KIT BOX

UNIT DUAL ONE

RP2040+ESP32

Development board

All the power in a single board

Manual de Prácticas

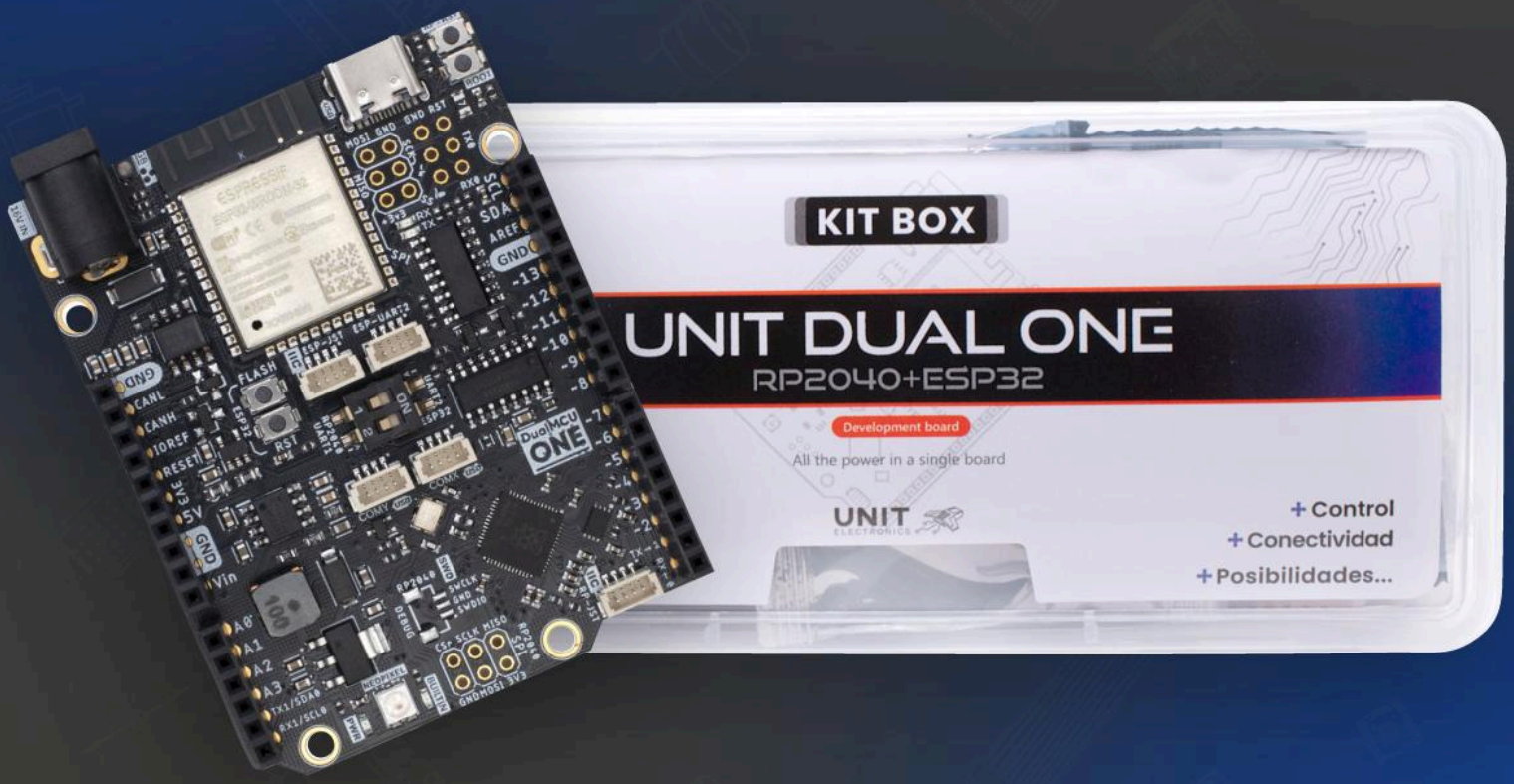


Imagen solo con fines ilustrativos. El producto real puede diferir.

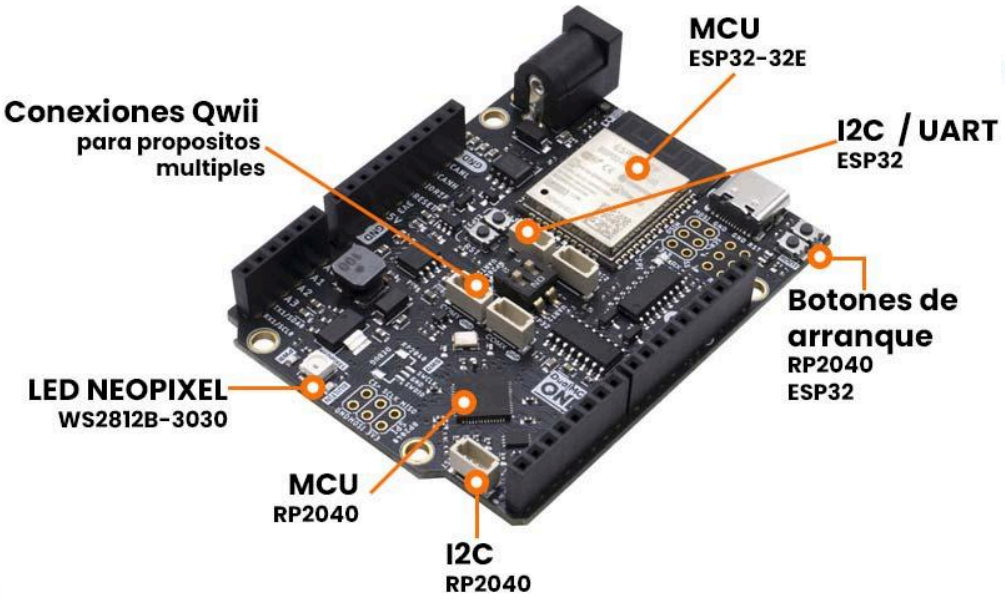
UNIT ELECTRONICS

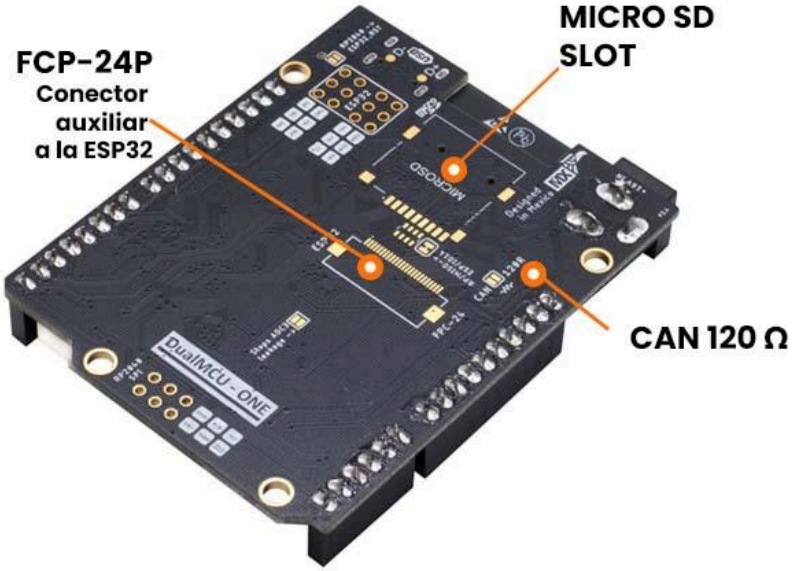
TIENDA DE COMPONENTES ELECTRÓNICOS

ÍNDICE

Primeros pasos con DUALMCU ONE

La DualMCU ONE es una placa de desarrollo que combina dos microcontroladores potentes: el RP2040 y el ESP32, intercomunicados a través de SPI, es la versión hermana de UNIT DualMCU pero con la configuración homóloga de pines de un Arduino UNO, obteniendo lo mejor de ambas tarjetas de desarrollo.



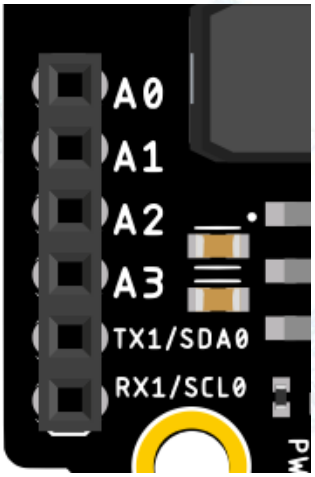


Distribución de pines

Analógicos

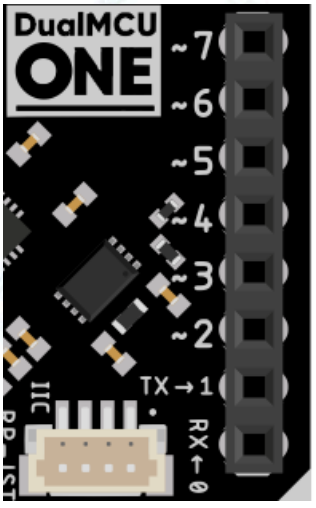
Los pines analógicos A0 a A3, TX1 y RX1 están asignados a los pines GPIO correspondientes en el microcontrolador RP2040.

PIN	Entrada/Salida RP2040
A0	26
A1	27
A2	28
A3	29
TX1 / SDA0	22
RX1 / SCL0	23



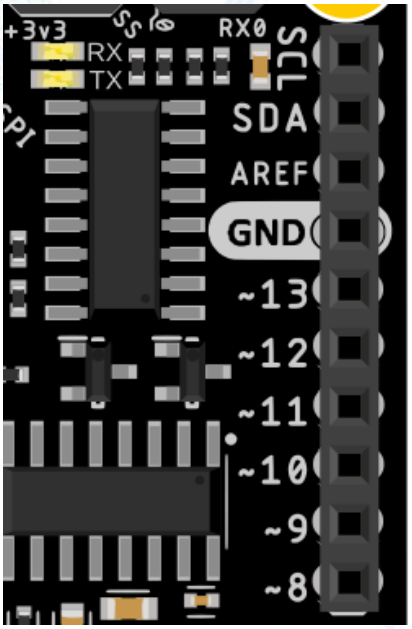
Distribución de pines digitales D0 a D7

PIN	Entrada/Salida RP2040
D8	GPIO2
D9	GPIO3
D10	GPIO17
D11	GPIO19
D12	GPIO16
D13	GPIO18
SDA0	GPIO20
SCL0	GPIO21



Distribución de pines digitales D8 a D13

PIN	Función	GPIO RP2040
D0	RX0	GPIO1
D1	TX0	GPIO0
D2	RX1	GPIO5
D3	TX1	GPIO4
D4	•	GPIO9
D5	•	GPIO11
D6	•	GPIO8
D7	•	GPIO10



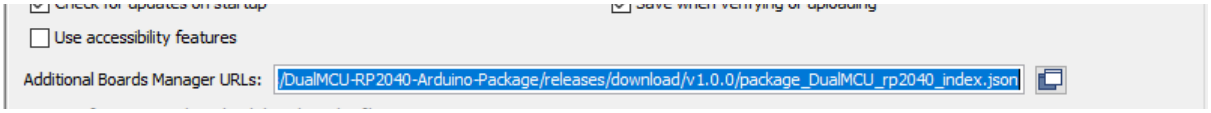
Consideraciones de Uso

El DualMCU-ONE funciona con niveles lógicos de 3,3 V en lugar de los 5 V típicos del Arduino Uno. Asegúrese de que cualquier protector o periférico conectado sea compatible con la lógica de 3,3 V para evitar posibles daños.

Instalación a través del IDE Arduino

1. Instale la IDE de Arduino (versión 1.8 o posterior). La versión actual está disponible en el sitio web de Arduino
2. Inicie Arduino y abra la ventana Preferencias.
3. Introduce uno de los enlaces de lanzamiento anteriores en el campo URL adicional del administrador de tableros . Puedes agregar varias URL separándolas con comas.

“https://raw.githubusercontent.com/UNIT-Electronics/Uelectronics-RP2040-Arduino-Package/main/package_Uelectronics_rp2040_index.json,https://github.com/UNIT-Electronics/Uelectronics-ESP32-Arduino-Package/releases/download/v1.0.0/package_Uelectronics_ESP32_index.json”



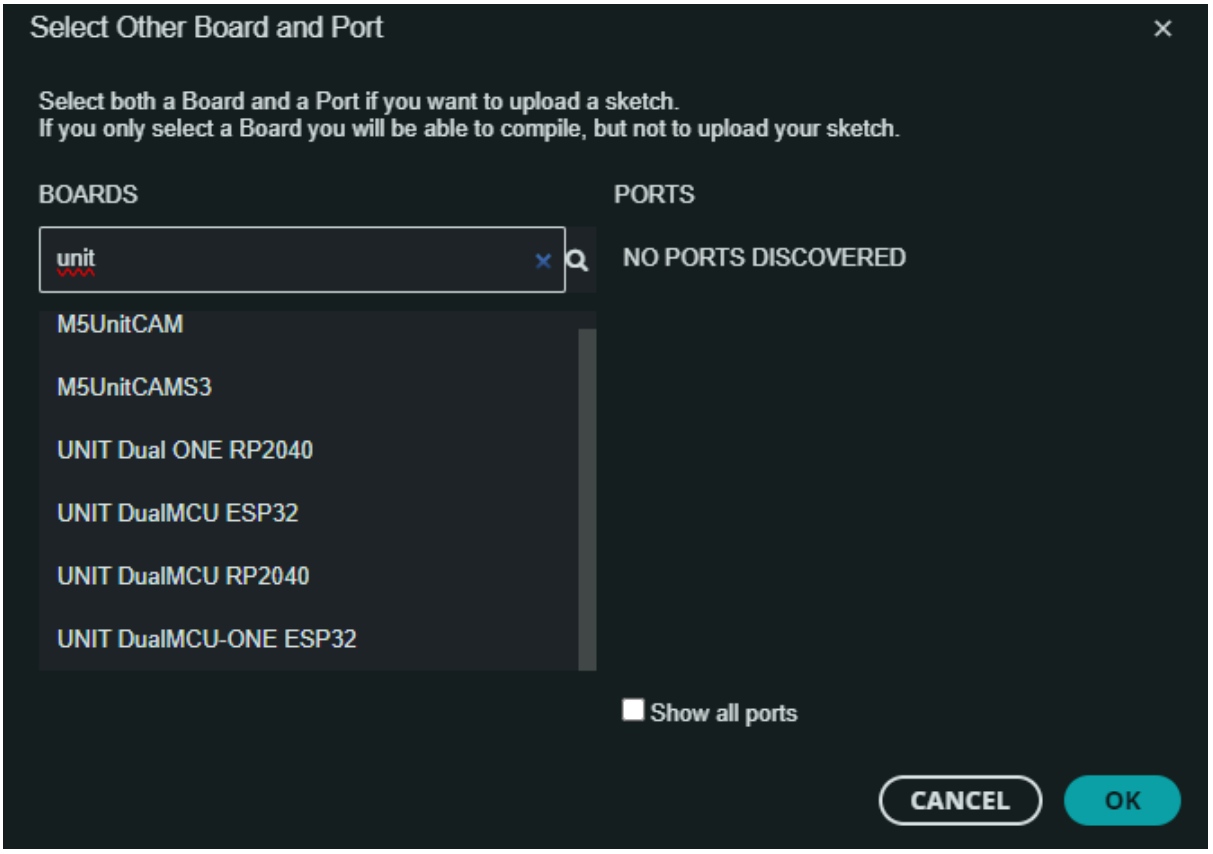
4. Pulse Aceptar para cerrar el cuadro de diálogo.
5. Vaya a Herramientas->Placas->Administrador de placas en el IDE
6. Escriba "**DUALMCU**” y aparecerá como opción “**UNIDAD Uelectronics RP2040**" en el cuadro de búsqueda y seleccione "Agregar":

Arduino ESP32 de Uelectronics

Posteriormente a la instalación de la placa DUALMCU, al seleccionar qué MCU será el que requiera ocupar verá que existen una colección de herramientas de software que permiten a los usuarios programar y controlar dispositivos utilizando el MCU ESP32 en la plataforma DualMCU ONE y Arduino.

El paquete incluye un conjunto de bibliotecas y herramientas para programar el ESP32 mediante Arduino (IDE).

Las que estaremos ocupando sera la correspondiente a UNIT Dual ONE RP2040 y UNIT DualMCU-ONE ESP32.



Lección 1 Entradas y salidas digitales

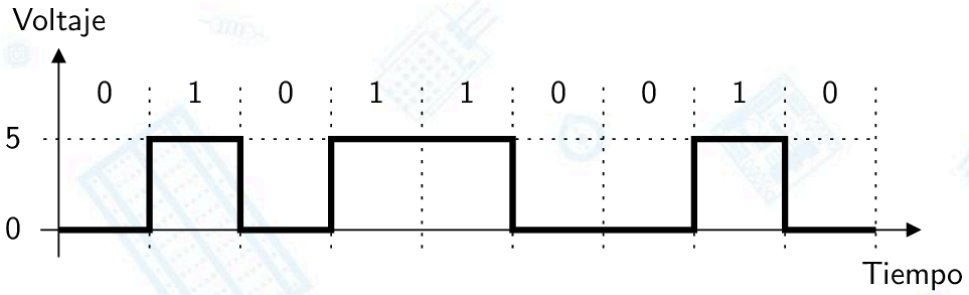
En esta lección aprenderás a encender un led al presionar dos botones utilizando el RP2040 integrado en la UNIT DUAL MCU a través de Arduino IDE.

Materiales

1.- Dual MCU ONE x1
2.- Protoboard x1
3.- Pulsadores x2
4.- Cables dupont Macho - Macho
6.- Led 5mm x1
7.- Resistencia 220 Ω ¼ W

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

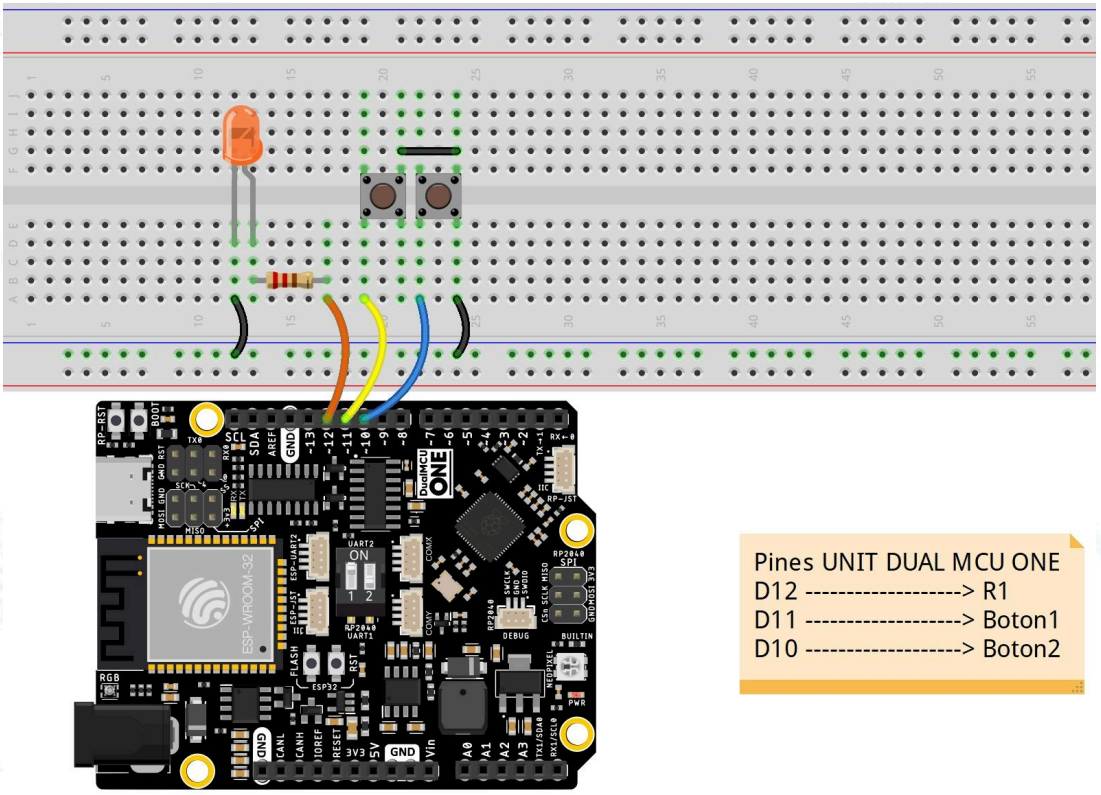
Una salida digital es un pin que se configura para *enviar* señales a dispositivos externos. El microcontrolador puede cambiar el estado de este pin a ALTO (1) o BAJO (0).

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.

Conexiones

Para la conexión consideraremos que el programa se declarara el par de pines como entrada digital , por lo cual se habilitaron resistencias pull up; esto nos evita el uso de resistencias externas. Por lo tanto cuando los dos botones sean presionados , internamente estaremos introduciendo un cero lógico y al dejarlos sin presionar se tendrá un uno lógico en dichos pines del microcontrolador.



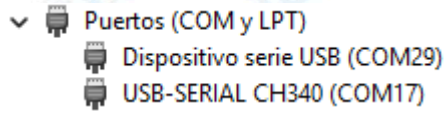
NOTA: Los pines en la UNIT DUAL MCU están rotulados como D3, D7, A0, A2, etc. pero al momento de escribir el código es importante notar que esos pines equivalen a un GPIO del RP2040 o del ESP32 por lo que deberás revisar el pinout

Código de funcionamiento

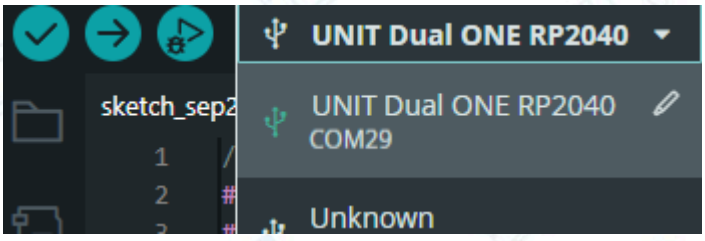
Carga el siguiente programa en la UNIT DUAL MCU ONE para el RP2040

```
// Pines de entrada para los pulsadores
#define boton1  19 // pin D11 GPIO19
#define boton2  17 // pin D10 D10 GPIO17
const int led = 16; // pin D12 GPIO16

void setup() {
  pinMode(boton1, INPUT_PULLUP);
  pinMode(boton2, INPUT_PULLUP);
  pinMode(led, OUTPUT);
}

void loop() {
  int estadoBoton1 = digitalRead(boton1);
  int estadoBoton2 = digitalRead(boton2);

//  Función NAND (&&): El LED solo enciende si ambos botones están presionados (ambos a cero)
  // Prueba cambiar la logica por una funcion NOR (||)
  if (estadoBoton1 == LOW && estadoBoton2 == LOW) {
    digitalWrite(led, HIGH);  // Enciende el LED
  } else {
    digitalWrite(led, LOW);   // Apaga el LED
  }
}
```

Para subir el código en tu placa Dual MCU ONE deberás revisar que puerto COM fue asignado al RP2040 y al ESP32, para ello abre tu administrador de dispositivos y busca la pestaña de Puertos COM y LPT y tras conectar tu placa a tu PC mediante USB tendrás algo similar a esto.



Donde el dispositivo nombrado como CH340 corresponde al ESP32 por lo que para el RP2040, en este caso sería el COM29
Dichos puertos los deberás ubicar en el Arduino IDE y seleccionar al que corresponde al RP2040



Lección 2 Entradas analógicas

En esta lección aprenderás a utilizar los convertidores analógicos mediante la lectura de tensión de un potenciómetro y de un sensor LM35 en el RP2040 y mediante sencillas operaciones matemáticas dentro del código en Arduino IDE se podrán obtener mediciones de tensión y temperatura.

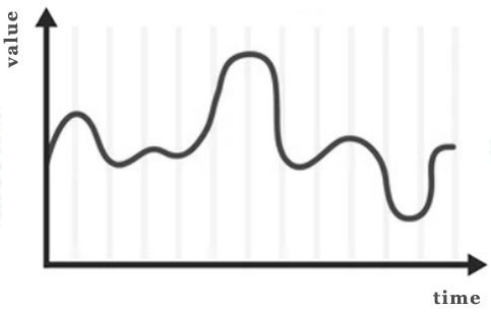
Materiales

1.- Dual MCU ONE x1
2.- LED 5mm x4
3.- Protoboard x1
4.- Potenciómetro x1
5.- Sensor de temperatura LM34 x1
6.- Cables Dupont Macho- Macho

Conocimientos previos

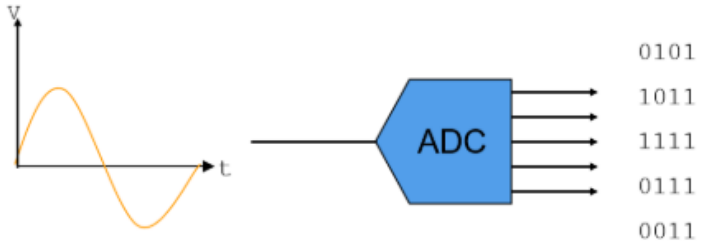
Entradas Analógicas

Las entradas analógicas son señales eléctricas que pueden tomar un rango continuo de valores dentro de un determinado intervalo. En contraste, las señales digitales 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 (2^8 = 256) niveles posibles.

Esto significa que la diferencia entre cada nivel será de aproximadamente:

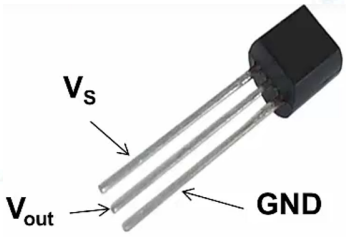
0.0195 V (5V / 256)

Si se utilizara un ADC de 10 bits (2^10 = 1024), la resolución sería mayor, y la diferencia entre niveles sería de:

0.0049 V (5V / 1024)

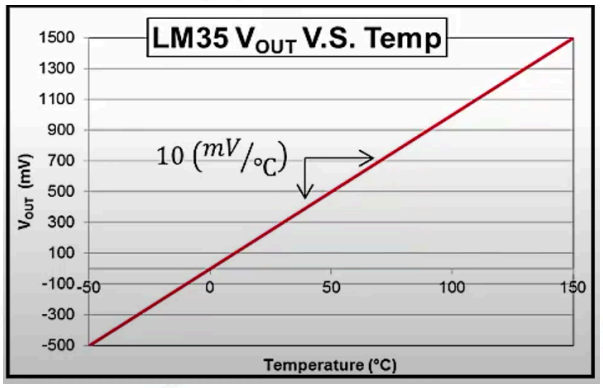
LM35

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).



Prácticas

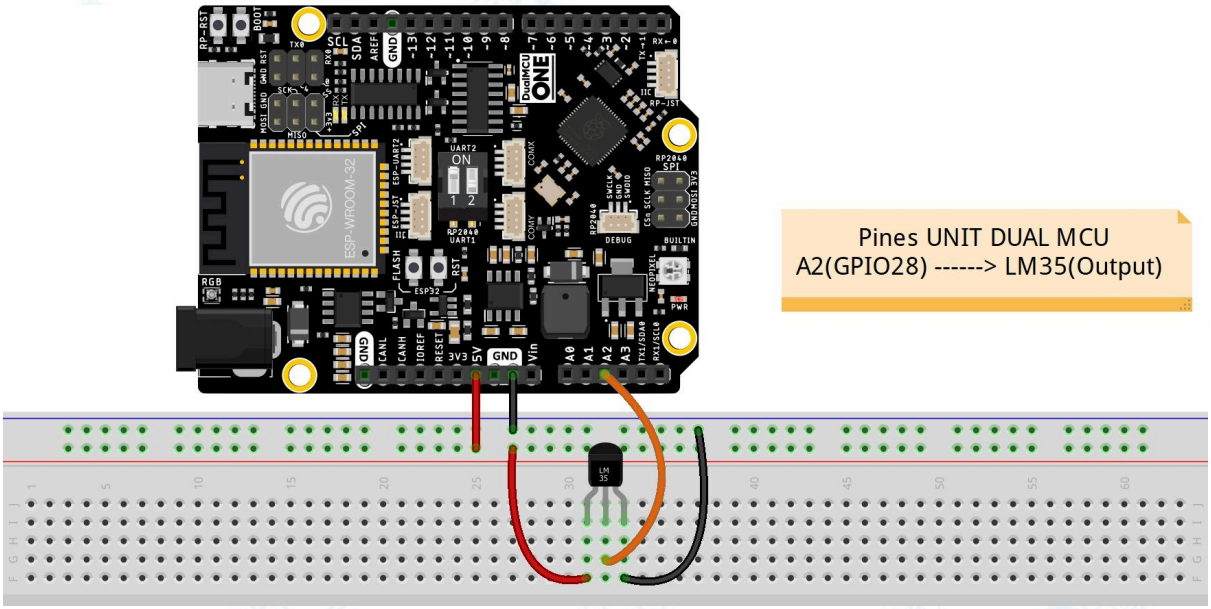
A continuación se presentaran 2 casos para obtener lecturas analogicas:

- Lecturas por medio del sensor LM35
- Vía Potenciómetro

1. **Sensor LM35**

En este primer circuito propuesto leeremos la temperatura en función de la tensión que entrega un sensor LM35, para lo anterior realizamos las siguientes conexiones.

Diagrama de conexión



Código de funcionamiento

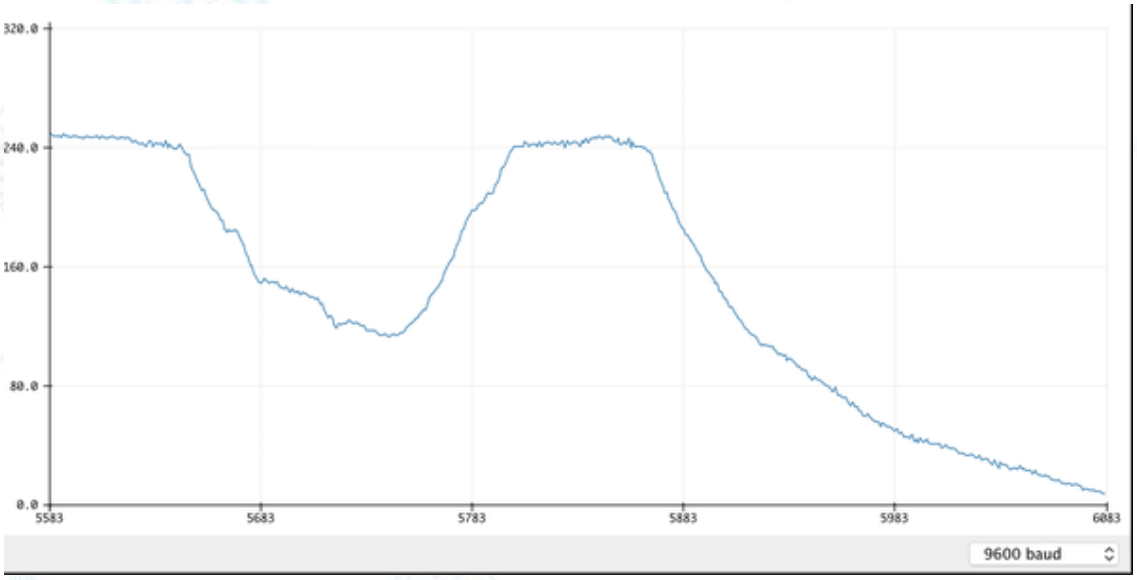
En este programa se observa cómo el **RP2040** procesa las lecturas del ADC que provienen del sensor LM35. Para una temperatura de 25 °C el LM35 entrega una tensión de 250 mV o 0.25 V pero necesitamos que esos 0.25 V se representan como 25 °C como se observa en la siguiente línea de código:

```
temp = ((analogRead(LM35) / 310.0) * 100);
```

```
// Define el pin analógico donde está conectado el sensor LM35
#define LM35 28  // GPIO28 → pin analógico A2 DUAL MCU ONE
// Variable para almacenar el valor calculado de temperatura
float temp = 0.0;
void setup() {
  // Configura el puerto serial para enviar datos al monitor serial
  Serial.begin(9600);  // Velocidad de comunicación: 9600 baudios
  // Configura el pin del LM35 como entrada analógica
  pinMode(LM35, INPUT);
}
void loop() {
  // Lee el valor analógico del pin (0 a 1023 para placas de 10 bits de
resolución)
  // Luego convierte ese valor a voltaje y después a temperatura en °C:
  // - El LM35 entrega 10 mV por cada grado Celsius
  // - Si el voltaje máximo es 3.3V y se usa una calibración aproximada →
divisor 310
  temp = ((analogRead(LM35) / 310.0) * 100);
   // Imprime el valor de temperatura en grados Celsius por el monitor
serial
  Serial.println(temp);
  // Espera 800 milisegundos antes de hacer otra lectura
  delay(800);}
```

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 el icono ubicado en la parte superior derecha del Arduino IDE.

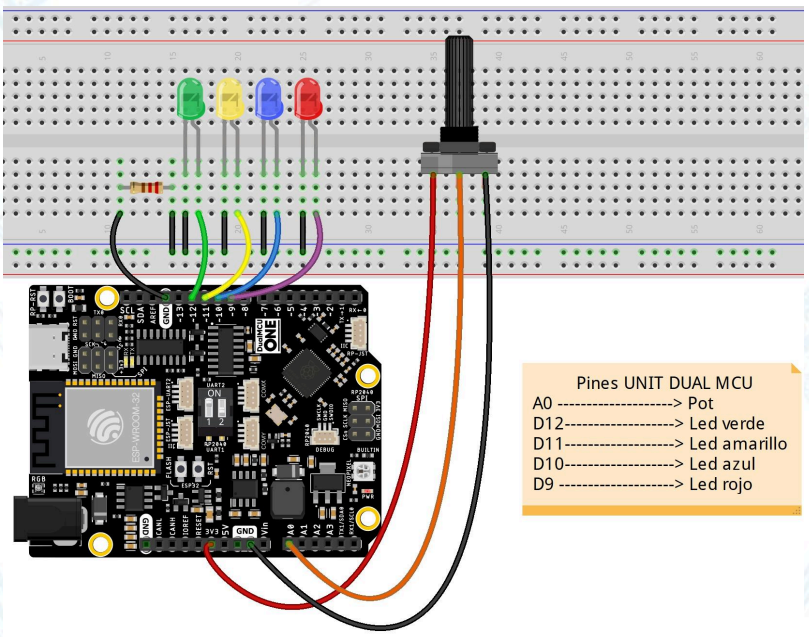
Y obtendrás la siguiente vista:



2. Vía Potenciómetro

⚠️Es importante que tengas cuidado al realizar las siguientes conexiones, sobre todo al conectar al potenciómetro el cual debe ser alimentado a 3.3v, ya que si lo conectamos a 5v se podría quemar el pin A2 porque el RP2040 no soporta tensiones mayores a 3.3v.⚠️

Diagrama de conexión



Código de funcionamiento

En este segundo programa realizamos el encendido de cuatro leds en función de la tensión que entrega un potenciómetro, de manera que tendremos un indicador LED de la señal analógica.
Te invitamos a realizar una combinación con el primer código, de manera que tengas un indicador LED de la temperatura leída por el sensor LM35.

```
#define pot 26 // GPIO26 es decir el pin A0 en la DUAL MCU ONE
// Definición de los pines donde están conectados los LEDs
#define ledVerde    16 // D12
#define ledAmarillo 19 // D11
#define ledAzul     17 // D10
#define ledRojo     3  // D9
// Variable para almacenar el valor de voltaje leído del potenciómetro
float voltaje = 0.0;
void setup() {
// Inicializa la comunicación serial para ver los datos por el monitor serial
  Serial.begin(9600);
  // Define el pin del potenciómetro como entrada
  pinMode(pot, INPUT);
  // Define los pines de los LEDs como salidas
  pinMode(ledVerde, OUTPUT);
  pinMode(ledAmarillo, OUTPUT);
  pinMode(ledAzul, OUTPUT);
  pinMode(ledRojo, OUTPUT);
}
void loop() {
  // Lee el valor analógico del potenciómetro (0 a 1023) y lo convierte a voltaje
  // Aquí se divide entre 310 como forma de calibración personalizada
  // (por ejemplo, si el voltaje máximo esperado es ~3.3V)
  voltaje = analogRead(pot) / 310.0;
  // Muestra el voltaje por el monitor serial
  Serial.println(voltaje);
  // Apaga todos los LEDs antes de decidir cuál encender
  digitalWrite(ledVerde, LOW);
  digitalWrite(ledAmarillo, LOW);
  digitalWrite(ledAzul, LOW);
  digitalWrite(ledRojo, LOW);
//Lógica de comparación para encender un LED dependiendo del nivel de voltaje
  if (voltaje < 1.0) {
    digitalWrite(ledVerde, HIGH);// Nivel bajo → LED verde
  } else if (voltaje >= 1.0 && voltaje < 2.0) {
    // Nivel medio-bajo → LED amarillo
    digitalWrite(ledAmarillo, HIGH);
  } else if (voltaje >= 2.0 && voltaje < 3.0) {
    digitalWrite(ledAzul, HIGH);    // Nivel medio-alto → LED azul
  } else {
    digitalWrite(ledRojo, HIGH);     // Nivel alto → LED rojo
  }
   // Espera un tiempo antes de hacer la siguiente lectura (para evitar parpadeos rápidos)
  delay(800);}
```

Lección 3 PWM

En esta lección aprenderás a utilizar la modulación por ancho de pulso del RP2040 para controlar el brillo de un LED y para controlar el ángulo de giro de un servomotor SG90 desde el microcontrolador y en función de un par de pulsadores para girar en un sentido o en otro

Materiales

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

Conocimientos previos

PWM

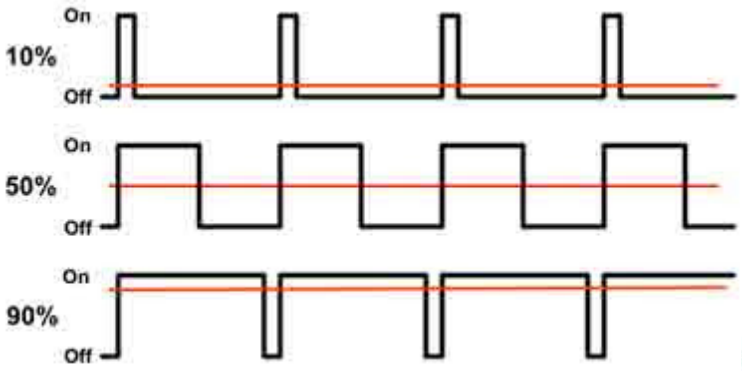
El PWM (Modulación por Ancho de Pulso, por sus siglas en inglés Pulse Width Modulation) es una técnica utilizada para controlar la potencia entregada a una carga eléctrica mediante una señal digital (encendido/apagado). Se utiliza en diversas aplicaciones, como el control de la intensidad de un LED,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

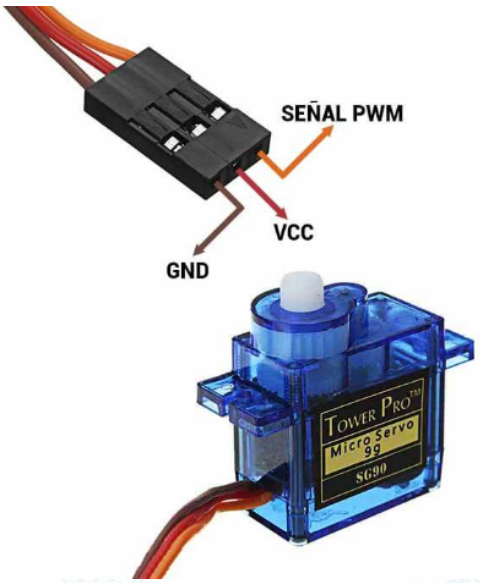


Algunas aplicaciones de la modulación por ancho de pulso son:

- **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.

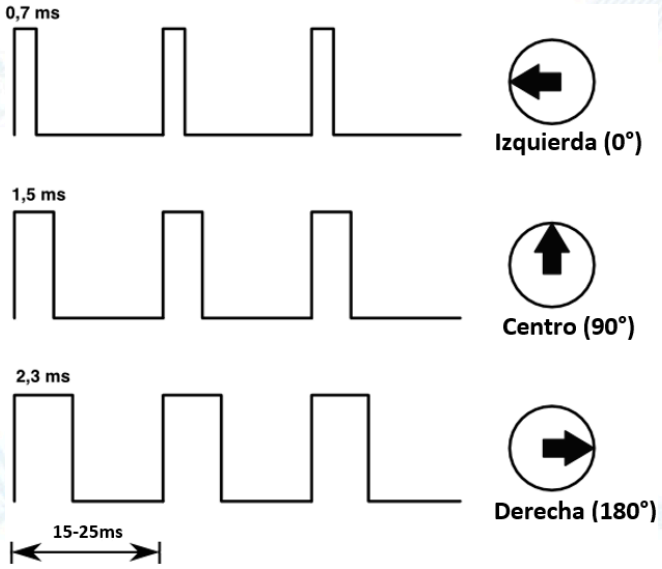
Servomotor

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.



Prácticas

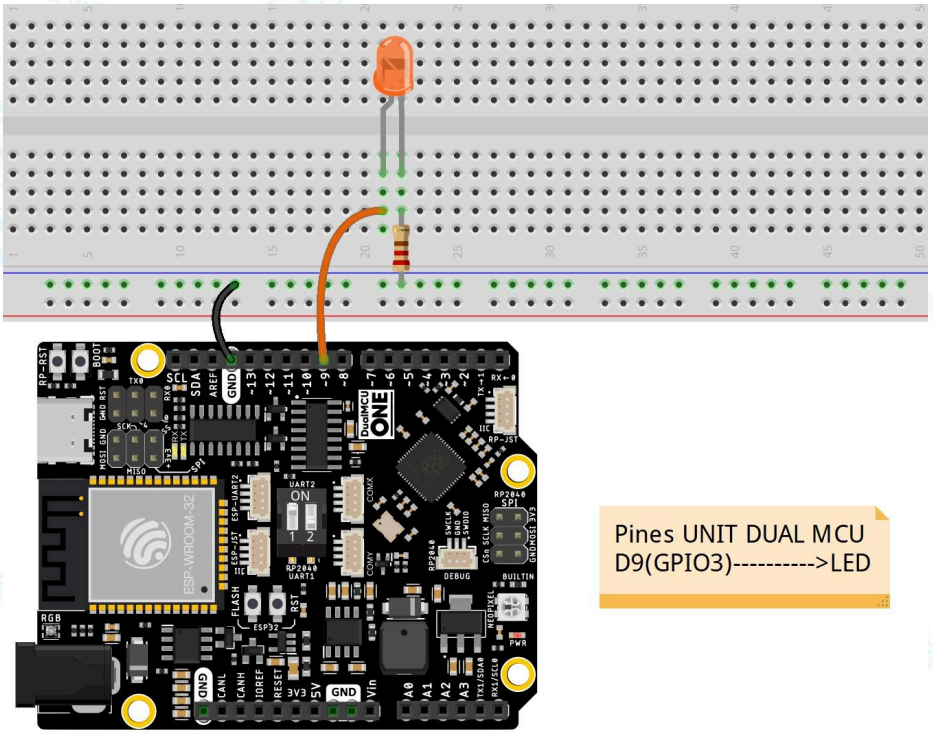
A continuación se presentan 2 casos para entender el funcionamiento del PWM en diferentes situaciones:

- Control de luz por PWM
- Control de un servomotor

1. **Control de luz por PWM**

En este primer diagrama se encuentra un led con su respectiva resistencia a GND para que puedas visualizar el aumento y disminución de la intensidad luminosa del led, esto debido a que el ciclo de trabajo de la señal PWM controla directamente la cantidad de potencia o voltaje promedio que se le entrega al LED.

Diagrama de conexiones



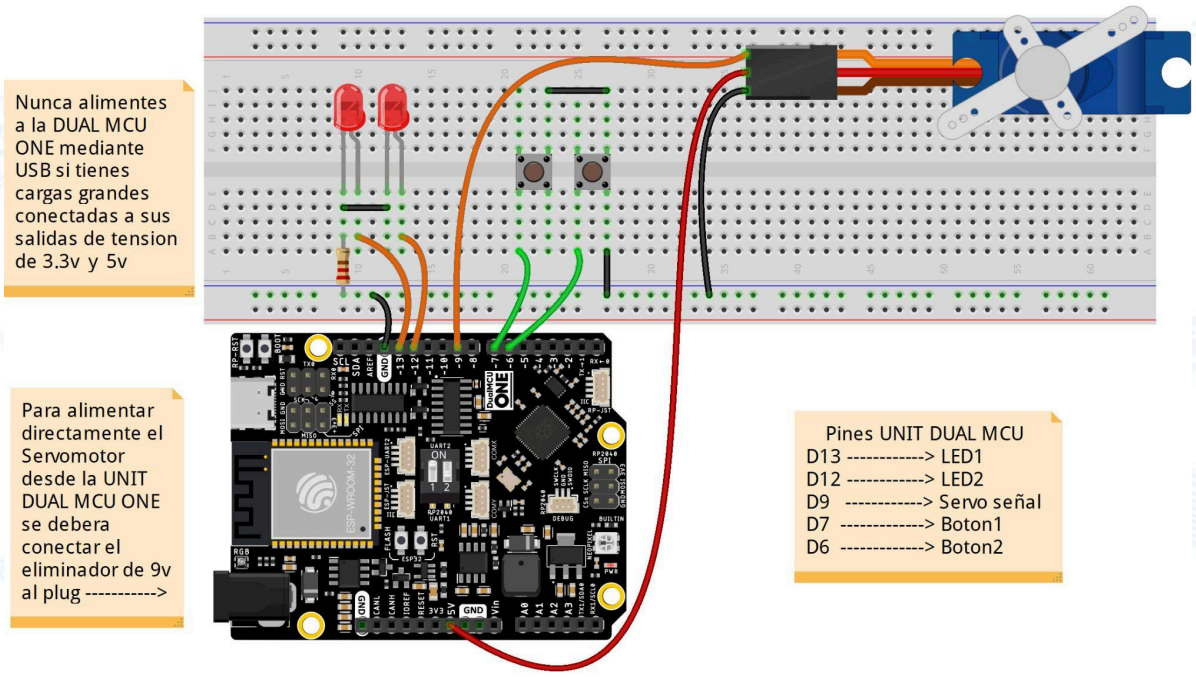
Código de funcionamiento 1

```
#define ledPWM 3  // Pin D9 PWM conectado al LED
int brillo = 0;          // Nivel de brillo actual (0 a 255)
int direccion = 5;       // Paso del cambio (positivo o negativo)
void setup() {
  pinMode(ledPWM, OUTPUT);
}
void loop() {
  analogWrite(ledPWM, brillo); // Aplica el valor de brillo al LED
  brillo += direccion;         // Aumenta o disminuye el brillo
  // Si llega a los extremos, invierte la dirección
  if (brillo <= 0 || brillo >= 255) {
    direccion = -direccion;
  }
   delay(20);   // Controla la velocidad de la transición (más bajo = más
rápido)
}
```

2. Control de un servomotor

Ahora el PWM controla la posición del servomotor espeficificamente el sentido de giro del servomotor mediante un par de pulsadores

Diagrama de conexión



Es importante hacer notar que el servomotor se encuentra alimentado directamente desde la UNIT DUAL MCU ONE a 5v y GND lo cual entra dentro de la capacidad de corriente (Hasta 2A) que nos puede proporcionar la tarjeta, pero cuidado, no te confíes que dicha corriente proviene de algún lado y ese lado no puede ser el puerto USB de tu PC, ya que podrías quemarlo. Por ello únicamente alimenta cargas que no rebasen los 2A desde la tarjeta cuando ésta se encuentre conectada a su vez por un eliminador capaz de entregar esa corriente de energía mediante el POWER JACK de la UNIT DUAL MCU ONE.

Código de funcionamiento

Para este código será importante 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.

```
// Pines de conexión

#define boton1 10       // GPIO10 Aumentar ángulo al pin D7
#define boton2 8       // GPIO8 Disminuir ángulo al pin D6
#define pinServo 3     // GPIO3 Señal PWM al servo al pin D9
#define led1 18        // GPIO18 pin D13
#define led2 16        // GPIO16 pin D12
int angulo = 0;        // Ángulo inicial del servo (grados)
void setup() {
  pinMode(boton1, INPUT_PULLUP);  // Botón conectado a GND
  pinMode(boton2, INPUT_PULLUP);  // Botón conectado a GND
  pinMode(pinServo, OUTPUT);      // Pin del servo como salida
  pinMode(led1, OUTPUT);       // Pin del led como salida
  pinMode(led2, OUTPUT);}        // Pin del led2 como salida

void loop() {
  // Si se presiona el botón 1, aumentar ángulo
  if (digitalRead(boton1) == LOW) {
   angulo+=20;
   angulo = constrain(angulo, 0, 180);
  digitalWrite(led1, HIGH);
  digitalWrite(led2, LOW);
  delayMicroseconds(10);} //Pequeña pausa para evitar que suba muy rápido
if (digitalRead(boton2) == LOW){//Disminuye angulo,al presionar el
botón2, ángulo
    angulo-=20;
    angulo = constrain(angulo, 0, 180);
    digitalWrite(led2, HIGH);
    digitalWrite(led1, LOW);
    delayMicroseconds(10); // Pausa para evitar que baje muy rápido
  } // Enviar la señal PWM manual al servo (20 ms cada ciclo)
```

```
for (int i = 0; i < 5; i++) { // Repetir varias veces para asegurar que el servo responda
      // Si se presiona el botón 2, disminuir ángulo
int pulso = map(angulo, 0, 180, 500, 2500);//Convertir ángulo a microsegundos (0° = 0.5 ms, 180° = 2.5 ms)
    digitalWrite(pinServo, HIGH);
    delayMicroseconds(pulso);
    digitalWrite(pinServo, LOW);
    delay(20 - (pulso / 1000));
      }
} // Resto del ciclo de 20 ms
```

En esta lección se implementaron las señales PWM mediante software; te invitamos a probar librerías dedicadas a desplegar el control PWM de una manera más sencilla como la librería **Servo.h** en Arduino IDE, a cuyas funciones prueba cambiando el valor del ángulo al que deseamos que se mueva el eje del servomotor y ¡listo!

Lección 4 Driver L298 control de motor DC

En esta lección aprenderás a controlar un motor de corriente continúa utilizando el RP2040 integrado en la UNIT DUAL MCU ONE programado en Arduino IDE a través del módulo de puente H L298

Materiales

1.- Dual MCU ONE x1
2.- Módulo puente H L298 x1
3.- Motor DC x1
4.- Cables Dupont Macho - hembra
5.- Potenciómetro x1

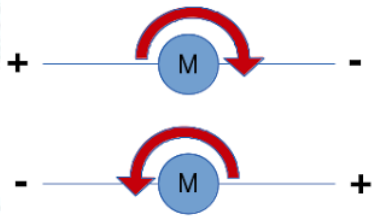
Conocimientos previos

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.

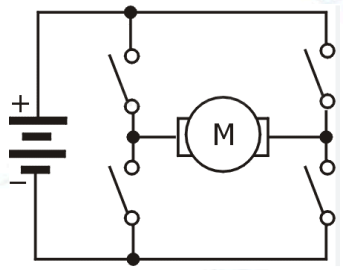


2. Uso de un puente H

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 sin la incómoda necesidad de intercambiar la polaridad hacia las conexiones físicas.

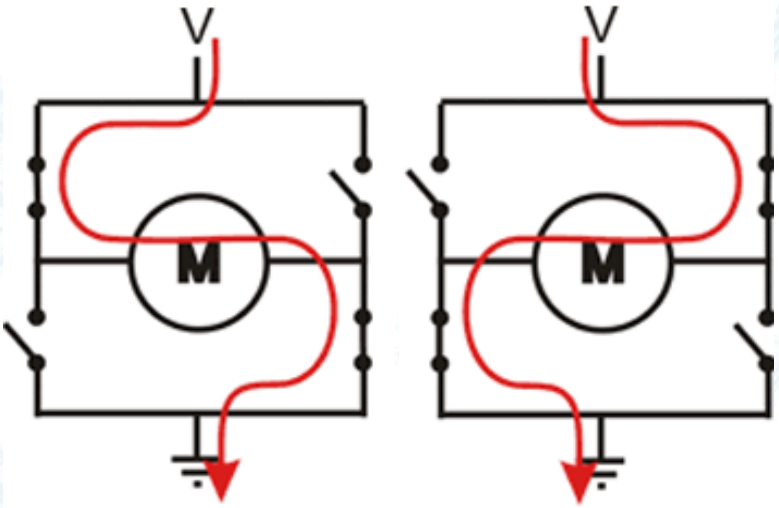
¿Cómo funciona un puente H?

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.



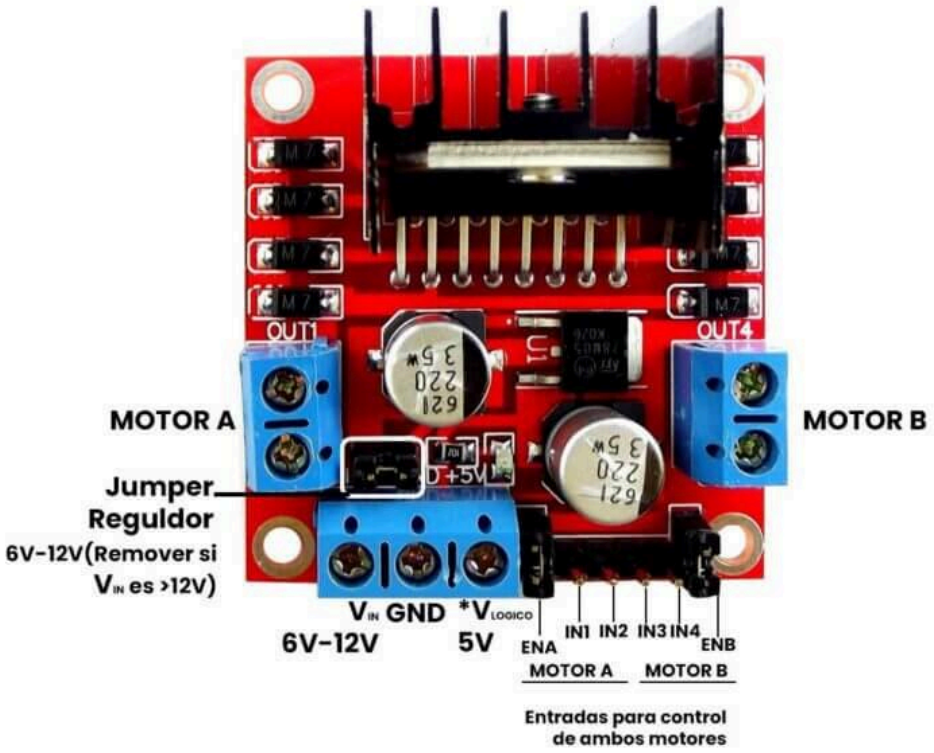
Módulo Puente H L298

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 Regulador Activado <Instalado>, funciona como salida de 5V
Jumper Regulador Desactivado<Removido>,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.⚠️

Prácticas

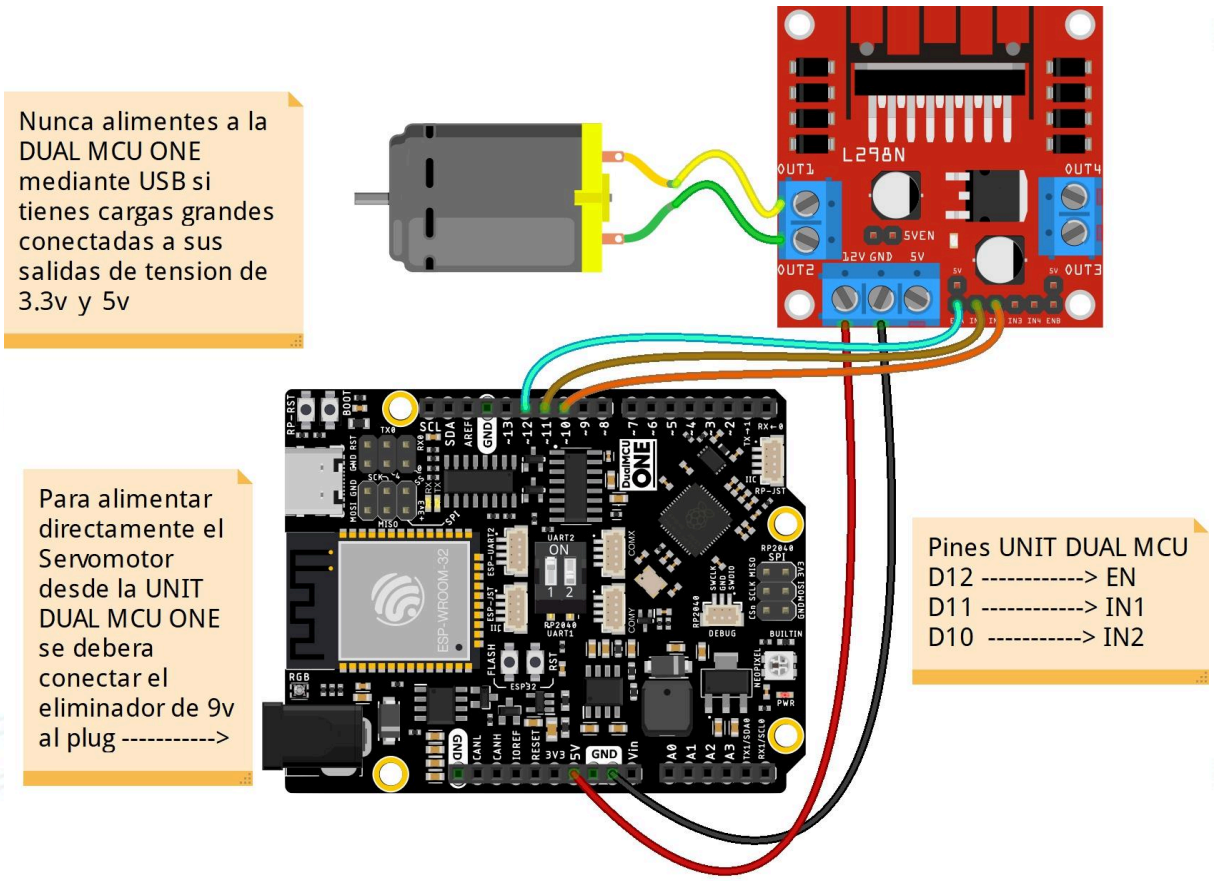
A continuación se presentan 2 casos para entender el funcionamiento driver L298 en diferentes situaciones:

- Control digital del motor
- Control analogico via potenciometro

1. **Control digital del motor**

En este primer circuito hacemos uso del RP2040 integrado en la Dual MCU ONE.

Es importante hacer notar que el motor a través del módulo L298 se encuentra alimentado directamente desde la UNIT DUAL MCU ONE a 5v y GND lo cual entra dentro de la capacidad de corriente (Hasta 2A) que nos puede proporcionar la tarjeta, pero cuidado, no te confíes, que dicha corriente proviene de algún lado y ese lado no puede ser el puerto USB de tu PC, ya que podrías quemarlo. **Por ello únicamente alimenta cargas que no rebasen los 2A desde la tarjeta cuando ésta se encuentre conectada a su vez por un eliminador capaz de entregar esa corriente de mediante el POWER JACK de la UNIT DUAL MCU ONE.**



Código de funcionamiento

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: `digitalWrite(l298n_enable, HIGH);` por medio del pin D12 (GPIO16) y controlamos el giro en un sentido si en el módulo tenemos un pulso alto en IN1 y uno bajo en IN2 y viceversa; lo que en el código logramos mediante las sentencias

```
digitalWrite(l298n_input1,HIGH);
digitalWrite(l298n_input1,LOW);
digitalWrite(l298n_input2,HIGH);
digitalWrite(l298n_input2,LOW);
```

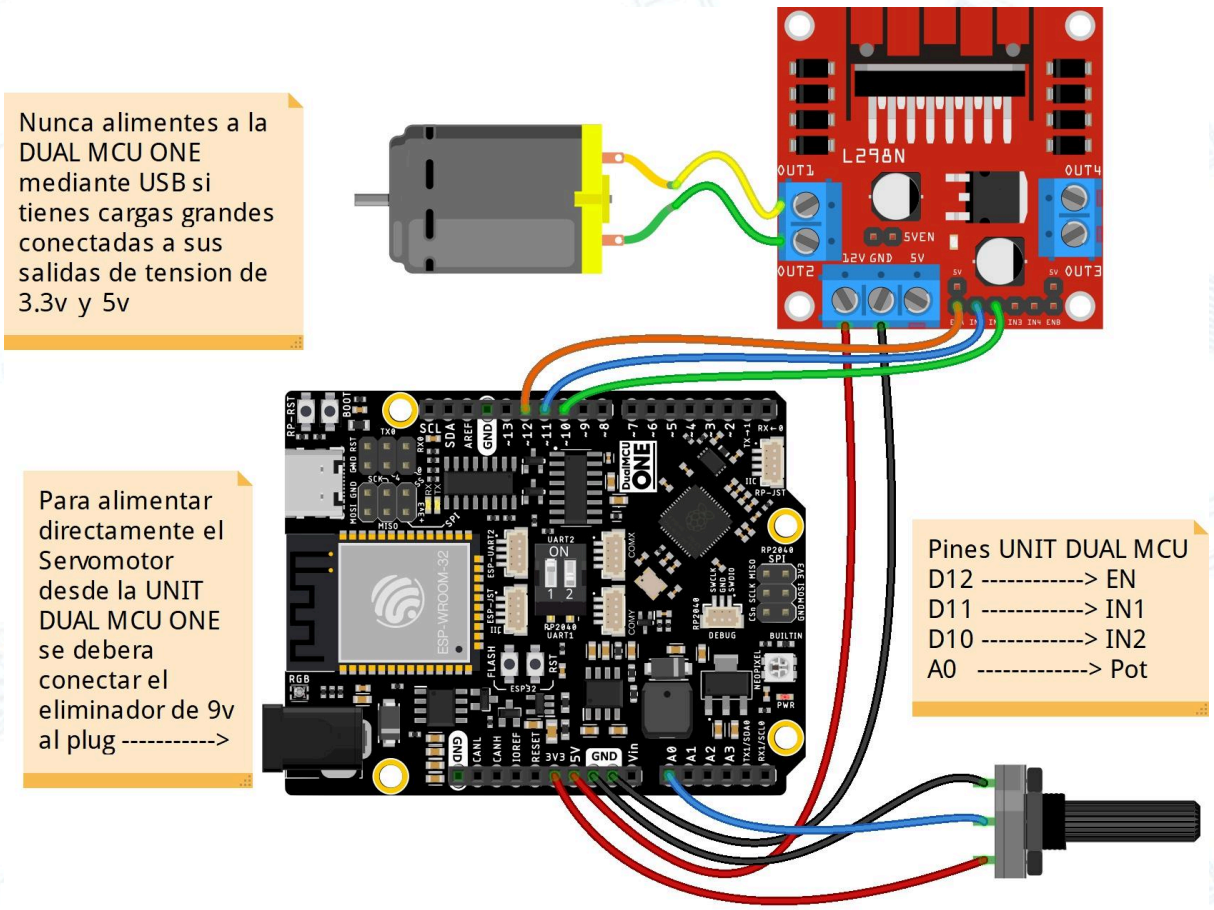
```
// Definición de pines
const int l298n_enable = 16; //EN del L298N (D12 para la DUAL MCU ONE)
const int l298n_input1 = 19; //IN1 del L298N (D11 para la DUAL MCU ONE)
const int l298n_input2 = 17; //IN2 del L298N (D10 para la DUAL MCU ONE)
void setup() {
  // Configura los pines como salida
  pinMode(l298n_enable, OUTPUT);
  pinMode(l298n_input1, OUTPUT);
  pinMode(l298n_input2, OUTPUT);
}
void loop() {
  // Habilita el motor
  digitalWrite(l298n_enable, HIGH);
  // Gira en un sentido
  digitalWrite(l298n_input1, HIGH);
  digitalWrite(l298n_input2, LOW);
  delay(2000); // Espera 20 segundos
  // Deshabilita el motor
  digitalWrite(l298n_enable, LOW);
  delay(1000); // Espera 1 segundo
  // Habilita el motor nuevamente
  digitalWrite(l298n_enable, HIGH);
  // Gira en el sentido contrario
  digitalWrite(l298n_input1, LOW);
  digitalWrite(l298n_input2, HIGH);
  delay(1000); // Espera 1 segundo
  // Inhabilita el motor
  digitalWrite(l298n_enable, LOW);
}
```

Control analogico via potenciometro

En esta segunda práctica controlamos 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.

⚠️ Ten cuidado con la alimentación a la que conectas el potenciómetro, ya que esta debe ser de 3.3v y no de 5v porque tanto el ESP32 como el RP2040 se quemarán si desde cualquier pin reciben pulsos que rebasen los 3.3v ⚠️

Diagrama de conexiones



Código de funcionamiento

En el siguiente programa es importante resaltar la secuencia en la que se logra controlar la velocidad de giro mediante el potenciómetro, ya que dentro del bucle while primero se obtiene la lectura de la tensión del potenciómetro usando la sentencia: `int potValue = analogRead(potPin)`,después ese valor se mapea y se escribe en la sentencia: `analogWrite(l298n_input4, pwmValue);`). De esta manera logramos que el ciclo de trabajo esté en función del valor de conversión del ADC proveniente del potenciómetro.

```
// Pines para el L298N y el potenciómetro
const int l298n_enable = 16;// EN del L298N(D12 para la DUAL MCU ONE)
const int l298n_input4 = 19;//IN1 del L298N(PWM)(D11 para DUAL MCU ONE)
const int l298n_input3 = 17;//IN2 del L298N (D10 para la DUAL MCU ONE)
const int potPin = 26;//Pin analógico hacia el potenciómetro(A0 para la
DUAL MCU ONE)
void setup() {
  pinMode(l298n_enable, OUTPUT);
  pinMode(l298n_input3, OUTPUT);
  pinMode(l298n_input4, OUTPUT);
  digitalWrite(l298n_enable, HIGH); // Habilita el motor
  Serial.begin(9600);
}
void loop() {
  // Lee el valor del potenciómetro (0 a 1023 en Arduino Uno)
  int potValue = analogRead(potPin);
  // Mapea el valor leído al rango del PWM (0 a 255 en Arduino)
  int pwmValue = map(potValue, 0, 1023, 0, 255);
  // Apaga IN2 para girar el motor en una dirección
  digitalWrite(l298n_input3, LOW);
  // Aplica PWM a IN1
  analogWrite(l298n_input4, pwmValue);
  // Imprime el valor para depuración
  Serial.print("PWM = ");
  Serial.println(pwmValue);
  delay(100); // Espera 100ms antes de la siguiente lectura
}
```

Lección 5 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 RP2040 de la UNIT DUAL MCU ONE.

Materiales

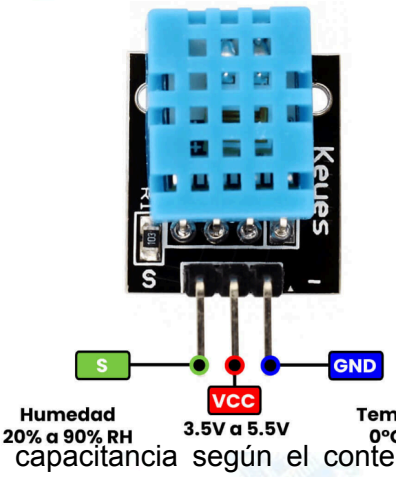
1.- DUAL MCU ONE x1
2.- Módulo KY-015 DHT11 x1
3.- Protoboard x1
4.- Cables jumper Macho-Hembra
5.- Resistencia 220 Ω a ¼ W
6.- LED 5mm x1

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.

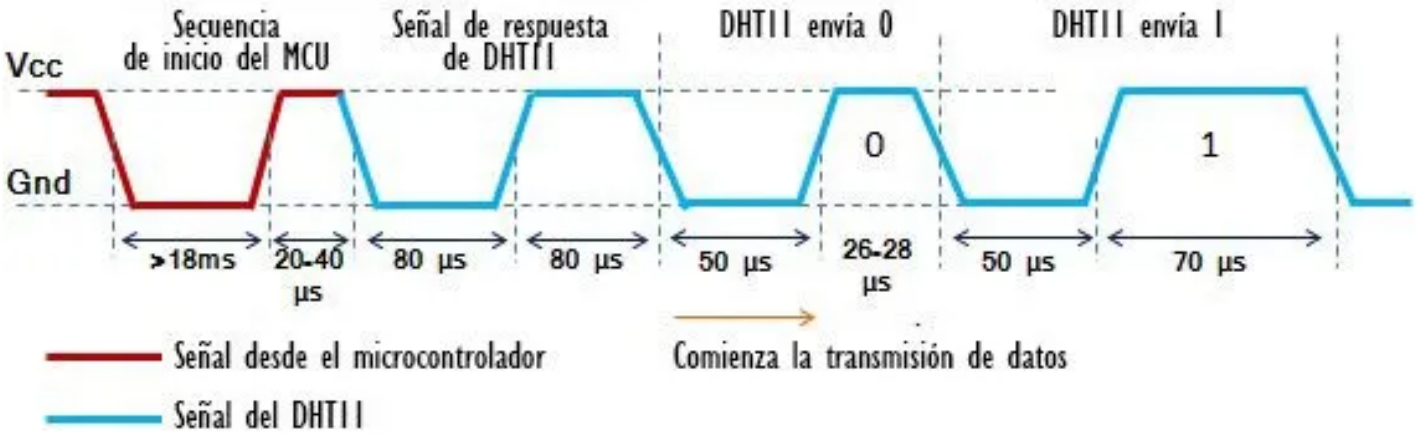
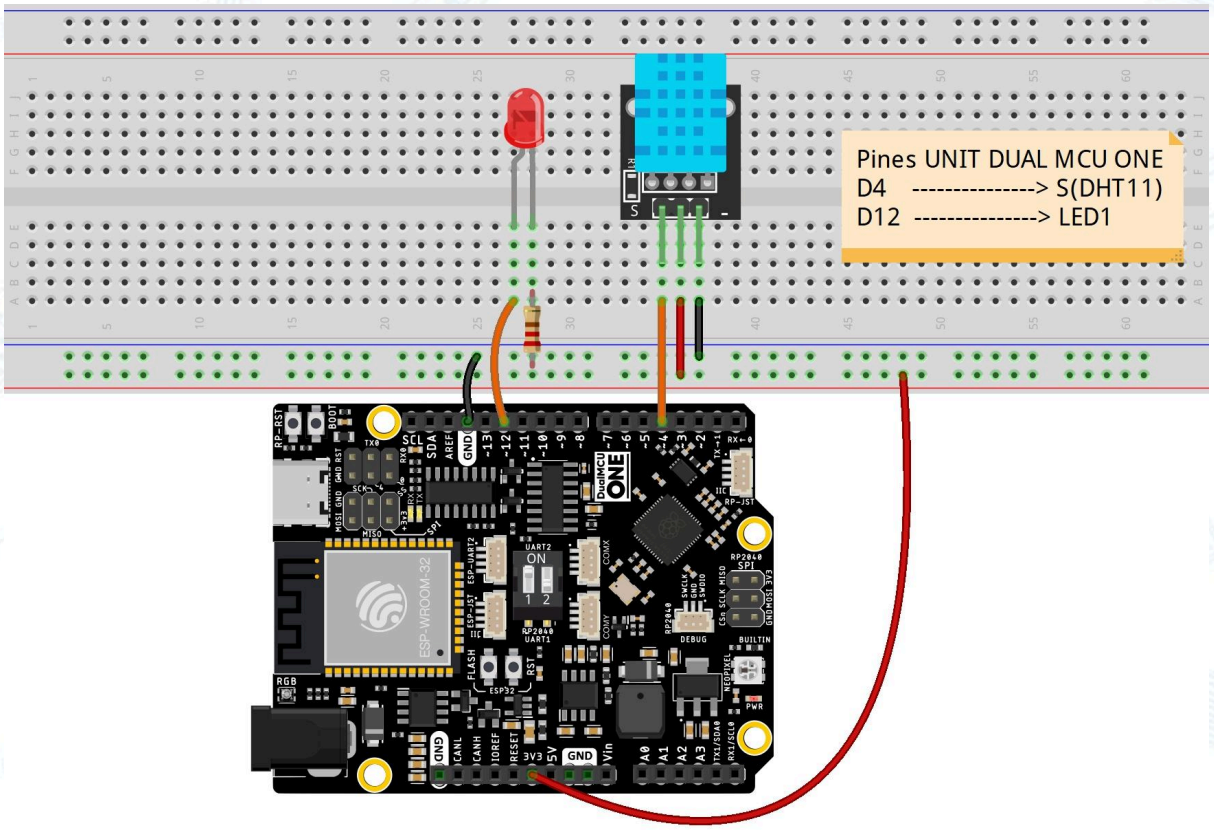


Diagrama de conexión



Código de funcionamiento

El siguiente programa usa la librería DHT.h la cual tendrás que incluir desde el Administrador de librerías

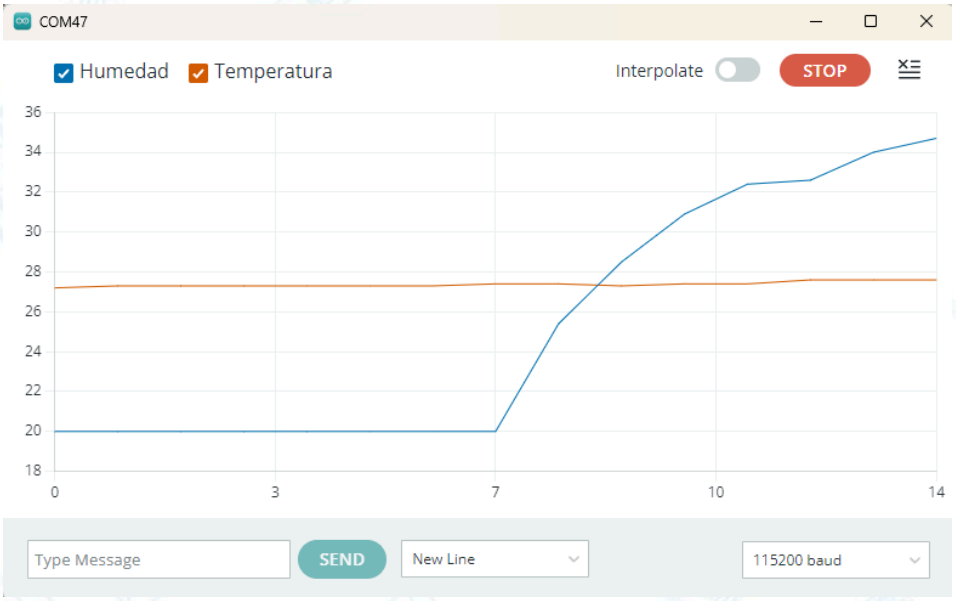
```
#include <DHT.h>
// Definiciones de pines
#define DHTPIN 9// Pin digital conectado al DHT11 (D4 en la UNIT DUAL MCU ONE)
#define DHTTYPE DHT11// Tipo de sensor
#define LED_PIN 16// Pin para el LED (D12 en la UNIT DUAL MCU ONE)
// Inicializa el sensor
DHT dht(DHTPIN, DHTTYPE);
unsigned long previousMillis = 0;
const long interval = 2000; // Intervalo de lectura en milisegundos (2 segundos)
void setup() {
  Serial.begin(9600);
  dht.begin();
  pinMode(LED_PIN, OUTPUT);
}
```

```
void loop() {
  unsigned long currentMillis = millis();
  // Verifica si han pasado 2 segundos desde la última lectura
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    // Lee temperatura y humedad
    float h = dht.readHumidity();
    float t = dht.readTemperature();
    // Verifica que la lectura sea válida
    if (isnan(h) || isnan(t)) {
      Serial.println("Error al leer del sensor DHT11");
      return;
    }
    Serial.print("Humedad:");
    Serial.print(h);
    Serial.print("%\t");    //Borra el sigo % si quieres graficar
    Serial.print("Temperatura:");
    Serial.print(t);
    Serial.println(" *C");  // Comenta esta línea si quieres graficar
    //Serial.println(); // Descomenta esta línea si quieres graficar
    // Control del LED según temperatura
    if (t > 28) {
      digitalWrite(LED_PIN, HIGH);}
 else {
      digitalWrite(LED_PIN, LOW);
            }
      }
}
```

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

```
Humedad: 30.00 %        Temperatura: 26.00 *C
Humedad: 32.90 %        Temperatura: 26.20 *C
Humedad: 32.80 %        Temperatura: 26.30 *C
Humedad: 32.10 %        Temperatura: 26.30 *C
Humedad: 31.50 %        Temperatura: 26.30 *C
Humedad: 31.20 %        Temperatura: 26.30 *C
```

Pero si requieres visualizar las gráficas deberás borrar el signo de porcentaje, comentar la línea `Serial.println(" *C");` y descomentar la línea `Serial.println();` debido a que el serial plotter no admite como símbolos válidos cualquier otro carácter que no sea un número entero o flotante


COM47
Humedad
Temperatura
Interpolate
STOP
36
34
32
30
28
26
24
22
20
18
0
3
7
10
14
Type Message
SEND
New Line
115200 baud

Lección 6 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

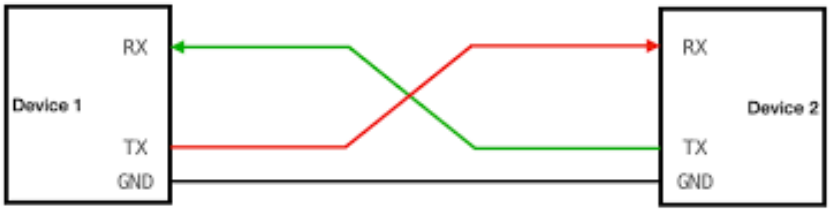
1.- DUAL MCU ONE x1
2.- Push button x1
3.- Protoboard x1
4.- Cables Dupont Macho - Macho

Conocimientos previos

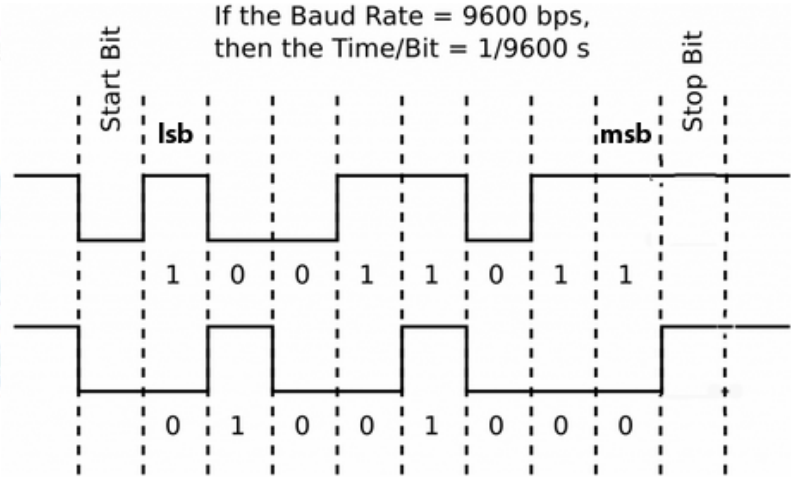
UART (Universal Asynchronous Receiver / Transmitter)

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.

Consulta el pinout de la UNIT DUAL MCU ONE para tener una referencia más precisa de la ubicación de los pines para los módulos UART, haciendo clic en el siguiente enlace: https://uelectronics.com/wp-content/uploads/2025/01/DualMCU-ONE-Pinout-V1.pdf

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.

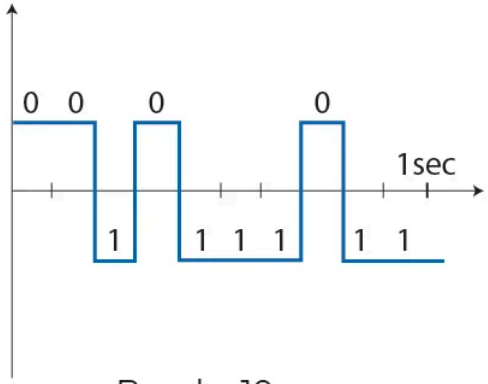
Baudios (Baud rate)

Es la cantidad de símbolos por segundo que se transmiten.

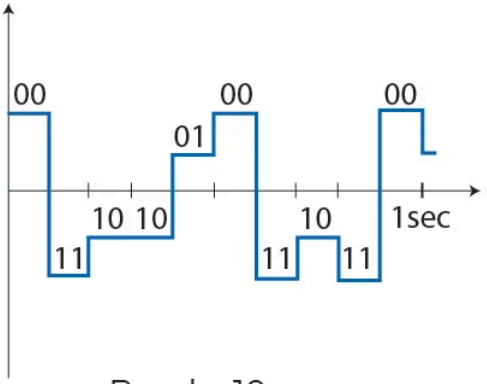
- Un símbolo es una unidad de señal transmitida (por ejemplo, un cambio de voltaje).
- En UART, cada símbolo suele equivaler a 1 bit, así que en la práctica:
    - Baudios = bps, en sistemas simples como UART.
- Pero en otros protocolos más complejos, cada símbolo puede representar más de 1 bit (por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

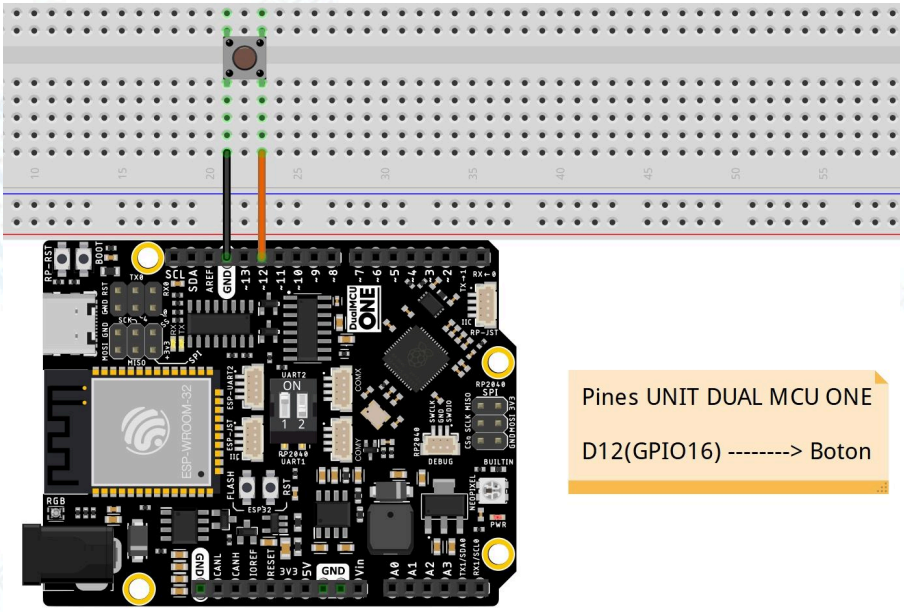


Baud = 10
Bit rate = 10 bps

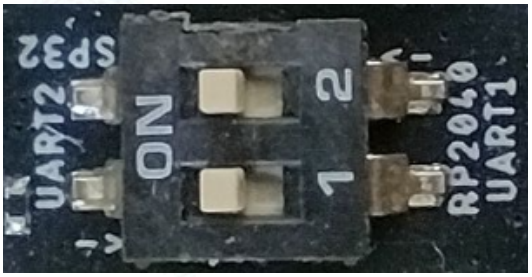


Baud = 10
Bit rate = 20 bps

Diagrama de conexión



Nota: Deberás ubicar el DIP Switch y colocar los botones selectores en las posiciones que se muestran en la imagen



De esta manera estaremos conectando los pines de TX y RX de ambos microcontroladores entre sí para establecer la comunicación UART.

Códigos de funcionamiento

Para esta lección tendremos dos programas, uno para el ESP32 y otro para el RP2040

Código para el RP2040

```
//Define el pin del botón (GPIO16, que corresponde a D12 en la DUAL MCU ONE)
#define BUTTON_PIN 16
// Variables para el antirrebote (debounce) del botón:
bool lastButtonState = HIGH;// Estado anterior del botón (HIGH por INPUT_PULLUP)
bool ledState = false;     // Estado virtual del LED (ON/OFF)
unsigned long lastDebounceTime = 0;  // Último tiempo de detección
const unsigned long debounceDelay = 300;  // Tiempo de espera para evitar rebotes (ms)
void setup() {
  Serial.begin(9600); // Inicia UART0 para depuración (Monitor Serial)
  Serial1.begin(9600); // Inicia UART1 (GPIO0=TX, GPIO1=RX por defecto)
  pinMode(BUTTON_PIN, INPUT_PULLUP);  // Botón pull-up interna
  Serial.println("RP2040 Transmisor listo");}
void loop() {
  // Lee el estado actual del botón
  bool currentButtonState = digitalRead(BUTTON_PIN);
  //  Recepción de mensajes por UART (opcional, para confirmaciones)
  if(Serial1.available() > 0) {
    String mensaje = Serial1.readStringUntil('\n');  // Lee hasta salto de línea
    mensaje.trim();  // Elimina espacios y caracteres especiales
    Serial.print("ACK: ");  // Confirma recepción (ej: "ACK: RECIBIDO")
    Serial.println(mensaje);}
//-Lógica del botón-Detecta flanco descendente(botón presionado,pasa de HIGH a LOW)
```

```
if (currentButtonState == LOW && lastButtonState == HIGH) {
    // Verifica que haya pasado el tiempo de debounce
    if (millis() - lastDebounceTime > debounceDelay) {
      ledState = !ledState; // Cambia el estado del LED (toggle)
      // Envía "ON" o "OFF" por UART según el estado
      if (ledState) {
        Serial1.println("ON"); // Envía "ON" por UART1 (GPIO0)
        Serial.println("Enviado: ON");// Confirma por Monitor Serial
      } else {
        Serial1.println("OFF");  // Envía "OFF" por UART1 (GPIO0)
        Serial.println("Enviado: OFF");
      }
 lastDebounceTime = millis();  // Reinicia el temporizador de debounce
    }
  }
  // Actualiza el estado anterior del botón
  lastButtonState = currentButtonState;
}
```

Código para el ESP32

```
// Definición del pin del LED (GPIO25, común en muchas placas ESP32)
#define LED 25
// Configuración del UART2 (HardwareSerial) para comunicación serial
HardwareSerial mySerial(2);
void setup() {// Inicializa el puerto serial USB (para depuración en el
Monitor Serial)
  Serial.begin(9600);
// Inicializa el UART2 con:
// - Baud rate: 9600
//  - Configuración: 8 bits de datos, sin paridad, 1 bit de stop
(SERIAL_8N1)
// - Pines: RX = GPIO16, TX = GPIO17 (pueden cambiarse según necesidades)
  mySerial.begin(9600, SERIAL_8N1, 16, 17);
  // Configura el pin del LED como salida
  pinMode(LED, OUTPUT);
    // Apaga el LED inicialmente (HIGH porque es común en lógica de
pull-up)
  digitalWrite(LED, HIGH);
  // Mensaje inicial para confirmar que el programa inició
  Serial.println("ESP32 UART Receiver");}
```

```
void loop() {
  // Verifica si hay datos disponibles en el UART2
  if (mySerial.available()) {
    // Lee el mensaje hasta encontrar un salto de línea ('\n')
    String message = mySerial.readStringUntil('\n');

      // Elimina espacios en blanco al inicio/final del mensaje (para
limpieza)
 message.trim();   // Muestra el mensaje recibido en el Monitor Serial
(USB)
 Serial.println("Received: " + message);
    // --- Lógica de control del LED ---
    if (message == "ON") {
digitalWrite(LED, LOW);  // Enciende el LED (LOW activa, común en lógica
pull-up)
Serial.println("LED ENCENDIDO DEBIDO A MENSAJE EN UART");
mySerial.write("DATO  RECIBIDO");     //  Envía  confirmación  por  UART
(opcional)
    }
    else if (message == "OFF") {
      digitalWrite(LED, HIGH);  // Apaga el LED
      Serial.println("LED APAGADO DEBIDO A MENSAJE EN UART");
    }
    else {
      // Mensaje si el comando no es "ON" ni "OFF"
      Serial.println("Comando no reconocido");
    }
  }
}
```

En este par de programas presta atención a la manera en que se realiza la comunicación mediante el protocolo UART las cuales son líneas específicas que requiere cada microcontrolador

Para inicializar los periféricos UART en el RP2040 utilizamos las siguientes sentencias dentro del void setup:

```
  Serial1.begin(9600);// Inicia UART1 (GPIO0=TX, GPIO1=RX por defecto)
```

Mientras que la misma acción para el ESP32 requiere de la creación de un objeto que llamamos “mySerial”

```
HardwareSerial mySerial(2); // Configuración del UART2 (HardwareSerial)
para comunicación serial
```

Y dentro del void setup inicializamos el objeto médiate el siguiente constructor:

```
  mySerial.begin(9600, SERIAL_8N1, 16, 17);
// Inicializa el UART2 con:
  // - Baud rate: 9600
   // - Configuración: 8 bits de datos, sin paridad, 1 bit de stop (SERIAL_8N1)
    // - Pines: RX = GPIO16, TX = GPIO17 (pueden cambiarse según necesidades)
```

Para enviar (escribir) algún dato por UART se realiza con las siguientes sentencias:

```
mySerial.write("DATO RECIBIDO"); // El ESP32 envia una cadena de texto al RP2040
Serial1.println("ON");  // EL RP2040 envía una cadena de texto al ESP32
```

Como puedes notar, la lógica es la misma para ambos microcontroladores, pero cada uno tiene una sintaxis propia, por lo que tendrás que revisar los códigos para notar las diferencias 🙂

Por otra parte, si abres el monitor serial que le corresponde al RP2040 y al ESP32 obtendrás los siguientes mensajes al presionar el botón, además de observar el encendido y apagado del LED integrado a la UNIT DUAL MCU ONE en el GPIO25 del ESP32

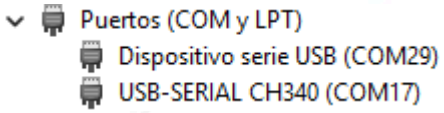
Cuando presionas el botón el RP2040 envía la palabra “ON” y el ESP32 le contesta con las palabras “DATO RECIBIDO” y al volver a presionar el botón se envía la palabra “OFF”

```
Enviado: ON
ACK: DATO RECIBIDO
Enviado: OFF
Enviado: ON
ACK: DATO RECIBIDO
```

El ESP32 recibe desde el RP2040 la palabra ON y confirma en el monitor serial mediante la cadena “LED ENCENDIDO DEBIDO A MENSAJE EN UART” y de manera análoga cuando se recibe la palabra “OFF”

```
Received: ON
LED ENCENDIDO DEBIDO A MENSAJE EN UART
Received: OFF
LED APAGADO DEBIDO A MENSAJE EN UART
Received: ON
LED ENCENDIDO DEBIDO A MENSAJE EN UART
```

Recuerda que para que obtengas ambas salidas a la vez en el monitor serial de Arduino debes abrir dos ventanas de tu Arduino IDE donde una reciba los datos desde el ESP32 y en otra del RP2040. Para ello debes identificar en qué puerto COM se encuentra conectado cada microcontrolador, por lo que deberás abrir tu administrador de dispositivos donde podrás ver una lista como la siguiente:



Donde el dispositivo que se refiere al CH340 siempre será el ESP32 y por descarte, el otro dispositivo es el RP2040. Lo anterior lo podemos comprobar revisando el diagrama de bloques de la tarjeta UNIT DUAL MCU ONE. Donde los datos USB (USB DATA) pasan primero por el CH340 antes de llegar al ESP32 mientras que pasan directamente hacia el RP2040

Este diagrama y mucha más información de la UNIT DUAL MCU ONE la podrás encontrar en el [GitHub]

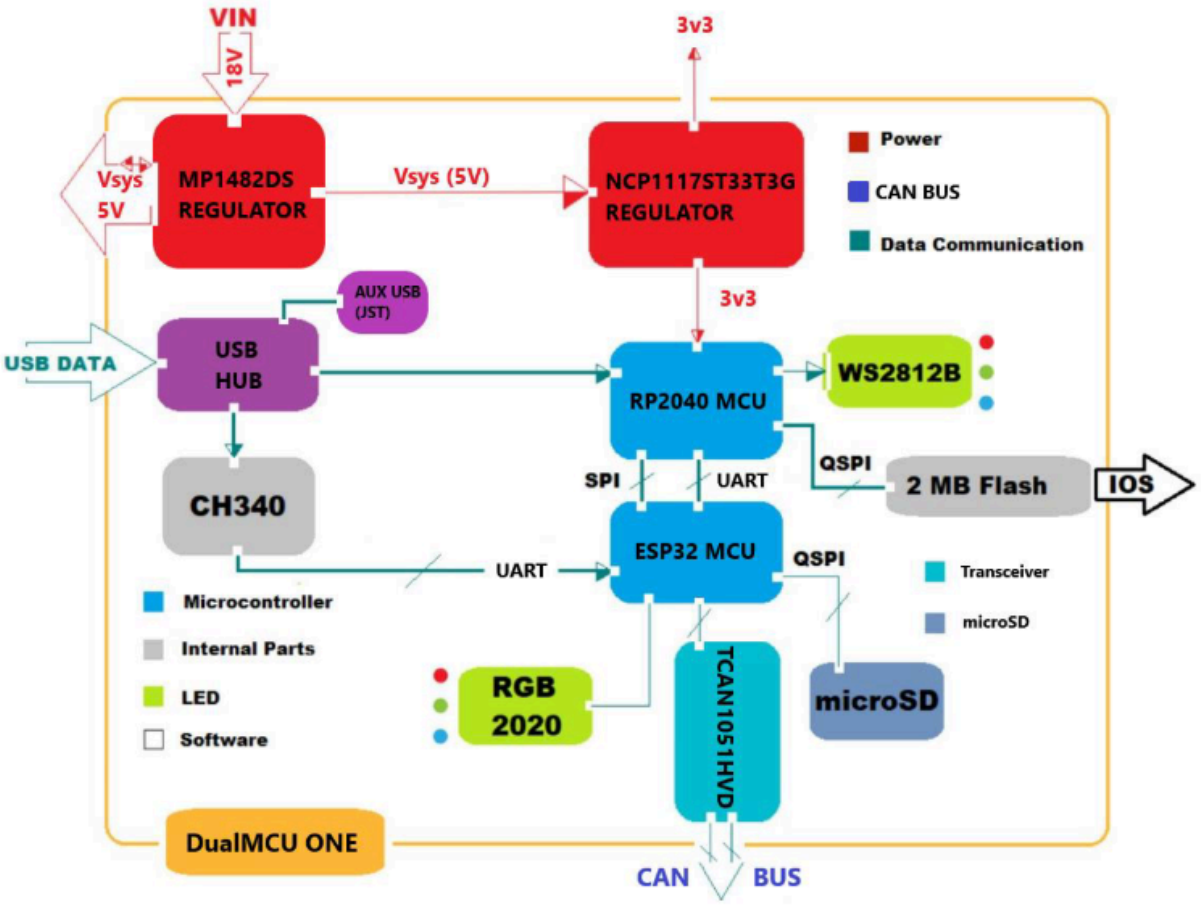
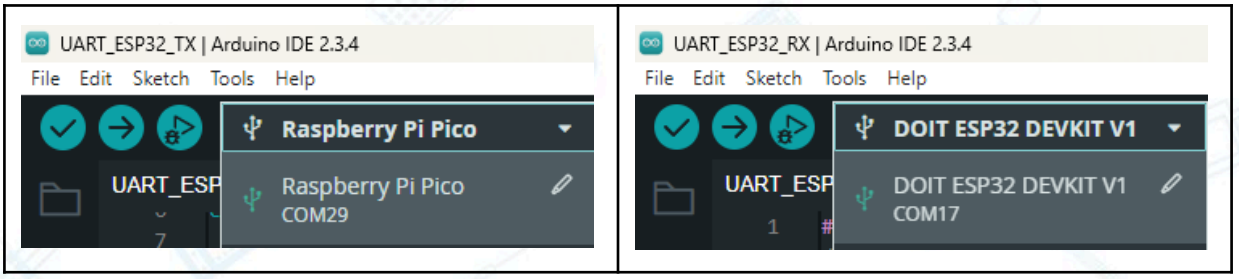


Diagrama de Bloques de DualMCU-ONE RP2040 + ESP32

Por último, configura cada ventana de tu Arduino IDE para recibir los datos de cada microcontrolador de acuerdo con el puerto COM previamente identificado.



Lección 7 Relé y servidor web

En esta lección implementaras el control de un relevador mediante un servidor web alojado en el ESP32. Para este fin haremos uso de ambos microcontroladores para demostrar la versatilidad de la UNIT DUAL MCU ONE en la que podemos dividir la ejecución de las tareas de comunicación y control al ESP32 y al RP2040 respectivamente, los cuales entre ellos se envían datos mediante UART.

Materiales

1.- Resistencias 10k, 22k, 100 todas a 1/4w
2.- Transistor BC547 x1
3.- Diodo 1N4007 x1
4.- LED´s 5mm x5
5.- Relevador SPDT 5V x1
9.- Protoboard x1
10.- DUAL MCU ONE x1
11.- Cables dupont Macho - Macho
12.- 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: para controlar dispositivos de alta potencia con señales de baja potencia.
- Protección de circuitos: como parte de sistemas de seguridad en equipos eléctricos.
- Controles de maquinaria y sistemas de control.

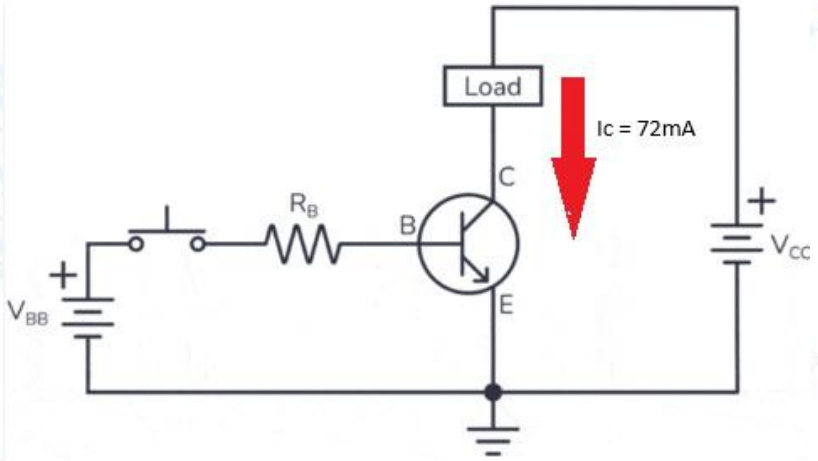
BJT como switch electrónico

Para lograr la activación del relevador tenemos que hacer fluir una corriente a través de la bobina del mismo. Dicha corriente es de 72mA como se observa en la imagen

Esto resulta problemático debido a que no debemos intentar conectar directamente el relevador a la tarjeta Mega 2560 ni a ningún microcontrolador o circuito integrado que no disponga de la suficiente corriente a la salida para manejar al relé.

Es por esto que utilizaremos un transistor BC547 el cual es del tipo NPN y puede manejar corrientes de 100mA como máximo.

Por lo tanto, el problema del manejo de la corriente se reduce al cálculo de una resistencia de base (Rb) que le permita al transistor lidiar con los 72mA (Icq) que necesita el relevador (Load).



Procedimiento

1.- Identificación de datos

Del circuito mostrado en la figura superior sabemos que la fuente de tensión VBB refiere al Mega 2560 y los 5V que esta puede dar en cada una de sus salidas digitales, por lo tanto,

$$V_{BB} = 5\,\mathrm{V}$$

Load hace referencia a la carga que se va a manejar, en este caso el relevador con los 72mA de corriente que tienen que transitar por el colector del transistor, por consiguiente

$$I_{CQ} = 72\,\mathrm{mA}$$

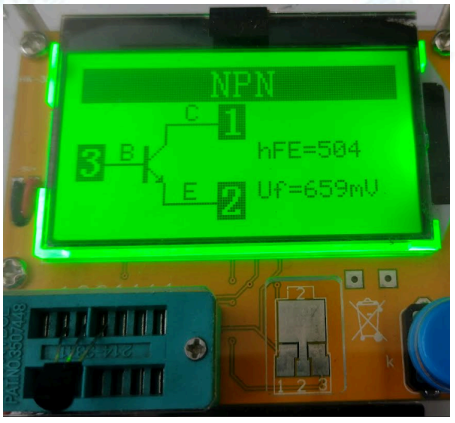
También necesitamos el dato de la tensión entre la base y el emisor que se produce cuando el transistor entra en conducción, que típicamente es de 0.7v, por ello:

$$V_{BE} = 0.7\,\mathrm{V}$$

Por último, necesitamos el dato de la ganancia de corriente del transistor BC547 en CD, identificada con los símbolos “hFe” o “β” que puedes obtener de la hoja de datos o midiendo directamente el transistor con un probador de componentes o un multímetro con la función hFe. El hFe que se obtuvo para el presente ejemplo fue de 504, por lo tanto:

$$h_{FE} = 504$$

Tal y como se observa en la imagen donde se obtuvo la medición



2.- Cálculo de la corriente de base

Reemplazando con los datos antes reunidos, calculamos la corriente de base por medio de la siguiente fórmula:

$$I_{BQ} = \frac{I_{CQ}}{h_{FE}}$$

por lo que tenemos:

$$I_{BQ} = \frac{72\,\text{mA}}{504} = 142\,\mu\text{A}$$

Esta corriente será la que la DUAL ONE tendrá que aportar para que el circuito funcione y como ves, se trata de un valor 500 veces menor que los 72 mA que el transistor va a manejar, así evitamos dañar al microcontrolador

3.- Cálculo de la resistencia de base

Finalmente, todos los datos hasta ahora recopilados y calculados los reemplazamos en la siguiente fórmula para obtener el valor de resistencia que nos garantice un correcto funcionamiento, además de seguro, ya que no existirán sobrecorrientes que puedan provocar daños al equipo.

$$R_B = \frac{V_{BB} - V_{BE}}{I_{BQ}}$$

Sustituyendo datos tendremos:

$$R_b = \frac{V_{BB} - V_{BE}}{I_{BQ}} = \frac{5V - 0.7V}{142\mu A} = 29,577\,\Omega$$

Dado que este valor de resistencia no es comercial, siempre utilizaremos el valor inferior más próximo este caso será una resistencia de base

$$R_b = 27\,k\Omega$$

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.

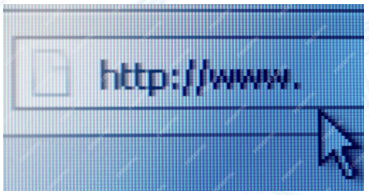
Cuando un navegador hace una solicitud HTTP a un servidor, la solicitud incluye varias partes importantes:

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.

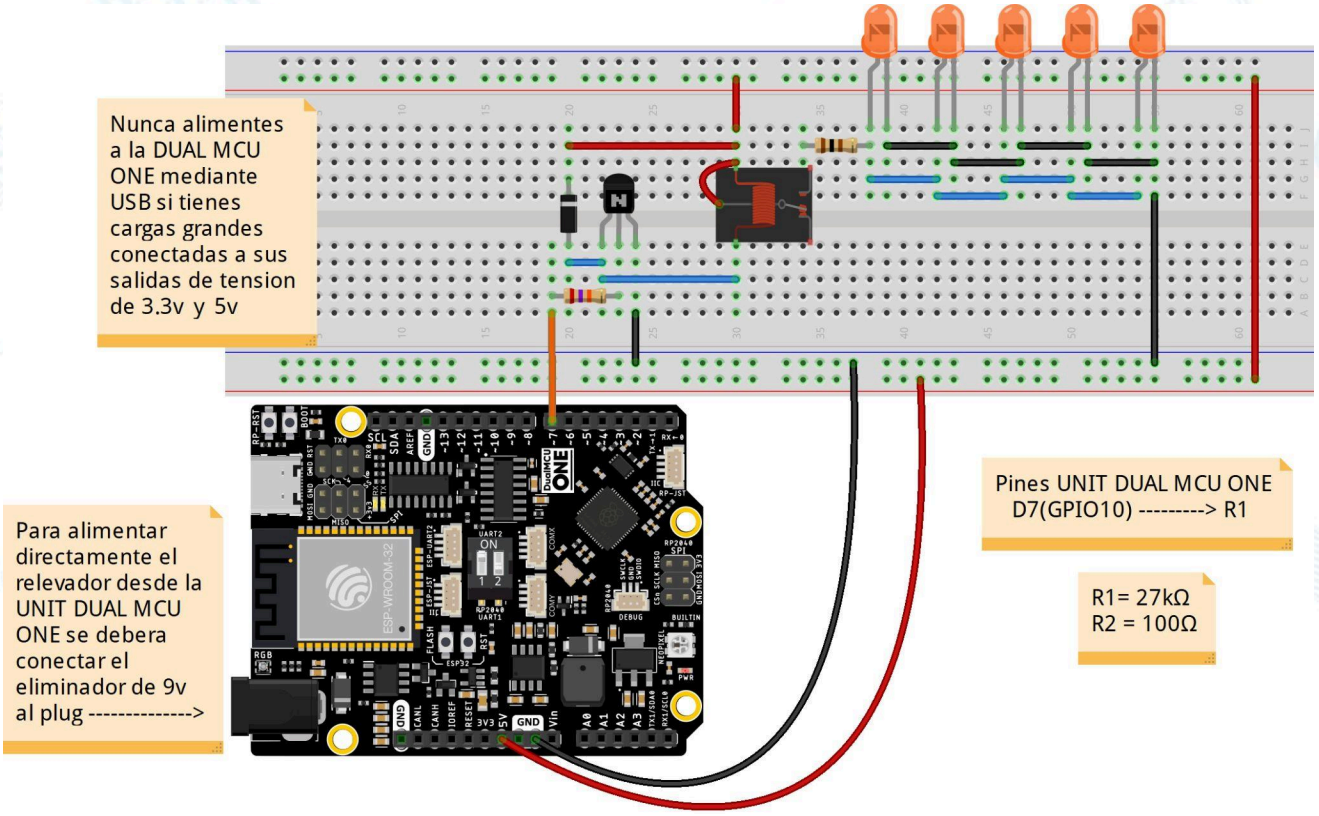
- El ESP32 se conecta a la red Wi-Fi y obtiene una dirección IP local (esto se muestra en el monitor serial).

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

Tal y como se observa a continuación, podemos alimentar el relevador y la carga que consiste de 5 LEDs, directamente desde la tarjeta UNIT DUAL MCU ONE la cual soporta hasta 2 A de corriente directa; misma que proviene del eliminador de 9 v que conectamos al plug de la tarjeta.



Códigos de funcionamiento

Como parte de la división de tareas, el RP2040 se encarga de la interacción con el exterior mediante el relevador y el ESP32 de la comunicación y el alojamiento del servidor web que recibe las órdenes para encender y apagar el relevador mediante métodos HTTP.

Código para el RP2040

En este programa te percatarás de su similitud con los vistos en la lección pasada, esto debido a que el RP2040 se encarga de enviar y recibir datos desde el ESP32 por medio de UART para activar o desactivar la GPIO10 (D7) conectada al circuito que controla el relevador.

```
// Define el pin del relevador (GPIO10 en el RP2040, corresponde al pin D7 en la DUAL MCU ONE)
#define RELE 10
void setup() {
  Serial.begin(9600);  // Inicia la comunicación por USB (UART0) para el monitor serial (depuración)
   Serial1.begin(9600); // Inicia UART1 (TX = GPIO0, RX = GPIO1), usado para recibir datos del ESP32
  pinMode(RELE, OUTPUT);//Configura el pin del relevador como salida
  Serial.println("RP2040 UART listo");// Mensaje de confirmación
}
void loop() {  // Verifica si hay datos disponibles para leer desde el ESP32 (vía UART1)
  if (Serial1.available()) {// Lee la cadena de texto recibida hasta el salto de línea '\n'
     String message = Serial1.readStringUntil('\n'); // Elimina espacios en blanco o caracteres no deseados al inicio/final
    message.trim(); // Muestra el mensaje recibido en el monitor serial
    Serial.println("Received: " + message);
    // ---------------------------
    // Lógica para controlar el RELE
    // ---------------------------
    // Si el mensaje es "ON", activa el relevador
    if (message == "ON") {
    digitalWrite(RELE, HIGH);  // Enciende el relevador (activa salida)
   Serial.println("RELE ACTIVADO DEBIDO A MENSAJE EN UART");  // Mensaje de confirmación
// Envía respuesta de confirmación por UART de regreso al ESP32
    Serial1.write("DATO RECIBIDO : RELE desactivado"); }
    // Si el mensaje es "OFF", apaga el relevador
    else if (message == "OFF") {
      digitalWrite(RELE, LOW); // Apaga el relevador (desactiva salida)
      Serial.println("RELE APAGADO DEBIDO A MENSAJE EN UART");
      // Envía respuesta de confirmación por UART de regreso al ESP32
      Serial1.write("DATO RECIBIDO : RELE ACTIVADO"); }
 // Si se recibe un mensaje diferente a "ON" u "OFF"
    else {
       Serial.println("Comando no reconocido"); // Mensaje de advertencia en consola
    }
 }
}
```

Código para el ESP32

El ESP32 tiene la tarea de alojar el servidor web con una página escrita en HTML, de administrar la conexión de los clientes HTTP y en función de la interacción cliente-servidor, enviar y recibir cadenas de texto mediante UART con el RP2040

```
#include <WiFi.h>  // Librería para manejo de WiFi en ESP32
// Variable para guardar el estado del relevador
String p = "off";
// Credenciales de la red WiFi
const char *ssid = "Escribe aquí el nombre de tu red";
const char *password = "Escribe aquí la contraseña de dicha red";
// Crear un servidor en el puerto 80 (HTTP)
WiFiServer server(80);
// Se declara el segundo puerto serial UART (UART2)
// Este se usará para comunicar con el microcontrolador RP2040
HardwareSerial mySerial(2);
void setup() {
Serial.begin(9600);//Inicia comunicación serial para depuración por USB
  Serial.println();
   Serial.println("Configurando WiFi...");// Conexión al WiFi con las
credenciales dadas
  WiFi.begin(ssid, password);
 // Espera hasta que el ESP32 se conecte al WiFi
  Serial.print("Conectandome");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Una vez conectado, se muestra la IP asignada al ESP32
  Serial.println();
  Serial.print("Conectado, IP: ");
  Serial.println(WiFi.localIP());
  // Inicia el servidor web
  server.begin();
  Serial.println("Servidor iniciado");
  // Inicializa UART2 (HardwareSerial 2)
  // Velocidad 9600 baudios, formato 8N1, pines RX = 16, TX = 17
  mySerial.begin(9600, SERIAL_8N1, 16, 17);
  Serial.println("Transmisor ESP32 por UART a RP2040");
}
void loop() {
  // Espera por un cliente HTTP
  WiFiClient client = server.available();
  if (client) {
    Serial.println("Nuevo cliente.");
    String currentLine = "";
    bool comandoEnviado = false;// Flag para asegurarse que solo se envía
una vez el comando
    // Mientras el cliente esté conectado
    while (client.connected()) {
      if (client.available()) {
```

```
 char c = client.read();  // Lee un carácter del cliente
        Serial.write(c);          // Lo muestra por el monitor serial
//Si se detecta un salto de línea, se interpreta que terminó una petición
HTTP
        if (c == '\n') {
          if (currentLine.length() == 0) {
            // Envío de cabecera HTTP
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
            // Contenido HTML de la página
            client.println("<!DOCTYPE html><html>");
client.println("<head><metaname=\"viewport\"content=\"width=device-width,
initial-scale=1\">");
client.println("<link          rel=\"icon\"          href=\"data:,\">");
client.println("<style>html{font-family: Helvetica; text-align: center;}
.button{border:none;  color:white;  padding:15px  32px;  font-size:16px;
margin:4px;      cursor:pointer;}      .button1{background-color:#4CAF50;}
.button2{background-color:#808080;}</style>");
client.println("</head><body><h1>Control de Relevador ESP32</h1>");
// Botón cambia según estado del relevador
if (p == "off") {
client.println("<button                class='button                button1'
onclick=\"location.href='/r1=ON'\">ENCENDER</button>");
 } else {
          client.println("<button          class='button          button2'
onclick=\"location.href='/r1=OFF'\">APAGAR</button>");
           }
 client.println("</body></html>");
 client.println();  // Fin del contenido HTML
 break;    // Sal del bucle después de enviar la respuesta
 } else {
 currentLine = ""; // Limpia la línea actual si no es el final
}
} else if (c != '\r') {
currentLine += c;  // Acumula caracteres para formar la línea completa
}// Si se detecta la petición para encender el relevador
 if (!comandoEnviado && currentLine.indexOf("GET /r1=ON") != -1) {
   p = "on";  // Actualiza estado
   mySerial.write("ON");  // Envia comando ON por UART al RP2040
   comandoEnviado = true;  // Marca como enviado
   // Si hay datos disponibles desde el RP2040, los lee y muestra
   if (mySerial.available() > 0) {
   String mensaje = mySerial.readStringUntil('\n');
    mensaje.trim();  // Elimina espacios o saltos innecesarios
    Serial.println("\n ACK: ");
    Serial.print(mensaje);
    Serial.print("\n"); }}
 // Si se detecta la petición para apagar el relevador
if (!comandoEnviado && currentLine.indexOf("GET /r1=OFF") != -1) {
          p = "off";  // Actualiza estado
```

```
 mySerial.write("OFF");  // Envia comando OFF por UART al RP2040
          comandoEnviado = true;  // Marca como enviado
          // Si hay datos disponibles desde el RP2040, los lee y muestra
           if (mySerial.available() > 0) {
String mensaje2 = mySerial.readStringUntil('\n');
mensaje2.trim();
Serial.println("\n ACK: ");
Serial.print(mensaje2);
Serial.print("\n");    } } } }
    // Espera mínima antes de cerrar la conexión
    delay(1);
    client.stop();  // Termina conexión con el cliente
    Serial.println("Cliente desconectado");
  }}
```

Tras cargar el programa, abre el monitor serial y si el ESP32 estableció conexión correctamente deberá mostrar una dirección IPV4 como la siguiente.

```
Configurando WiFi...
Conectandome...
Conectado, IP: 192.168.3.135
Servidor iniciado
Transmisor ESP32 por UART a RP2040
```

Dicha IP la ingresamos en el navegador web de nuestra preferencia y obtendrá la siguiente página web, alojada en el ESP32 donde podrá activar y desactivar el relevador al presionar el botón que aparece en la página.

Control de Relevador ESP32

ENCENDER

Lección 8 Control por mensajería en WhatsApp

En esta lección aprenderás a implementar el control para una salida digital mediante mensajes de texto desde Whatsapp, además de la visualización de dichos mensajes mediante una OLED utilizando ambos microcontroladores integrados en el UNIT DUAL MCU ONE.

Materiales

1.- DUAL MCU ONE x1
2.- OLED SSD1306 x1
3.- Cables Dupont Macho - Hembra

Conocimientos previos

OLED
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.

Se usan mucho en teléfonos inteligentes, televisores modernos, relojes inteligentes y, por supuesto, en proyectos de electrónica como la pantalla de 128x64 . Es una tecnología bastante avanzada y visualmente muy atractiva, especialmente cuando se trata de mostrar gráficos o texto en un dispositivo pequeño.

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:

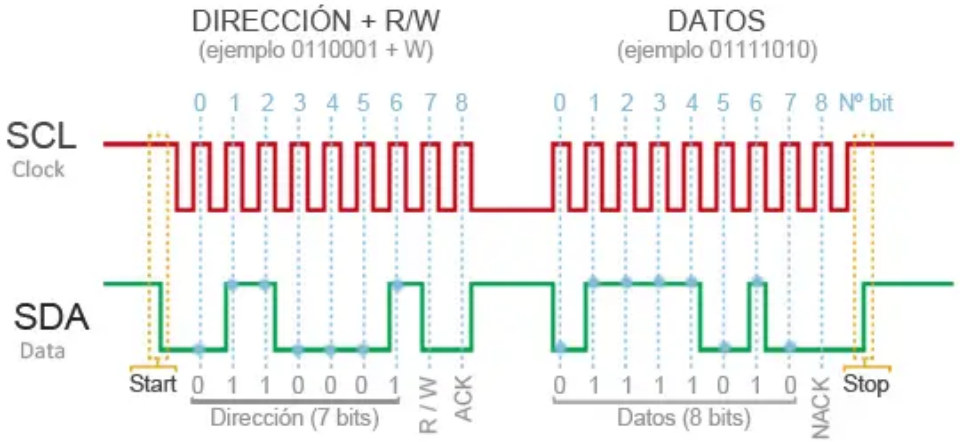
ACK (0): Acknowledgement → "He recibido correctamente el byte."
NACK (1): No Acknowledge → "No he recibido correctamente el byte o no quiero más datos."

5. Datos (8 bits por cada byte)

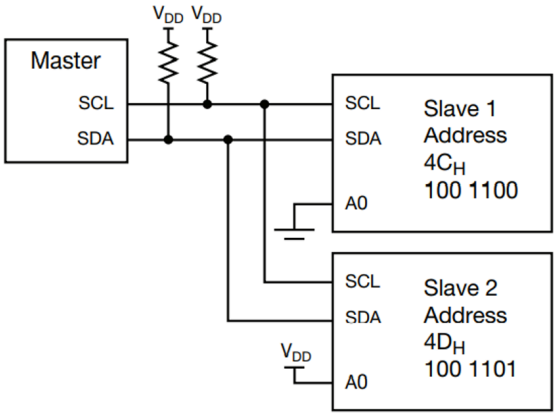
Los datos siempre se transmiten en **bloques de 8 bits** (1 byte).
Después de cada byte, el receptor envía otro bit ACK/NACK.

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.

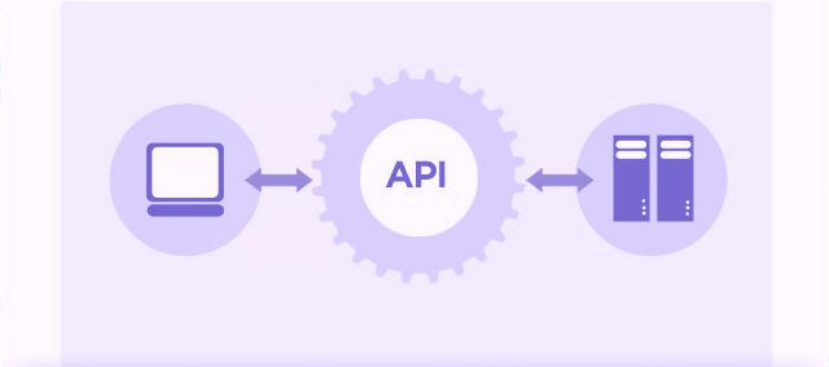


Los dos hilos del BUS interfaz de comunicación I2C PIC son líneas de colector abierto, por lo que se deben utilizar unas resistencias PULL UP (resistencias conectadas a positivo) para asegurar un nivel alto cuando NO hay dispositivos conectados al BUS I2C.



API

Una API (siglas de Application Programming Interface, en inglés) es un conjunto de reglas y definiciones que permite que diferentes programas o sistemas se comuniquen entre sí. Podrías imaginarla como un puente o un traductor que facilita la interacción entre dos piezas de software que, por sí solas, no se entenderían directamente.



Cuando los desarrolladores crean un sistema, una aplicación o una plataforma, pueden diseñar para que otros programas puedan interactuar con ella sin necesidad de conocer cómo está hecha por dentro. La API se encarga de exponer sólo lo necesario: las funciones disponibles, los datos que se pueden enviar o recibir, y cómo deben organizarse esos datos.

Es como un menú en un restaurante: tú no necesitas saber cómo se cocina un platillo, solo eliges lo que quieres y das la orden. La cocina (el software interno) se encarga del resto. En este sentido, la API es ese menú y también la forma en que haces el pedido.

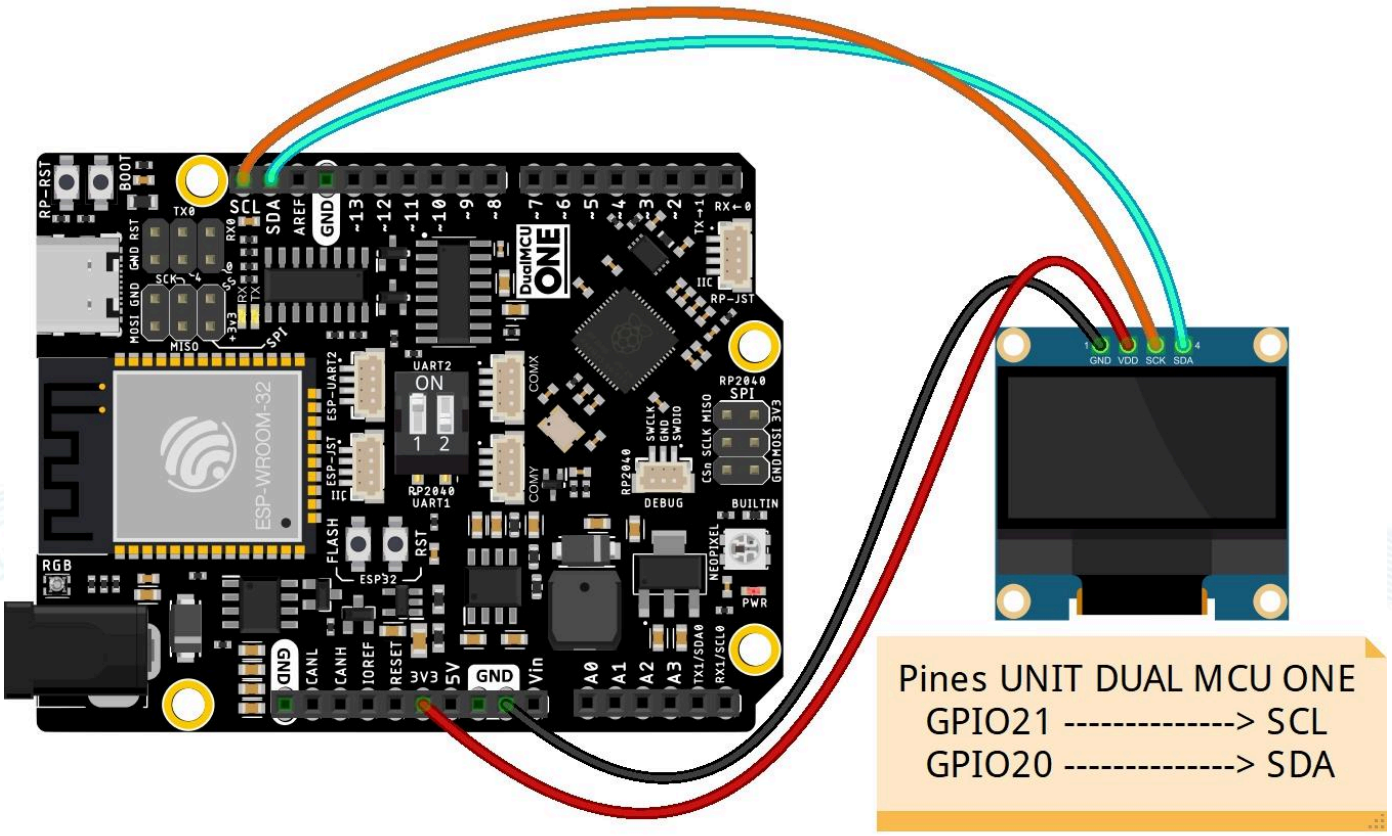
Por ejemplo, cuando usas una app para consultar el clima, esa app no tiene todos los datos meteorológicos por sí sola. En su lugar, se comunica con una API de un servicio meteorológico que sí posee esa información. La aplicación le pregunta a la API: “¿Cuál es el clima actual en esta ciudad?”, y la API devuelve una respuesta estructurada (como un mensaje en formato JSON), con la temperatura, la humedad, etc. La aplicación solo se encarga de mostrarlo al usuario de forma visual y entendible.

Las APIs no solo sirven para consultar información. También permiten realizar acciones: enviar un mensaje, hacer una compra, validar un pago, iniciar sesión con una cuenta de Google, entre muchas otras. En todos esos casos, tu aplicación hace una solicitud a través de la API del servicio externo, y este le responde con un resultado o ejecuta una acción. Así, puedes integrar funciones complejas en tus programas sin tener que programarlas desde cero.

En la práctica moderna, muchas APIs se ofrecen a través de internet, a lo que se llama comúnmente "APIs web" o "REST APIs". Esto ha permitido que diferentes servicios —aunque estén en servidores distintos y escritos en distintos lenguajes— puedan trabajar juntos de forma ordenada. Es la base del funcionamiento de muchas aplicaciones modernas, desde redes sociales hasta tiendas en línea y servicios bancarios.

En resumen, una API es la forma en la que los programas se hablan entre sí. Define las palabras, las frases, y el tono del diálogo, permitiendo que múltiples sistemas trabajen juntos sin conflictos, con seguridad, y de forma eficiente.

Diagrama de conexiones

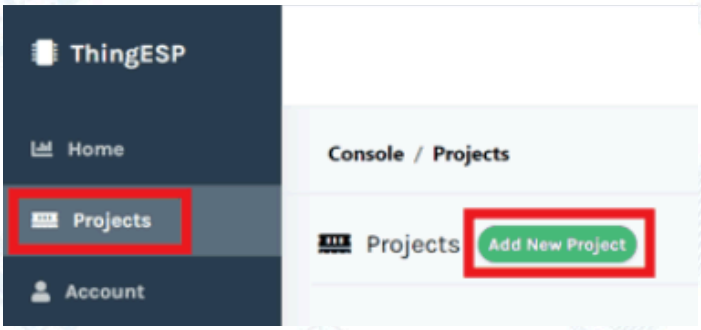


Configuración Previa

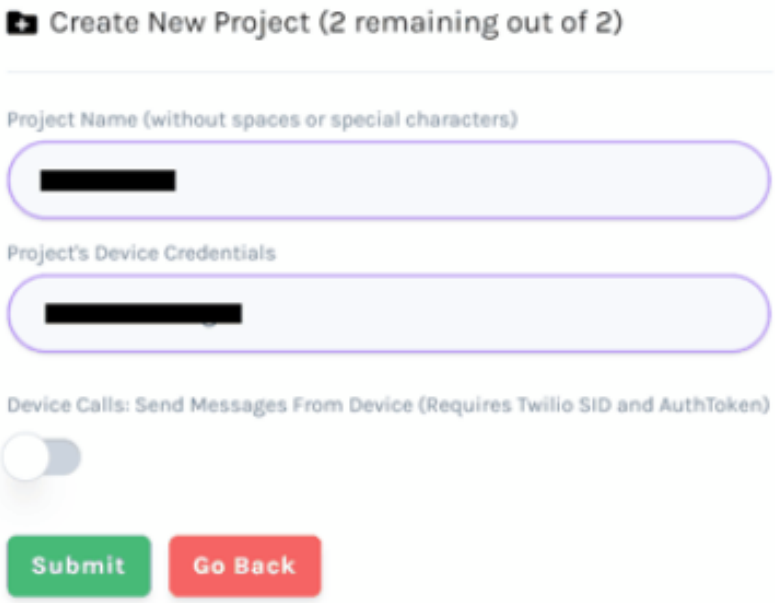
Antes de continuar con los programas para el RP2040 y para el ESP32 debemos crear un proyecto y configurarlo en las plataformas de la API ThingESP y Twilio

Comencemos con **ThingESP**

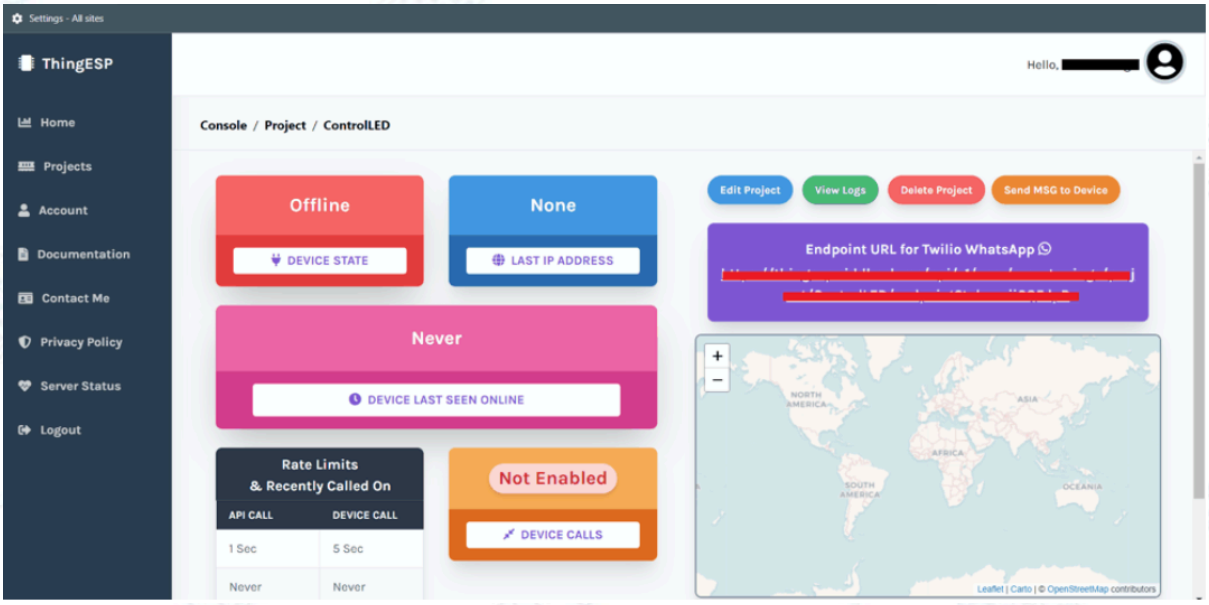
1.- Entra al siguiente enlace y crea una cuenta **https://thingesp.siddhesh.me/#/**
2.- Click en **Projects** y después en **Add New Project**



3.- Establece un nombre y contraseña para tu proyecto y haz click en **Submit**. Ten presentes estos datos al igual que tu nombre de usuario, ya que, serán usados después en el programa para el ESP32 en Arduino IDE 👀

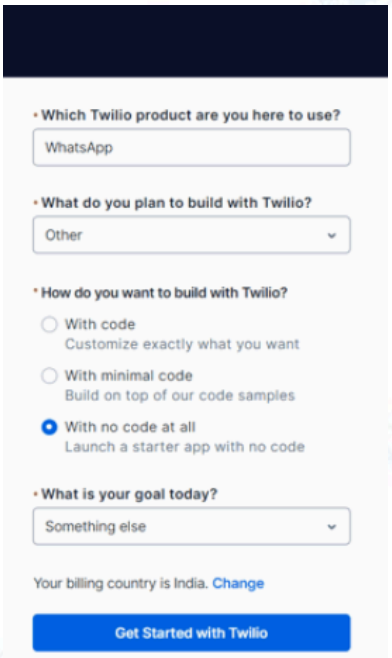


4.- Ya creado el proyecto ThingESP te provee de un URL el cual copia y guárdalo que después será usado en la plataforma de Twilio 👀

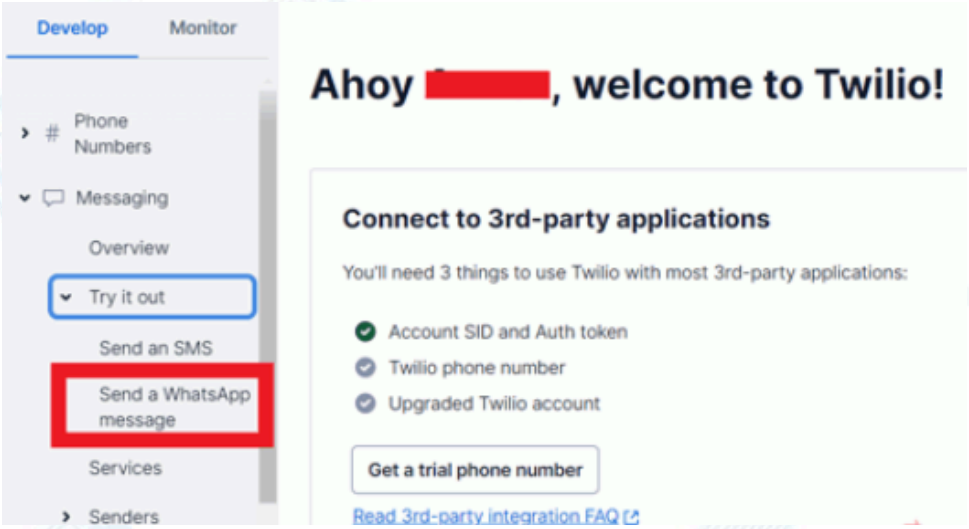


Continuemos con **Twilio**

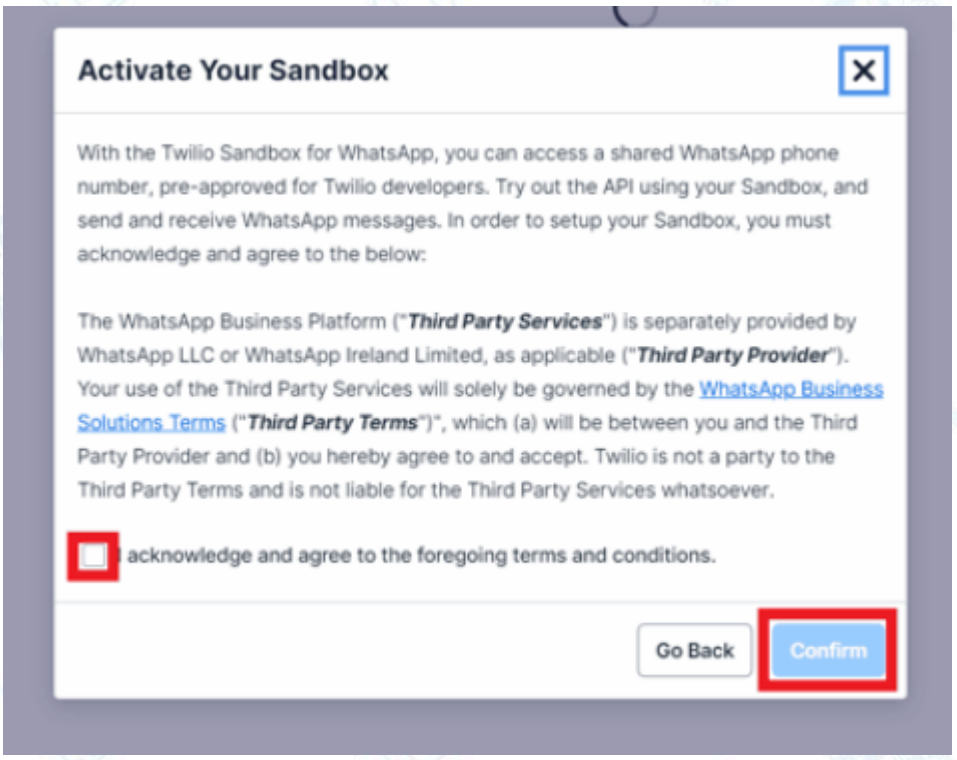
1.- Ingresa al siguiente enlace https://www.twilio.com/en-us y crea una cuenta
2.- Al crear la cuenta te pedirá tu número de teléfono y eventualmente llegaras a una ventana donde deberás colocar los campos como se muestra en la siguiente imagen:



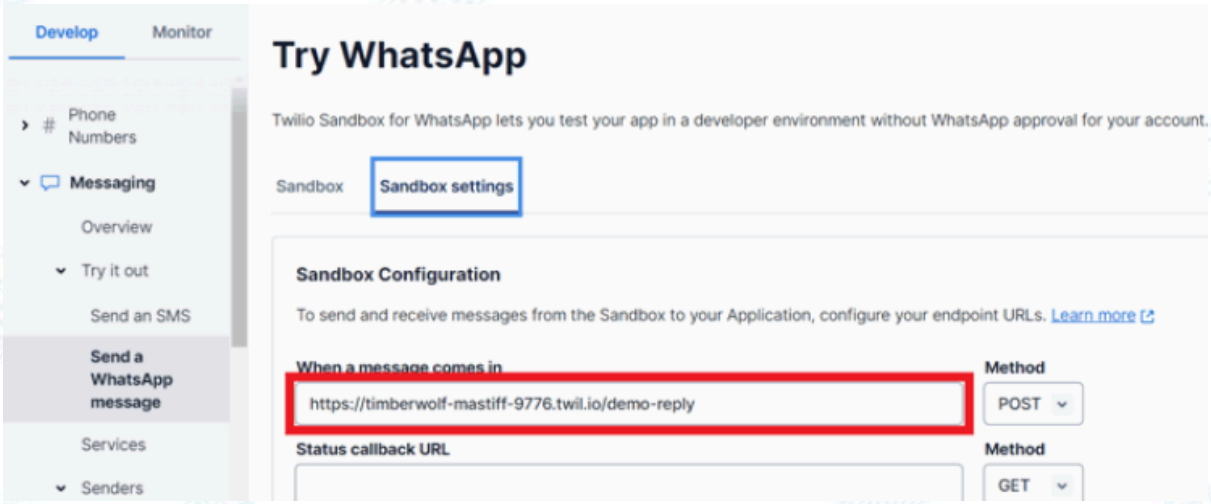
3.- Una vez creada tu cuenta, ve a la parte izquierda de la pantalla, click en **Try it out** y en **Send a WhatsApp message**



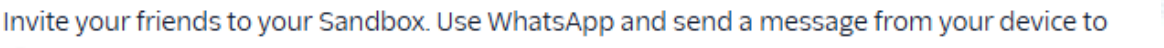
4.- Marca la casilla y click en **Confirm**



5.- Ahora ve a la pestaña **Sandbox Settings** y pega el URL que **ThingESP** te dio y da click en **Save**

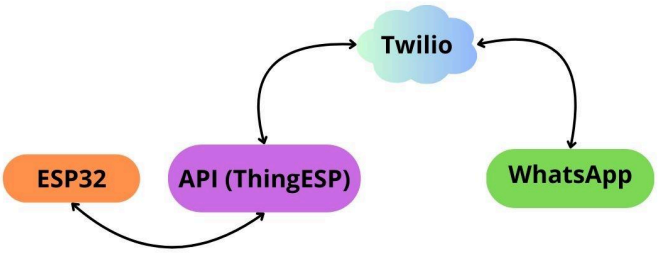


6.- Scrollea hasta abajo y ubica el número de teléfono que Twilio te provee, guárdalo en tus contactos y en WhatsApp envía el mensaje de activación que en este caso fue **join contain-grain**



Código de funcionamiento

El ESP32 interactúa con una API (ThingESP) que enlaza al dicho microcontrolador con una plataforma de comunicaciones en la nube (Twilio) que integra las funciones de mensajería con WhatsApp.



Para hacer funcionar el siguiente programa, deberás ingresar el usuario, nombre del proyecto y contraseña del proyecto que configuraste en la plataforma ThingESP, además de las credenciales de tu red WiFi

```
// Librerías necesarias para conectividad WiFi y para usar la plataforma ThingESP
#include <WiFi.h>
#include <ThingESP.h>
// Creamos un objeto de tipo ThingESP32 los datos ingresados en ThingESP
// Estos datos deben coincidir con los configurados en la plataforma ThingESP
ThingESP32 thing("usuario", "nombre del proyecto", "contraseña del proyecto");
// Creamos una instancia del puerto serial 2 para comunicar al ESP32 con el RP2040 por UART
// Puedes usar cualquier par de pines compatibles del ESP32 para esto
HardwareSerial mensaje_al_RP2040(2);  // UART2
// Variables para control de tiempo (no se usan activamente en este código, pero pueden ser útiles después)
unsigned long previousMillis = 0;
const long INTERVAL = 6000; // 6 segundos
// Función de configuración que se ejecuta una sola vez al encender el ESP32
void setup()
{
```

```
  // Iniciamos el puerto serial principal (USB) para depuración con el monitor serie
  Serial.begin(115200);
   // Configuramos el nombre y la contraseña de la red WiFi a la que el ESP32 debe conectarse
  thing.SetWiFi("Nombre de la red WiFi", "contraseña de la red");
  // Inicializamos la conexión del ESP32 con la plataforma ThingESP
  thing.initDevice();
  // Inicializamos la comunicación UART con el RP2040 a través del puerto serial 2
  // Parámetros:
  // - Velocidad de 115200 baudios
  // - Formato 8N1 (8 bits de datos, sin paridad, 1 bit de parada)
  // - RX en el pin GPIO16, TX en el pin GPIO17
  mensaje_al_RP2040.begin(115200, SERIAL_8N1, 16, 17);
}
// Función que maneja los comandos recibidos desde la plataforma ThingESP
// Recibe una cadena ("query") con el comando enviado desde la app o web
// Devuelve un mensaje de respuesta que se mostrará en la app de ThingESP
String HandleResponse(String query)
{
  // Si el usuario envía "encender luz"
  if (query == "encender luz") {
      // Enviamos un mensaje por UART al RP2040 indicando que se debe encender la luz
    mensaje_al_RP2040.write("LUZ ENCENDIDA ;)\n");
    // Enviamos una respuesta de confirmación a ThingESP
    return "HECHO: Luz encendida";
  }
  // Si el usuario envía "apagar luz"
  else if (query == "apagar luz") {
      // Enviamos un mensaje por UART al RP2040 indicando que se debe apagar la luz
    mensaje_al_RP2040.write("LUZ APAGADA :C\n");
    // Enviamos una respuesta de confirmación a ThingESP
    return "HECHO: Luz apagada";
  }
  // Si se recibe un comando que no es reconocido
  else {
```

```
      // Enviamos un mensaje al RP2040 indicando que el comando fue
inválido
    mensaje_al_RP2040.write("THA FUCK?\n");
    // Enviamos una respuesta de error a ThingESP
    return "Mensaje invalido"; }}
// Bucle principal del programa, se ejecuta constantemente
void loop()
{
    // Esta función se encarga de verificar si hay nuevos comandos
recibidos desde ThingESP
  // y automáticamente llama a HandleResponse() con el texto del comando
  thing.Handle();
}
```

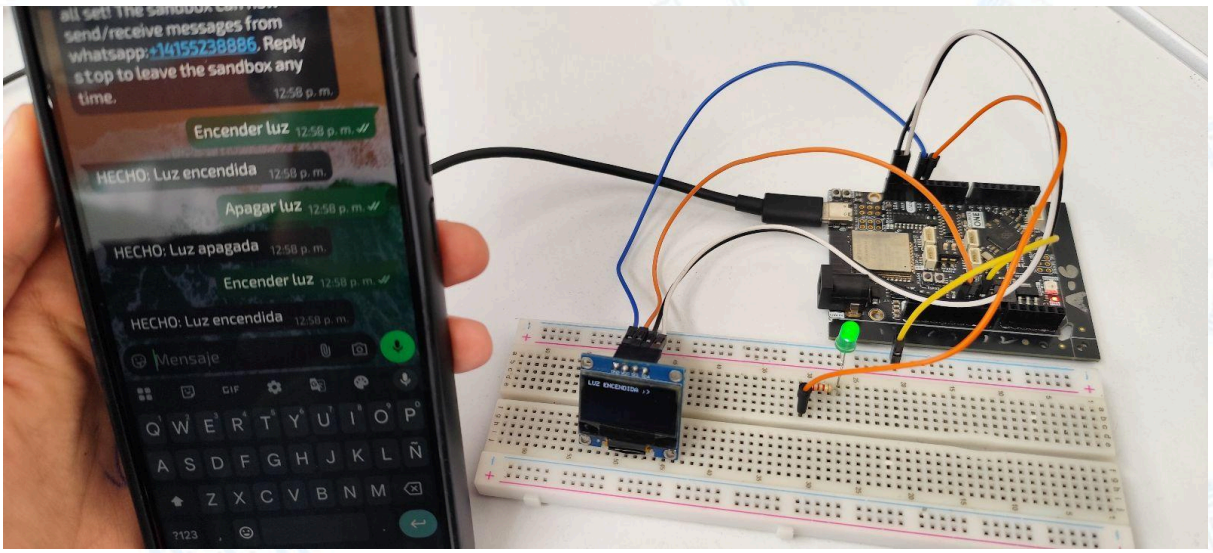
Por su parte, el RP2040 recibe mediante UART cadenas de texto según lo que haya recibido el ESP32 desde WhatsApp y en función de las mismas se enciende o se apaga un led integrado en la tarjeta UNIT DUAL MCU ONE y se muestra un mensaje en una pantalla OLED 128x64 SSD1306

```
// Incluimos las librerías necesarias para manejar la pantalla OLED por
I2C
#include <Wire.h>                    // Comunicación I2C
#include <Adafruit_GFX.h>           // Librería gráfica de Adafruit (texto,
formas, etc.)
#include <Adafruit_SSD1306.h>       // Librería específica para pantallas
OLED con chip SSD1306
// Dimensiones de la pantalla OLED (usualmente 128x64 píxeles)
#define ANCHO_PANTALLA 128
#define ALTO_PANTALLA 64
// Definimos el pin donde está conectado el LED integrado en la UNIT DUAL
MCU ONE
#define LED 29  // GPIO 29, puedes cambiarlo si conectaste el LED a otro
pin
// Creamos el objeto 'display' para manejar la OLED
// Usa el bus I2C predeterminado (Wire) y sin pin de reset (-1)
Adafruit_SSD1306 display(ANCHO_PANTALLA, ALTO_PANTALLA, &Wire, -1);
// Función de configuración, se ejecuta una sola vez al iniciar el RP2040
void setup() {
   Serial.begin(115200);         // Comunicación serial con la computadora
(depuración vía USB)
```

```
  Serial1.begin(115200);          // Comunicación UART con el ESP32 (usa pines TX1/RX1 del RP2040)
  Wire.begin();                   // Inicializamos la comunicación I2C para la OLED
  pinMode(LED, OUTPUT);           // Configuramos el pin del LED como salida
  digitalWrite(LED, LOW);         // Apagamos el LED inicialmente
  // Inicializamos la pantalla OLED (dirección I2C: 0x3C)
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("No se encontró la pantalla OLED"));
      while (true);   // Si no se detecta la pantalla, el programa se detiene
  }
  // Limpiamos pantalla y configuramos estilos de texto
  display.clearDisplay();                 // Borra cualquier contenido en la pantalla
  display.setTextSize(1);                 // Tamaño de texto normal
    display.setTextColor(SSD1306_WHITE);     // Texto en blanco (OLED monocromática)
}
// Función principal, se ejecuta en bucle constantemente
void loop() {
  // Verificamos si llegó un mensaje desde el ESP32 por el puerto UART (Serial1)
  if (Serial1.available() > 0) // Leemos la línea completa (hasta salto de línea) enviada desde el ESP32
    String mensaje = Serial1.readStringUntil('\n');
    mensaje.trim();  // Eliminamos espacios extra o saltos de línea
    // Mostramos el mensaje en el monitor serial (depuración)
    Serial.print("ACK: ");      // Prefijo de confirmación
    Serial.println(mensaje);    // Mostramos el mensaje recibido
// Encendemos o apagamos el LED según el contenido del mensaje recibido
    if (mensaje == "LUZ ENCENDIDA ;)") {
      digitalWrite(LED, HIGH);  // Encendemos el LED
    }
    else if (mensaje == "LUZ APAGADA :C") {
      digitalWrite(LED, LOW);   // Apagamos el LED
    }
    // Mostramos el mensaje en la pantalla OLED
  display.clearDisplay();         // Limpiamos la pantalla
```

```
  display.setCursor(0,  0);// Colocamos el cursor en la esquina superior
izquierda
   display.println(mensaje);      // Escribimos el mensaje
   display.display();             // Mostramos el contenido
   delay(3000);  // Esperamos 3 segundos antes de actualizar la pantalla
   }}
```

Una vez que todo esté configurado y los programas cargados podemos enviar mensajes en WhatsApp para encender y apagar la luz y el mismo sistema nos responde con mensajes de confirmación 😀🎉



Lección 9 Control de luz por comandos de voz

En esta lección aprenderás a implementar un sencillo control de una GPIO del RP2040 integrado en la UNIT DUAL MCU ONE 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

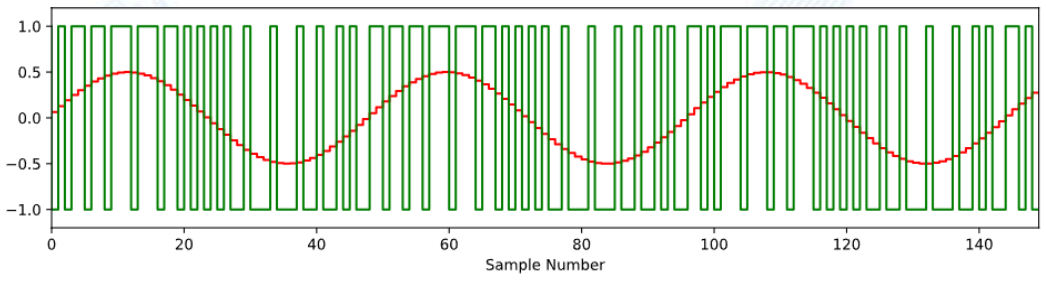
PDM (Modulación por Densidad de Pulsos) es una técnica de codificación de audio digital en la que la información de la señal analógica se representa como una secuencia de pulsos binarios (1s y 0s).

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:

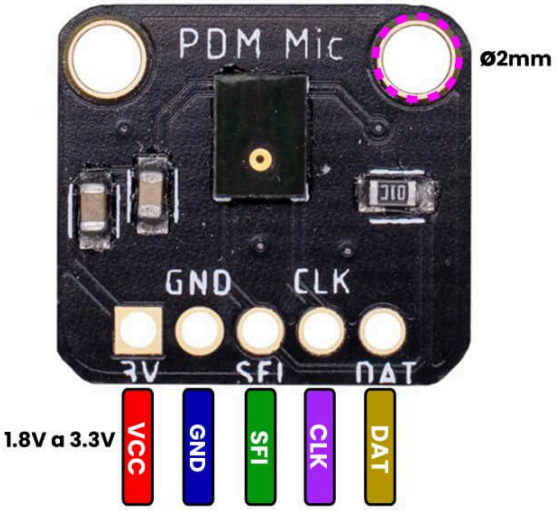
Si la señal de entrada es **alta**, se generan **más unos (1)**.
Si la señal de entrada es **baja**, se generan **más ceros (0)**.

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.

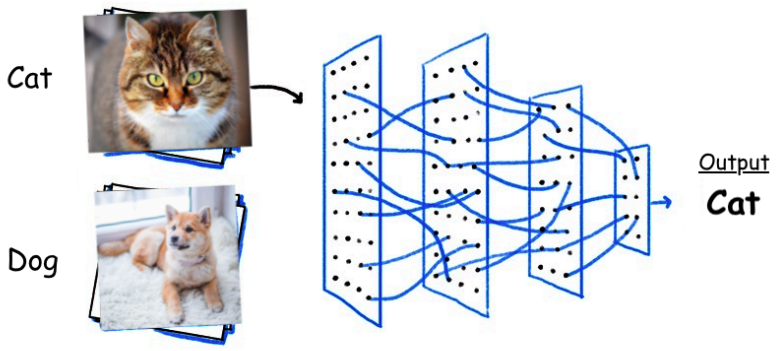


Machine learning

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".

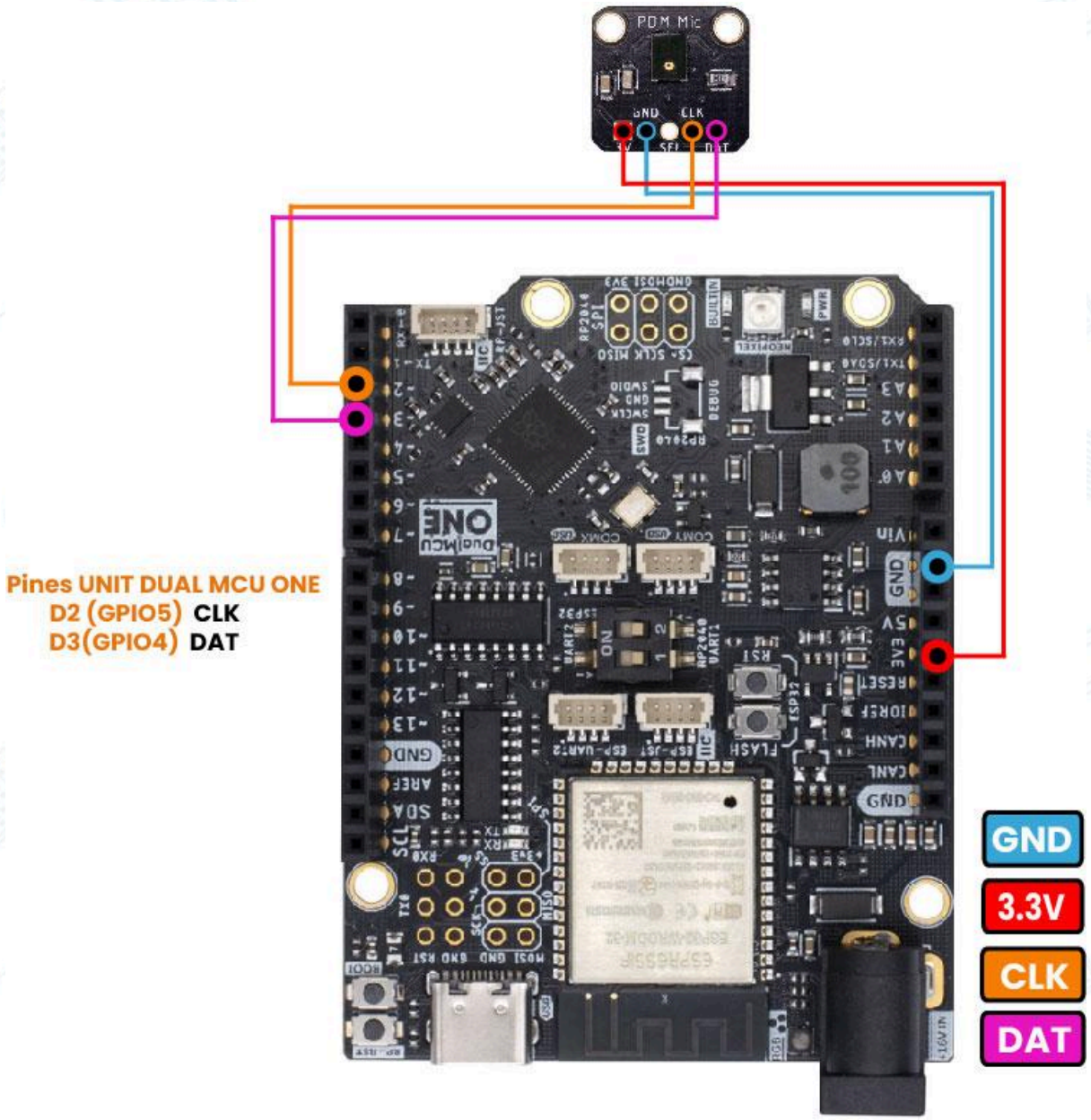
5. **Evaluación:** Se prueba el modelo con datos que no ha visto antes para ver qué tan bien ha aprendido.

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:

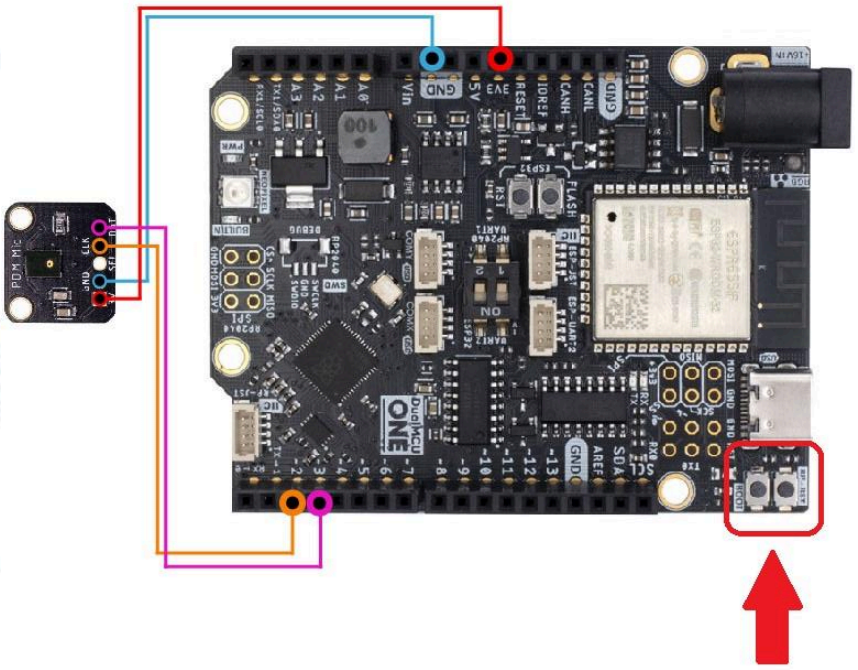
- Configurar su RP2040 como un micrófono USB para entrenar al modelo de machine learning

- 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



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.

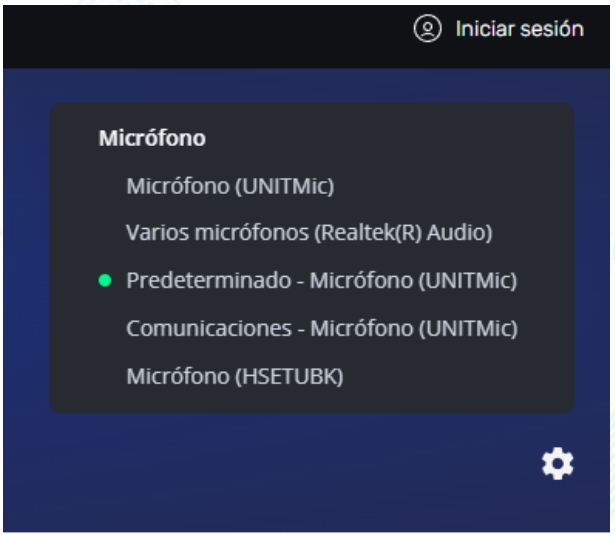
Tras lo cual, en su explorador de archivos se abrirá una unidad de almacenamiento similar a la siguiente

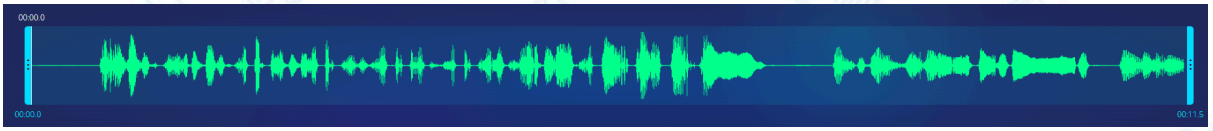
En dicha unidad deberá arrastrar y soltar el archivo **DualONE_USBMic_v1.uf2** que podrá descargar en el repositorio de la lección
https://github.com/UNIT-Electronics/UNIT-ONE-MCU-/tree/main/L9_Comandos_de_voz

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





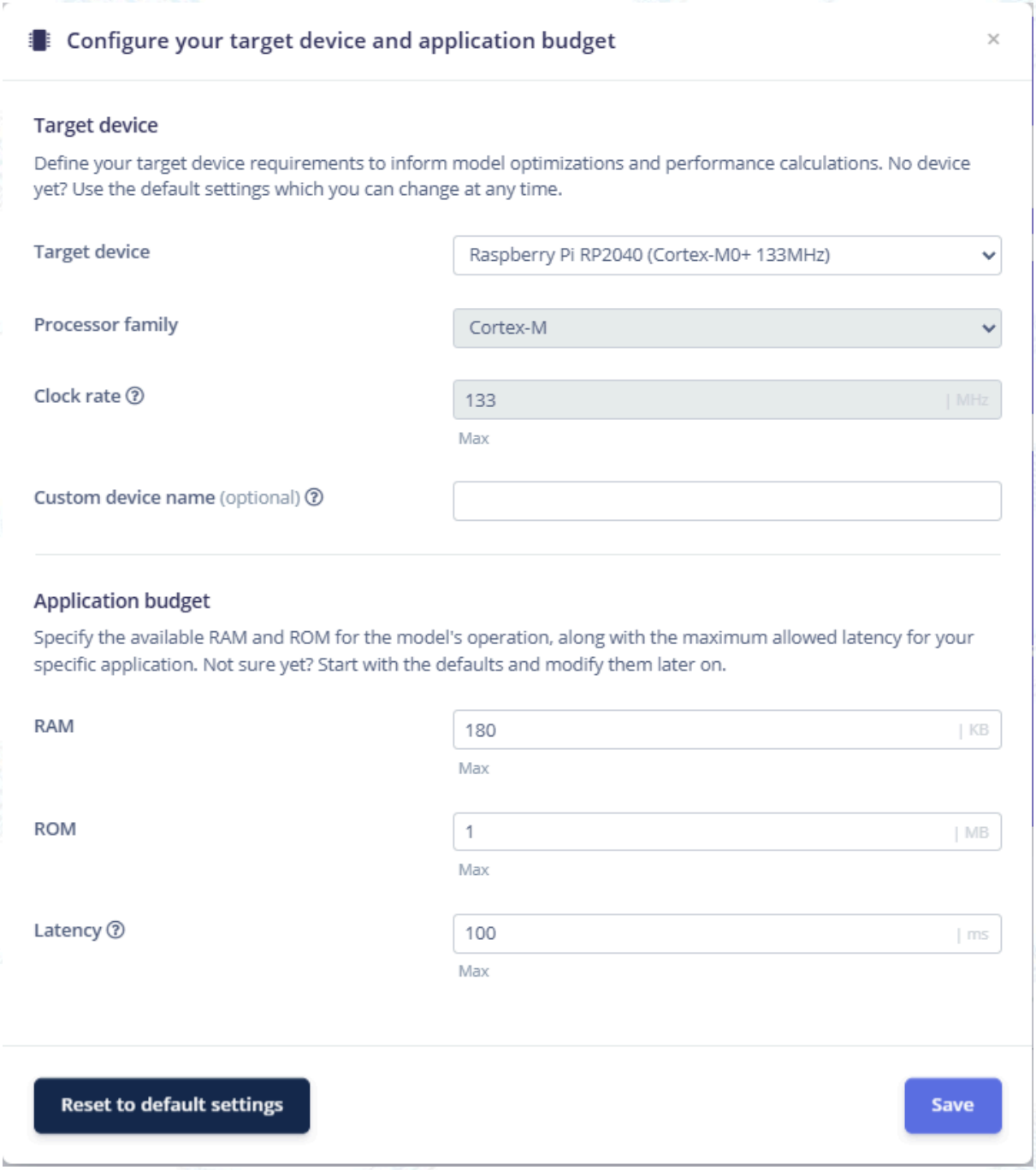
Para volver a la normalidad su RP2040, es decir, que vuelva a ser reconocida como un puerto COM en su PC, presione la misma combinación de botones y cargue un Blink desde su Arduino IDE al puerto UF2

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



Lección Extra: CAN BUS

En esta lección demostrativa aprenderás a establecer la comunicación entre dos UNIT DUAL MCU ONE y un ESP32 mediante el protocolo CAN

Materiales

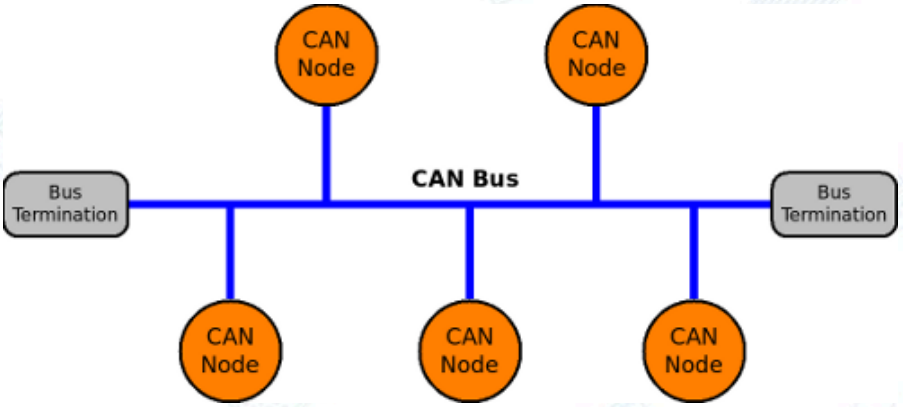
1.- DUAL MCU ONE x2
2.- ESP32 x1
3.- Protoboard x1
4.- Módulo MCP2515 x1
5.- Cables dupont Macho - Macho
6.- Cables dupont Macho - Hembra
7.- Resistencia 120 Ω ¼ W x1 (Opcional)

Conocimientos previos

Protocolo CAN

El protocolo CAN (Controller Area Network) es un sistema de comunicación en red desarrollado originalmente por Bosch en la década de 1980, específicamente para la industria automotriz. Su propósito principal era permitir que diversos microcontroladores y dispositivos electrónicos dentro de un automóvil pudieran comunicarse entre sí de manera confiable, sin requerir un complejo sistema de cables punto a punto.

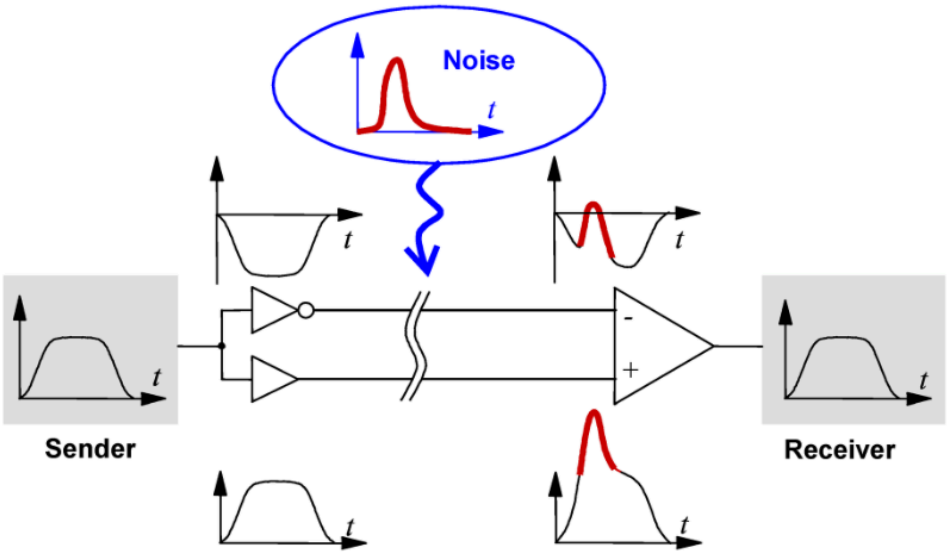
El protocolo CAN es una **red de difusión** (broadcast) en la que todos los nodos (dispositivos conectados al bus) reciben todos los mensajes que se transmiten en el medio, aunque solo respondan a los que les interesan. Es una red **multimaestro**, es decir, cualquier nodo puede iniciar una transmisión si el bus está libre.



Algunas características clave del protocolo CAN son:

- **Alta inmunidad al ruido** y capacidad de detección de errores.
- **Acceso no destructivo al bus**: permite que múltiples nodos intenten transmitir simultáneamente sin colisiones dañinas.
- **Velocidad de transmisión típica**: hasta 1 Mbps (en CAN estándar).
- **Topología en bus**: todos los nodos están conectados al mismo par de cables.

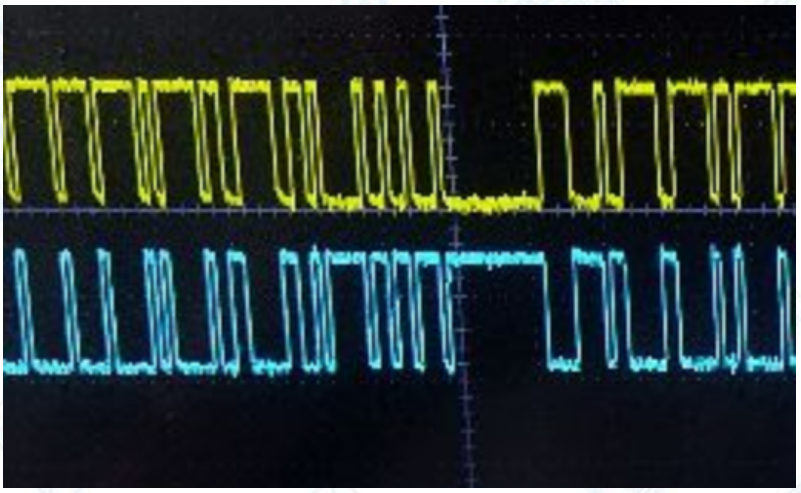
El nivel físico de CAN normalmente utiliza un bus diferencial de dos hilos, conocido como CAN_H y CAN_L. Esta señalización diferencial mejora la inmunidad al ruido electromagnético. Cuando se transmite un bit dominante (0), la línea CAN_H está a un voltaje más alto que CAN_L. Un bit recesivo (1) ocurre cuando ambas líneas tienen aproximadamente el mismo voltaje.



Los nodos se conectan a este bus a través de transceptores CAN, que convierten las señales digitales del microcontrolador en las señales diferenciales del bus.

En la imagen se observa el oscilograma de las dos líneas diferenciales que componen al CAN bus, donde la traza amarilla corresponde a CAN-H y la traza azul a CAN - L enviando una trama de bits a través del bus

.



Manejo de Errores

El protocolo CAN tiene mecanismos avanzados de detección y gestión de errores, tales como:

- Errores de bit: cuando un nodo detecta un bit diferente al transmitido.
- Errores de formato: si la estructura del mensaje es inválida.
- Errores de ACK: si ningún nodo reconoce un mensaje recibido.
- Errores de CRC: si hay discrepancias en el código de verificación.
- Errores de sobrecarga: si un nodo no puede procesar datos a tiempo.

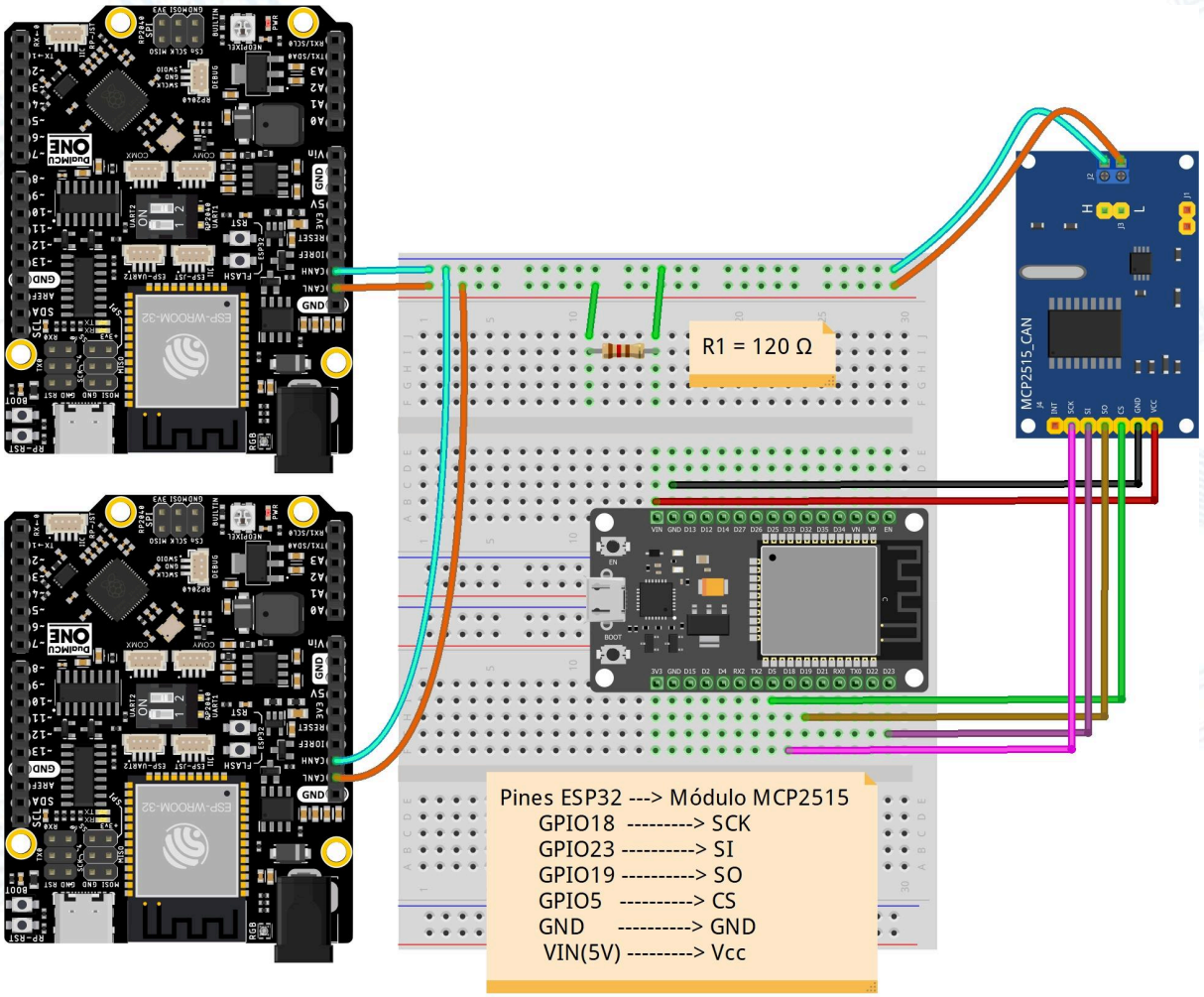
Cuando un error es detectado, el nodo interrumpe el mensaje y transmite una trama de error, notificando a los demás. Los nodos también pueden entrar en modos de operación degradada si detectan demasiados errores, como el modo pasivo (escucha pero no transmite) o el modo bus-off (desconexión temporal del bus para evitar interrupciones graves).

Aplicaciones del protocolo CAN

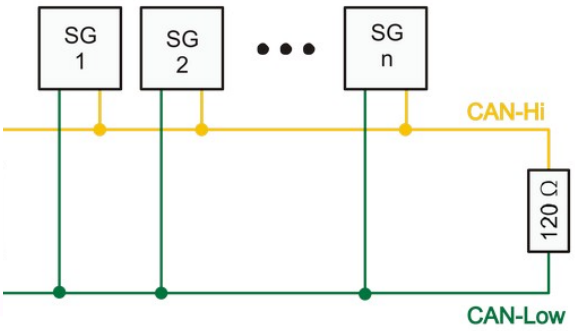
Aunque fue creado para automóviles, el protocolo CAN se ha expandido a muchas otras áreas debido a su confiabilidad y robustez. Tales como la automatización industrial (redes de sensores y actuadores), sistemas médicos (como equipos de imágenes y dispositivos implantables), aeronáutica y ferrocarriles, robótica y domótica

Diagrama de conexiones

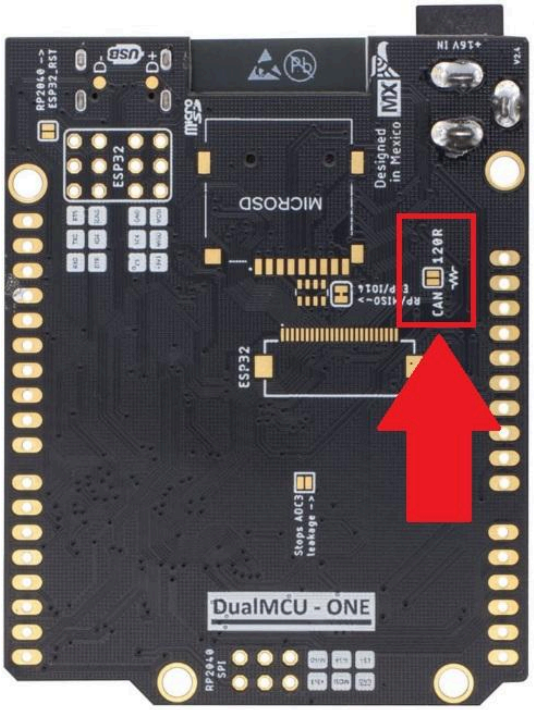
Con el siguiente circuito podremos lograr la comunicación entre tres ESP32 donde es destacable la reducción de conexiones que se logra al usar las tarjetas UNIT DUAL MCU ONE en comparación con la misma implementación, pero con un ESP32 en su clásica presentación DOIT DEV KIT que forzosamente requiere del uso externo del módulo MCP2515



Por otra parte, es indispensable el uso de una resistencia de 120Ω conectada entre los pines del bus (CANH y CANL), ya que sin ella la comunicación no se lleva a cabo



Pero para el desarrollo de esta lección esto se simplifica al realizar un puente de soldadura en los pads ubicados en el anverso de las tarjetas UNIT DUAL MCU ONE para habilitar una resistencia integrada dentro de la placa



Códigos de funcionamiento

Para la realización de esta lección es necesario cargar 3 programas a tres microcontroladores, dos ESP32 integrados, cada uno en par de UNIT DUAL MCU ONE y un último ESP32 en su clásica tarjeta de desarrollo DOIT DEV KIT. Dichos programas los puedes descargar en el siguiente enlace:
https://github.com/UNIT-Electronics/UNIT-ONE-MCU-/tree/main/CAN%20bus

La comunicación en este código consiste en que una de las DUAL MCU ONE va a actuar como maestro y va a transmitir un mensaje al ID 0x123 que corresponde a la otra DUAL MCU ONE y otro mensaje al ID 0x0F que tiene asignado el ESP32 de la tarjeta DOIT DEV KIT
Aquí observamos el mensaje en el monitor serie del maestro donde envía un par de mensajes cada uno a un ID en específico

```
✅ ✅Mensaje transmitido
✅ Transmisión exitosa.
✅ Transmisión exitosa.
Enviando: 01 02 03 04
✅ Mensaje transmitido
Enviando: 04 03 02 01
```

El esclavo con el ID 0x0F recibe e imprime el mensaje en su monitor serie e ignora el otro mensaje dirigido al ID 0x123

```
Mensaje en formato estándar
ID: f6
Bytes: 0 = 04, 1 = 03, 2 = 02, 3 = 01,
```

El esclavo con el ID 0x123 recibe e imprime el mensaje en su monitor serie e ignora el otro mensaje con un ID que no es el suyo

```
Mensaje recibido con ID 0x123: 01 02 03 04
```

Tal y como se dijo anteriormente, las tramas de bits de las que consta el protocolo CAN nos permite detectar errores, como es el caso del siguiente mensaje en el monitor serie del maestro donde detecta que lo mensajes no fueron recibidos por los esclavos, esto debido a la desconexión de los mismos

```
✅ ✅Mensaje transmitido
⚠️ Error en el bus CAN.
Errores en el bus: 50
❌ Falló la transmisión.
TX en cola: 0 | TX errores: 0 | TX fallidas: 50
```

De esta manera logramos una comunicación robusta entre múltiples dispositivos conectados al mismo canal, a pesar de interferencias electromagnéticas con el medio, lo que convierte a este protocolo en una opción adecuada para entornos industriales o en aplicaciones rodadas de ruido eléctrico

Gracias por su preferencia

Manual elaborado por el Equipo UNIT Electronics
Septiembre 2025