

Modern IoT Technology

An Introduction to ESP32

What is a ESP32

- ESP microcontrollers refer to a series of versatile and widely used microcontrollers developed by Espressif Systems
- The ESP (Espressif Systems' Platform) family is particularly well-known for its integration of Wi-Fi.
- Some models, Bluetooth capabilities,
- Popular choices for IoT (Internet of Things) applications.
- Among the notable ESP microcontrollers are the SP8266 and ESP32

Why ESP32?

- IoT technologies have proven their worth over the years, ESP32 is definitely one of those tool has:
 - Its price tag and availability
 - Wi-Fi and Bluetooth in a single SoC (System on Chip)
 - Many peripheral interfaces, different power modes, and cryptographic hardware acceleration
 - Variants for different requirements, in terms of both chips and modules
 - A huge community
 - And finally, native integration with top cloud infrastructures

Hardware

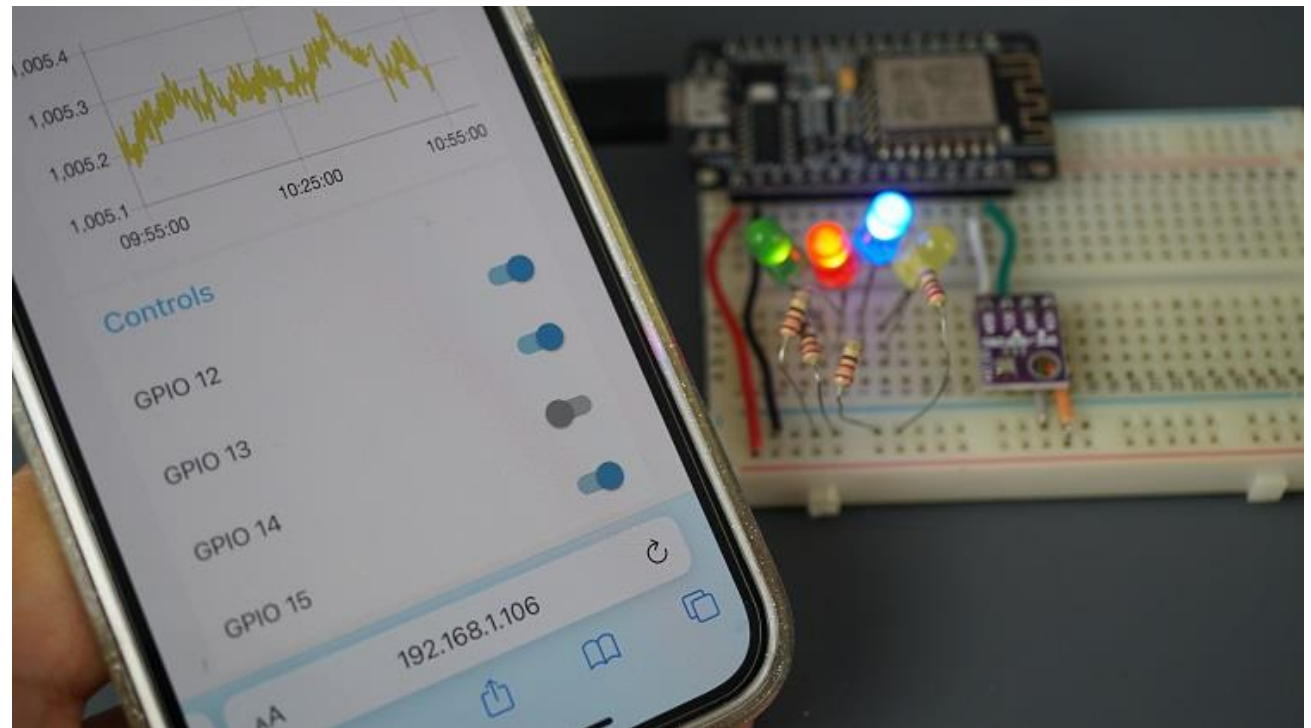
- The ESP32 Series: The ESP32 emerged in 2016 as a successor to the ESP8266. It is based on the Xtensa dual-core 32-bit LX6 processor. It supports both WiFi and Bluetooth 4.2 along with different I/O
- The ESP32-Sx Series: The S-series first emerged in 2020 with a more recent Xtensa LX7 processor. There are options for both single- and dual-core devices. Also, Bluetooth 5.0 is supported and also optional.
- The ESP32-Cx Series: The C-series also first emerged in 2020, incorporating the open-source RISC-V processor. The C-series ESP includes options for both single-core and dual-core processors. Support for Bluetooth 5.0 and WiFi 6
- The ESP32-Hx Series: The H-series first announced in 2021. The H-series is similar to the C-series except that it adds more connectivity options. These options include the Thread and Zigbee protocols.
- The ESP32-Px Series: The P-series is the most recent series and was announced in 2024. The P-series is RISC-V-based and targets AI applications. The P-series does not incorporate any connectivity for secure use cases. This series offers also multi-core functionality. The leading P4 device will include a dual-core RISC-V processor for high-performance workloads. In addition, the same P4 device will incorporate a single-core RISC-V core for low-power workloads

Development kits/boards

- ESP32-C3-DevKitM-1
- ESP32-C3-DevKitC-02
- ESP32-C3-DevKit-RUST-1
- ESP32-C3-AWS-ExpressLink-DevKit
- ESP32-C3-Lyra
- The difference in the onboard components and the number of pins that each provides.

Applications

- With low power consumption, ESP32 is an ideal choice for IoT devices in the following area:
- Smart Home
- Industrial Automation
- Health Care
- Consumer Electronics
- Smart Agriculture
- Cameras for Video Streaming
- Speech Recognition
- Image Recognition

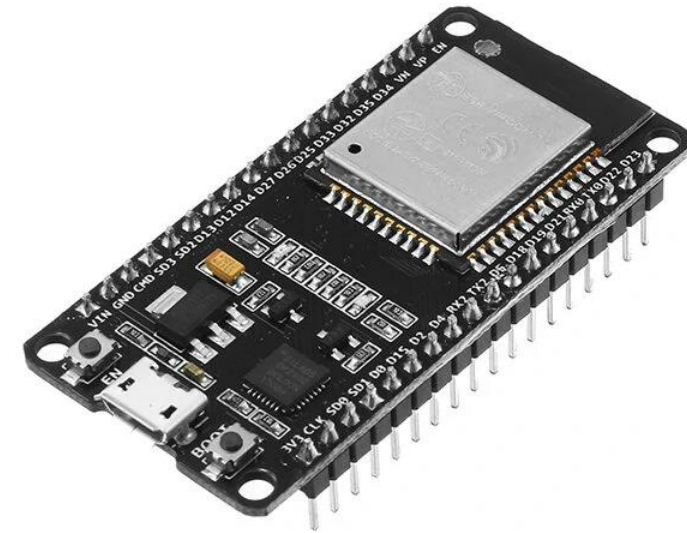
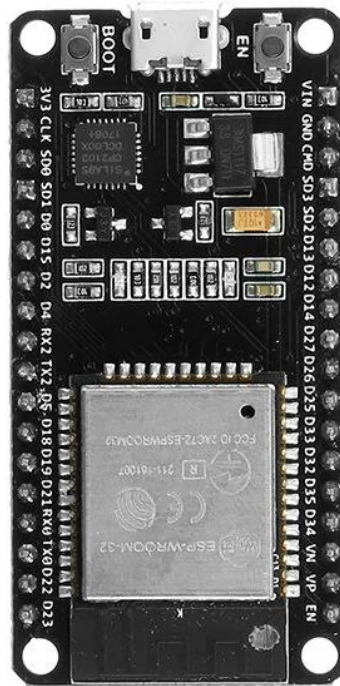


Review

- <https://makeradvisor.com/esp32-development-boards-review-comparison>

ESP32 development board

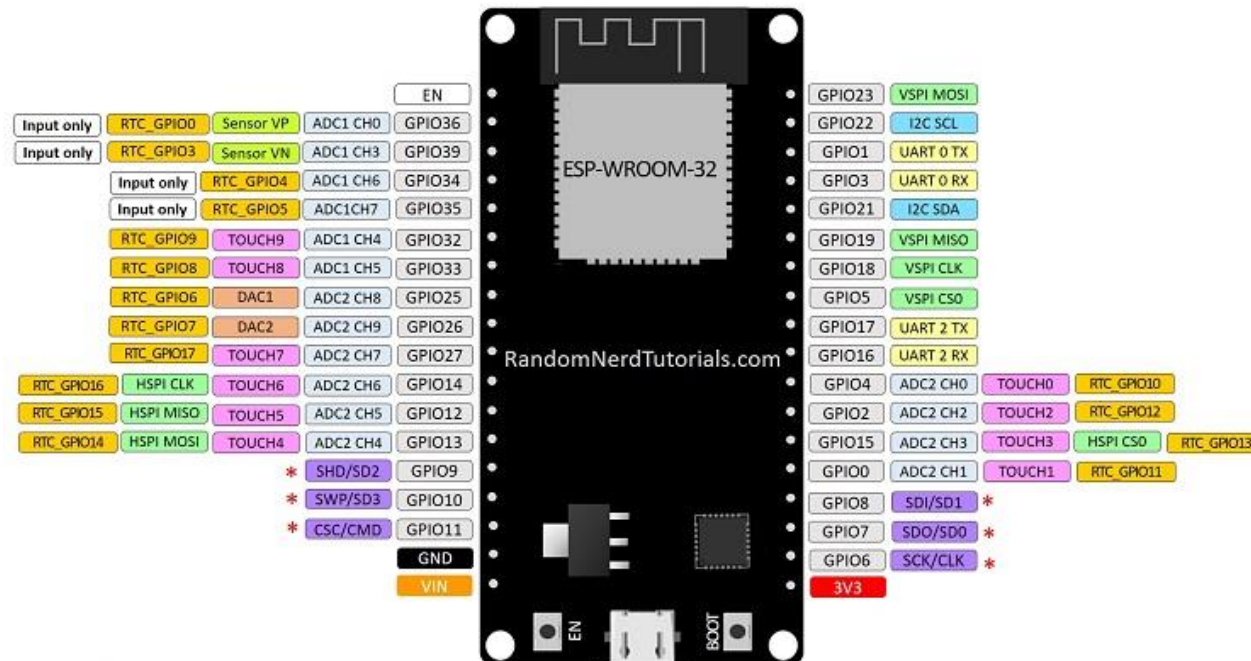
- ESP32 development board for beginners
- [DOIT ESP32 DEVKIT V1 Board](#) (Wi-Fi and Bluetooth)



Number of cores	2 (dual core)
Wi-Fi	2.4 GHz up to 150 Mbits/s
Bluetooth	BLE (Bluetooth Low Energy) and legacy Bluetooth
Architecture	32 bits
Clock frequency	Up to 240 MHz
RAM	512 KB
Pins	30, 36, or 38 (depending on the model)
Peripherals	Capacitive touch, ADC (analog to digital converter), DAC (digital to analog converter), I2C (Inter-Integrated Circuit), UART (universal asynchronous receiver/transmitter), CAN 2.0 (Controller Area Network), SPI (Serial Peripheral Interface), I2S (Integrated Inter-IC Sound), RMII (Reduced Media-Independent Interface), PWM (pulse width modulation), and more.
Built-in buttons	RESET and BOOT buttons
Built-in LEDs	built-in blue LED connected to GPIO2; built-in red LED that shows the board is being powered
USB to UART bridge	CP2102

ESP32 GPIOs Pinout

ESP32 DEVKIT V1 – DOIT version with 36 GPIOs



* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and CSC/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

ESP32 Peripherals

GPIO	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot, must be LOW to enter flashing mode
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED, must be left floating or LOW to enter flashing mode
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot, strapping pin
6	X	X	connected to the integrated SPI flash
7	X	X	connected to the integrated SPI flash
8	X	X	connected to the integrated SPI flash
9	X	X	connected to the integrated SPI flash
10	X	X	connected to the integrated SPI flash
11	X	X	connected to the integrated SPI flash

12	OK	OK	boot fails if pulled high, strapping pin
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot, strapping pin
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	

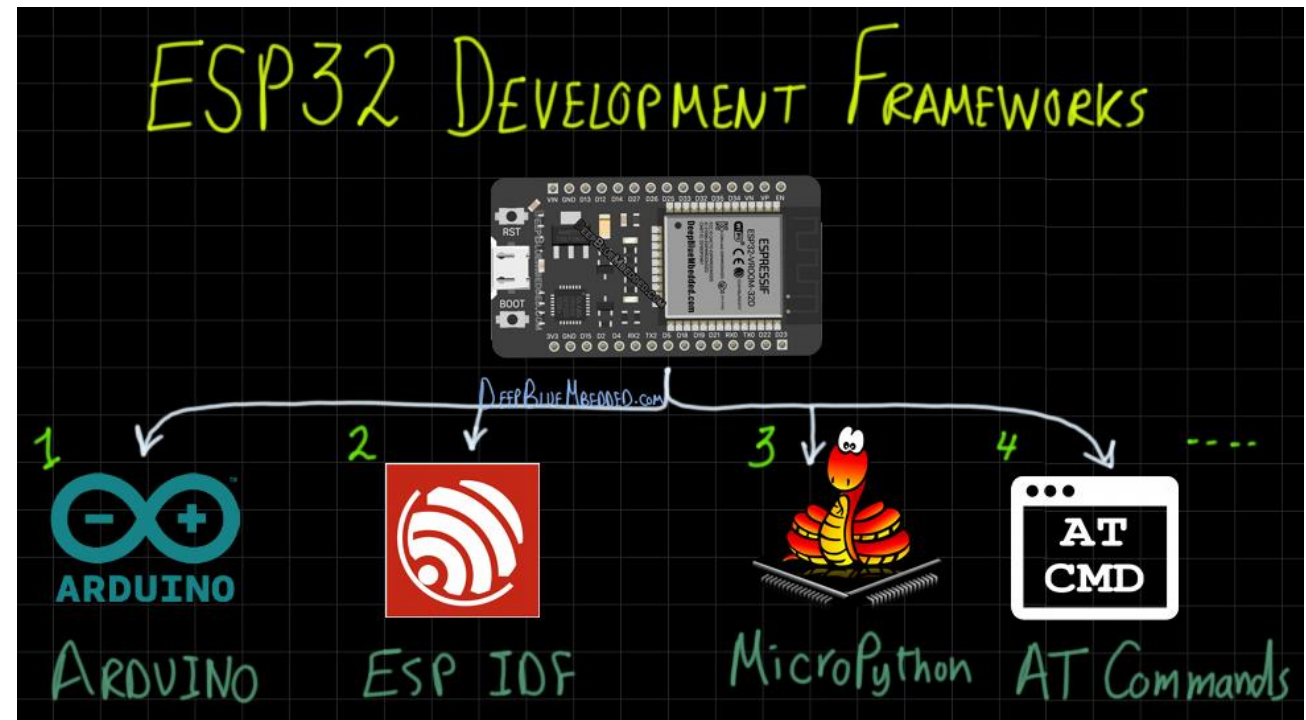
ESP32 Peripherals

- The pins highlighted in green are OK to use
- Pins highlighted in yellow are OK to use, but may have an unexpected behavior mainly at boot
- The pins highlighted in red are not recommended to use as inputs or outputs

32	OK	OK	
33	OK	OK	
34	OK		input only
35	OK		input only
36	OK		input only
39	OK		input only

Development Framework

- Arduino C/C++ using the Arduino core for the ESP32
- Espressif IDF (IoT Development Framework)
- Micropython
- JavaScript
- LUA

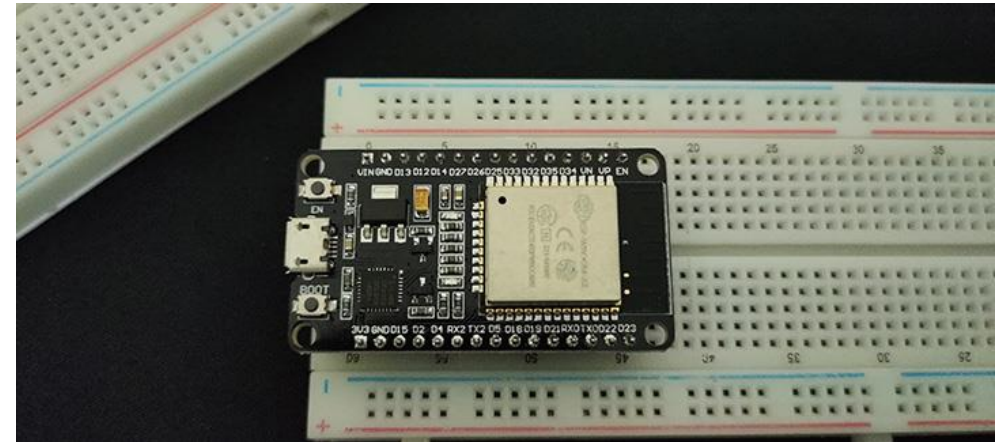


ESP32 LED Blinking

```
#define LED_BUILTIN 2 // On-Board LED Pin

void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

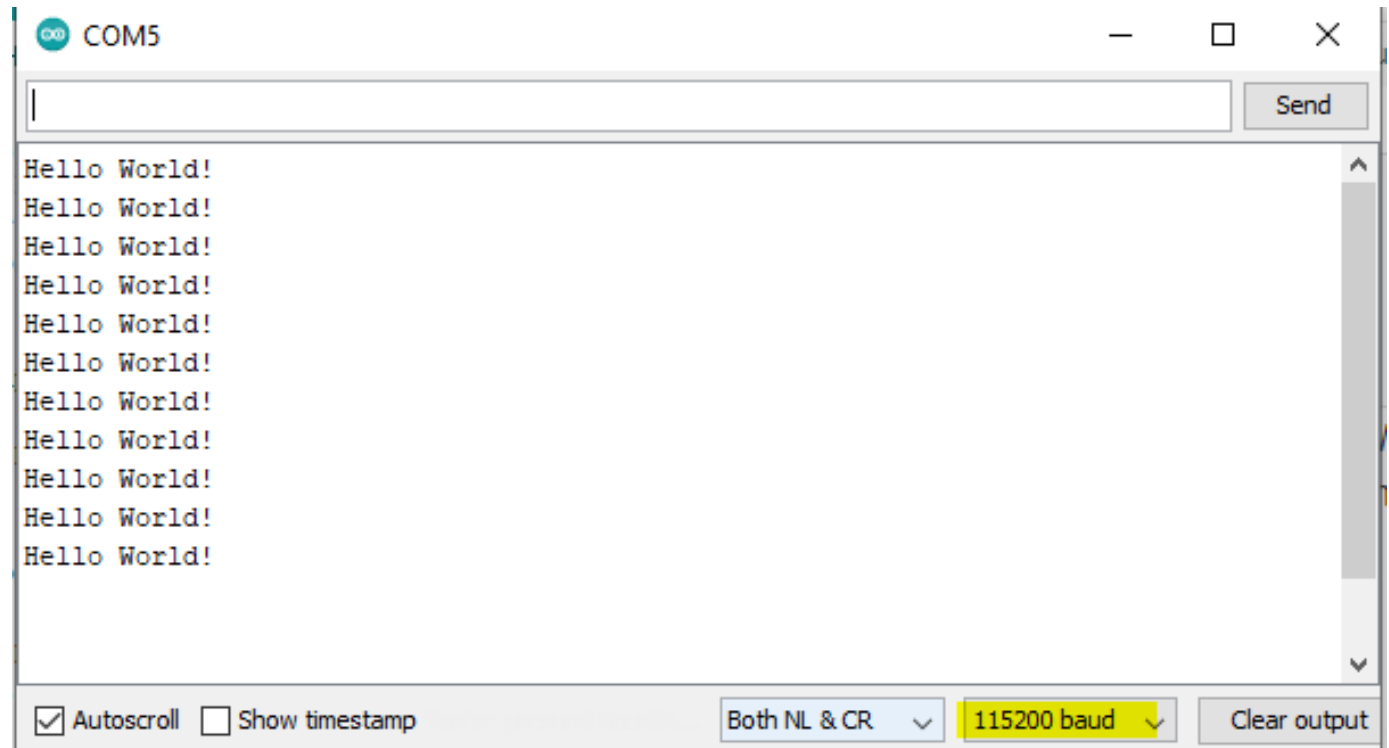
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on
  (HIGH is the voltage level)
  delay(250);
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by
  making the voltage LOW
  delay(250);
}
```



Serial Print For Debugging

```
void setup()
{
  Serial.begin(115200);
}

void loop()
{
  Serial.println("Hello World!");
  delay(1000);
}
```



ESP32 Serial Read Functions

```
int incomingByte = 0; // for incoming serial data

void setup() {
  Serial.begin(115200); // opens serial port, sets data rate to 115200
  bps
}

void loop() {
  // send data only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

    // say what you got:
    Serial.print("ESP32 received: ");
    Serial.println(incomingByte);
  }
}
```

ESP32 PWM

- The ESP32 has an LED PWM controller with 6 to 16 independent channels

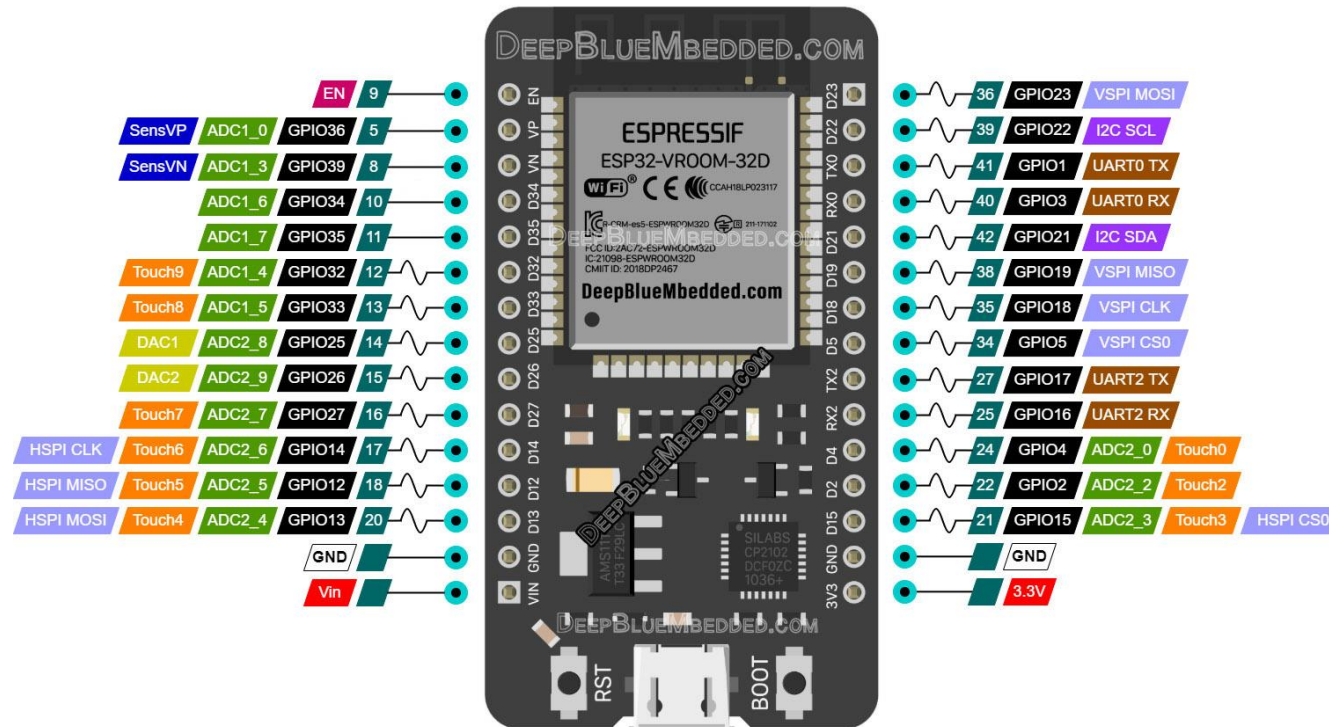
```
void analogWrite(uint8_t pin, int value);
```

- `void analogWrite(2, 180);`

ESP32 PWM

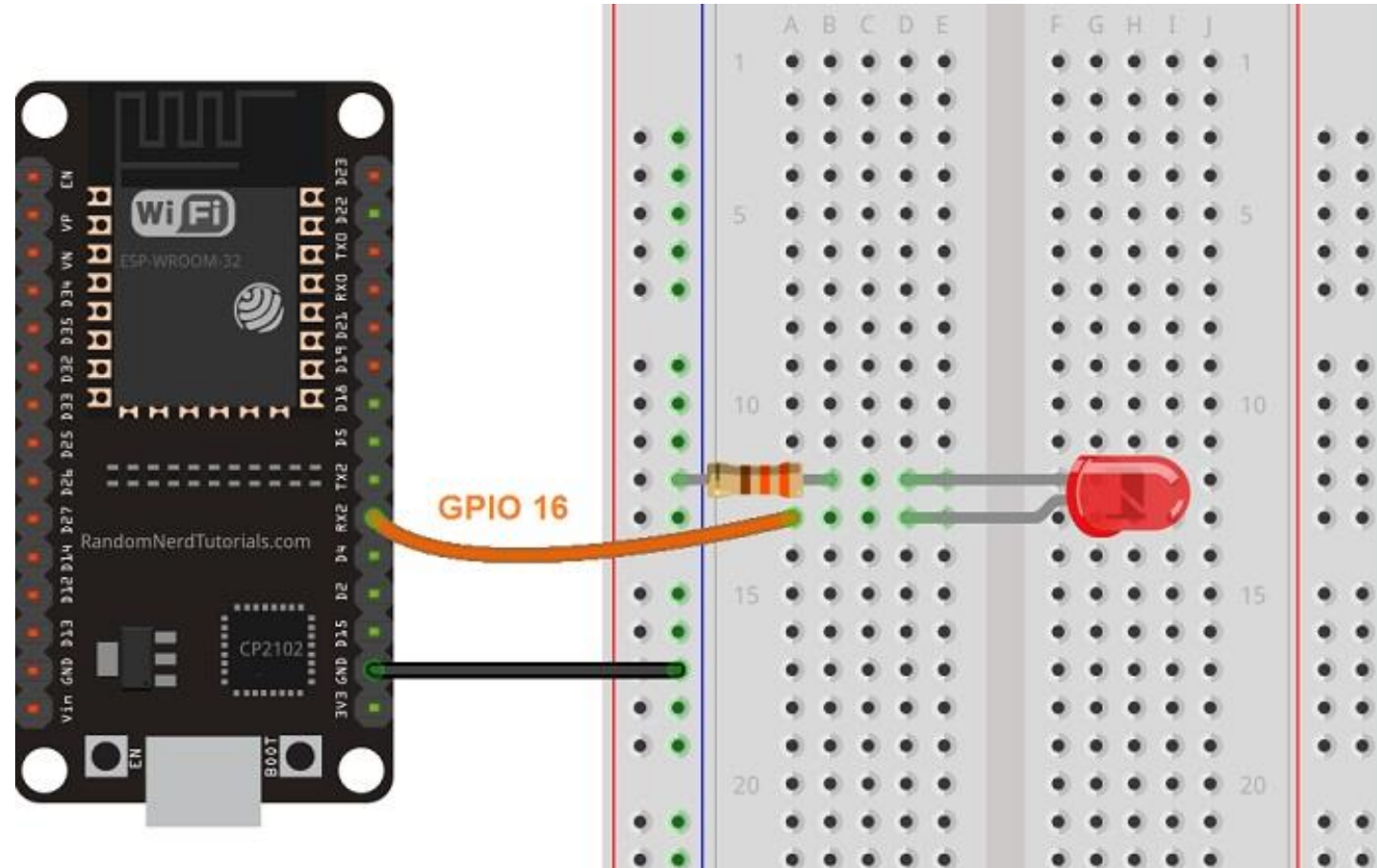
ESP32 DEVKit v1 - DOIT - Pinout

Board With 30 GPIOs



Nguyen Ngoc Le

ESP32 PWM Example



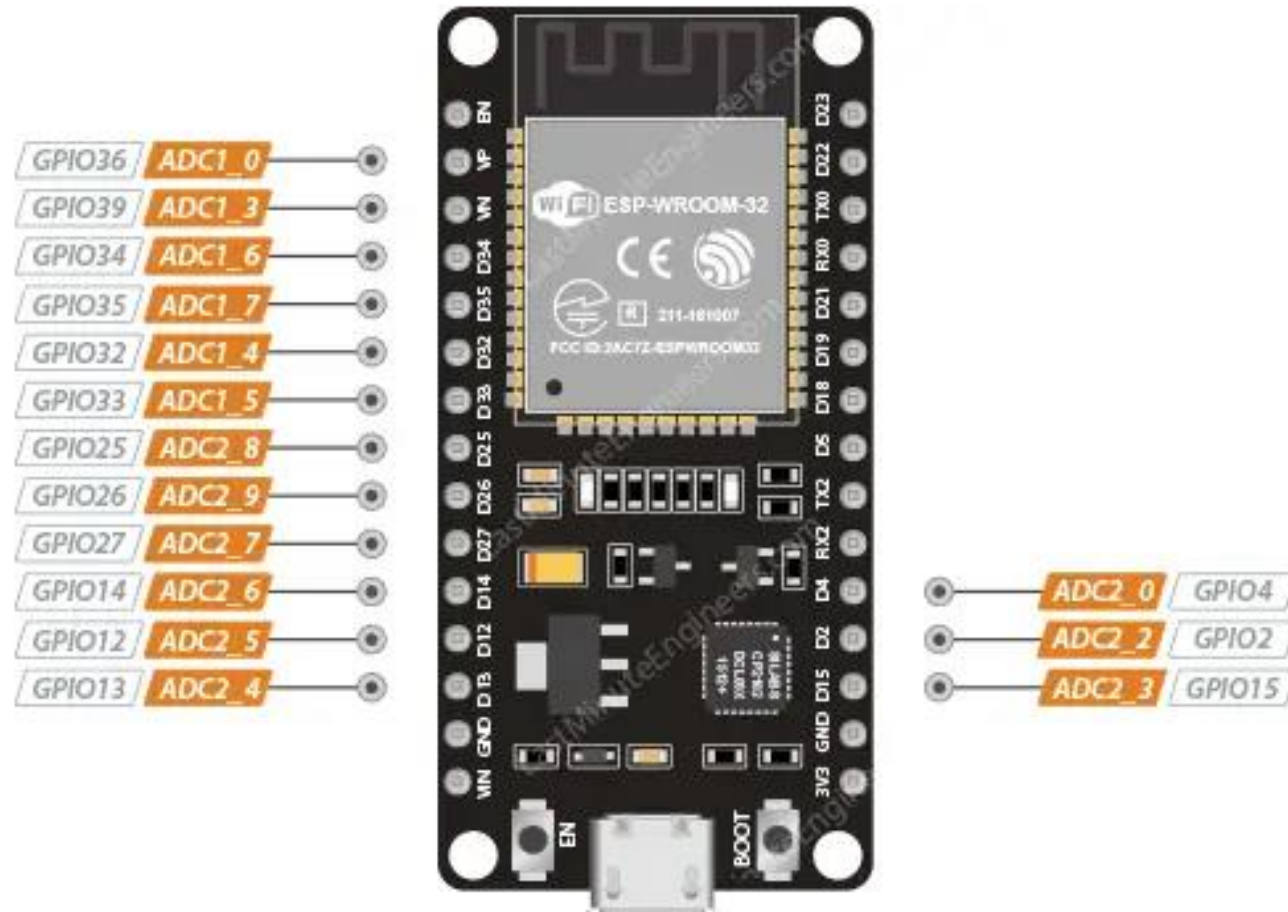
ESP32 PWM Example

```
// the number of the LED pin
const int ledPin = 16; // 16 corresponds to GPIO 16
void setup() {
  // set the LED as an output
  pinMode(ledPin, OUTPUT); }

void loop(){
  // increase the LED brightness
  for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
    // changing the LED brightness with PWM
    analogWrite(ledPin, dutyCycle);
    delay(15); }

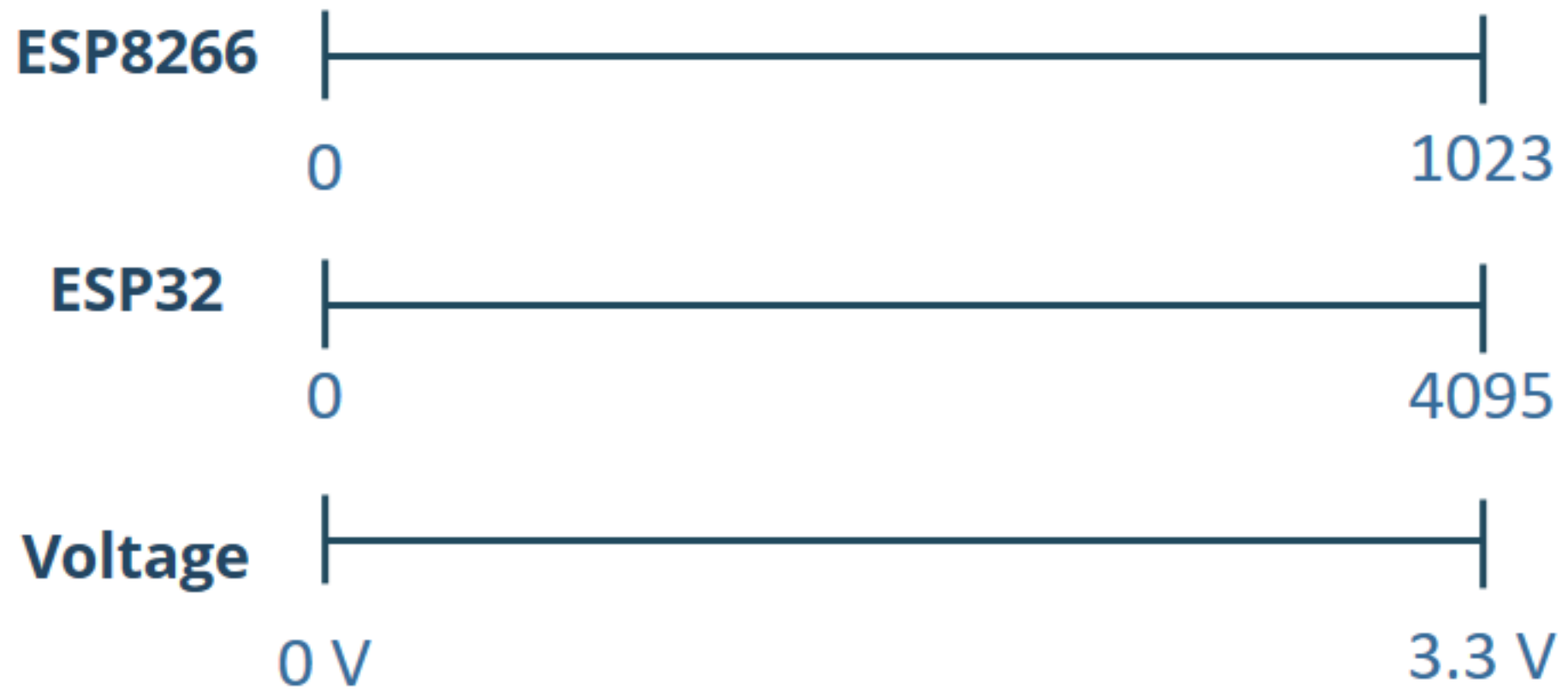
  // decrease the LED brightness
  for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
    // changing the LED brightness with PWM
    analogWrite(ledPin, dutyCycle);
    delay(15); }
}
```

ESP32 ADC

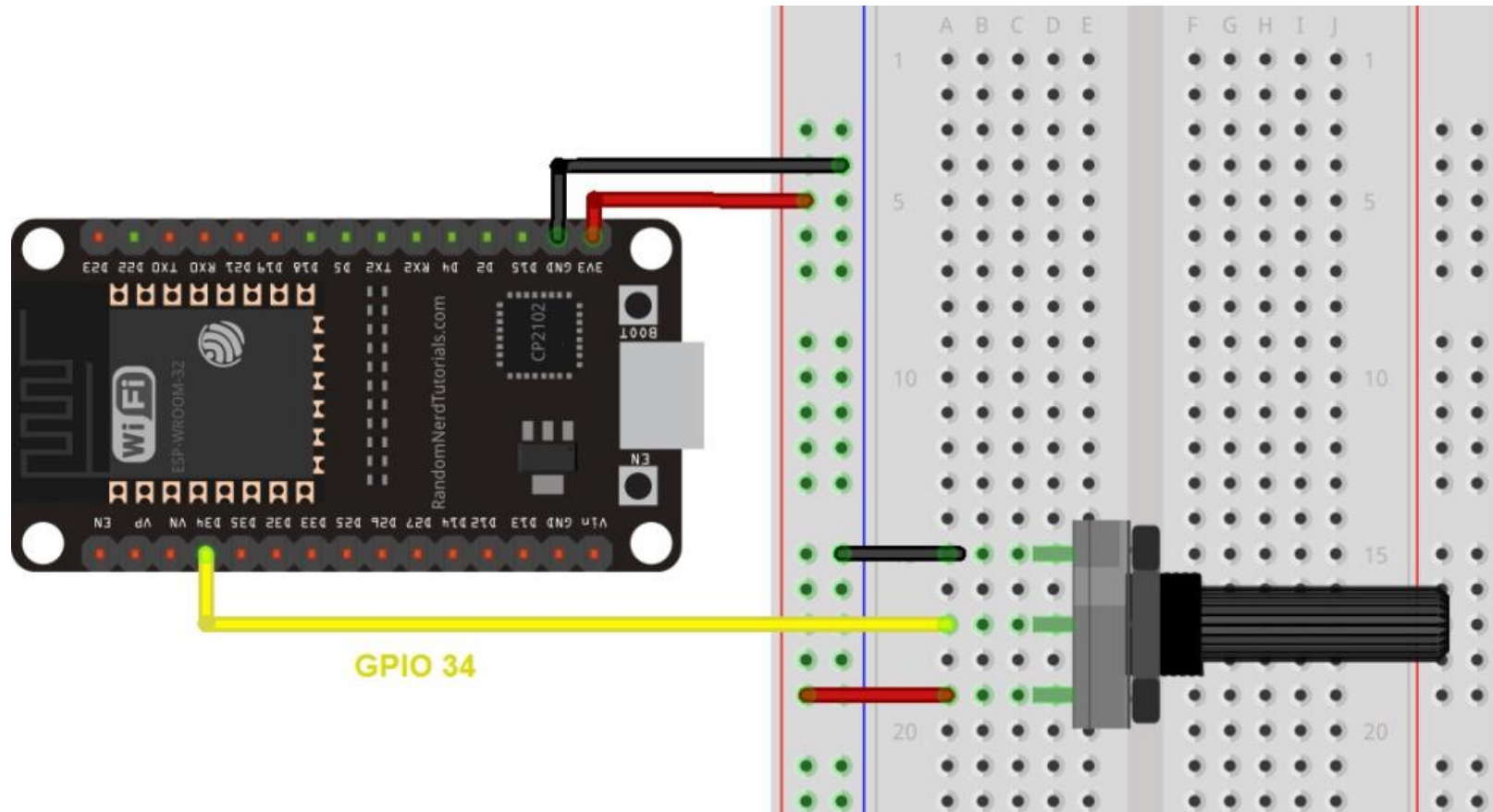


ESP32 ADC Resolution

- a **12-Bit ADC**
- Readings in the range (0 – 4095)
- Change the resolution bits
- `AnalogSetWidth(width)`: set the sample bits and resolution. It can be a value between 9 (0 – 511) and 12 bits (0 – 4095). Default is 12-bit resolution.



ESP32 ADC - Potentiometer



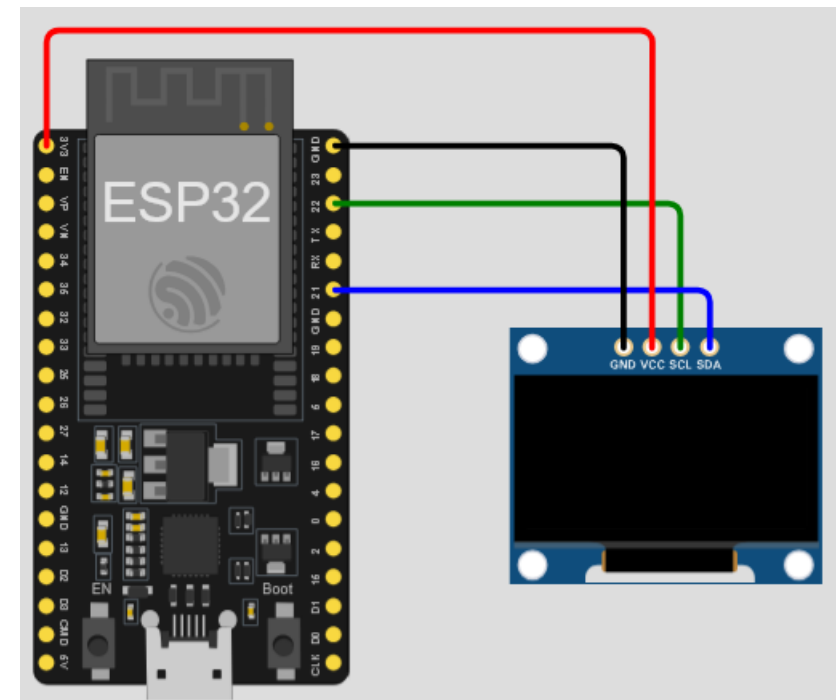
OLED

```
from machine import Pin, I2C
import ssd1306

# ESP32 Pin assignment
i2c = I2C(0, scl=Pin(22), sda=Pin(21))

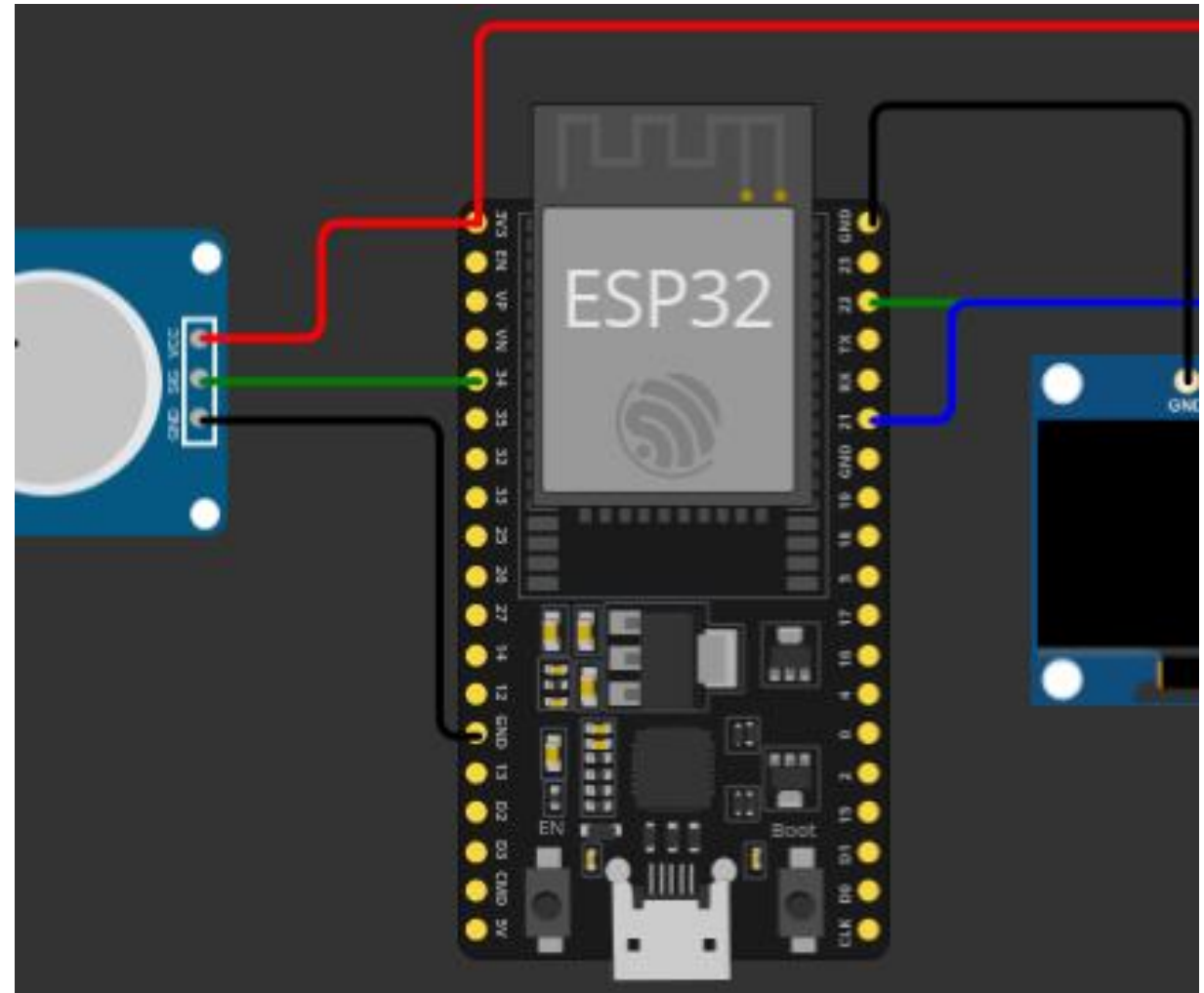
oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width,
                           oled_height, i2c)

oled.text('Hello, Wokwi!', 10, 10)
oled.show()
```



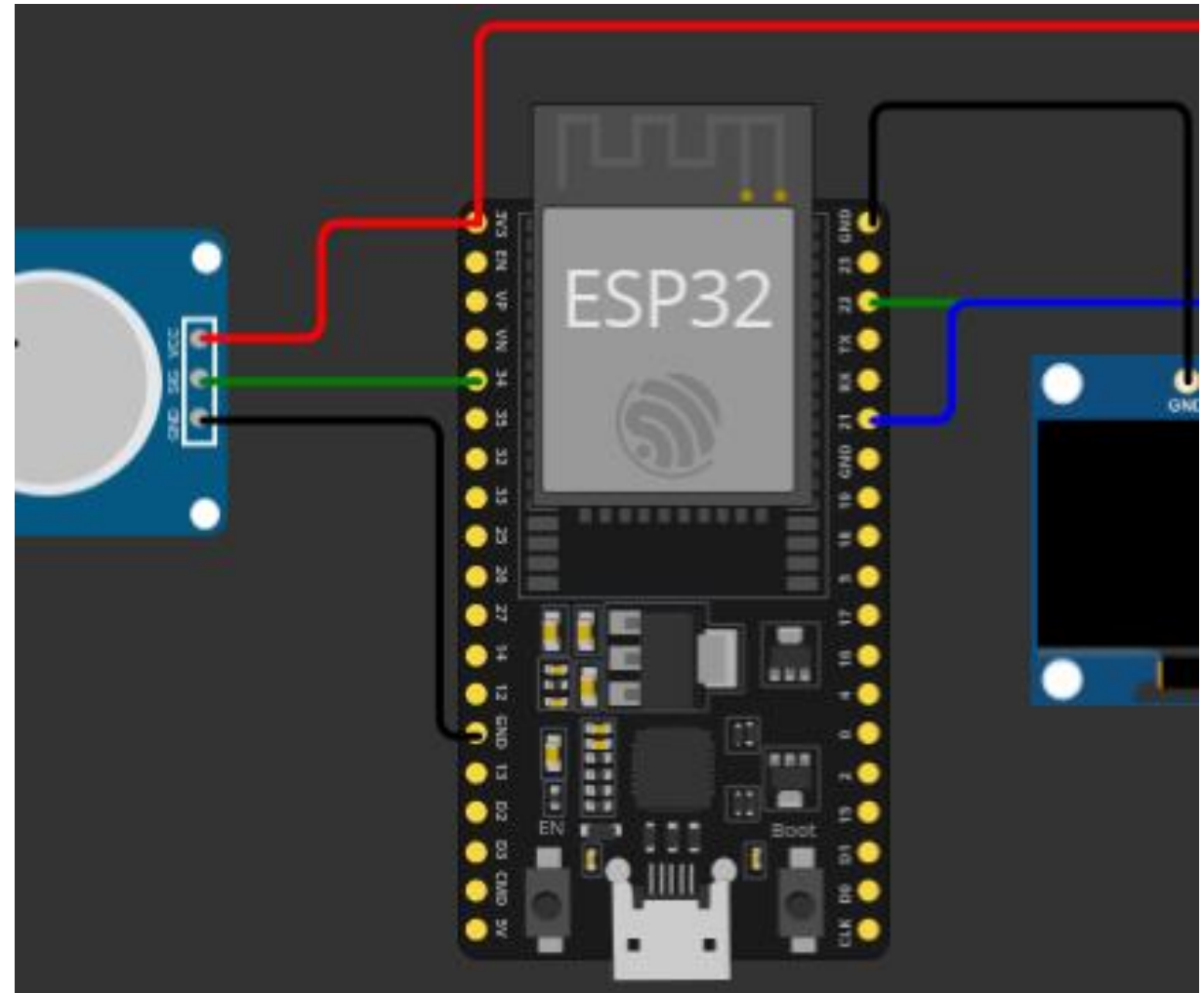
ESP32 ADC

- Display value on OLED
-



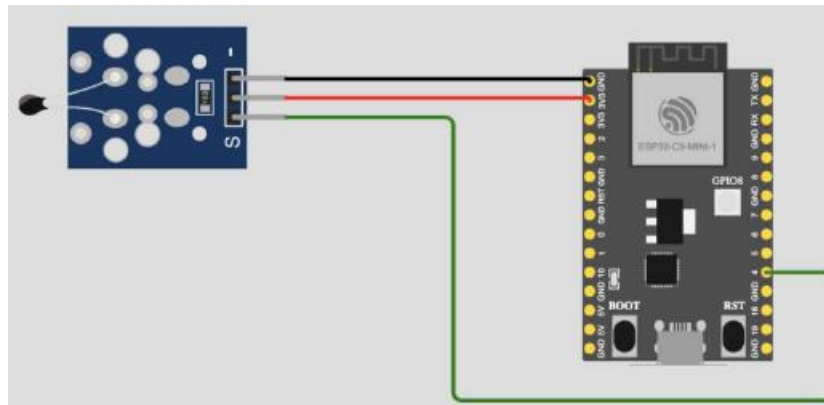
ESP32 ADC

- `pot = ADC(Pin(34))`
- `pot.atten(ADC.ATTN_11DB)#Full range: 3.3v`
- - `while True:`
 - `pot_value = pot.read()`
 - `print(pot_value)`
 - `sleep(0.1)`
-

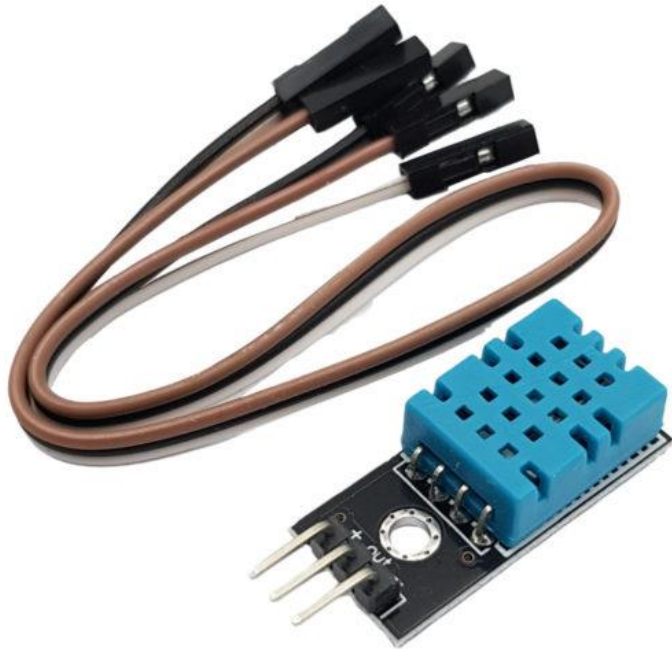


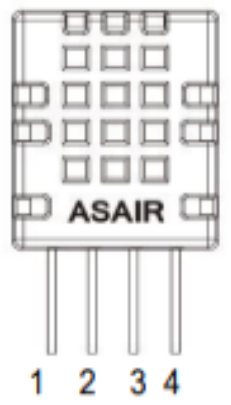
Exercise

- Components: An NTC Temperature Sensor
- The wiring schematic is depicted in Figure:
 - Temperature sensor signal pin connecting to pin gpio4
 - Temperature sensor Vcc pin to 3.3V board power.
 - Temperature sensor Gnd pin to board Gnd.



DHT20 temperature sensor



Pins	Name	Describe	
1	VDD	Power supply (2.2v to 5.5v)	
2	SDA	Serial data bidirectional port	
3	GND	Ground	
4	SCL	Serial clock bidirectional port	

Exercise

- Display temperature using DHT20 temperature sensor