ESP32

O microcontrolador ESP32 é uma ferramenta que, além de apresentar os recursos básicos padrões como envio de sinais digitais e analógicos, também apresenta integração com Wi-Fi e conectividade Bluetooth para uma ampla gama de aplicativos. Sua programação é feita na linguagem C++ e pode ser feita através da própria Arduino IDE, um compilador já bastante reconhecido no mercado.

A programação do ESP32 é basicamente idêntica a do Arduino e do ESP8266, apenas muda algumas configurações na IDE. Vamos mostrar como configurar o ESP32.

PROGRAMANDO O ESP32

Para programar o ESP32 podemos usar o Arduino IDE, mas precisaremos configurar o site das preferências para instalar a placa no ESP32.

Site da Placa

Vá em: Arquivo ⇒ Preferências ⇒ Cole o Link ⇒ URLs Adicionais. Agora cole o link

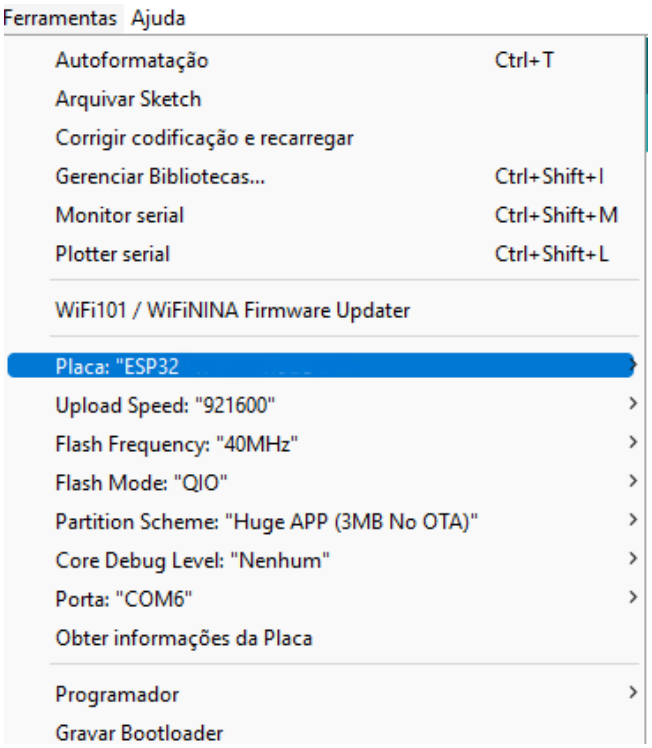
- https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

Instalando a Placa

Vá em: Ferramentas ⇒ Placa ⇒ Gerenciador de Placas ⇒ ESP32 (pesquisar) ⇒ Instalar (Selecione a Placa ESP32)

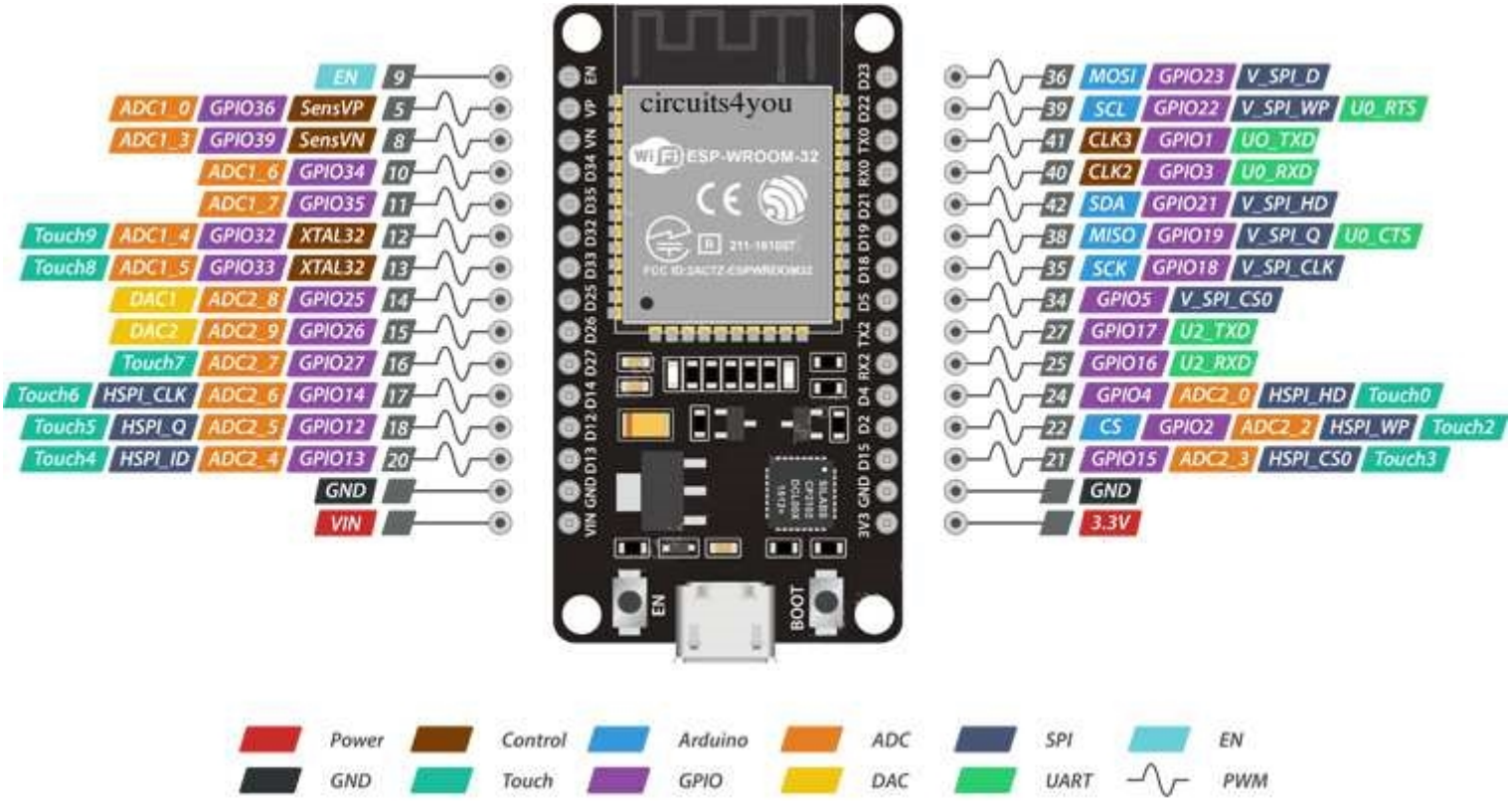
Configurações Recomendadas

As configurações recomendadas (aba ferramentas da IDE) estão mostradas na figura abaixo



PINAGEM

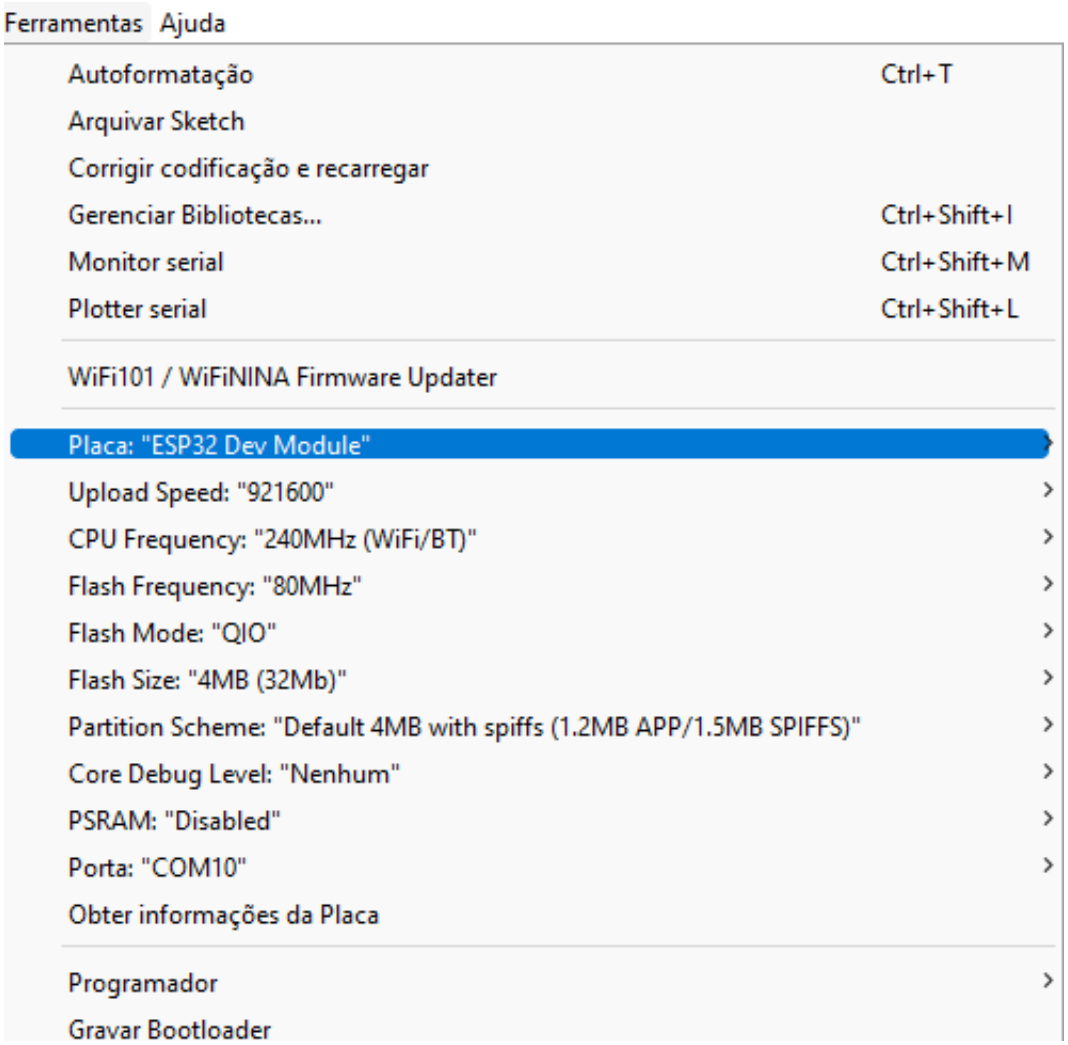
Existem vários modelos de ESP32, então vão existir também diversas pinagens. Assim sendo, deveremos saber qual é o modelo de seu ESP32 para encontrar a ilustração da pinagem desejada, geralmente essa informação se encontra debaixo do próprio ESP, sendo necessária uma pesquisa na internet para encontrá-la. A figura abaixo mostra a pinagem do ESP32 DEV KIT V1.



No ESP32, os pinos GPIO são para envio de sinais digitais e analógicos (PWM) e são enumerados pelo nome do pino (se quisermos, por exemplo, usar o GPIO15, vamos usar o número 15 na programação).

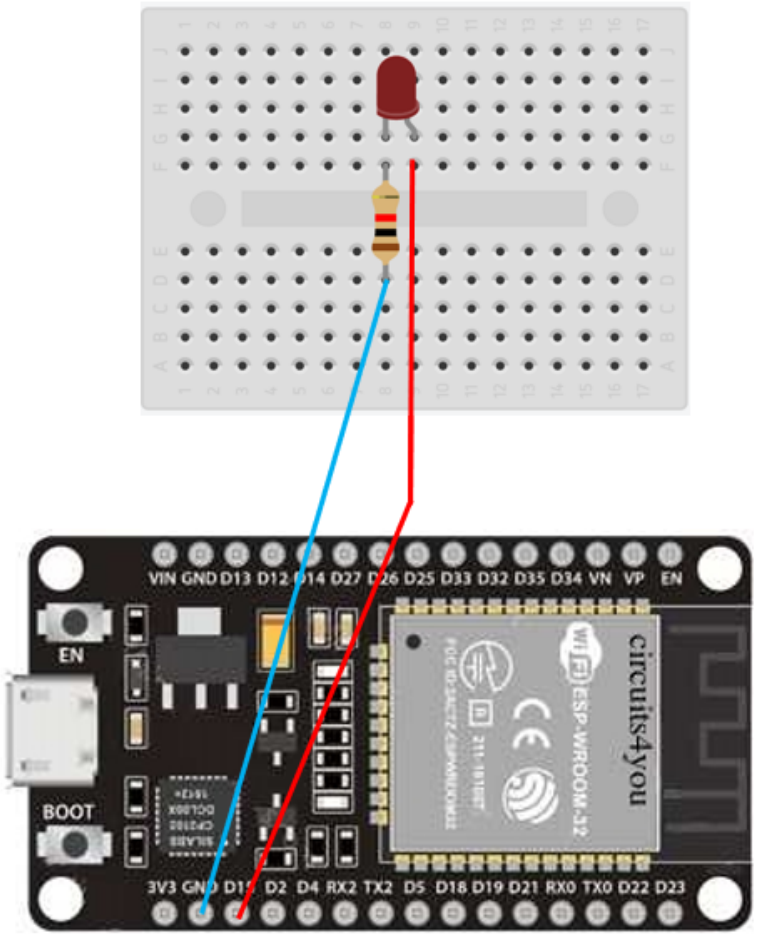
Os pinos ADC servem para a leitura de sinais analógicos, como por exemplo a voltagem enviada por um potenciômetro que pode ser regulada. Tais pinos podem então ler uma tensão de 0 a 3.3 Volts, mas essa funcionalidade está apenas nos pinos GPIO com o ADC (vermelho).

Antes de compilar o código, deveremos selecionar a placa de seu modelo ESP32 na aba "Ferramentas", como mostrar a figura abaixo.



ACENDENDO UM LED

Vamos conectar o ESP32 em uma protoboard. Para encaixar melhor, é melhor juntar duas protoboards e conectar o ESP entre elas, como mostrado na figura abaixo. Conectando o LED no pino GPIO15 e depois o outro no GND que é o negativo.



Após enviar o comando HIGH ao pino ?, uma corrente de 40 mA em 3.3 Volts passa pelo LED, o acende e retorna ao negativo, pelo GND. A programação é semelhante a do Arduino e está mostrada na figura abaixo, fazendo o LED piscar a cada 1 segundo.

```
#define LED 15

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, HIGH);
  delay(1000);

  digitalWrite(LED, LOW);
  delay(1000);
}
```

ENVIO DE SINAIS PWM

A função PWM está presente em todos os pinos GPIO do ESP32, onde podemos enviar uma tensão específica de 0 a 3.3 Volts por um sinal analógico.

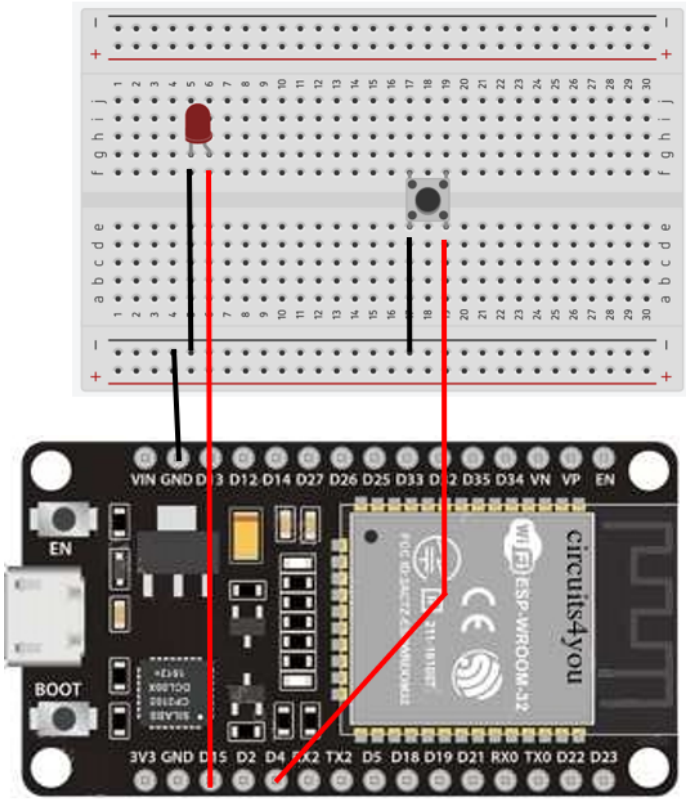
Primeiro precisamos declarar o pino do LED como um pino PWM usando uma variável canal que recebe 0 (pode ser qualquer valor), e então estabelecendo sua frequência para 5000 a ele, ou seja, o LED vai usar uma frequência de 5000 sinais digitais por segundo para o envio do sinal analógico.

```
#define LED 15

int canal = 0;            // CANAL PARA USAR O PWM DO PINO 15
int frequencia = 5000;    // FREQUÊNCIA PWM DO LED
int bits = 8;             // 2^8 = 256, SINAL DE 0 A 255

void setup() {
  ledcSetup(canal, frequencia, bits);  // INICIA O CANAL 0
  ledcAttachPin(LED, canal);           // CONECTA O LED AO CANAL 0
}

void loop() {
  for(int x=0; x<256; x++){  // DE 0 (0 VOLTS) A 255 (3.3 VOLTS)
    ledcWrite(canal, x);     // ENVIO DO SINAL ANALÓGICO
    delay(50);
  }

}
```

A função ledcWrite(canal, sinalAnalogico) recebe um valor de 0 a 255 que é justamente a intensidade do brilho do LED, que aumenta no laço for para aumentar o sinal analógico em 1 a cada repetição, amentando assim o brilho do LED a cada 50 milissegundos.

BOTÕES

O botão funciona como uma chave, quando ninguém o aperta essa chave está aberta e nenhuma corrente passa sobre ele.



Como vamos **ler** um dado digital do botão, vamos declará-lo como um pino de entrada (INPUT), e também como um pino INPUT PULLUP para evitar pontos flutuantes e falsas leituras, mas declará-lo como INPUT_PULLUP irá inverter a leitura, fazendo com que o 0 seja **botão pressionado** e 1 seja **botão não pressionado**.

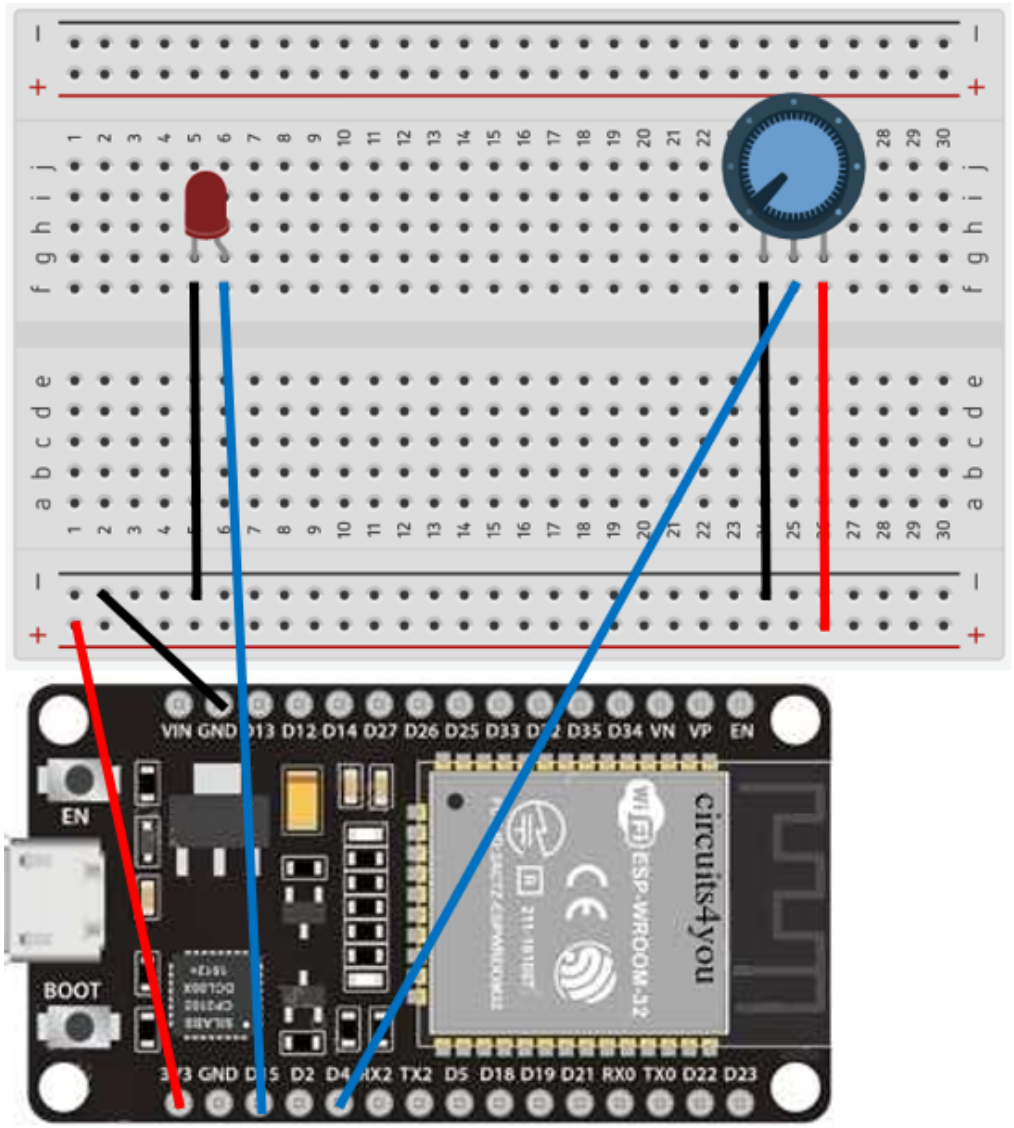
```
#define LED 15
#define BOTAO 4

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BOTAO, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  bool estado = digitalRead(BOTAO);

  if(estado == 0){
    Serial.println("BOTÃO PRESSIONADO!");
    digitalWrite(LED, HIGH);
  }else{
    digitalWrite(LED, LOW);
  }
}
```

USANDO POTENCIÔMETROS

O potenciômetro é um componente onde é possível ajustar a sua resistência elétrica, então podemos ler seus valores e usá-los em nossos projetos.

Agora usaremos um potenciômetro com o LED, o potenciômetro é conectado em um pino analógico do ESP32, sendo os pinos GPIO com a função ADC que está escrito em vermelho na filmagem. Nele é possível ajustar manualmente a voltagem que eu mando para o circuito girando a sua rodinha.

O potenciômetro tem 3 pinos, sendo o do meio o pino ADC/analógico que lerá o valor ajustado manualmente, os pinos dos cantos são o GND e o VCC (3.3 Volts), onde podemos conectá-los em qualquer ordem, mas isso irá inverter o sentido de crescimento da rodinha.



O potenciômetro gera um valor entre 0 e 4095 que é lido pelo comando analogRead(pino), e o comando ledcWrite (pino, intensidade) vai enviar um pulso entre 0 (0 Volts) até 255 (3.3 Volts). Perceba que para ajustar o brilho do LED devemos colocá-lo em um pino PWM. O potenciômetro deve receber o sinal do pino analógico, mais os 3.3V e GND.

```
#define LED 15
#define POT 4
const int canal = 0;

void setup() {
  pinMode(LED, OUTPUT);
  Serial.begin(9600);

  ledcSetup(canal, 5000, 8);
  ledcAttachPin(LED, canal);
}

void loop() {
  int intensidade = analogRead(POT);                // LEITURA DO POTENCIÔMETRO
  intensidade = map(intensidade, 0, 4095, 0, 255); // ALTERANDO PARA INTERVALO PWM

  Serial.println(intensidade);                      // IMPRIMINDO O VALOR DA INTENSIDADE
  ledcWrite(canal, intensidade);                    // ENVIANDO O SINAL PWM AO CANAL (LED)
}
```

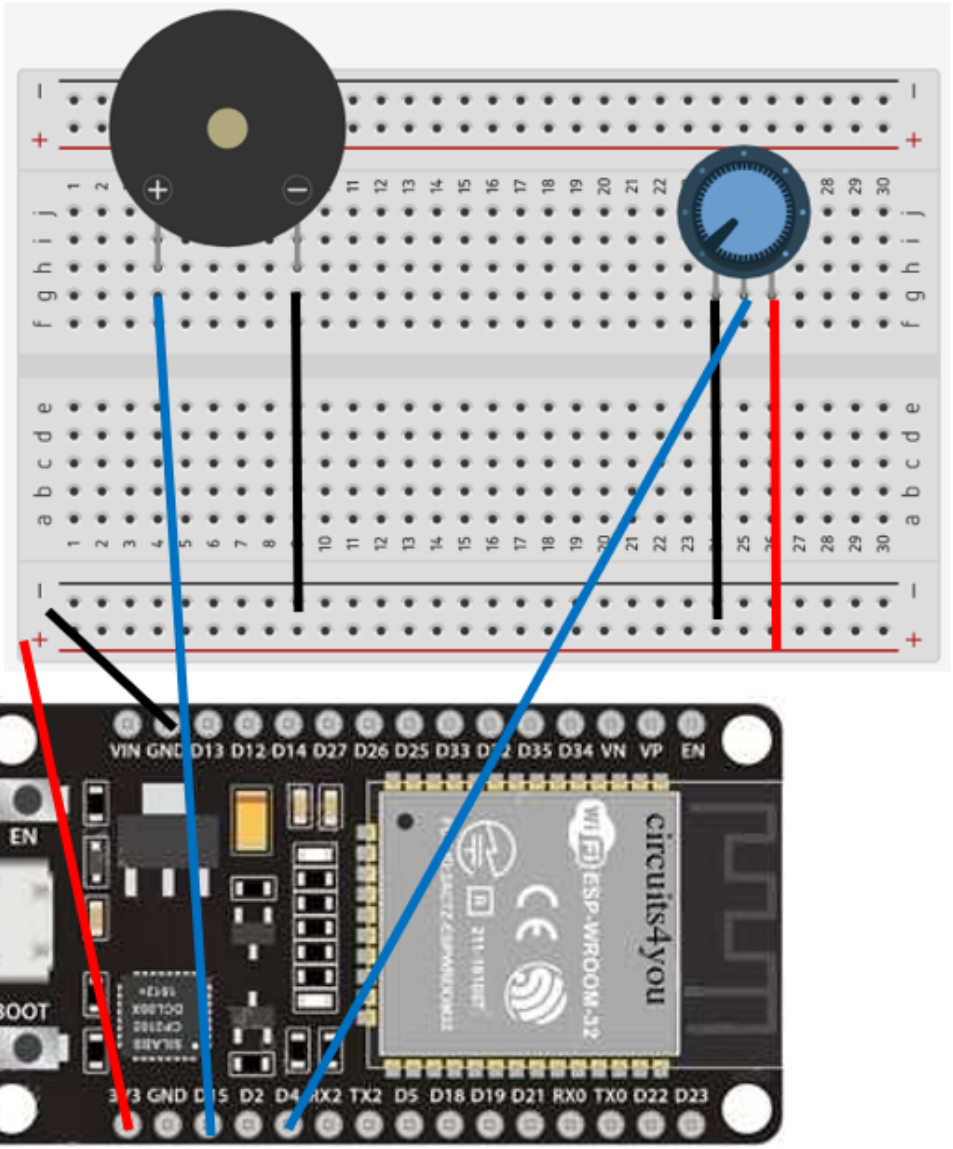
Usamos também a função map(valor, Xo, X, Yo, Y) para mapear linearmente o intervalo lido (0 a 4095) para o intervalo PWM desejado (0 a 255).

BUZZER

Os sonorizadores são dispositivos que têm a capacidade de produzir sons com quando são submetidos a uma certa voltagem, e faremos isso usando o Arduino.

Buzzer

Para usar o sonorizador no ESP32, lembre-se de que alguns possuem polaridade, ou seja, o lado positivo deve ser conetado aos 3.3 V e o negativo ao GND, se ficar muito barulhento, você pode adicionar resistores ao circuito a fim de diminuir o volume que o sonorizador faz. Vamos agora substituir o LED do projeto anterior por um buzzer, e assim, controlar o volume do sonorizador com o valor ajustado.




```
#define LED 15
#define POT 4
const int canal = 0;

void setup() {
  pinMode(LED, OUTPUT);
  Serial.begin(9600);

  ledcSetup(canal, 5000, 8);
  ledcAttachPin(LED, canal);
}

void loop() {
  int intensidade = analogRead(POT);                // LEITURA DO POTENCIÔMETRO
  intensidade = map(intensidade, 0, 4095, 0, 255); // ALTERANDO PARA INTERVALO PWM

  Serial.println(intensidade);                      // IMPRIMINDO O VALOR DA INTENSIDADE
  ledcWrite(canal, intensidade);                    // ENVIANDO O SINAL PWM AO CANAL (LED)
}
```