ESP32: Internal Details and Pinout

by Fernando Koyanagi

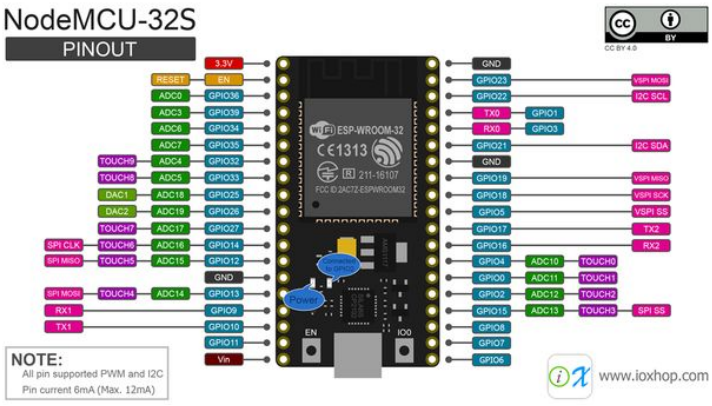
In this article, we will talk about the internal details and the pinning of ESP32. I will show you how to correctly identify the pins by looking at the datasheet, how to identify which of the pins work as an OUTPUT / INPUT, how to have an overview about the sensors and peripherals that the ESP32 offers us, in addition to the boot. Therefore, I believe that, with the video below, I will be able to answer several questions that I have received in messages and comments about the ESP32 references, among other information.

https://youtu.be/xmiL49UjfzI

Step 1: NodeMCU ESP-WROOM-32

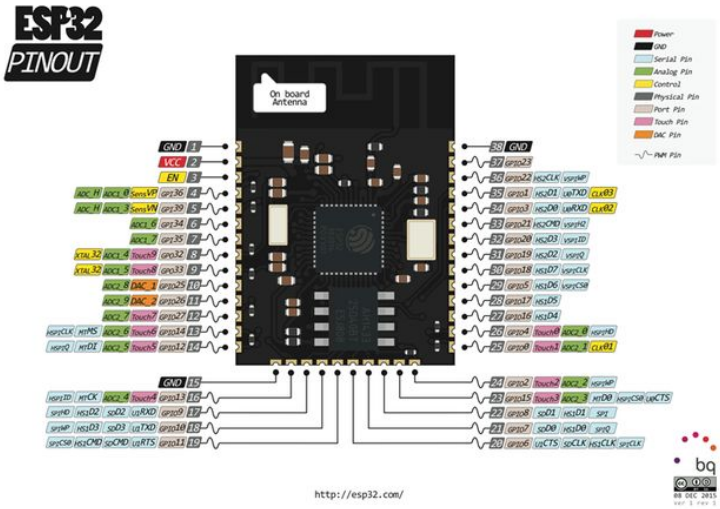
Here we have the PINOUT of the

WROOM-32 that serves as a good reference for when you program. It is important to pay attention to General Purpose Input / Output (GPIOs), that is, programmable data input and output ports, which can still be an AD converter or a Touch pin, such as GPIO4, for example. This also occurs with the Arduino, where the input and output pins can also be PWM.



Step 2: ESP-WROOM-32

In the image above, we have the ESP32 itself. There are several types of inserts with different characteristics according to the manufacturer.



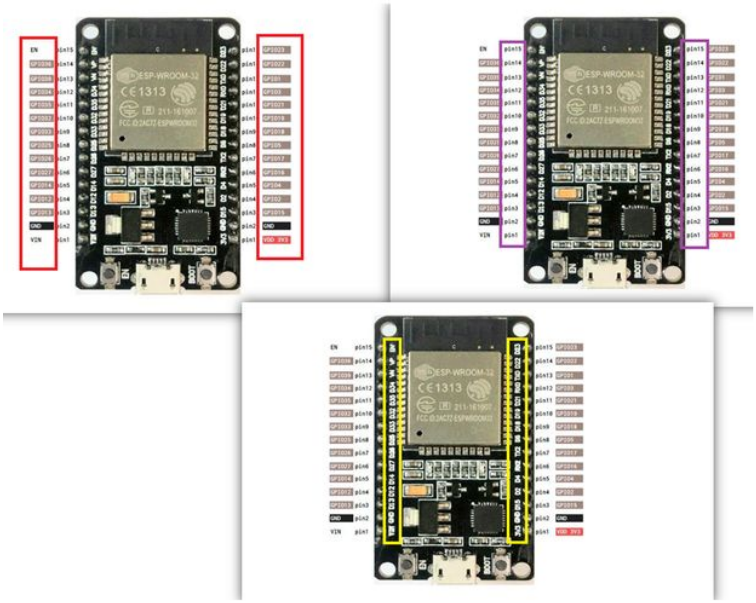
Step 3: But, What Is the Correct Pinout for Me to Use for My ESP32?

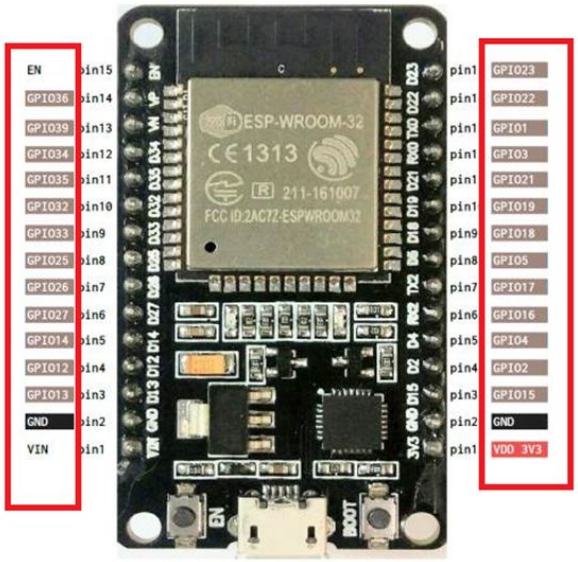
ESP32 is not difficult. It's so easy that we can say that there is no didactic concern in your environment. However, we need to be didactic, yes. If you want to program in Assembler, that is okay. But, engineering time is expensive. So, if everything that is a technology supplier gives you a tool that takes time to understand its workings, this can easily become a problem for you, because all this will increase the engineering time, while the product is becoming increasingly expensive. This explains my preference for easy things, those that can make our day to day easier, because time is important, especially in today's busy world.

Returning to the ESP32, in a datasheet, as in the one above, we have the correct pin identification in the highlights. Often, the label on the chip doesn't match the actual number of the pin, as we have three situations: the GPIO, the serial number, and also the code of the card itself.

As shown in the example below, we have a connection of a LED in the ESP and the correct mode of configuration:

Notice that the label is TX2, but we must follow the correct identification, as highlighted in the previous image. Therefore, the correct identification of the pin will be 17. The image shows just how close the code should stay.

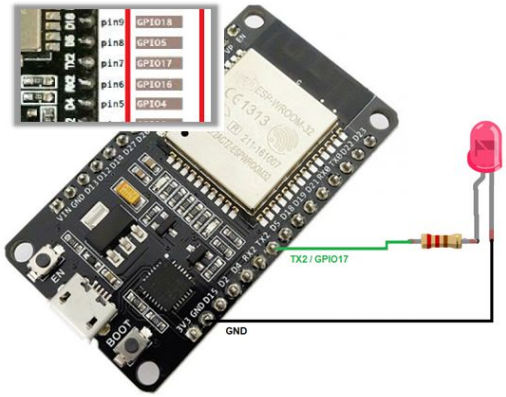





```ino
const int pinoLED = 17; //pino que o LED foi conectado

void setup() {
   pinMode(pinoLED, OUTPUT); //define o pino 17 como saida
}

void loop() {
  //inverte o estado do LED
  digitalWrite(pinoLED, !digitalRead(pinoLED));
  delay(1000);
}
```



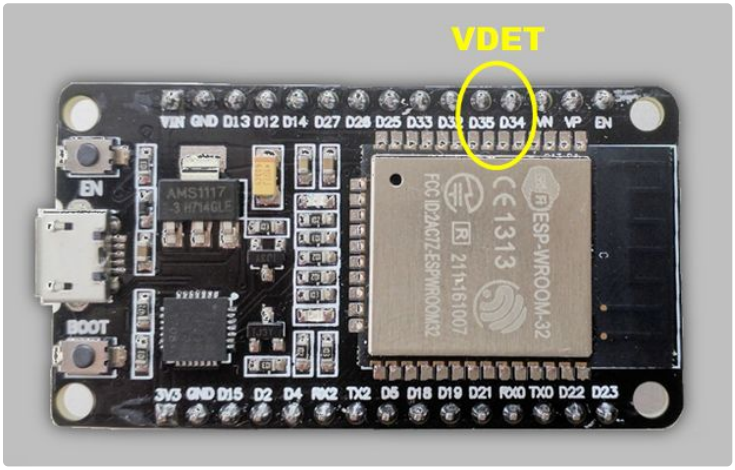
Step 4: INPUT / OUTPUT

When performing INPUT and OUTPUT tests on the pins, we obtained the following results:

INPUT didn’t work only on GPIO0.

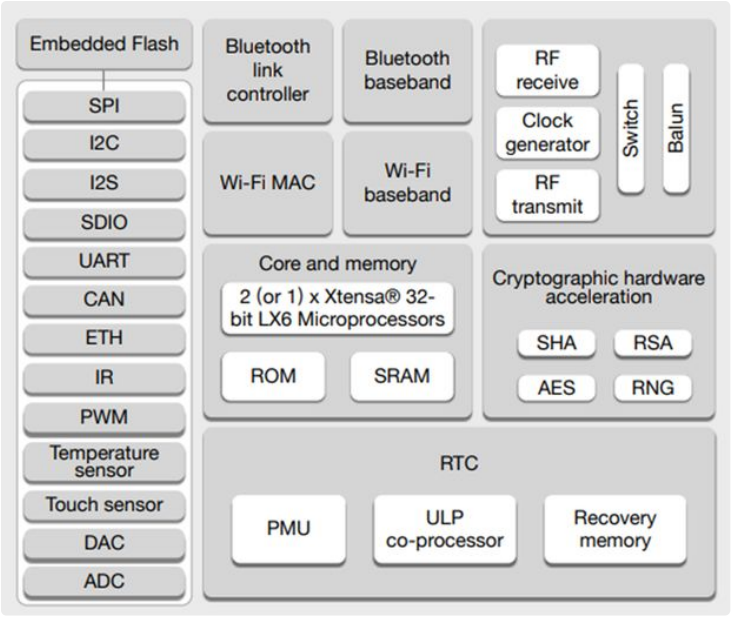
OUTPUT didn’t work only on the GPIO34 and GPIO35 pins, which are VDET1 and VDET2, respectively.

* The VDET pins belong to the power domain of the RTC. This means that they can be used as ADC pins and that the ULP-coprocessor can read them. They can only be entries and never exits.



Step 5: Block Diagram

This diagram shows that the ESP32 has dual core, a chip area that controls WiFi, and another area that controls Bluetooth. It also has hardware acceleration for encryption, which allows connection to LoRa, a long-distance network that allows for a connection of up to 15km, using an antenna. We also observe the clock generator, real time clock, and other points involving, for example, PWM, ADC, DAC, UART, SDIO, SPI, among others. This all makes the device quite complete and functional.



Step 6: Peripherals and Sensors

The ESP32 has 34 GPIOs that can be assigned to various functions, such as:

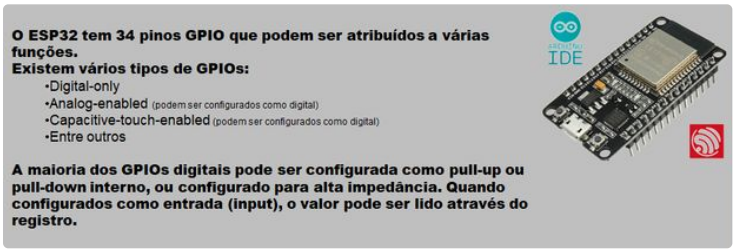
Digital-only;

Analog-enabled (can be configured as digital);

Capacitive-touch-enabled (can be configured as digital);

And others.

It is important to note that most digital GPIOs can be configured as internal pull-up or pull-down, or configured for high impedance. When set as input, the value can be read through the register.



Step 7: GPIO

Analog-to-Digital Converter (ADC)

The Esp32 integrates 12-bit ADCs and supports measurements on 18 channels (analog-enabled pins). The ULP-coprocessor in the ESP32 is also designed to measure voltages while operating in sleep mode, which allows for low power consumption. The CPU can be awakened by a threshold setting and / or through other triggers.

Digital-to-Analog Converter (DAC)

Two 8-bit DAC channels can be used to convert two digital signals to two analog voltage outputs. These dual DACs support the power supply as an input voltage reference and can drive other circuits. Dual channels support independent conversions.

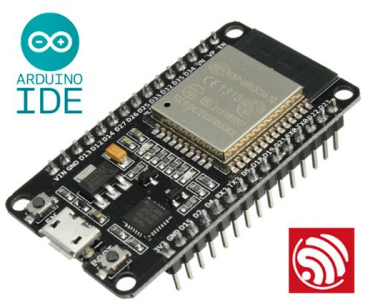
Step 8: Sensors

Touch Sensor

The ESP32 has 10 capacitive detection GPIOs that detect induced variations when touching or approaching a GPIO with a finger or other objects.

The ESP32 also has a Temperature Sensor and an Internal Hall Sensor, but to work with them, you have to change the settings of the registers. For more details, see the technical manual through the link:

https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf



Capacitive-sensing signal name	Pin name	
T0	GPIO4	
T1	GPIO0	
T2	GPIO2	
T3	MTDO	GPIO15
T4	MTCK	GPIO13
T5	MTD1	GPIO12
T6	MTMS	GPIO14
T7	GPIO27	GPIO27
T8	32K_XN	GPIO33
T9	32K_XP	GPIO32

Algumas placas (como essa da imagem, escondem o GPIO0)

Step 9: Watchdog

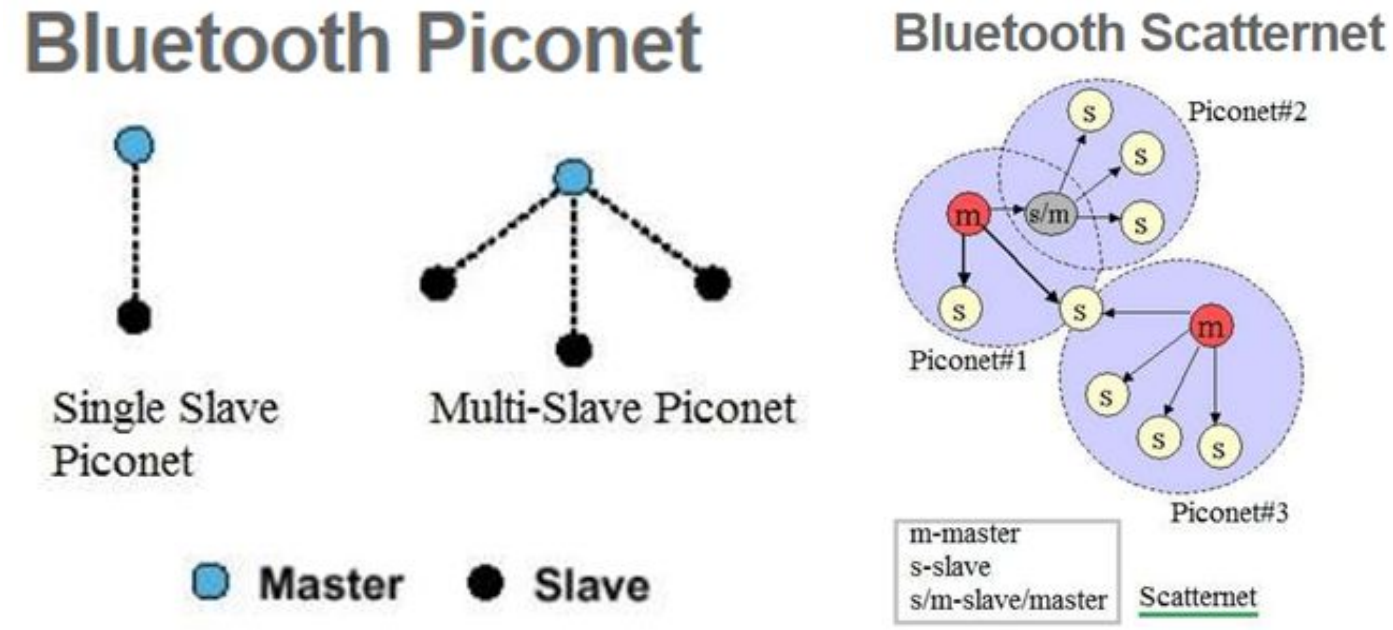
The ESP32 has three surveillance timers: one on each of the two timer modules (called the Primary Watchdog Timer, or MWDT) and one on the RTC module (called RTC Watchdog Timer or RWDT).

Step 10: Bluetooth

Bluetooth Interface v4.2 BR / EDR and Bluetooth LE (low energy)

The ESP32 integrates a Bluetooth connection controller and Bluetooth baseband, which perform baseband protocols and other low-level link routines, such as modulation / demodulation, packet processing, bit-stream processing, frequency hopping, etc.

The connection controller operates in three main states: standby, connection, and sniff. It allows multiple connections and other operations, such as inquiry, page, and secure simple pairing, and thus allows for Piconet and Scatternet.



Step 11: Boot

On many development boards with embedded USB / Serial, esptool.py can automatically reset the board to boot mode.

ESP32 will enter the serial boot loader when the GPIO0 is kept low on the reset. Otherwise, it will run the program in flash.

GPIO0 has an internal pullup resistor, so if it is without a connection, it will go high.

Many boards use a button labeled "Flash" (or "BOOT" on some Espressif development boards) that leads the GPIO0 downward when pressed.

GPIO2 should also be left unconnected / floating.

In the image above, you can see a test that I performed. I put the oscilloscope on all the pins of the ESP to see what happened when it was turned on. I discovered that when I get a pin, it generates oscillations of 750 microseconds, as shown in the highlighted area on the right side. What can we do about this? We have several options, like giving a delay with a circuit with a transistor, a door expander, for example. I point out that GPIO08 is reversed. The oscillation exits upwards and not downwards.

Another detail is that we have some pins that start in High, and others in Low. Therefore, this PINOUT is a reference to when the ESP32 turns on, especially when you are working with a load to trigger, for example, a triac, a relay, a contactor, or some power.

GPIO0 Input	Mode
Low/GND	ROM serial bootloader for esptool.py
High/VCC	Normal execution mode