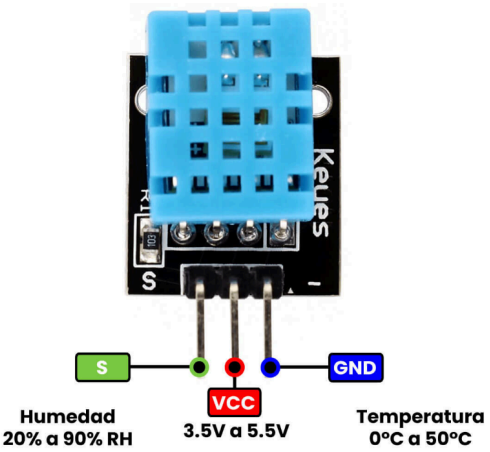
1.- DUAL MCU ONE x1
2.- Módulo KY-015 DHT11 x1
3.- Protoboard x1
4.- Cables jumper Macho-Hembra
5.- Resistencia 220 Ω a ¼ W
6.- LED 5mm x1

Conocimientos previos

Sensor DHT11

El sensor DHT11 es un sensor utilizado para medir temperatura y humedad relativa en proyectos de electrónica y sistemas de automatización del hogar. Sus principales características son:

Parámetro	DHT11
Alimentación	3Vdc ≤ Vcc ≤ 5Vdc
Señal de Salida	Digital
Rango de medida Temperatura	De 0 a 50 °C
Precisión Temperatura	±2 °C
Resolución Temperatura	0.1°C
Rango de medida Humedad	De 20% a 90% RH
Precisión Humedad	4% RH
Resolución Humedad	1%RH
Tiempo de sensado	1s
Tamaño	12 x 15.5 x 5.5mm



El DHT11 tiene un sensor capacitivo que cambia su capacitancia según el contenido de humedad en el aire. Cuanto mayor es la humedad, mayor es la capacitancia del sensor. Este valor es convertido en una señal digital, además utiliza un termistor que cambia su resistencia en función de la temperatura. A medida que la temperatura aumenta o disminuye, cambia la resistencia del termistor. Este valor también es convertido en una señal digital.

Una vez que el sensor toma las mediciones, las transmite al microcontrolador mediante una señal digital de un solo cable.

El DHT11 utiliza un protocolo de comunicación unidireccional y secuencial sobre un solo pin de datos. El protocolo se basa en pulsos de tiempo específicos que el microcontrolador debe interpretar para obtener las lecturas.

El proceso de lectura se realiza de la siguiente forma:

1. **Inicio de la Lectura**
    - El microcontrolador envía una señal de inicio al DHT11 (en forma de un pulso bajo en el pin de datos durante 18ms).
    - El DHT11 responde con una señal de inicio que dura 80ms en bajo y luego un pulso de 80ms en alto, indicando que está listo para enviar datos.
2. **Transmisión de Datos**
    - El sensor comienza a enviar los datos de forma secuencial:
        - Primero, 8 bits de la humedad relativa.
        - Luego, 8 bits de la temperatura en grados Celsius.
        - Después, 8 bits de la parte decimal de la temperatura.
        - Finalmente, 8 bits de la parte decimal de la humedad.
    - Cada bit se envía con un pulso de tiempo:
        - Un pulso alto indica un bit 1.
        - Un pulso bajo indica un bit 0.
3. **El microcontrolador debe medir el tiempo de estos pulsos para deducir los valores de temperatura y humedad.**

4. **Validación de los Datos:** Después de que el DHT11 envía los 40 bits de datos (5 bytes), el microcontrolador puede validar los valores comprobando que la suma de los 4 primeros bytes sea igual al quinto byte. Si la suma es correcta, los datos son válidos.

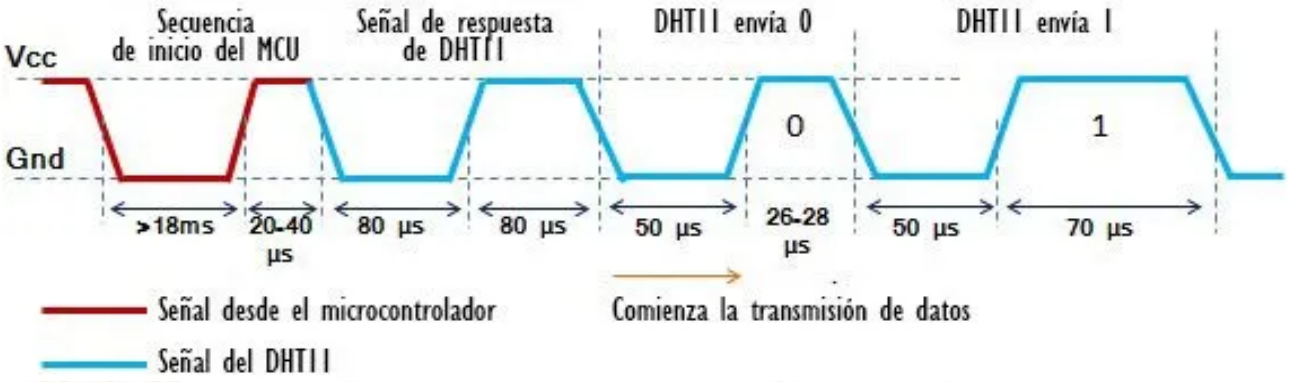
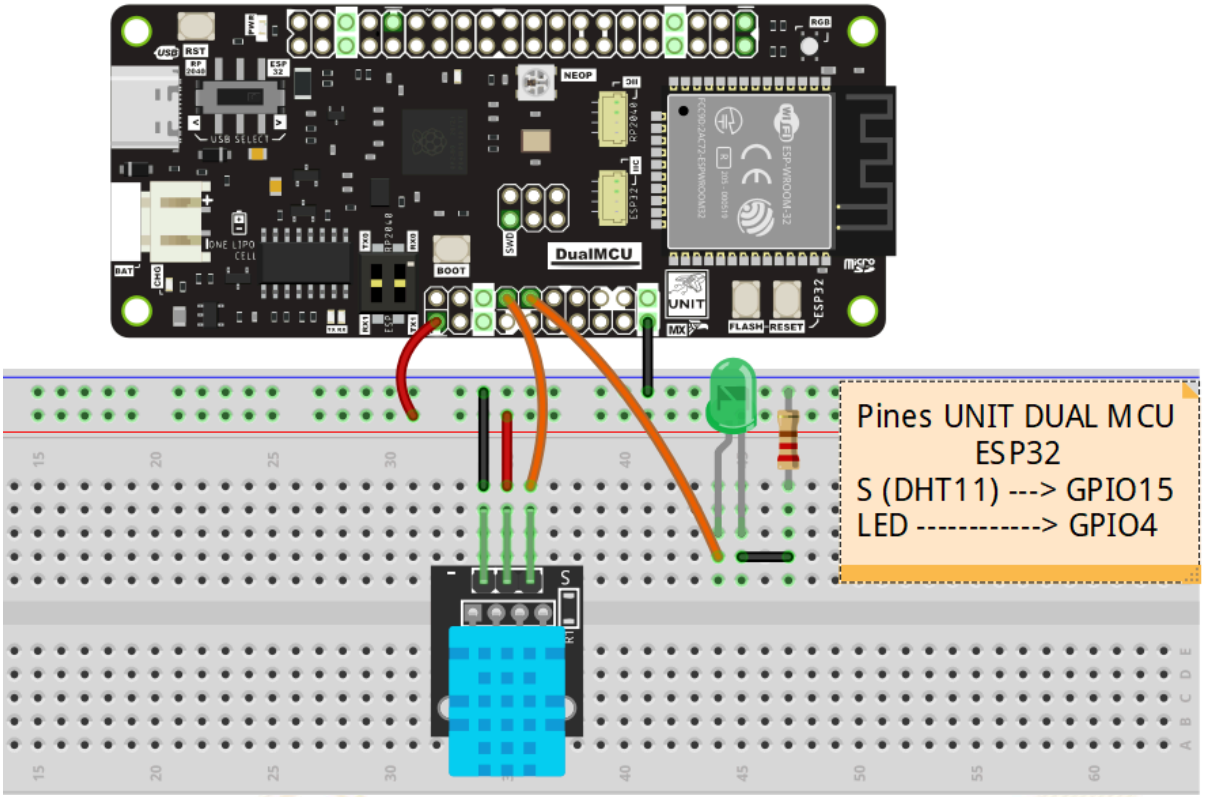


Diagrama de conexión



Código de funcionamiento

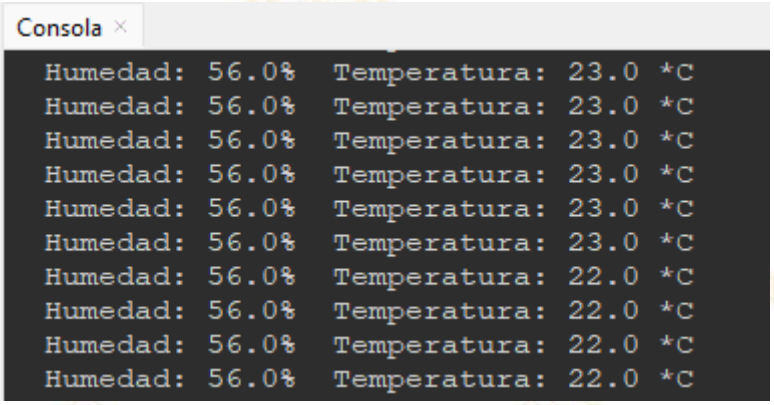
El siguiente programa usa la librería DHT.h la cual tendrás que incluir desde el Gestor de paquetes de Thonny.

Python

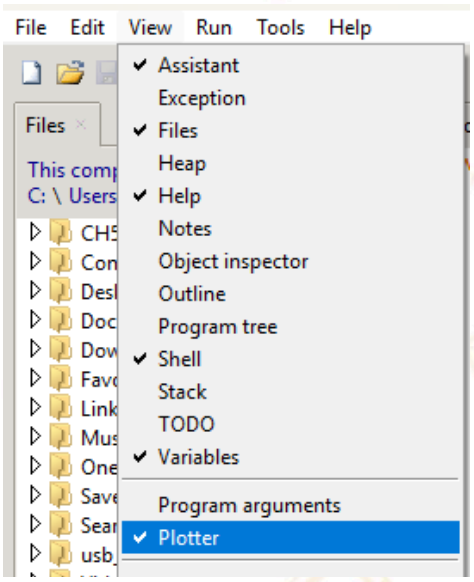
```python
from machine import Pin
import dht
import time

# Definiciones de pines
DHTPIN = 15  # Pin digital al DHT11 (D4 en la UNIT DUAL MCU ONE)
LED_PIN = 4  # Pin para el LED (D12 en la UNIT DUAL MCU ONE)
# Inicializa el sensor DHT11 y el LED
dht_sensor = dht.DHT11(Pin(DHTPIN))
led = Pin(LED_PIN, Pin.OUT)
previous_millis = 0
interval = 2000  # Intervalo de lectura en milisegundos (2 segundos)
def setup():
    print("Iniciando sistema DHT11...")
def loop():
    global previous_millis
    current_millis = time.ticks_ms()
    # Verifica si han pasado 2 segundos desde la última lectura
    if time.ticks_diff(current_millis, previous_millis) >= interval:
        previous_millis = current_millis
        try:
            # Lee temperatura y humedad
            dht_sensor.measure()
            t = dht_sensor.temperature()
            h = dht_sensor.humidity()
            # Verifica que la lectura sea válida
            if h is None or t is None:
                print("Error al leer del sensor DHT11")
                return
    print("Humedad: {:.1f}%\tTemperatura: {:.1f} *C".format(h, t))
            # Control del LED según temperatura
            if t > 28:
                led.on()
            else:
                led.off()
        except Exception as e:
            print("Error en la lectura del sensor:", e)
```

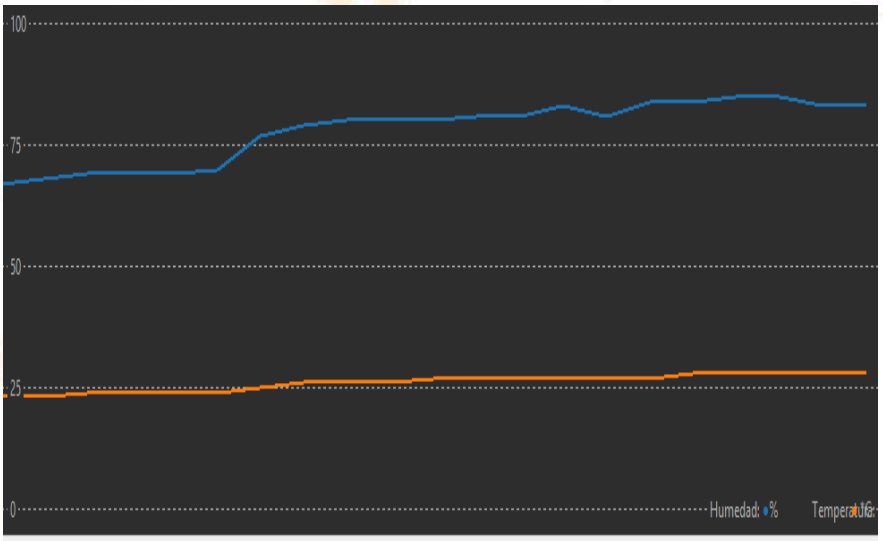
```python
# Programa principal
if __name__ == "__main__":
    setup()
    while True:
        loop()
        time.sleep_ms(100)  # Pequeña pausa
```

Tras cargar tu código en el monitor serial obtendrás una salida como la siguiente imagen donde la humedad está en porcentaje y la temperatura en grados Celsius



Recuerda que si requires visualizar los datos en gráfica, acude al menú de **"View">"Plotter"**





Lección 5 PIR y pantalla OLED

En esta lección se implementará un contador de las veces que se ha detectado la presencia de personas por medio del PIR HC-SR501 , este contador se visualizará en una pantalla OLED ssd1306

Para este fin haremos solo el microcontrolador RP2040, para ejecutar la detección de personas y la visualización en la pantalla OLED

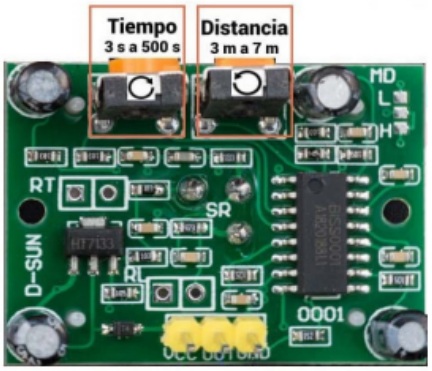
Materiales

1.- PIR HC-SR501 x1
2.- Pantalla OLED SSD1306 x1
3..- Protoboard x1
4.- DUAL MCU ONE x1
5.- Cables dupont Macho - Macho

Conocimientos previos

Sensor HC-SR501

El HC-SR501 al detectar movimiento entrega una señal de voltaje de 3.3V a la salida adecuada para los pines de la DualMCU; a pesar de que el movimiento sea constante no se activa todo el tiempo. El PIR cuenta con dos perillas de ajuste debajo del sensor, con ellas podremos ajustar el Tiempo y Distancia de trabajo



Estas perillas son importantes, ya que hay un tiempo de retardo antes de volver a detectar movimiento puede ajustarse desde 3 segundos hasta 500 segundos.

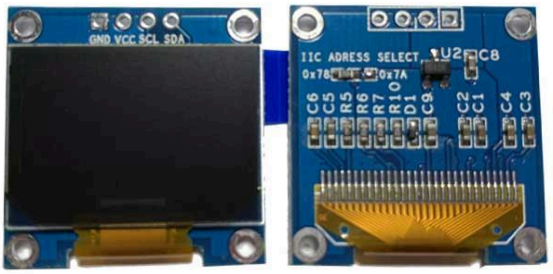
La otra perilla ajusta la distancia de detección que va desde los 3 m hasta 7 m a un ángulo de detección de 110°. Lo último que debes de saber de este módulo, es que cuenta con dos modos de configuración que puedes activar mediante los pines que se encuentran a un costado del sensor marcados como H y L.

JUMPER L : 1 Solo Disparo, cuando ocurre una detección de movimiento (el cual llamaremos 'evento'), la salida del sensor se activa durante el tiempo que se haya ajustado a través del potenciómetro correspondiente

JUMPER H: Disparos repetitivos, cada evento será detectado generando un nuevo tiempo de activación. Si transcurridos 30 s ocurre un segundo evento, entonces se sumarán 60 s al tiempo transcurrido, dando un total de 90 s continuos con la salida activa.

Y así, cada evento adicional, sumará un tiempo de 60 s. Pasados 3s el dispositivo regresa a su funcionamiento normal.

OLED



Es una tecnología de pantalla cuyo nombre significa "diodo orgánico emisor de luz". A diferencia de las pantallas tradicionales como las LCD, que necesitan una luz de fondo para iluminar la imagen, las pantallas OLED funcionan de una manera más avanzada: cada píxel se ilumina por sí solo. Esto significa que cuando una parte de la pantalla necesita mostrar color negro, simplemente ese píxel se apaga por completo, logrando un negro profundo y un contraste mucho mayor.

Gracias a eso, las pantallas OLED ofrecen imágenes más vibrantes, colores intensos y un consumo de energía más eficiente, sobre todo cuando se muestran imágenes oscuras. También permiten fabricar pantallas muy delgadas, porque no requieren una capa de retroiluminación.

Una **pantalla OLED 128x64 con controlador SSD1306** es un tipo de pantalla **pequeña** (generalmente de 0.96 pulgadas) usada comúnmente en proyectos electrónicos con microcontroladores.

Donde **128x64** es la **resolución**. Tiene 128 píxeles horizontales y 64 verticales, o sea **8.192 píxeles en total** y **SSD1306** es el **chip controlador** que se encarga de manejar la pantalla. Permite la comunicación con el microcontrolador a través de **I2C** o **SPI**, que son protocolos de comunicación digital.

Para el desarrollo de esta lección usaremos el protocolo de comunicación I2C que a continuación se detalla.

Protocolo I2C

El bus de comunicaciones I2C, conocido como Inter-Integrated Circuit, es un protocolo que permite la comunicación entre múltiples dispositivos mediante solo dos hilos nombrados como SDA (Serial Data) y SCL (Serial Clock Line) que corresponden al hilo de datos seriales y al hilo de la señal de reloj que sincroniza la comunicación.

Para poder reconocer cada uno de los dispositivos conectados a los dos hilos del bus I2C, a cada dispositivo se le asigna una **dirección**.

En este tipo de comunicaciones el **MAESTRO** es el que tiene la **iniciativa** en la transferencia y este es quien decide con quien se quiere conectar para enviar y recibir datos y también decide cuándo finalizar la comunicación.

Una transmisión típica de I²C consiste en los siguientes pasos y bits:

1. Start Bit (bit de inicio)

El maestro pone **SDA de alto a bajo mientras SCL está alto**.
Esto indica a todos los dispositivos en el bus que va a comenzar una transmisión.

2. Dirección del esclavo (7 bits o 10 bits)

El maestro envía 7 (o 10) bits que identifican al esclavo con el que quiere comunicarse.
Cada esclavo en el bus tiene una **dirección única**.

3. Bit de lectura/escritura (1 bit)

Bit que indica si el maestro desea:
0: Escribir al esclavo (transmitir datos hacia el esclavo).
1: Leer del esclavo (recibir datos del esclavo).

4. ACK/NACK (1 bit)

Después de cada byte enviado, el receptor debe responder con un bit:

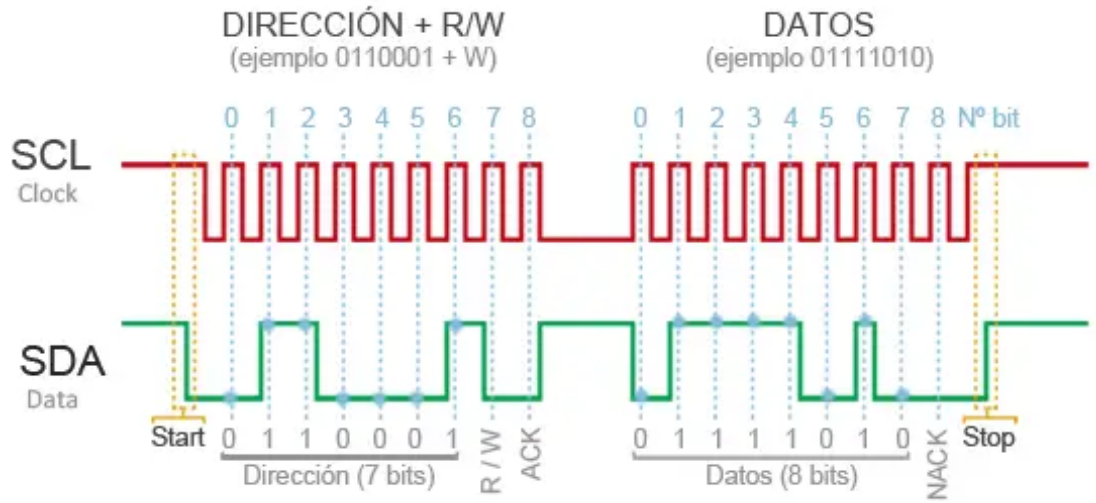
ACK (0): Acknowledgement → "He recibido correctamente el byte."
NACK (1): No Acknowledge → "No he recibido correctamente el byte o no quiero más datos."

5. Datos (8 bits por cada byte)

Los datos siempre se transmiten en **bloques de 8 bits** (1 byte).
Después de cada byte, el receptor envía otro bit ACK/NACK.

6. Stop Bit (bit de parada)

El maestro pone **SDA de bajo a alto mientras SCL está alto**.
Esto indica que la transmisión ha terminado.



Los dos hilos del BUS interfaz de comunicación I2C PIC son líneas de colector abierto, por lo que se deben utilizar unas resistencias PULL UP (resistencias conectadas a positivo) para asegurar un nivel alto cuando NO hay dispositivos conectados al BUS I2C.

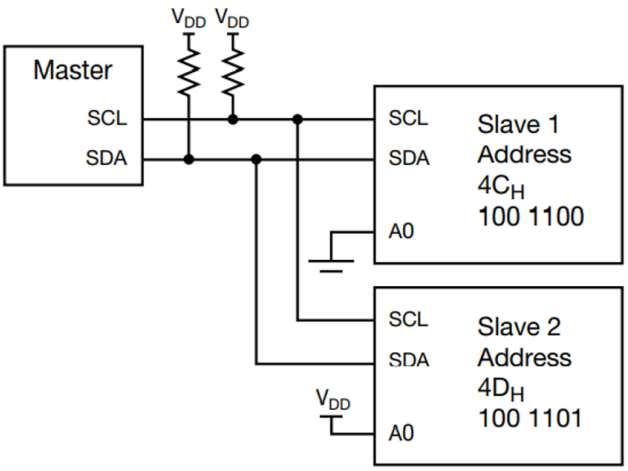
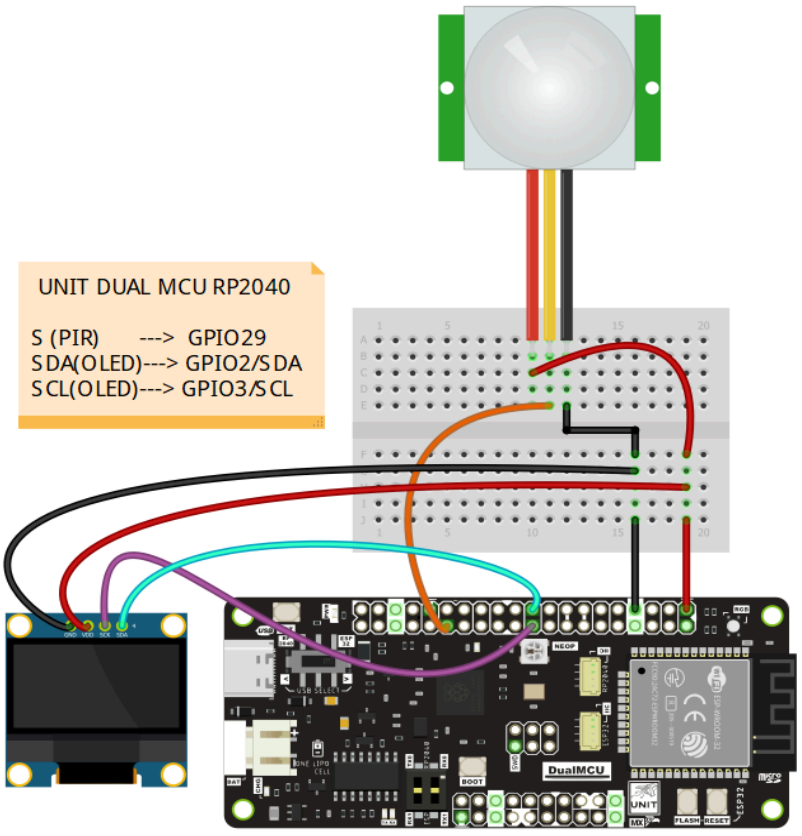
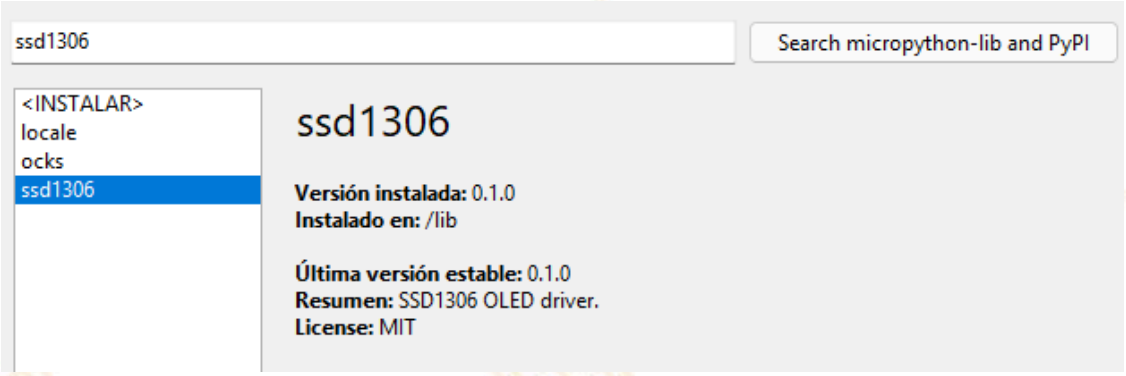


Diagrama de conexión



Configuración Previa

Para hacer uso de la pantalla OLED en el entorno de Thonny , tendremos que descargar la librería (**Gestor de Paquetes**)



Posterior a la instalación , realizaremos un testeo en la pantalla OLED SSD1306 realizando un contador del 1 al 10 y mostrando los datos de la pantalla.

```python
from machine import Pin, I2C
import ssd1306
import time
import framebuf

# Configuración I2C para RP2040 Pines SDA=2, SCL=3
i2c = I2C(1, scl=Pin(3), sda=Pin(2), freq=400000)

# Dirección I2C del display OLED (normalmente 0x3C)
OLED_ADDRESS = 0x3C

# Verificar dispositivos I2C conectados
print("Buscando dispositivos I2C...")
dispositivos = i2c.scan()
print("Dispositivos encontrados:", [hex(addr) for addr in
dispositivos])

if OLED_ADDRESS not in dispositivos:
        print(f"Error: No se encontró el OLED en la dirección
{hex(OLED_ADDRESS)}")
    print("Verifica las conexiones:")
    print("SDA -> Pin 2")
    print("SCL -> Pin 3")
    print("VCC -> 3.3V")
    print("GND -> GND")
```

Python

```
else:
    print(f"OLED encontrado en {hex(OLED_ADDRESS)}")

    # Inicializar display (128x64 píxeles)
    oled = ssd1306.SSD1306_I2C(128, 64, i2c, addr=OLED_ADDRESS)
    # Limpiar pantalla
    oled.fill(0)
    oled.show()
    # Función para mostrar texto con diferentes tamaños
    def mostrar_texto(texto, x=0, y=0, clear=True):
        if clear:
            oled.fill(0)
        oled.text(texto, x, y)
        oled.show()
    # Demostración de funcionalidades
    try:
        # 1. Texto simple
        mostrar_texto("OLED SSD1306", 0, 0)
        time.sleep(2)
        # 2. Contador
        for i in range(10, -1, -1):
            oled.fill(0)
            oled.text("Contador:", 0, 0)
            oled.text(str(i), 50, 25)
            oled.text("segundos", 30, 40)
            oled.show()
            time.sleep(1)
        # 3. Mensaje final
        mostrar_texto("Test completado!", 0, 25)
        time.sleep(2)
        # 4. Pantalla de información permanente
        while True:
            oled.fill(0)
            oled.text("OLED Conectado", 0, 0)
            oled.text(f"Addr: {hex(OLED_ADDRESS)}", 0, 16)
            oled.text("SDA:Pin2 SCL:Pin3", 0, 32)
            oled.text("RP2040 OK!", 0, 48)
            oled.show()
            time.sleep(5)
    except KeyboardInterrupt:
        oled.fill(0)
        oled.text("Apagando...", 0, 25)
        oled.show()
        time.sleep(1)
        oled.poweroff()
        print("Display apagado")
```

Código de funcionamiento

En el siguiente código encontrarás cómo se utiliza la comunicación por I2D y la detección de presencia por parte del sensor PIR , todo el funcionamiento desde la MCU RP2040. Te recordamos que deberás realizar la configuración de hardware y firmware para que puedas usar solo el RP2040.

Visualmente la OLED mostrará el número de veces que el PIR se ha accionado.

Python

```python
import ssd1306
import time
from machine import Pin, I2C

# Configuración
sensor_pir = Pin(29, Pin.IN, Pin.PULL_DOWN)
i2c = I2C(1, scl=Pin(3), sda=Pin(2), freq=400000)

# Inicializar OLED
try:
    oled = ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)
    oled_activo = True
    print("OLED inicializado correctamente")
except:
    oled = None
    oled_activo = False
    print("OLED no detectado, continuando sin display")

detecciones = 0

def actualizar_display(estado):
    """Actualiza el display OLED"""
    if oled_activo:
        oled.fill(0)
        oled.text("Sensor PIR", 20, 0)
        oled.text(f"Estado: {estado}", 0, 20)
        oled.text(f"Detec: {detecciones}", 0, 35)
        oled.text(time.strftime("%H:%M:%S"), 15, 50)
        oled.show()

def manejar_deteccion(pin):
    """Maneja la detección de movimiento"""
    global detecciones
    detecciones += 1
    print(f"Movimiento detectado! #{detecciones}")
```

Python

```
if oled_activo:
        # Mostrar alerta
        oled.fill(0)
        oled.text("! MOVIMIENTO !", 5, 15)
        oled.text(f"Contador: {detecciones}", 15, 35)
        oled.show()
        time.sleep(2)

# Configurar interrupción
sensor_pir.irq(trigger=Pin.IRQ_RISING, handler=manejar_deteccion)

# Calibración
print("Calibrando sensor PIR...")
actualizar_display("CALIBRANDO")
time.sleep(30)

print("Sistema listo")
actualizar_display("MONITOREANDO")

# Bucle principal
try:
    while True:
        actualizar_display("ACTIVO")
        time.sleep(5)
except KeyboardInterrupt:
    print("Sistema detenido")
    if oled_activo:
        oled.fill(0)
        oled.show()
```

Lección 6 Activación de un Relevador por un servidor Web

En esta lección implementaremos el control de un relevador mediante un servidor web alojado en el ESP32. Para este fin haremos solo el **MCU ESP32.**

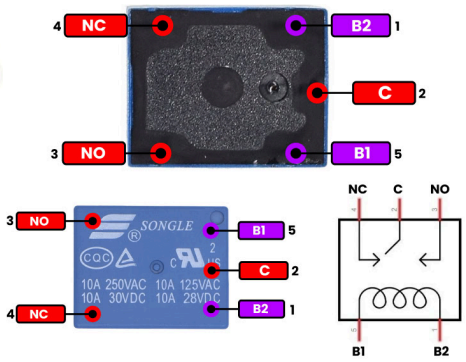
Materiales

1.- Resistencias 10k, 22k, 100 todas a 1/4w
2.- Transistor BC547 x1
3.- LED x1
4.- Relevador SPDT 5V x1
5.- Protoboard x1
6.- DUAL MCU ONE x1
7.- Cables dupont Macho - Macho
8.- Eliminador 9V 1A con Plug x1

Conocimientos previos

Relevador

Un relevador (también conocido como relé) es un dispositivo electromecánico que se utiliza para abrir o cerrar circuitos eléctricos. Funciona como un interruptor controlado por una señal eléctrica. Cuando una corriente pasa por la bobina del relevador, esta crea un campo magnético que mueve una palanca o un conjunto de contactos, lo que a su vez cambia el estado del circuito (encender o apagar). Los relevadores se usan comúnmente en aplicaciones como: Automatización de sistemas, control de dispositivos de alta potencia con señales de baja potencia, protección de circuitos o controles de maquinaria.

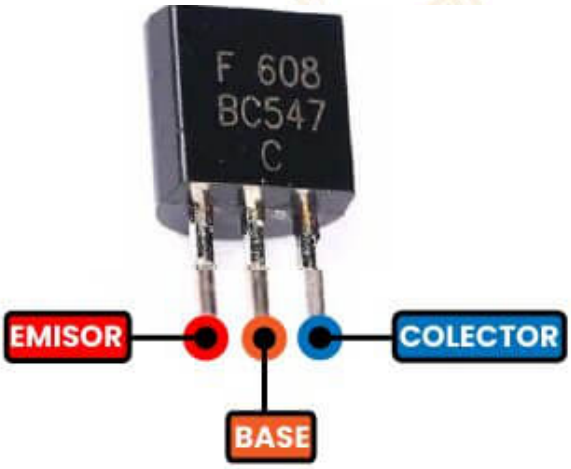


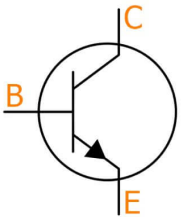
PIN	NOMBRE	DESCRIPCIÓN
1	B2	Bobina terminal 2
2	C	Contacto común
3	NO	Normalmente abierto (Normal Open)
4	NC	Normalmente cerrado (Normal Close)
5	B1	Bobina terminal 1

BJT como switch electrónico

Para lograr la activación del relevador tenemos que hacer fluir una corriente a través de la bobina del mismo

BJT BC547 nos ayudará a elevar la corriente que requiere el relevador para poder funcionar, ya que de lo contrario, si lo conectamos directamente a la tarjeta de desarrollo DUAL MCU, es probable que dañemos el GPIO o la tarjeta completa ya que no dispone de la suficiente corriente a la salida para manejar al relé.





BC547B - NPN	
1	EMISOR
2	BASE
3	COLECTOR

Servidor web

Un servidor web es un sistema que almacena, procesa y entrega contenido web a través de Internet. En términos simples, es una computadora o un software que maneja las solicitudes de los usuarios (como las que provienen de un navegador web) y les envía de vuelta el contenido solicitado, como páginas HTML, imágenes, videos, archivos y otros recursos.

¿Cómo funciona un servidor web?

1. Solicitud del cliente: Cuando un usuario escribe una URL en su navegador (por ejemplo, https://www.ejemplo.com), el navegador realiza una solicitud HTTP al servidor web correspondiente. Esta solicitud incluye detalles sobre el recurso que el navegador quiere obtener (una página web, una imagen, etc.).

2. Respuesta del servidor: El servidor web recibe esta solicitud y responde enviando los recursos solicitados. En el caso de una página web, el servidor podría enviar un archivo HTML, CSS, JavaScript, imágenes o cualquier otro archivo necesario para mostrar la página en el navegador.

3. Interacción y manejo de peticiones: Los servidores web también pueden procesar solicitudes más complejas, como formularios enviados por los usuarios o peticiones que requieren bases de datos, y generar respuestas dinámicas en función de estos datos.

Protocolo HTTP

El protocolo HTTP (HyperText Transfer Protocol) es un protocolo de comunicación que se utiliza para la transferencia de información en la web. Es el lenguaje que los navegadores web (como Google Chrome, Firefox, etc.) y los servidores web utilizan para intercambiar datos (como páginas HTML, imágenes, videos, etc.) a través de Internet.

Cuando un navegador hace una solicitud HTTP a un servidor, la solicitud incluye varias partes importantes:

1. Método HTTP: Es el tipo de acción que se solicita. Los más comunes son:

- GET: Solicita datos de un servidor (como una página web o una imagen).
- POST: Envía datos al servidor (por ejemplo, cuando envías un formulario).
- PUT: Actualiza recursos en el servidor. ○ DELETE: Elimina recursos en el servidor.
- HEAD: Solicita los encabezados de una respuesta sin los datos del cuerpo.

2. URL: Es la dirección del recurso al que se quiere acceder, como http://www.ejemplo.com/index.html.

3. Versión del protocolo: Indica qué versión de HTTP se está utilizando, por ejemplo, HTTP/1.1.

4. Encabezados: Los encabezados proporcionan información adicional sobre la solicitud, como el tipo de navegador que está haciendo la solicitud o los tipos de contenido aceptables. ESP32 como servidor web El ESP32 puede funcionar como un servidor web gracias a su capacidad para conectarse a redes Wi-Fi y procesar solicitudes HTTP.

Al actuar como servidor, el ESP32 puede servir contenido web, como páginas HTML, imágenes, o incluso procesar datos recibidos desde un navegador o cualquier cliente HTTP, como solicitudes para controlar dispositivos (sensores, LEDs, motores, etc.) a través de una interfaz web.

¿Cómo funciona un ESP32 como servidor web?

Un servidor web en el ESP32 sigue el modelo cliente-servidor: el ESP32 actúa como servidor que recibe las solicitudes HTTP de un cliente (como un navegador web o una aplicación móvil) y responde con datos (como una página HTML o información de sensores).

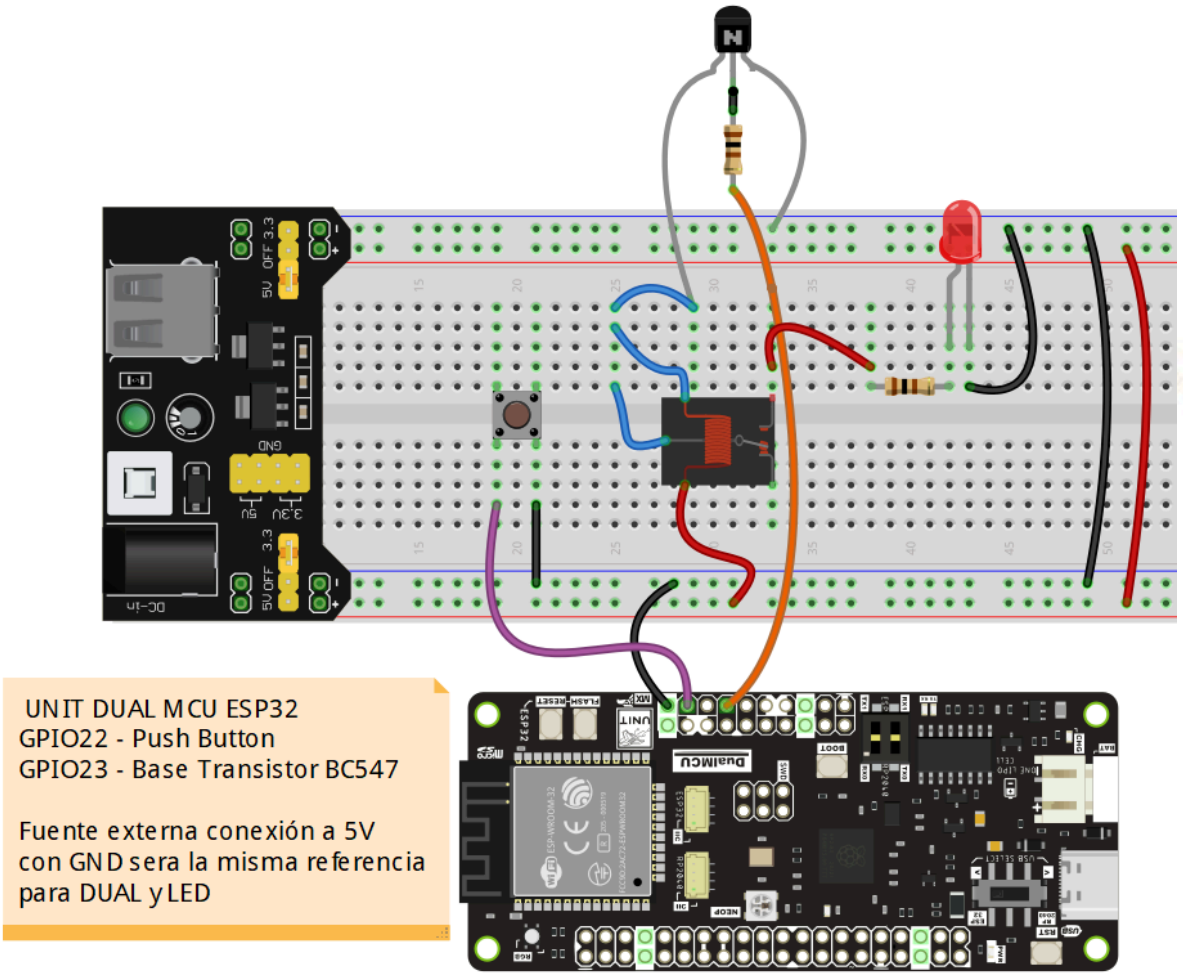
1. Conexión Wi-Fi: El ESP32 debe conectarse a una red Wi-Fi para poder funcionar como servidor web. El primer paso es establecer esta conexión, utilizando la librería WiFi.h.

- El ESP32 se conecta a la red Wi-Fi y obtiene una dirección IP local (esto se muestra en el monitor serial).

2. Creación del servidor web: El siguiente paso es crear un servidor HTTP en el ESP32. Esto lo logramos utilizando librerías como WebServer.h o ESPAsyncWebServer.h. Un servidor web básico usa el puerto 80 (que es el estándar para HTTP) para recibir las solicitudes.

Diagrama de conexión

El siguiente circuito nos ayudará a poder controlar primero de manera manual con ayuda del push button el encendido y apagado del relevador que a su vez está conectado a un led. La siguiente conexión y programación fue considerando que se conectó al pin del relevador NO, el cual es , normalmente abierto. También se puede conectar el led al pin NC (normalmente cerrado), pero considera este cambio a la lógica de la programación.



UNIT DUAL MCU ESP32
GPIO22 - Push Button
GPIO23 - Base Transistor BC547

Fuente externa conexión a 5V
con GND sera la misma referencia
para DUAL y LED

Código de funcionamiento

Con el siguiente código cargado en el ESP32, podremos controlar el relevador con el pulsar del push button.

Python

```python
from machine import Pin
import time
# Configuración de pines
RELE_PIN = 23
BUTTON_PIN = 22
# Inicializar pines
rele = Pin(RELE_PIN, Pin.OUT)
button = Pin(BUTTON_PIN, Pin.IN, Pin.PULL_UP)
# Estado inicial
rele.value(0)
rele_state = False
print("Control de rele con botón")
print("Presiona el botón para encender/apagar el rele")
while True:
    # Leer estado del botón
    if button.value() == 1:  # Botón presionado (pull-up)
        time.sleep(0.05)  # Anti-rebote
        if button.value() == 1:  # Confirmar
            # Cambiar estado del rele
            rele_state = not rele_state
            rele.value(0 if rele_state else 1)

            estado = "ENCENDIDO" if rele_state else "APAGADO"
            print(f"rele {estado}")

            # Esperar a que se suelte el botón
            while button.value() == 1:
                time.sleep(0.01)

    time.sleep(0.1)
```

Código en ESP32 para activacion via WiFi

Python

```python
from machine import Pin
import network
import socket
import time
# Configuración WiFi
# Configuración WiFi
SSID = "COLOCA TU SSID"
PASSWORD = "COLOCA TU PASSWORD"
# Configuración del relé
RELAY_PIN = 23
relay = Pin(RELAY_PIN, Pin.OUT)
relay.value(0)  # Iniciar apagado
# Conectar a WiFi
def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print('Conectando a WiFi...')
        wlan.connect(SSID, PASSWORD)
        # Esperar conexión
        for i in range(20):
            if wlan.isconnected():
                break
           time.sleep(1)
     if wlan.isconnected():
       print('WiFi conectado!')
        print('IP:', wlan.ifconfig()[0])
        return wlan.ifconfig()[0]
    else:
        print('Error conectando a WiFi')
        return None
# Página web HTML
def web_page():
    relay_state = "APAGADO" if relay.value() else "ENCENDIDO"
    relay_color = "#FF6600" if relay.value() else "#3B83BD"
    html = """<html>
    <head>
```

Python

```
<title>Control via WiFi</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <style>
            body {font-family: Arial; text-align: center; margin: 0 auto;
padding: 20px;}
            .button {padding: 15px 30px; font-size: 20px; margin: 10px;
border: none; border-radius: 5px; cursor: pointer;}
            .on {background-color: #3B83BD; color: white;}
            .off {background-color: #FF6600; color: white;}
            .status {font-size: 24px; margin: 20px; padding: 10px;
border-radius: 5px;}
        </style>
    </head>
    <body>
        <h1>Control via WiFi</h1>
        <div class="status" style="background-color: """ + relay_color +
""";">
            Estado: """ + relay_state + """
        </div>
        <p>
            <a href="/on"><button class="button on">ENCENDER</button></a>
            <a href="/off"><button class="button off">APAGAR</button></a>
        </p>
    </body>
    </html>"""
    return html

# Servidor web principal
def start_server():
    ip = connect_wifi()
    if ip is None:
        return

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(('', 80))
    s.listen(5)

    print(f'Servidor iniciado en http://{ip}')
```

```python
while True:
        conn, addr = s.accept()
        print('Cliente conectado desde:', addr)

        request = conn.recv(1024)
        request = str(request)

        # Controlar el relé según la solicitud
        if '/on' in request:
            relay.value(0)
            print("Relé ENCENDIDO")
        elif '/off' in request:
            relay.value(1)
            print("Relé APAGADO")

        # Enviar página web
        response = web_page()
        conn.send('HTTP/1.1 200 OK\n')
        conn.send('Content-Type: text/html\n')
        conn.send('Connection: close\n\n')
        conn.sendall(response)
        conn.close()

# Iniciar servidor
start_server()
```

Vista del monitor

```
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Conectando a WiFi...
WiFi conectado!
IP: 192.168.3.173
Servidor iniciado en http://192.168.3.173
Cliente conectado desde: ('192.168.3.54', 39962)
Relé APAGADO
Cliente conectado desde: ('192.168.3.54', 39964)
Relé APAGADO
Cliente conectado desde: ('192.168.3.54', 39966)
Relé ENCENDIDO
```

Vista desde la IP generada

Lección 7 Driver L298 control de motor DC

En esta lección aprenderás a controlar un motor de corriente continúa utilizando el **RP2040** y el **ESP32** utilizando el módulo para control puente H: L298.

Materiales

1.- Dual MCU x1
2.-Fuente para protoboard x1
3.- Protoboard x1
4.- Módulo puente H L298 x1
5.- Motor DC x1
6.- Cables dupont Macho - hembra
7.- Potenciómetro x1

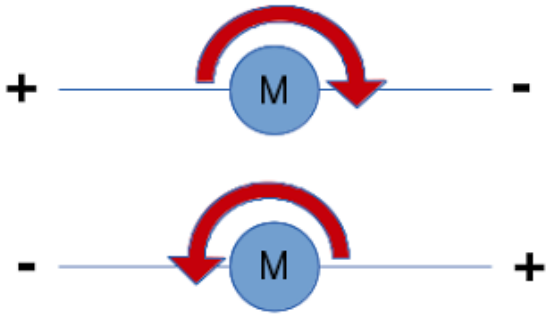
Conocimientos previos

Control del giro de un motor DC

Para cambiar el sentido de giro de un motor de corriente continua (DC), se logra invirtiendo la polaridad de la corriente que llega al motor. Hay dos formas principales de hacerlo:

1. Invertir las conexiones de los cables de alimentación:

El motor de corriente continua tiene dos terminales de entrada: uno positivo (+) y uno negativo (-). Si inviertes las conexiones, es decir, conectas el terminal positivo a lo que originalmente estaba conectado al terminal negativo, y viceversa, el motor cambiará el sentido de su giro.

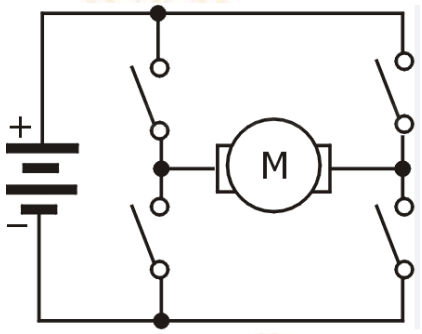


2. Uso de un puente H

Un puente H es un circuito electrónico cuya función principal es permitir que la corriente fluya en ambas direcciones a través del motor que a través de un microcontrolador o circuito lógico digital se lleva a cabo evitando intercambiar la polaridad hacia las conexiones físicas.

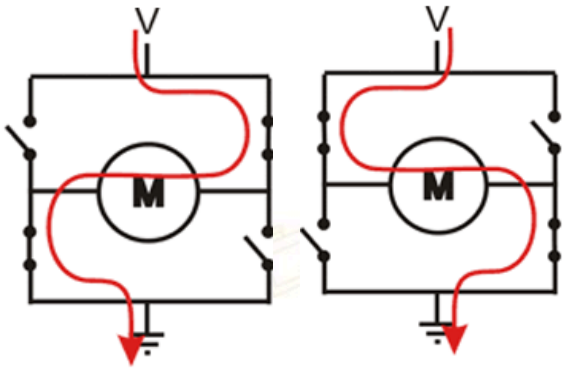
¿Cómo funciona un puente H?

Un puente H está formado por cuatro interruptores (transistores o relés) dispuestos de tal manera que se forma una figura de "H". Los interruptores están conectados de la siguiente manera:



Cuando los interruptores se abren o se cierran en diferentes combinaciones, puedes hacer que la corriente fluya de manera que el motor gire en una dirección u otra.

Observa los diferentes caminos que toma la corriente a través del motor dependiendo de los interruptores que se encuentren cerrados, dicho cambio en la dirección de la corriente y, por lo tanto, de la polaridad que recibe el motor en sus bornes se traduce en un cambio en el giro del mismo.



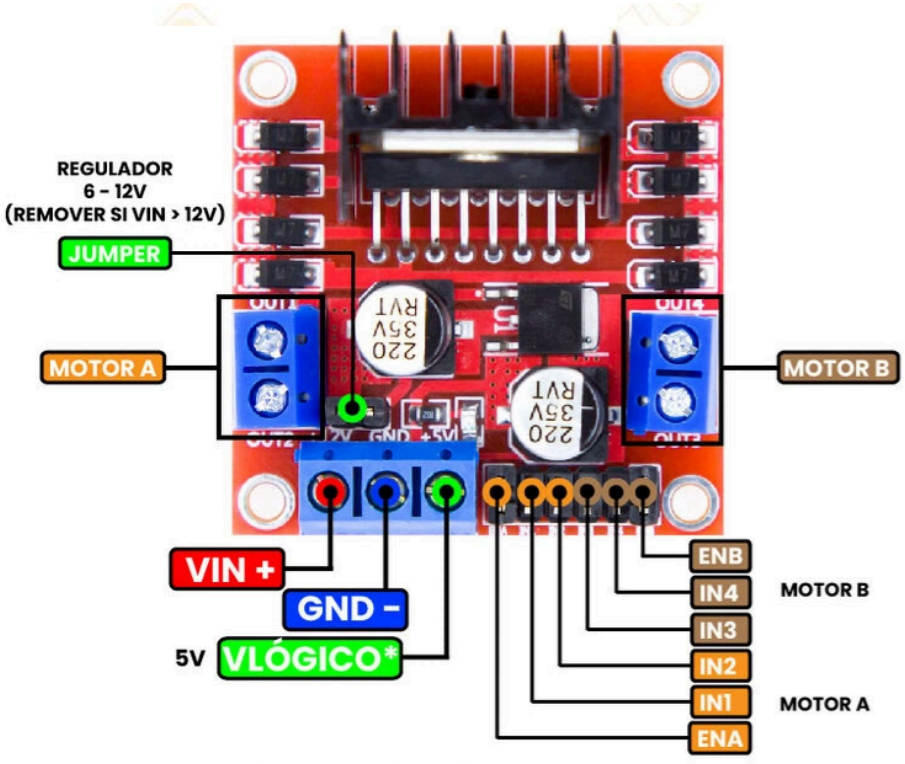
Módulo puente H L298

El módulo **L298** es un controlador que permite controlar la dirección y la velocidad de motores de corriente continua (DC), ya que internamente posee dos puentes H que permiten controlar 2 motores DC de hasta 2A mediante señales TTL que se pueden obtener de microcontroladores y tarjetas de desarrollo como en este caso la UNIT Dual MCU.

Tiene integrado un regulador de voltaje de 5 volts encargado de alimentar la parte lógica del L298N, el cual se activa a través de un jumper y se puede usar para alimentar la etapa de control.

Pines del módulo puente H L298

- **VIN**: Fuente de voltaje para los motores (por ejemplo, **12V**).
- **5V (Pin 15)**: Conectar a 5V para alimentar la parte lógica del módulo.
- **IN1, IN2, IN3, IN4**: Entradas de control de dirección para los motores.
- **ENA, ENB**: Habilitación de los motores A y B respectivamente. Se habilitan conectando a 5v y se deshabilitan a GND
- **MOTOR A y MOTOR B**: Salidas para los motores
- **GND**: Tierra común para la alimentación y señales.



*Jumper Instalado: Funciona como salida de 5V.
Jumper Desactivado: Funciona como entrada de 5V (alimenta el módulo).

Alimentación del Módulo L298

En el módulo notarás la presencia de un jumper el cual si está conectado habilita un regulador de 5v. Dicho jumper establece el siguiente comportamiento para la alimentación del L298.

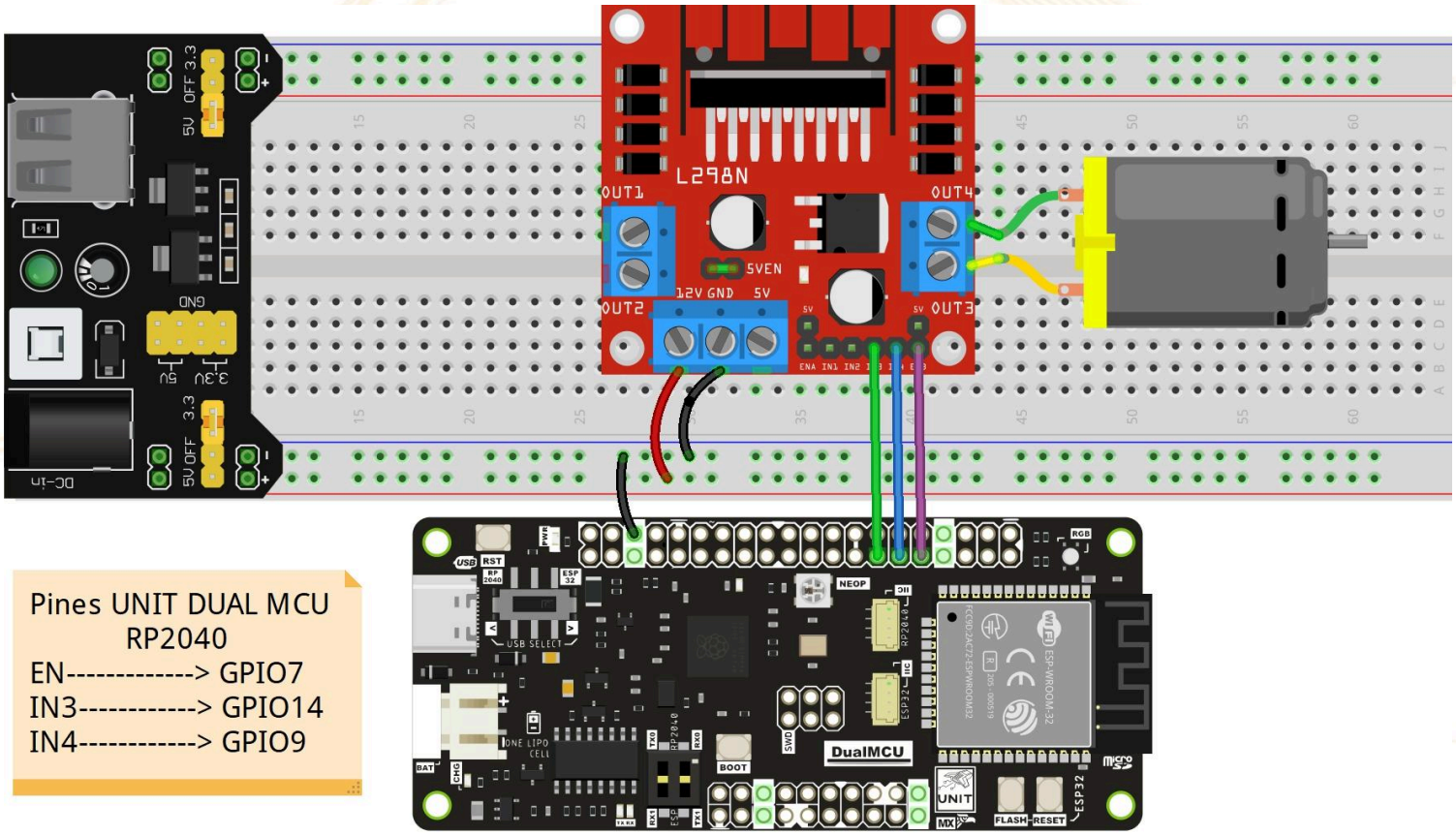
Jumper conectado: En esta configuración el módulo se alimenta usando **una** fuente de 5v a 12v en el pin **Vin** y el tercer pin de la bornera de alimentación llamado **+Vlogico** funcionará como una **salida** de 5v.

Jumper desconectado: En esta segunda configuración el módulo se alimenta con **dos** fuentes, una de 5v a 12v en el pin **Vin** que se encargará de alimentar a los motores, y otra fuente de 5v en el pin **+Vlogico** que actúa como **entrada** y energiza la lógica del módulo.

⚠️No conectes tensión de entrada al pin de +5V si tienes activado el regulador de tensión con el jumper colocado porque es posible que se dañe módulo.⚠️Diagrama de conexión 1

Diagrama de conexiones RP2040

En este primer circuito hacemos uso de los pines del **RP2040.** Por otra parte, es importante señalar el uso del módulo fuente de alimentación para energizar al motor a través del Driver, ya que si esto se hiciera desde los pines de alimentación de la tarjeta de desarrollo, esta podría quemarse.



Pines UNIT DUAL MCU RP2040
EN----------> GPIO7
IN3----------> GPIO14
IN4----------> GPIO9

Código de funcionamiento RP2040

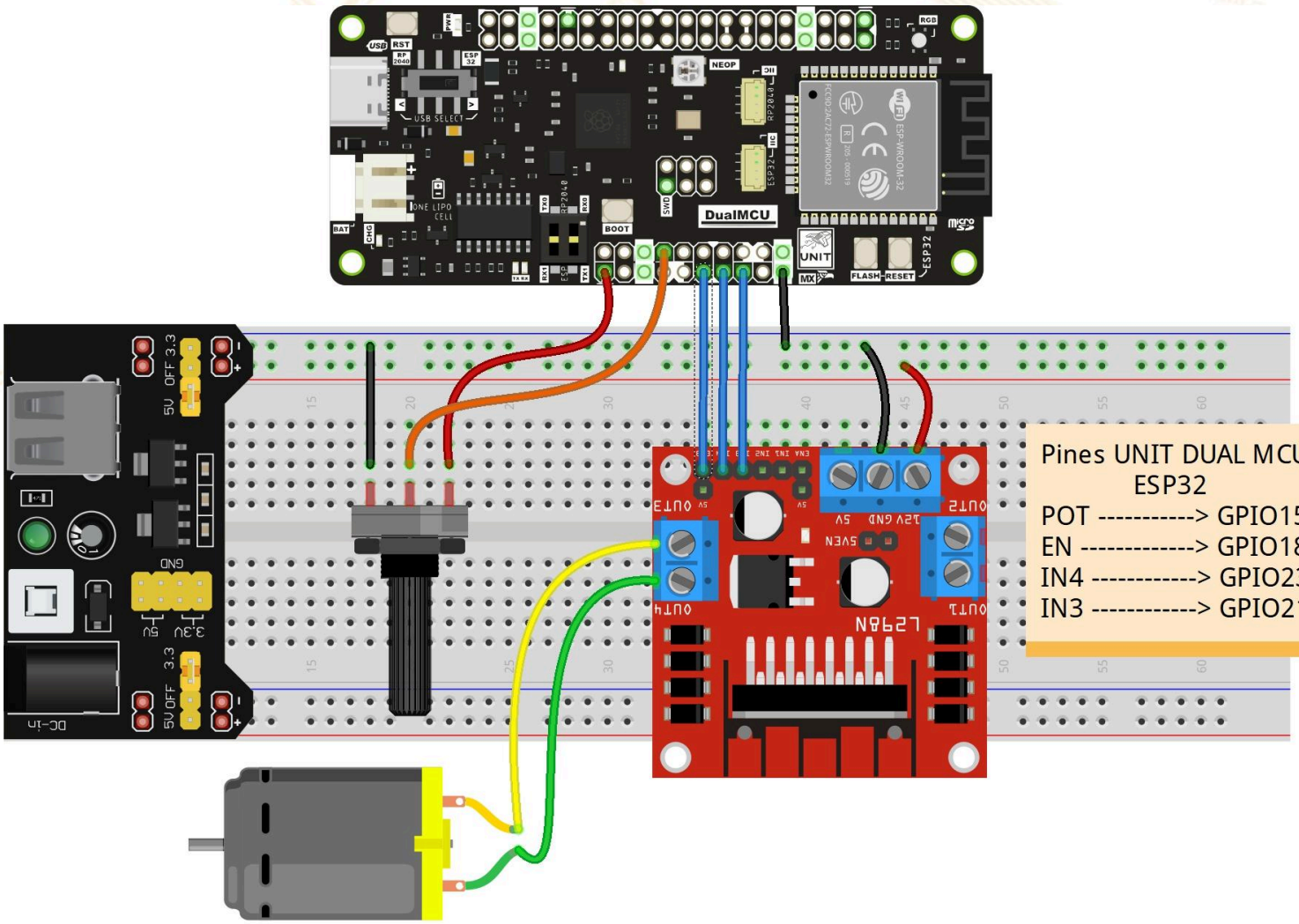
Con este primer programa lograremos controlar la dirección del giro del motor DC a través del RP2040.
Para lograr lo anterior habilitamos el motor B en el módulo L298 enviando un pulso alto en el pin EN mediante la línea **l298n_enable.on()** por medio del pin 7 y controlamos el giro en un sentido si en el módulo tenemos un pulso alto en IN3 y uno bajo en IN4 y viceversa; lo que en el código logramos mediante las sentencias **l298n_input1.on(), l298n_input1.off(), l298n_input2.on() y l298n_input2.off().**

Python

```python
from machine import Pin
import time
# Configura los pines para controlar el L298N
l298n_enable = Pin(7, Pin.OUT)  # Conecta a EN del L298N
l298n_input1 = Pin(14, Pin.OUT)  # Conecta a IN1 del L298N
l298n_input2 = Pin(9, Pin.OUT)  # Conecta a IN2 del L298N
while True:
# Habilita el motor
 l298n_enable.on()
# Control del motor
# El motor gira en un sentido
 l298n_input1.on()
 l298n_input2.off()
#Espera 1s
 time.sleep(20)
#Deshabilita el motor
 l298n_enable.off()
#Espera 1s
 time.sleep(1)
#Habilita el motor
 l298n_enable.on()
# Control del motor, el motor gira en el sentido contrario
 l298n_input1.off()
 l298n_input2.on()
#Espera 1s
 time.sleep(1)
 l298n_enable.off() # Inhabilita el motor
```

Diagrama de conexiones ESP32

En este segundo diagrama hacemos uso de los pines para ESP32 para controlar la velocidad de giro del motor DC a través de una señal PWM mediante la lectura que el ADC del microcontrolador recibe de un potenciómetro.



Código de funcionamiento ESP32

En el siguiente programa es importante resaltar la secuencia en la que se logra controlar la velocidad de giro mediante el potenciómetro. Dentro del bucle **while** primero se obtiene la lectura de la tensión del potenciómetro usando la sentencia **pot.read(),** después ese valor se mapea y se escribe en la sentencia **pw1.duty(velocidad_motor).**

De esta manera logramos que el ciclo de trabajo esté en función del valor de conversión del ADC proveniente del potenciómetro.

Python

```
from machine import Pin, PWM, ADC
from utime import sleep
# Configura los pines para controlar el L298N
l298n_enable = Pin(18, Pin.OUT)  # Conecta a EN del L298N
l298n_input4 = Pin(23, Pin.OUT)  # Conecta a IN1 del L298N
l298n_input3 = Pin(21, Pin.OUT)  # Conecta a IN2 del L298N
pot = ADC(15)  # Potenciómetro conectado a ADC en el pin 15
pot.atten(ADC.ATTN_11DB)  # Configuración de atenuación para el rango
de 0 a 3.6V
# Habilita el motor
l298n_enable.on()

while True:
    # Lee el valor del potenciómetro (valor de 0 a 4095)
    valor = pot.read()

    # Mapea el valor del potenciómetro (de 0-4095 a 0-65535) para PWM
    velocidad_motor = int(valor * 0.25)

    # Configura el PWM en IN4
    pwm1 = PWM(l298n_input4)
    pwm1.freq(10000)  # Frecuencia de 1 kHz
    pwm1.duty(velocidad_motor)  # Ajusta el ciclo de trabajo según la
lectura del potenciómetro

     # Si quieres que el motor gire en un sentido, IN2 debe estar
apagado
    l298n_input3.off()

    # Imprime el valor de ADC y el valor del PWM
    print("ADC = {}  PWM = {}".format(valor, velocidad_motor))
    sleep(0.1)
```

Lección 8 Comunicación UART entre el ESP32 y el RP2040

En esta lección aprenderás a comunicar los dos microcontroladores integrados en la UNIT DUAL MCU ONE mediante el protocolo serial UART para encender un led en el **ESP32** ante el envío y recepción de cadenas de texto tras presionar un botón conectado al **RP2040**.

Materiales

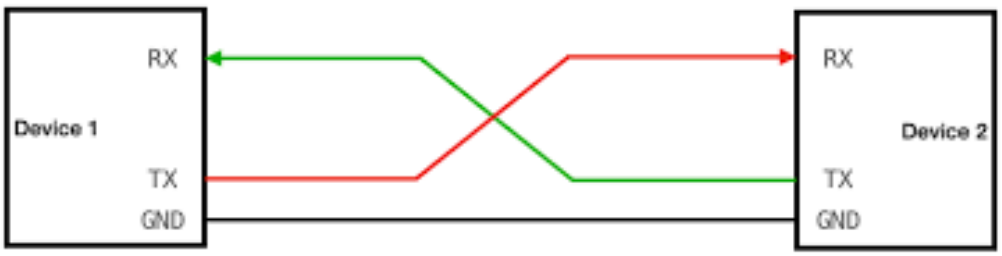
1.- DUAL MCU ONE x1
2.- Push button x1
3.- Protoboard x1
4.- Cables Dupont Macho - Macho

Conocimientos previos

UART (Universal Asynchronous Receiver / Transmitter)

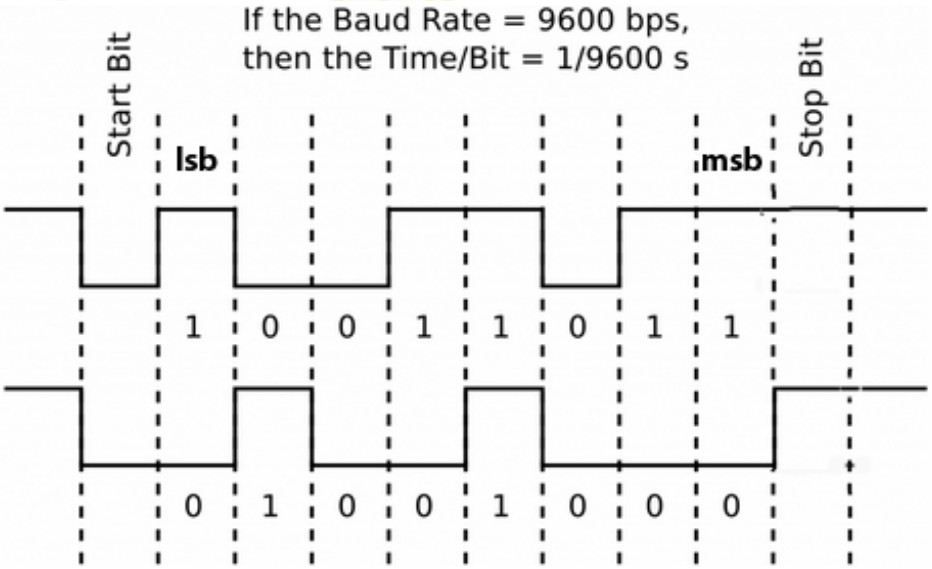
Por sus siglas del inglés, Receptor/Transmisor Asíncrono Universal, es un protocolo de comunicación en serie asíncrono, que permite la transmisión de datos entre dos dispositivos sin necesidad de una señal de reloj compartida, lo que lo hace ideal para comunicaciones simples entre microcontroladores, sensores, módulos Bluetooth, GPS, etc.

Utiliza solo dos hilos entre el transmisor y el receptor para transmitir y recibir en ambas direcciones. Ambos extremos tienen una conexión a masa.



La comunicación en UART puede ser **simplex** (los datos se envían en una sola dirección), **semidúplex** (cada extremo se comunica, pero solo uno al mismo tiempo), o **dúplex completo** (ambos extremos pueden transmitir y recibir simultáneamente).

En UART, los datos se transmiten en forma de tramas de bits donde el estado (Alto o bajo) y posición de cada uno de ellos representan diferente información dentro de la trama.



Campo	Cantidad / Valor	Función	Detalles
Bit de inicio	Siempre 0 (nivel bajo)	Señala el comienzo de una nueva trama.	La línea está en alto cuando inactiva; el transmisor la baja a nivel bajo para indicar el inicio.
Bits de datos	De 5 a 9 bits (típicamente 8)	Contienen la información real a transmitir.	Se envían del LSB (bit menos significativo) al MSB (bit más significativo).
Bit de paridad	Opcional	Verifica si el número de bits en '1' es par o impar.	- **Paridad par:** Total de bits en 1 debe ser par. - **Paridad impar:** Total debe ser impar. - Puede omitirse.
Bits de parada	1, 1.5 o 2 bits	Indican el fin de la trama.	Siempre en nivel alto (1). Permiten al receptor procesar el byte antes de recibir el siguiente.

Debido a que este protocolo es asíncrono, es decir, que no comparten una línea de pulso de reloj, el transmisor y el receptor deben tener la misma velocidad de transmisión o **baud rate.**

El ESP32 tiene 3 módulos UART disponibles (UART0, UART1 y UART2) a un baud rate máximo de 5Mbps, donde UART0 no lo podemos usar porque se encuentra reservado para el monitor serial con la computadora

Por su parte, el RP2040 tiene dos módulos UART (UART0 y UART1) a un máximo de 1.5Mbps donde al igual que en el ESP32, UART0 está reservado para la comunicación serial con la computadora.

bps (Bit rate)

Los bps significan "bits por segundo", y representa la cantidad real de bits (0 o 1) que se transmiten por segundo en una línea de comunicación.

- Es una medida de velocidad de transferencia de datos.
- Incluye datos útiles y bits de control (como bits de inicio, parada, y paridad).
- Es lo que te indica cuánta información cruda se está moviendo.

Ejemplo: Si una línea transmite 9600 bps, significa que se están enviando 9600 bits por segundo en total.

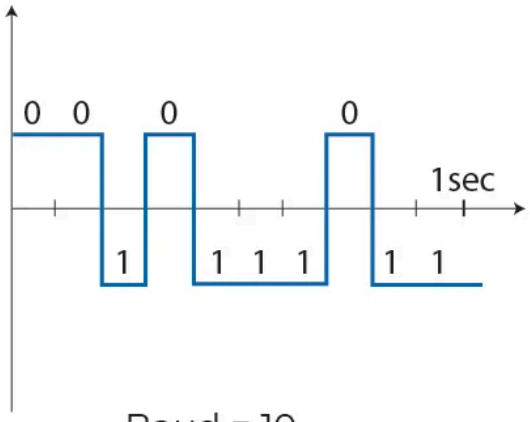
Baudios (Baud rate)

Es la cantidad de símbolos por segundo que se transmiten.

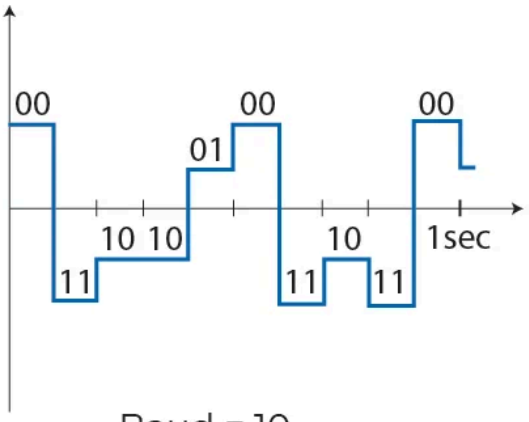
- Un símbolo es una unidad de señal transmitida (por ejemplo, un cambio de voltaje).
- En UART, cada símbolo suele equivaler a 1 bit, así que en la práctica:
    - Baudios = bps, en sistemas simples como UART.
- Pero en otros protocolos más complejos, cada símbolo puede representar más de 1 bit (por ejemplo, en modulaciones como QAM o PSK).

Ejemplo: En UART, si tienes una tasa de 9600 baudios, normalmente estás transmitiendo 9600 bps, porque 1 símbolo = 1 bit.

- Pero en una red moderna (como Wi-Fi o módems), podrías tener 2400 baudios pero 9600 bps, si cada símbolo representa 4 bits (porque 24=162^4 = 1624=16 niveles posibles)



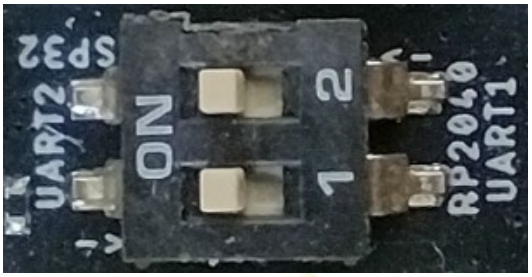
Baud = 10
Bit rate = 10 bps

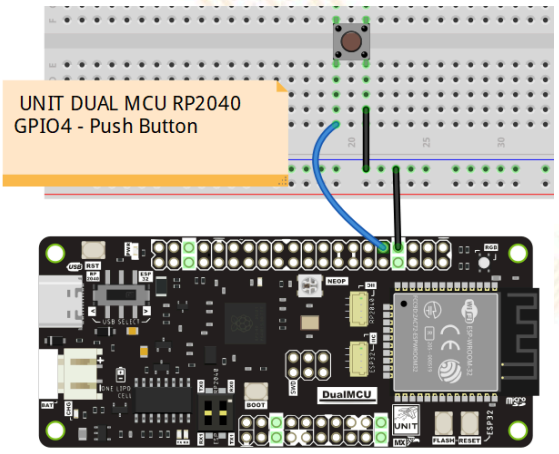


Baud = 10
Bit rate = 20 bps

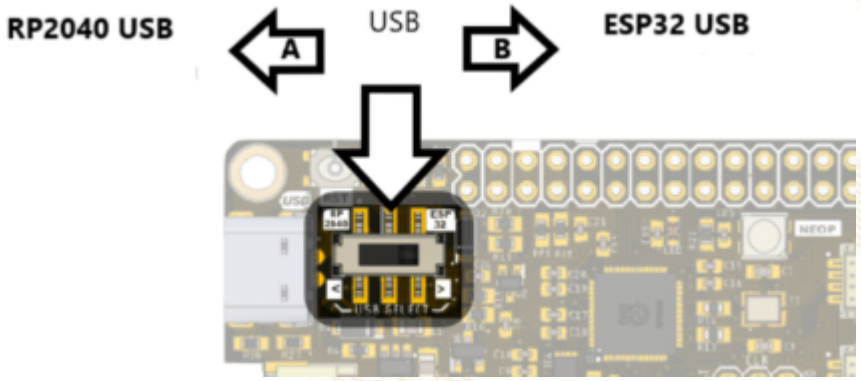
Diagrama de conexión

Para esta práctica usaremos la siguiente configuración en los pines de control UART . En donde el DIP switch tendrá que ser activado como se muestra en la figura. De esta manera estaremos conectando los pines de TX y RX de ambos microcontroladores entre sí para establecer la comunicación UART. Además de ocupar solo un botón conectado a los pines del **RP2040**.





Y realizaremos la configuración para utilizar primero el RP2040



Códigos de funcionamiento RP2040

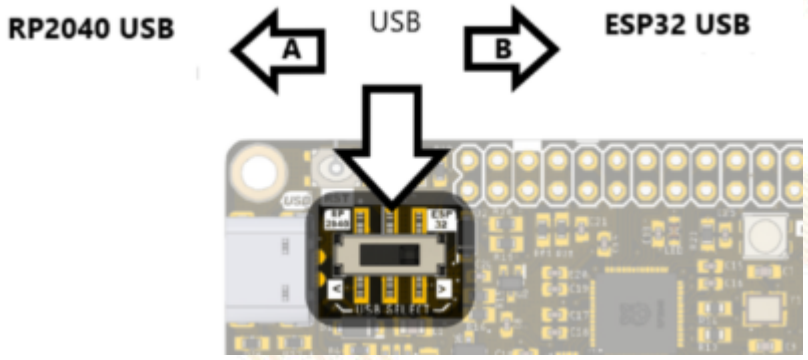
Python

```
import time
from machine import UART, Pin
import ujson
# Configuración UART
uart1 = UART(0, baudrate=115000, tx=Pin(0, Pin.OUT), rx=Pin(1,
Pin.IN))
# Configuración del botón
boton=Pin(4, Pin.IN, Pin.PULL_UP)#Botón al pin 4 con resistencia
pull-up interna
# Secuencia de LEDs
led_sequence = ["rojo", "verde", "azul"]
indice_actual = 0  # Índice para seguir la posición en la secuencia
estado_led_actual = "apagado"  # Estado actual del LED

# Variables para el control del botón
ultimo_estado_boton = 1  # Estado inicial (sin presionar)
ultimo_tiempo = 0
debounce_time = 50  # Tiempo de debounce en milisegundos
def enviar_estado_led(led, accion):
    """Envía el estado del LED a través del UART"""
    datos = {
        "led_actual": led,
        "accion": accion}
    txData = ujson.dumps(datos)
    uart1.write(txData + '\n\r')
    print(txData)
```

Python

```
def manejar_boton():
    """Maneja el evento del botón con debounce"""
        global ultimo_estado_boton, ultimo_tiempo, indice_actual,
estado_led_actual
    estado_actual = boton.value()
    tiempo_actual = time.ticks_ms()

    # Detectar flanco descendente (botón presionado) con debounce
    if estado_actual == 0 and ultimo_estado_boton == 1:
            if time.ticks_diff(tiempo_actual, ultimo_tiempo) >
debounce_time:

# Botón presionado - cambiar estado del LED
            if estado_led_actual == "apagado":
                # Encender el LED actual
                led_actual = led_sequence[indice_actual]
                enviar_estado_led(led_actual, "encender")
                estado_led_actual = "encendido"
                print(f"LED {led_actual} encendido")
            else:
                # Apagar el LED actual y pasar al siguiente
                led_actual = led_sequence[indice_actual]
                enviar_estado_led(led_actual, "apagar")
                print(f"LED {led_actual} apagado")


                # Avanzar al siguiente LED en la secuencia
                            indice_actual = (indice_actual + 1) %
len(led_sequence)
                estado_led_actual = "apagado"
            ultimo_tiempo = tiempo_actual
        ultimo_estado_boton = estado_actual

print("Sistema listo. Presiona el botón para controlar los LEDs.")
print("Secuencia: " + " -> ".join(led_sequence))
while True:
    manejar_boton()
    time.sleep(0.01)  # Pequeña pausa para evitar sobrecarga
```

Posterior a esta carga en el MCU RP2040, realizaremos la configuración para usar el MCU ESP32.



Códigos de funcionamiento ESP32

Python

```
import ujson
from machine import UART, Pin
uart0 = UART(1, baudrate=115000, tx=Pin(17, Pin.OUT), rx=Pin(16, Pin.IN))
led_rojo = Pin(4, Pin.OUT)#Pin GPIO5 como salida para el LED rojo
led_verde = Pin(26, Pin.OUT)#Pin GPIO18 como salida para el LED verde
led_azul = Pin(25, Pin.OUT)#Pin GPIO19 como salida para el LED azul
def ejecutar_accion(accion, pin_led):
    if accion == "encender":
        pin_led.on()  #Enciende el LED
    elif accion == "apagar":
        pin_led.off()  #Apaga el LED
def recibir_json():
    rx_data = b''  #Inicializa una cadena de bytes vacía
    while True:
        if uart0.any():
           byte_received = uart0.read(1)#Lee un byte desde el UART
            rx_data += byte_received
  #Verifica si el carácter de nueva línea indica el final del JSON
            if byte_received == b'\n':
                try:
                    # Intenta cargar el JSON
                    json_data = ujson.loads(rx_data.decode('utf-8'))
                    print("JSON recibido:", json_data)
# Extrae los valores de 'accion' y 'led_actual' del JSON
                    accion = json_data.get('accion', '')
                        led_actual = json_data.get('led_actual', '')
# Ejecuta la acción indicada en el JSON para cada LED
```

Python

```
 if led_actual == "rojo":
                        ejecutar_accion(accion, led_rojo)
                    elif led_actual == "verde":
                        ejecutar_accion(accion, led_verde)
                    elif led_actual == "azul":
                        ejecutar_accion(accion, led_azul)
                print("--led recibido:", led_actual, "accion:", accion)
                        return json_data
                    except ValueError as e:
                        print("Error al parsear JSON:", e)
# Reinicia la cadena si hay un error en el JSON
                    rx_data = 'b'
# Ejemplo de uso
while True:
    data = recibir_json() # Realiza acciones con el JSON recibido
```

Lección 9 Control de luz por comandos de voz

En esta lección aprenderás a implementar un sencillo control de una GPIO del **RP2040** mediante comandos de voz captados por un micrófono PDM que serán procesados por un programa de machine learning.

Materiales

1.- DUAL MCU ONE x1
2.- UNIT MP34DT05TR-A Módulo Micrófono PDM x1
3.- Cables Dupont Macho - Macho x4

Conocimientos previos

PDM

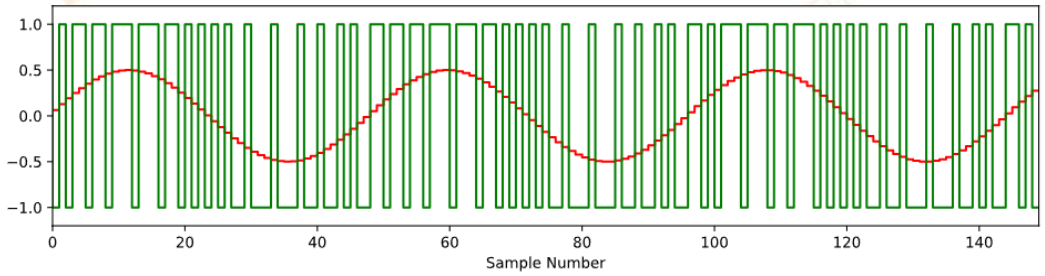
PDM (Modulación por Densidad de Pulsos) es una técnica de codificación de audio digital en la que la información de la señal analógica se representa como una secuencia de pulsos binarios (1s y 0s).

A diferencia de otros métodos (como PCM o PWM), en PDM la información está en la densidad de los pulsos, no en su amplitud o duración.

La conversión a PDM típicamente se hace usando un **modulador Sigma-Delta (ΣΔ)**. Este dispositivo compara continuamente la señal analógica con una señal de retroalimentación y genera un flujo de bits:

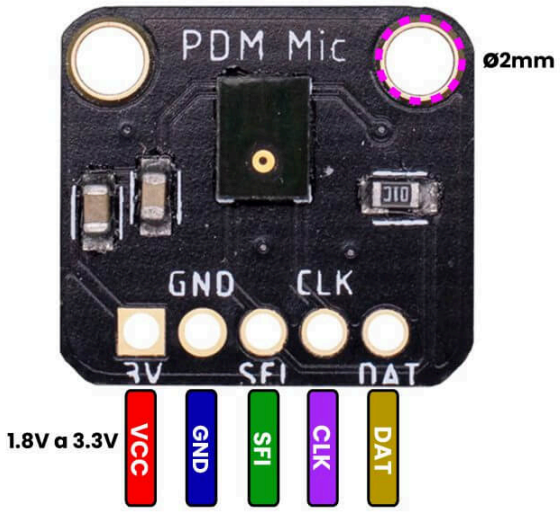
Si la señal de entrada es **alta**, se generan **más unos (1)**.
Si la señal de entrada es **baja**, se generan **más ceros (0)**.

La **densidad de unos** en un intervalo dado representa el valor promedio de la señal analógica durante ese intervalo.



Se usa para **convertir señales de audio analógicas en señales digitales** que luego pueden ser procesadas por un microcontrolador, grabadas o transmitidas.

Para el desarrollo de esta lección se hace uso del UNIT MP34DT05TR-A Módulo Micrófono PDM cuyo transductor está construido basado en la tecnología MEMS (Sistemas MicroElectromecánicos) los cuales son dispositivos **muy pequeños**, del tamaño de micras (µm), que combinan **componentes mecánicos y electrónicos** integrados en un solo chip de silicio que para el caso de este módulo de este micrófono contiene internamente un diafragma que vibra con el sonido. Esa vibración cambia la **capacitancia** entre el diafragma y una placa fija (como un condensador variable) dicha variación se convierte en una señal eléctrica la cual se amplifica y provee de una salida PDM.

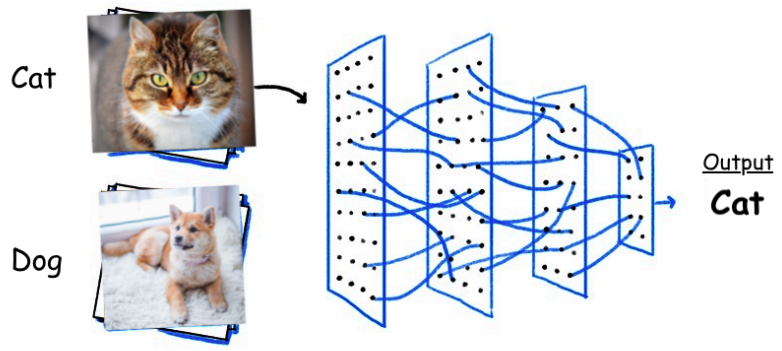


Machine learning

Machine Learning (ML) es una rama de la inteligencia artificial (IA) que **permite a las computadoras aprender de los datos** y **mejorar su rendimiento sin ser programadas explícitamente para cada tarea específica**.

En lugar de escribir instrucciones paso a paso, se le dan ejemplos (datos) a un algoritmo para que este **aprenda patrones** y pueda hacer **predicciones o tomar decisiones** basadas en nueva información.

Imagina que quieres que una computadora **reconozca si una imagen es de un gato o de un perro**. En programación tradicional, tendrías que decirle: "si tiene orejas puntiagudas y bigotes, entonces es un gato"... Lo cual es complicado y poco fiable.



Con **Machine Learning**, simplemente le das al sistema **miles de imágenes etiquetadas** (algunas de gatos, otras de perros), y el sistema **aprende por sí solo** a distinguirlas basándose en esos ejemplos.

Las etapas básicas de un sistema de machine learning son:

1. **Recolección de datos:** Se necesitan muchos datos: imágenes, textos, números, registros, etc.

2. **Preprocesamiento de los datos:** Los datos deben limpiarse, normalizarse y prepararse. Esto incluye eliminar errores, convertir texto en números, etc.

3. **Elección del modelo:** Hay muchos tipos de algoritmos de Machine Learning (como árboles de decisión, redes neuronales, máquinas de vectores de soporte, etc.).

4. **Entrenamiento del modelo:** Se alimenta al modelo con los datos y se ajustan sus parámetros para que "aprenda".

5. **Evaluación:** Se prueba el modelo con datos que no ha visto antes para ver qué tan bien ha aprendido.

6. **Predicción:** Una vez entrenado, el modelo puede hacer predicciones con nuevos datos.

7. **Ajustes y mejoras:** Se puede mejorar el modelo ajustando parámetros o cambiando el tipo de algoritmo.

Diagrama de conexiones

Para la realización de esta lección deberás seguir el diagrama que a continuación se muestra, donde solo utilizaremos cuatro de los cinco pines del módulo PDM, los cuales dos hilos son Vcc y GND, mientras que los dos hilos restantes corresponden a los datos (DAT) y a la señal de reloj (CLK).

Para realizar dichas conexiones será necesario soldar 4 cables dupont Macho-Macho a tu módulo PDM.

Código de funcionamiento

Para poder implementar el control de una GPIO utilizando comandos de voz deberás instalar la librería "Comandos_de_voz_interfacing" y después compilar y cargar al RP2040 de tu UNIT DUAL MCU ONE el archivo L9_Comandos_de_voz.ino que se encuentra en el siguiente repositorio de GitHub:

https://github.com/UNIT-Electronics/UNIT-ONE-MCU-/tree/main/L9_Comandos_de_voz

NOTA: ⚠️La primera vez que cargas el código, la compilación puede tardar 40 minutos como mínimo para de allí en adelante tener tiempos más razonables como con cualquier otro programa en Arduino IDE⚠️

Configuración de Edge impulse

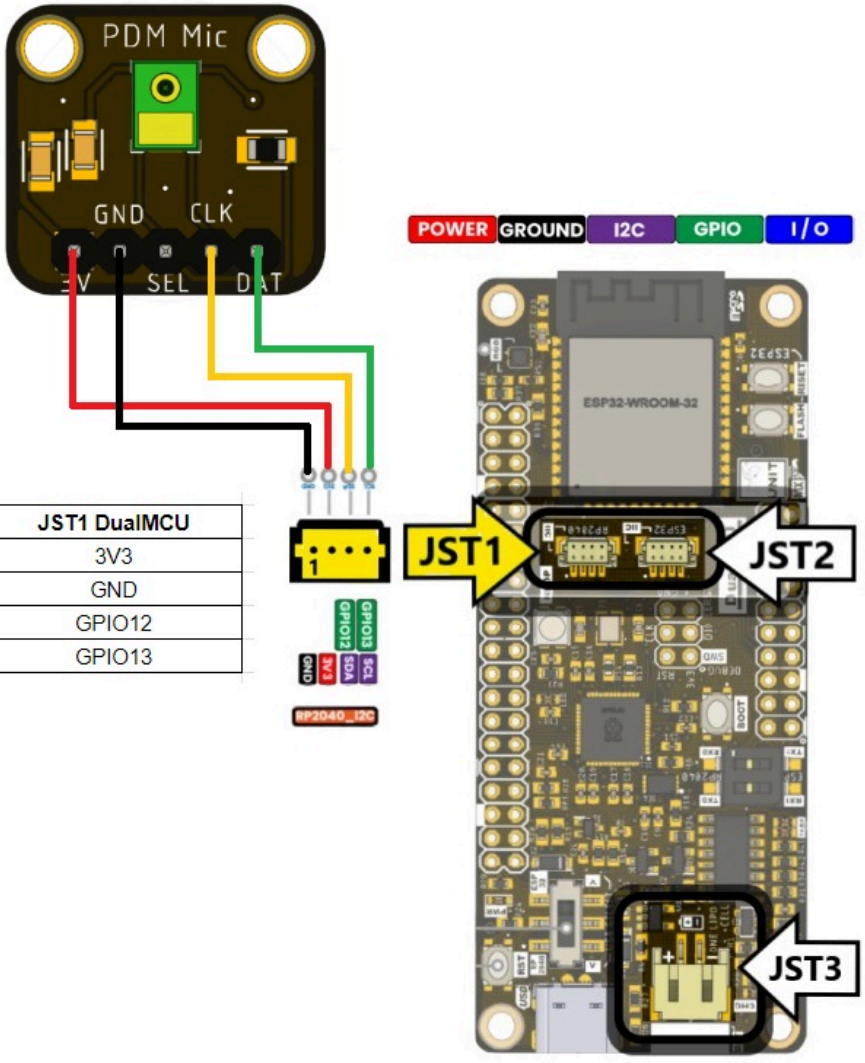
Si deseas realizar tus propios códigos de machine learning, crea una cuenta en **Edge impulse dando clic en este enlace** https://edgeimpulse.com/ y sigue el tutorial que la misma plataforma te brinda con las únicas observaciones de:

- Configurar su RP2040 como un micrófono USB para entrenar al modelo de machine learning
- Configurar el microcontrolador al cual desea cargar el programa y la librería que posteriormente Edge impulse le permite descargar tras entrenar y generar un modelo de machine learning.

Para configurar su tarjeta como micrófono USB

Al momento de que la plataforma le pide grabar muestras de audio (las palabras que desea sean detectadas) es indispensable que utilice el mismo micrófono PDM para la realización de esta lección para este propósito se puede convertir toda la placa UNIT DUAL MCU ONE en un micrófono extra para su PC, solo siga estos pasos:

1.- Ubique los botones de Boot y Reset correspondientes al RP2040 en la tarjeta, señalados en la siguiente imagen



UNIT PDM MP34DT05	JST1 DualMCU
3V	3V3
GND	GND
CLK	GPIO12
DAT	GPIO13

2.- Conecte la placa a su PC por USB y a continuación presione la siguiente combinación de botones rápidamente

1.- Reset
2.- Boot
3.- Mantenga ambos presionados
4.- Suelte Reset, mantenga Boot
5.- Repita una segunda vez desde el paso 1 al 4 rápidamente.

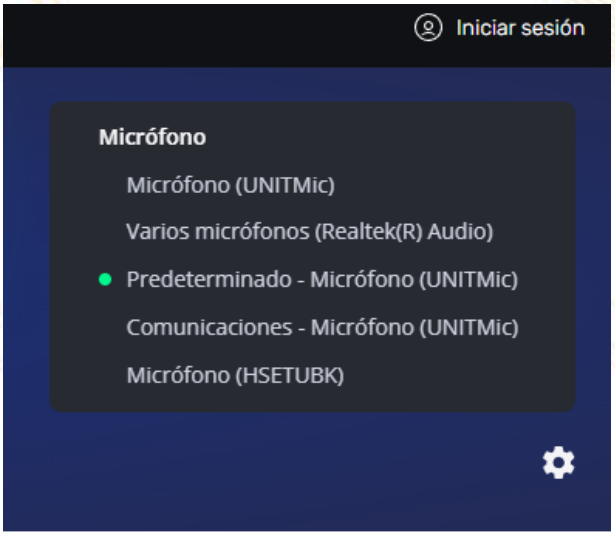
Tras lo cual, en su explorador de archivos se abrirá una unidad de almacenamiento similar a la siguiente

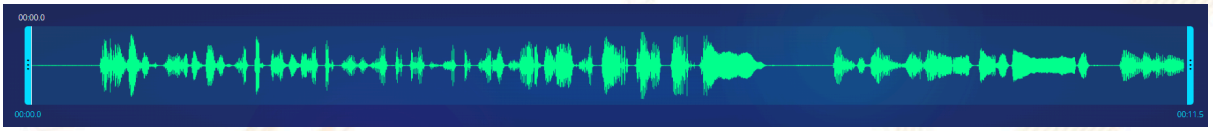
En dicha unidad deberá arrastrar y soltar el archivo **DualONE_USBMic_v1.uf2** que podrá descargar en el repositorio de la lección https://github.com/UNIT-Electronics/UNIT-ONE-MCU-/tree/main/L9_Comandos_de_voz

Con lo anterior su tarjeta será reconocida como un micrófono USB, puede revisar lo anterior en su administrador de dispositivos

Y comprobar su funcionamiento en el siguiente sitio web sugerido: https://online-voice-recorder.com/es/

Únicamente seleccione el UNITMic y listo, ya podrá usar su tarjeta como un micrófono USB





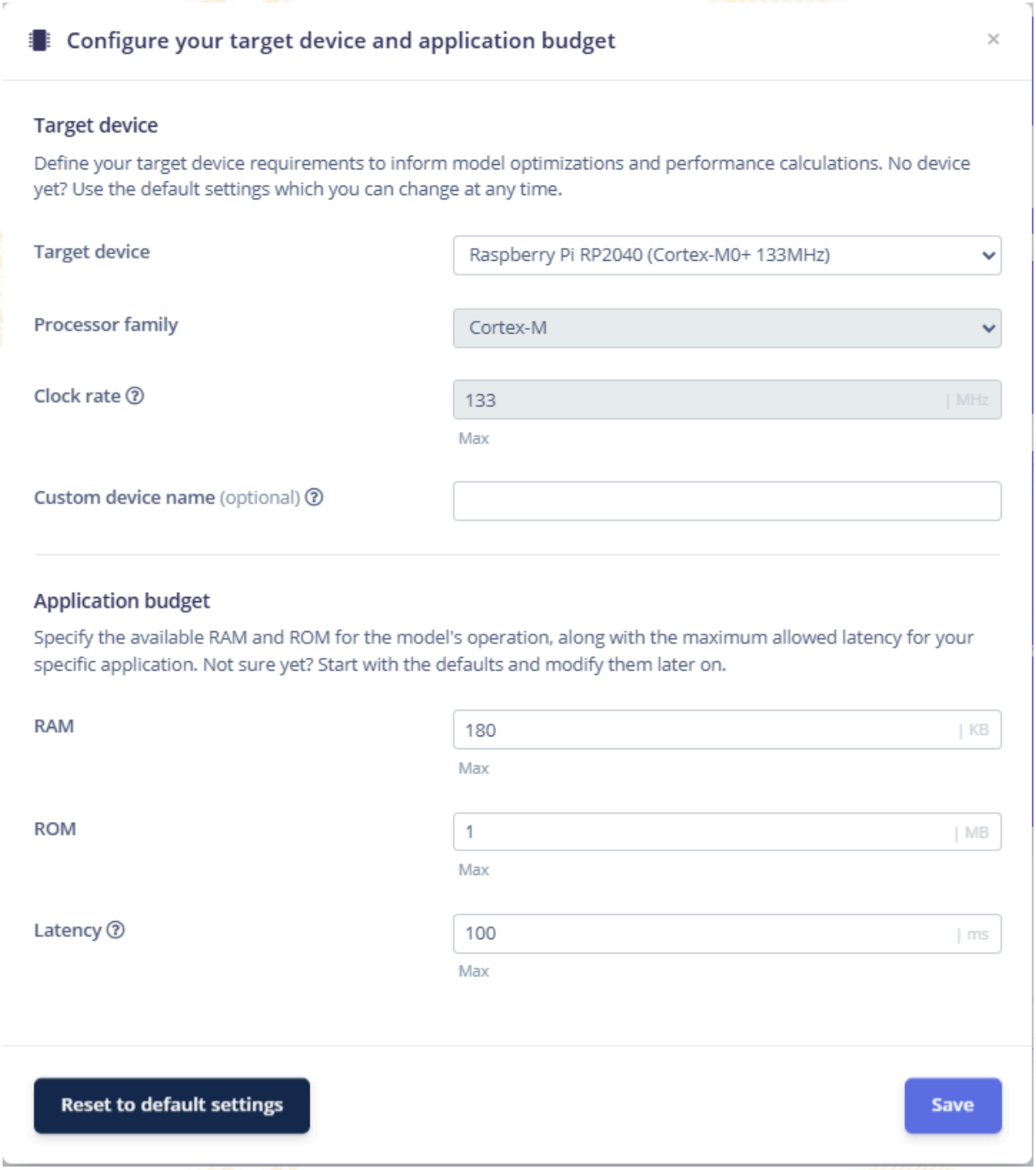
Para volver a la normalidad su RP2040, es decir, que vuelva a ser reconocida como un puerto COM en su PC, presione la misma combinación de botones y cargue un Blink desde su Arduino IDE al puerto UF2

Una vez que se haya terminado de volcar el programa, presione el botón de Reset y su RP2040 volverá a ser reconocido como un puerto COM en su PC

Para configurar el microcontrolador en la plataforma Edge Impulse

1.- Haz clic en este botón ubicado en la parte superior derecha de la ventana, donde podremos configurar el microcontrolador y sus capacidades de memoria. ⚠️ Esto es crucial debido a que los modelos de machine learning requieren recursos que fácilmente pueden exceder las cantidades de moria de un microcontrolador⚠️

Por ello se recomienda tener la siguiente configuración para el RP2040 donde bajamos la disponibilidad de RAM y ROM a 180KB y 1M respectivamente.
Da clic en el botón **Save** para guardar la configuración



Gracias por su preferencia

Manual elaborado por el Equipo UNIT Electronics
Septiembre 2025