ESP32 Pinout Reference: Which GPIO pins should you use?

Specific GPIO works in the same way regardless of the development board you’re using.

The ESP32 Peripherals Include:

* 18 Analog-to-Digital Converter (ADC) channels.
* 3 SPI interfaces.
* 3 UART interfaces.
* 2 I2C interfaces.
* 16 PWM output channels.
* 2 Digital-to-Analog Converters (DAC).
* 2 I2S interfaces.
* 10 Capacitive sensing GPIOs.

The ADC and DAC are assigned to specific static pins. However, you can decide which pins are UART, I2C, SPI, PWM, etc-you just need to assign them in the code. This is possible due to the ESP32 chip’s multiplexing feature.

GPIOs 34 t 39 are GPIs- input only pins. These pins don’t have internal pull-up or pull-down resistors. They can’t be used as outputs, so these pins only function as inputs.

GPIO 6 to GPIO 11 are exposed in some ESP32 development boards. However, these pins are connected to the integrated SPI flash on the ESP-WROOM-32 chip and are not recommended for other uses.

Capacitive touch GPIOs.

The ESP32 has 10 internal capacitive touch sensors. These can sense variations in anything that holds an electrical charge, like the human skin.

Those internal touch sensors are connected to these GPIos.

GPIO 4,0,2,15,13,12,14,27,33 and 32.

**ADC.**

The sep32 has 18\*12 bits ADC input channels. These are the GPIOs that can be used as ADC and respective channels.

GPIO – 36-39

32-35

4,0,2,15,13,12,14,27,25 and 26.

**DAC**

There are 2\*8 bits DAC channels on the ESP32 to convert digital signals to 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-processing is running.

**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 suing the wire library, you just need to call:

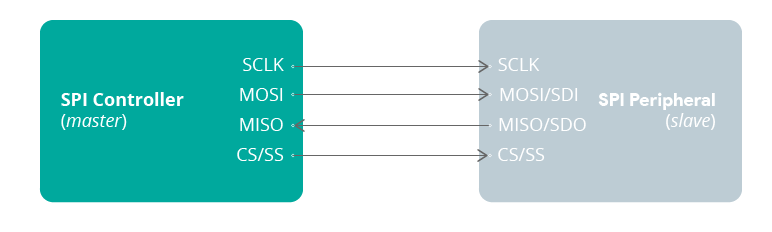
Wire.begin(SDA, SCL);

**SPI.**

**Serial Peripheral Interface.**

This is a synchronous serial data protocol used by microcontrollers to communicate with one or more peripherals. For example, your ESP32 board-communicating with a sensor that supports SPI or with another microcontroller.

In an SPI communication, there is always a controller that controls the peripheral devices. Data can be sent and received simultaneously. This means that the master can send data to a slave, and a slave can send data to the master at the same time.



You have only one master, which will be a microcontroller (the ESP32) but you can have multiple slaves. A slave can be a sensor, a display, a microSD card, etc.

This means you have an ESP32 connected to multiple sensors, but the same sensor can’t be connected to multiple ESP32 boards simultaneously.

For SPI communication you need four lines:

* MISO: Master In Slave Out.
* MOSI: Master Out Slave In.
* SCK: Serial Clock.
* CS/SS: Chip Select.

Interrupts.

All GPIOs can be configured as interrupts.

**Strapping Pins.**

The ESP32 chip has the following strapping pins:

* GPIO 0 (must be low to enter boot mode)
* GPIO 2 (must be floating or LOW during boot)
* 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 pf these pins. The board puts the pins in the right state for flashing or boot mode.

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.

**Pins HIGH at Boot.**

Some GPIOs change their state to HIGH or output PWM signals at boot or reset. Thus 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.

**Enable (EN)**

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

**GPIO current drawn.**

The absolute maximum current drawn per GPIO is 40mA according to the “Recommended Operating Conditions” section in the ESP32 datasheet.

**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.