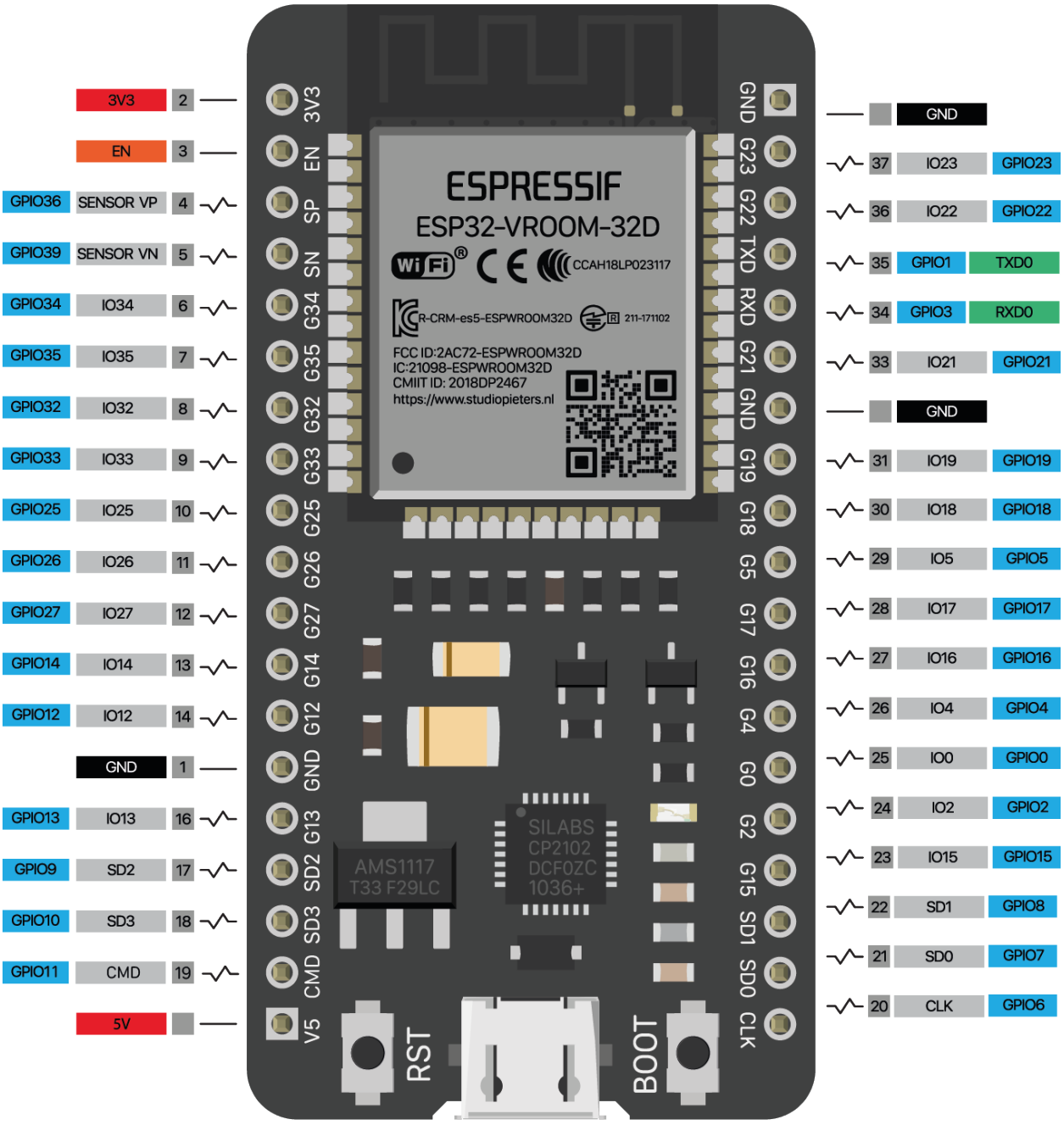
Adapted from: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

Getting Started with the ESP32

The ESP32 is a powerful microcontroller that can be used for embedded systems. There are various boards and chipsets. This tutorial covers this board:

Pinout



Adapted from: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

Powering the board

The ESP32 can be powered by one of two ways:

- USB
- connecting the 5V pin to 5V-12V

Do not power the ESP32 with both USB and 5V pin! This is dangerous.

Use only one power source at a time either:

- USB
- Injecting 5V-12V into the 5V pin

But not at the same time!

Pins

As you can see from the pinout diagram, there are many pins that serve different functions. Some are POWER, some are GROUND, some are GPIO. Let's go over these pins.

First some basics about electricity:

Electricity likes to flow from a power pin to the ground.

When the board is plugged in by USB to a standard 5V charger block or a computer, the 3V3 pin will have 3.3V available, and the 5V pin will have 5V available.

We will typically build circuits to **interface (i.e., connect)** with the board using the power pins or GPIO pins, and will complete the circuit by having all voltage eventual return to a GND pin. There are three GND pins on this board.

Adapted from: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

This board also features General-Purpose Input-Output pins, also known as GPIO. When we write software for the ESP32, we can configure any GPIO pin to be either **output** (makes 3.3V available), or **input** (can take in a voltage from .5V up to 3.3V).

It is important to know that these GPIO pins have limitations in both voltage and current. Exceeding these limitations can cause damage to that pin, making it unusable, or can cause damage to the ESP32, rendering the entire board useless.

Do not connect a load (i.e., external circuit) that demands more than 40mA (milli amps). Hence, we don't ever connect GPIO pins directly to any DC motors.

Do not input a voltage higher than 3.3V into the GPIO pins. One should always reduce the voltage to about 3V or less.

Though there are many GPIO pins on the board, not all GPIO pins can be used as **output**.

Input Only: There are six GPIO pins which can only be inputs. These cannot be configured as output pins:

- GPIO34
- GPIO35
- GPIO36
- GPIO37
- GPIO38
- GPIO39

Adapted from: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

Never use: These pins are used for other purposes, and should never be used:

- GPIO 6
- GPIO 7
- GPIO 8
- GPIO 9
- GPIO 10
- GPIO 11

Capacitive Touch: The ESP32 has 10 internal capacitive touch sensors. These can sense variations in anything that holds an electrical charge, like the human skin. So they can detect variations induced when touching the GPIOs with a finger. These pins can be easily integrated into capacitive pads and replace mechanical buttons. The capacitive touch pins can also be used to wake up the ESP32 from deep sleep.

Those internal touch sensors are connected to these GPIOs:

- T0 (GPIO 4)
- T1 (GPIO 0)
- T2 (GPIO 2)
- T3 (GPIO 15)
- T4 (GPIO 13)
- T5 (GPIO 12)
- T6 (GPIO 14)
- T7 (GPIO 27)
- T8 (GPIO 33)
- T9 (GPIO 32)

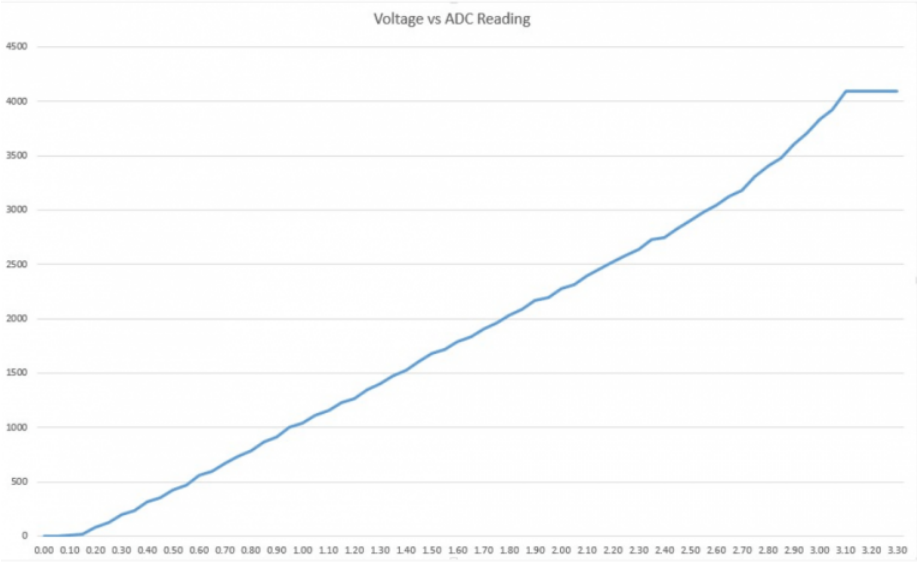
Analog to Digital Converter (ADC): The ESP32 has 18 x 12 bits ADC input channels (while the ESP8266 only has 1x 10 bits ADC). These are the GPIOs that can be used as ADC and respective channels:

- ADC1_CH0 (GPIO 36)
- ADC1_CH1 (GPIO 37)
- ADC1_CH2 (GPIO 38)
- ADC1_CH3 (GPIO 39)
- ADC1_CH4 (GPIO 32)
- ADC1_CH5 (GPIO 33)
- ADC1_CH6 (GPIO 34)
- ADC1_CH7 (GPIO 35)
- ADC2_CH0 (GPIO 4)
- ADC2_CH1 (GPIO 0)
- ADC2_CH2 (GPIO 2)
- ADC2_CH3 (GPIO 15)
- ADC2_CH4 (GPIO 13)
- ADC2_CH5 (GPIO 12)
- ADC2_CH6 (GPIO 14)
- ADC2_CH7 (GPIO 27)
- ADC2_CH8 (GPIO 25)
- ADC2_CH9 (GPIO 26)

The ADC input channels have a 12-bit resolution. This means that you can get analog readings ranging from 0 to 4095, in which 0 corresponds to 0V and 4095 to 3.3V. You can also set the resolution of your channels on the code and the ADC range.

The ESP32 ADC pins don’t have a linear behavior. You’ll probably won’t be able to distinguish between 0 and 0.1V, or between 3.2 and 3.3V.

You need to keep that in mind when using the ADC pins. You'll get a behavior similar to the one shown in the following figure.



Digital to Analog Converter (DAC): There are 2 x 8 bits DAC channels on the ESP32 to convert digital signals into analog voltage signal outputs. These are the DAC channels:

- DAC1 (GPIO25)
- DAC2 (GPIO26)

RTC GPIOs: There is RTC GPIO support on the ESP32. The GPIOs routed to the RTC low-power subsystem can be used when the ESP32 is in deep sleep. These RTC GPIOs can be used to wake up the ESP32 from deep sleep when the Ultra Low Power (ULP) co-processor is running. The following GPIOs can be used as an external wake up source.

- RTC_GPIO0 (GPIO36)
- RTC_GPIO3 (GPIO39)
- RTC_GPIO4 (GPIO34)
- RTC_GPIO5 (GPIO35)

- RTC_GPIO6 (GPIO25)
- RTC_GPIO7 (GPIO26)
- RTC_GPIO8 (GPIO33)
- RTC_GPIO9 (GPIO32)
- RTC_GPIO10 (GPIO4)
- RTC_GPIO11 (GPIO0)
- RTC_GPIO12 (GPIO2)
- RTC_GPIO13 (GPIO15)
- RTC_GPIO14 (GPIO13)
- RTC_GPIO15 (GPIO12)
- RTC_GPIO16 (GPIO14)
- RTC_GPIO17 (GPIO27)

PWM

The ESP32 LED PWM controller has 16 independent channels that can be configured to generate PWM signals with different properties. All pins that can act as outputs can be used as PWM pins (GPIOs 34 to 39 can't generate PWM).

To set a PWM signal, you need to define these parameters in the code:

- Signal's frequency;
- Duty cycle;
- PWM channel;
- GPIO where you want to output the signal.

I2C: The ESP32 has two I2C channels and any pin can be set as SDA or SCL. When using the ESP32 with the Arduino IDE, the default I2C pins are:

- GPIO 21 (SDA)

- GPIO 22 (SCL)

If you want to use other pins when using the wire library, you just need to call:

```
Wire.begin(SDA, SCL);
```

SPI

By default, the pin mapping for SPI is:

SPI	MOSI	MISO	CLK	CS
VSPI	GPIO 23	GPIO 19	GPIO 18	GPIO 5
HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

Interrupts

All GPIOs can be configured as interrupts.

Strapping Pins

The ESP32 chip has the following strapping pins:

- GPIO 0
- GPIO 2
- GPIO 4
- GPIO 5 (must be HIGH during boot)
- GPIO 12 (must be LOW during boot)
- GPIO 15 (must be HIGH during boot)

These are used to put the ESP32 into bootloader or flashing mode. On most development boards with built-in USB/Serial, you don’t need to

worry about the state of these pins. The board puts the pins in the right state for flashing or boot mode. More information on the ESP32 Boot Mode Selection can be found here.

However, if you have peripherals connected to those pins, you may have trouble trying to upload new code, flashing the ESP32 with new firmware, or resetting the board. If you have some peripherals connected to the strapping pins and you are getting trouble uploading code or flashing the ESP32, it may be because those peripherals are preventing the ESP32 from entering the right mode. Read the Boot Mode Selection documentation to guide you in the right direction. After resetting, flashing, or booting, those pins work as expected.

Pins HIGH at Boot

Some GPIOs change their state to HIGH or output PWM signals at boot or reset. This means that if you have outputs connected to these GPIOs you may get unexpected results when the ESP32 resets or boots.

- GPIO 1
- GPIO 3
- GPIO 5
- GPIO 6 to GPIO 11 (connected to the ESP32 integrated SPI flash memory – not recommended to use).
- GPIO 14
- GPIO 15

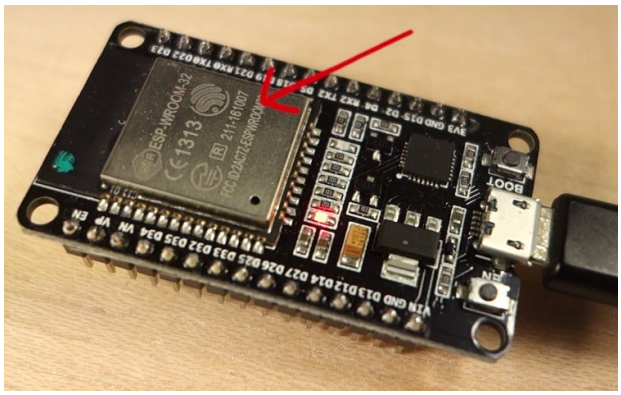
Adapted from: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

Enable (EN)

Enable (EN) is the 3.3V regulator's enable pin. It's pulled up, so connect to ground to disable the 3.3V regulator. This means that you can use this pin connected to a pushbutton to restart your ESP32, for example.

ESP32 Built-In Hall Effect Sensor

The ESP32 also features a built-in hall effect sensor that detects changes in the magnetic field in its surroundings.



A hall effect sensor can detect variations in the magnetic field in its surroundings. The greater the magnetic field, the greater the sensor's output voltage. The hall effect sensor can be combined with a threshold detection to act as a switch, for example. Additionally, hall effect sensors are mainly used to:

- Detect proximity;
- Calculate positioning;
- Count the number of revolutions of a wheel;
- Detect a door closing;
- And much more.