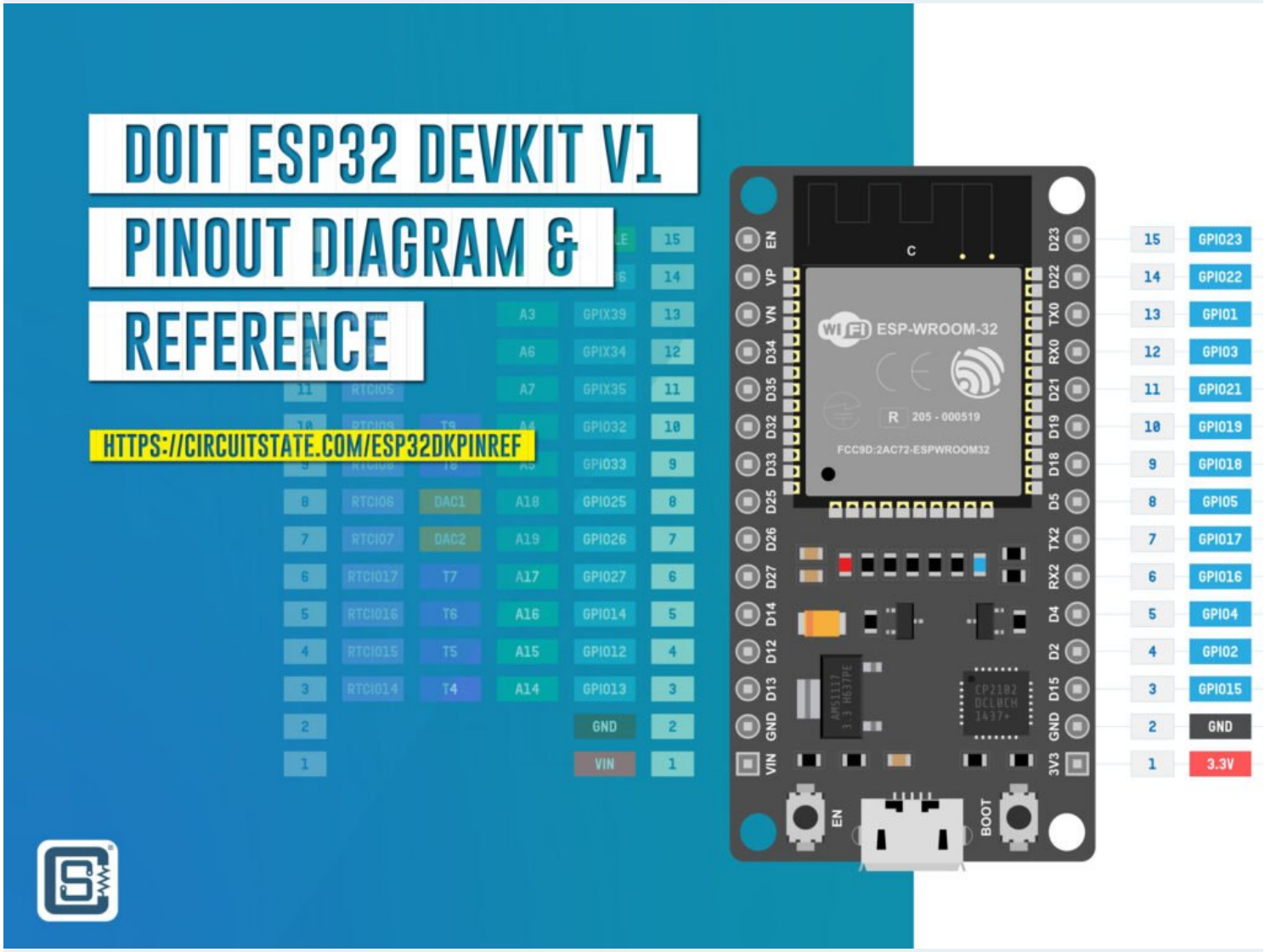


DOIT ESP32 DevKit V1 Pinout Diagram & Reference

The **DOIT ESP32 DevKit V1** is probably the most famous development board based on the equally popular ESP32 Wi-Fi SoC from *Espressif*. In fact, the DevKit V1 is more popular than any official board from Espressif. On this page, you will find a beautiful pinout diagram crafted by *CIRCUITSTATE* and other pin references for the ESP32 DevKit V1 board. If you are new to the ESP32 Wi-Fi and Bluetooth SoC, we have a great tutorial to get you started.

Contents

1. Pinout Diagram
   1.1 PNG
   1.2 PDF
2. Pinout Reference
   2.1 Power & Control
   2.2 GPIO
   2.3 Strapping
   2.4 Pull-Up & Pull-Down
   2.5 LED
   2.6 UART
   2.7 SPI
   2.8 ADC
   2.9 DAC
   2.10 Touch Sensor
   2.11 I2C

2.12 PWM

2.13 I2S

2.14 CAN/TWAI

2.15 JTAG

2.16 External Interrupts

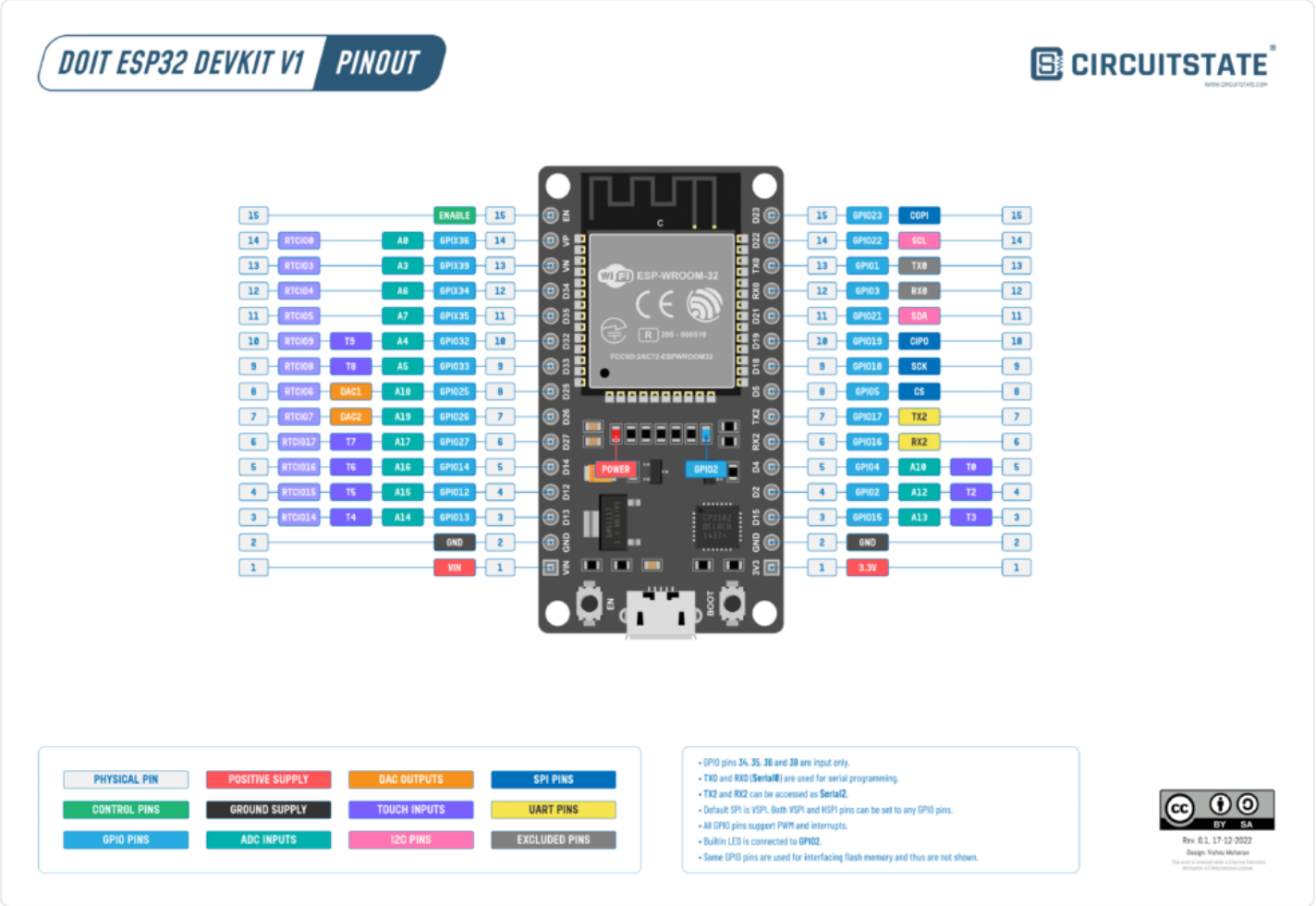
2.17 Ethernet MAC

3. Links

4. Short Link

Latest Revision: r0.1, 17-12-2022
Design by: *Vishnu Mohanan*

Pinouts are based on the latest documentation from Espressif. While we try our best to be accurate and up-to-date here, we can not guarantee correctness. Please also double-check the pin assignments with that from the official documentation. If you found any errors here, please let us know in the comments. We will update our designs ASAP.



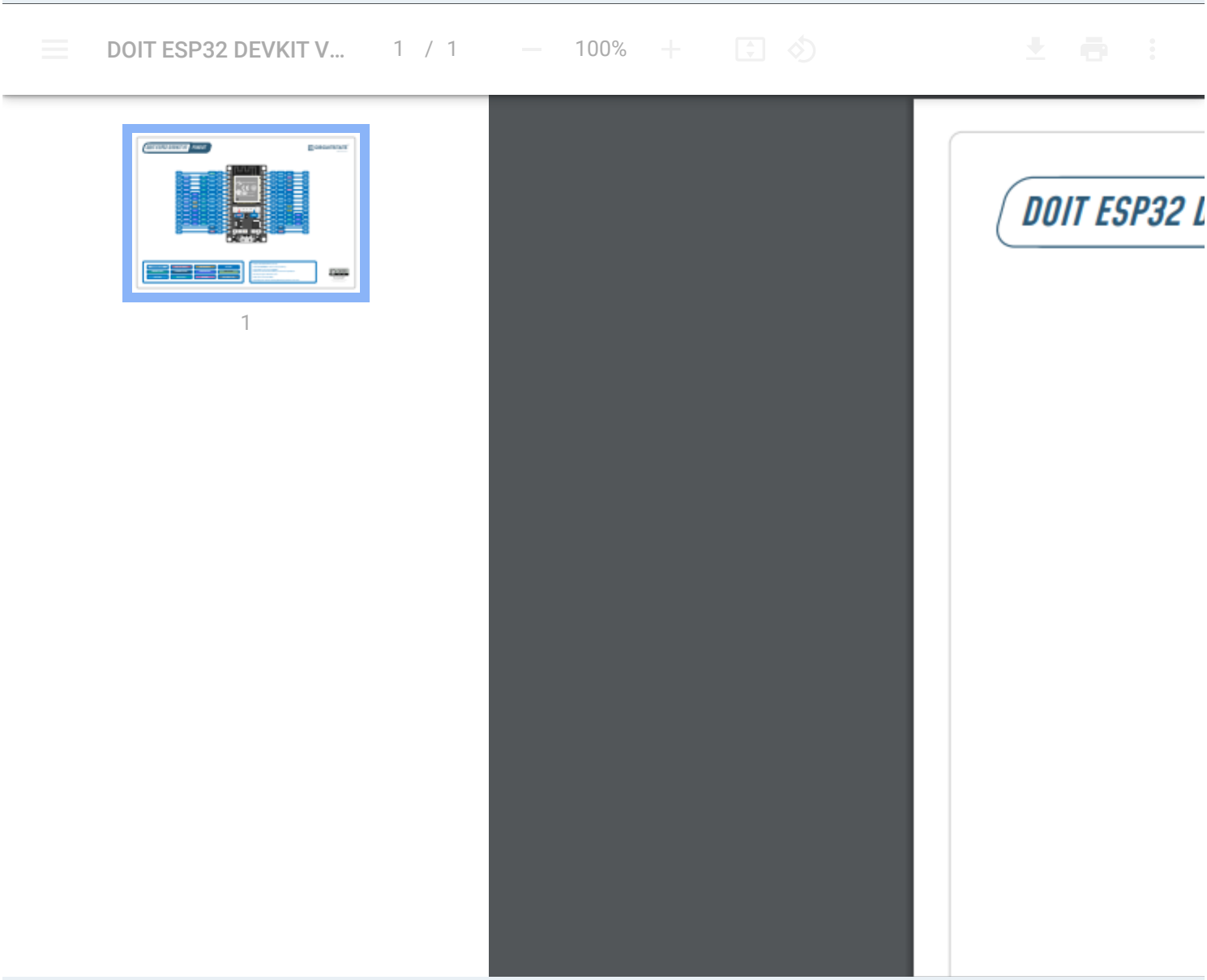
DOIT ESP32 DevKit V1 pinout diagram.

Even though this pinout is specifically created based on the particular board we have in hand, there are many variants of the same board in the market. The first ever ESP32 DevKit V1 we bought has the DOIT company logo, the line “ESP32 DEVKIT V1” and the website address www.doit.am. But that variant has all pins of the **ESP-WROOM-32** module broken out including the SPI interface used for the flash memory. Most of the boards we can get from online shops today do not have those unused pins which shorten the board length. Regardless of the type of board you have, our pinout diagrams will be still applicable as long as they use an ESP32 chip or module.

Also since these boards did not come with official pin numbering or pinouts, there is no consensus over where to start counting. So to make it easier for you to identify the pins by simply counting, we have numbered each row of pins on either side from 1 to 15. But you should not rely on the pin numbering alone. Always use the GPIO number to identify the pins and their functions. We have limited this pinout diagram to include only those references that are relevant to the **Arduino ESP32**

core. We have not used any of the official pin names. For that, you can always refer to the official documentation.

PDF preview may not load on mobile devices. Click the link to open an interactive preview, or download it directly.



ESP32-DevKit-V1-Pinout-r0.1-CIRCUITSTATE-Electronics Download

Pinout Reference

Power & Control

There are two positive power supply input pins and one control pin on the DOIT ESP32 DevKit V1.

Pin Name	Function
VIN	The input of the 3.3V positive voltage regulator. Supply voltage in the range of 4 to 12V.
3.3V	Output from the voltage regulator. You can also supply 3.3V to this pin if you have one. But do not supply both VIN and 3V3 together.
GND	Ground (Negative) supply pins.
ENABLE	This is the reset pin. Connecting this pin to GND will reset the ESP32. This pin is normally pulled-up. The EN button will pull it LOW when you press it.

ESP32 power supply and reset pins

GPIO

There are **34 GPIO** pins available on the ESP32 chip. These pins are named from `0` to `39`. But doesn't that make the count 40? No, because GPIOs 20, 24, 28, 29, 30, and 31 are not accessible. Also, not all of these pins are broken from the module or the board. But it is good to have a reference to know what is what. Below is the default ESP32 GPIO function matrix taken from the official documentation. Note that, many of the peripheral functions can be mapped to any of the GPIO pins using the **GPIO Mux** block of the ESP32.

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Function 5	Reset	No
0	GPIO0	GPIO0	CLK_OUT1	GPIO0	–	–	EMAC_TX_CLK	3	R
1	U0TXD	U0TXD	CLK_OUT3	GPIO1	–	–	EMAC_RXD2	3	–
2	GPIO2	GPIO2	HSPIWP	GPIO2	HS2_DATA0	SD_DATA0	–	2	R
3	U0RXD	U0RXD	CLK_OUT2	GPIO3	–	–	–	3	–
4	GPIO4	GPIO4	HSPIHD	GPIO4	HS2_DATA1	SD_DATA1	EMAC_TX_ER	2	R
5	GPIO5	GPIO5	VSPICS0	GPIO5	HS1_DATA6	–	EMAC_RX_CLK	3	–
6	SD_CLK	SD_CLK	SPICLK	GPIO6	HS1_CLK	U1CTS	–	3	–

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Function 5	Reset	No
7	SD_DATA_0	SD_DATA0	SPIQ	GPIO7	HS1_DATA0	U2RTS	—	3	—
8	SD_DATA_1	SD_DATA1	SPID	GPIO8	HS1_DATA1	U2CTS	—	3	—
9	SD_DATA_2	SD_DATA2	SPIHD	GPIO9	HS1_DATA2	U1RXD	—	3	—
10	SD_DATA_3	SD_DATA3	SPIWP	GPIO10	HS1_DATA3	U1TXD	—	3	—
11	SD_CMD	SD_CMD	SPICS0	GPIO11	HS1_CMD	U1RTS	—	3	—
12	MTDI	MTDI	HSPIQ	GPIO12	HS2_DATA2	SD_DATA2	EMAC_TXD3	2	R
13	MTCK	MTCK	HSPID	GPIO13	HS2_DATA3	SD_DATA3	EMAC_RX_ER	2	R
14	MTMS	MTMS	HSPICLK	GPIO14	HS2_CLK	SD_CLK	EMAC_TXD2	3	R
15	MTDO	MTDO	HSPICS0	GPIO15	HS2_CMD	SD_CMD	EMAC_RXD3	3	R
16	GPIO16	GPIO16	—	GPIO16	HS1_DATA4	U2RXD	EMAC_CLK_OUT	1	—
17	GPIO17	GPIO17	—	GPIO17	HS1_DATA5	U2TXD	EMAC_CLK_180	1	—
18	GPIO18	GPIO18	VSPICLK	GPIO18	HS1_DATA7	—	—	1	—
19	GPIO19	GPIO19	VSPIQ	GPIO19	U0CTS	—	EMAC_TXD0	1	—
21	GPIO21	GPIO21	VSPIHD	GPIO21	—	—	EMAC_TX_EN	1	—
22	GPIO22	GPIO22	VSPIWP	GPIO22	U0RTS	—	EMAC_TXD1	1	—
23	GPIO23	GPIO23	VSPID	GPIO23	HS1_STROBE	—	—	1	—
25	GPIO25	GPIO25	—	GPIO25	—	—	EMAC_RXD0	0	R
26	GPIO26	GPIO26	—	GPIO26	—	—	EMAC_RXD1	0	R
27	GPIO27	GPIO27	—	GPIO27	—	—	EMAC_RX_DV	0	R
32	32K_XP	GPIO32	—	GPIO32	—	—	—	0	R
33	32K_XN	GPIO33	—	GPIO33	—	—	—	0	R
34	VDET_1	GPIO34	—	GPIO34	—	—	—	0	R,
35	VDET_2	GPIO35	—	GPIO35	—	—	—	0	R,
36	SENSOR_VP	GPIO36	—	GPIO36	—	—	—	0	R,

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Function 5	Reset	No
37	SENSOR_CAPP	GPIO37	–	GPIO37	–	–	–	0	R,
38	SENSOR_CAPN	GPIO38	–	GPIO38	–	–	–	0	R,
39	SENSOR_VN	GPIO39	–	GPIO39	–	–	–	0	R,

ESP32 GPIO function matrix

The ”Reset” column shows each pad’s default configurations after reset:

- **0** – IE = 0 (input disabled)
- **1** – IE =1 (input enabled)
- **2** – IE = 1, WPD = 1 (input enabled, pull-down resistor)
- **3** – IE = 1, WPU = 1 (input enabled, pull-up resistor)

Notes column indicates,

- **R** – Pad has RTC/analog functions via RTC_MUX
- **I** – Pad can only be configured as input GPIO. These input-only pads do not feature an output driver or internal pull-up/pull-down circuitry.

In the Arduino environment, you can invoke the pins just using their respective numbers from 0 to 39. Below is the list of pins you can use safely.

GPIO	Input?	Output?	Notes
0	NO	YES	Pull LOW to enter bootloader mode.
1	NO	YES	`TX0` of serial port for programming and printing debug messages.
2	YES	YES	Connected to the onboard LED, must be left floating or LOW to enter flashing mode.
3	YES	NO	`RX0` of serial port for programming and printing debug

GPIO	Input?	Output?	Notes
			messages.
4	YES	YES	
5	YES	YES	Strapping pin
6	NO	NO	Flash memory interface. Do not use.
7	NO	NO	Flash memory interface. Do not use.
8	NO	NO	Flash memory interface. Do not use.
9	NO	NO	Flash memory interface. Do not use.
10	NO	NO	Flash memory interface. Do not use.
11	NO	NO	Flash memory interface. Do not use.
12	YES	YES	Strapping pin. Booting can fail if pulled HIGH (for 3.3V memories) due to brownout.
13	YES	YES	
14	YES	YES	
15	YES	YES	Pulling LOW mutes the debug messages through the serial port.
16	YES	YES	
17	YES	YES	
18	YES	YES	
19	YES	YES	
21	YES	YES	

GPIO	Input?	Output?	Notes
22	YES	YES	
23	YES	YES	
25	YES	YES	
26	YES	YES	
27	YES	YES	
32	YES	YES	
33	YES	YES	
34	YES	NO	Input only
35	YES	NO	Input only
36	YES	NO	Input only
39	YES	NO	Input only

ESP32 Arduino GPIO pins

Strapping

Every ESP32 chips have a **bootloader** inside the **Read-Only-Memory (ROM)** which is a program that monitors the state of the chip when you power it on. The bootloader can check for different inputs and put the chip into different configurations. The pins monitored by the bootloader are called strapping pins. There are five strapping pins in ESP32. These strapping pins exhibit other behaviors during the booting process. So you should be careful not to interfere with the pins.

Pin Name	Function
`GPIO0`	ESP32 will enter the serial bootloader when `GPIO0` is held low on reset. Otherwise, it will run the program in flash memory. This pin is internally pulled-up.

Pin Name	Function
`GPIO2`	This pin must also be either left unconnected/floating, or driven LOW, in order to enter the serial bootloader. In normal boot mode (`GPIO0` = HIGH), `GPIO2` is ignored. This pin is also connected to the onboard LED.
`GPIO12` / `MTDI`	This pin selects the flash voltage during boot. If driven HIGH, the flash voltage (VDD_SDIO) is 1.8V and not the default 3.3V. The pin has an internal pull-down, so unconnected means the flash voltage is 3.3V. May prevent flashing and/or booting if 3.3V flash is used and this pin is pulled HIGH, causing the flash to brownout.
`GPIO15` / `MTDO`	This pin can be used to mute the debug messages printed by ESP32 during booting. If driven LOW, silences the boot messages printed by the ROM bootloader. The pins have an internal pull-up, so if the pin is unconnected the normal message will be printed to serial.
`GPIO5`	This pin along with the `MTDO` pins determines the bus timing of the SDIO peripheral. This is internally pulled up.

ESP32 strapping pins

After the booting process, all strapping pins return to their default normal functions. Additionally, the `TX0` (`GPIO1`) and `RX0` (`GPIO3`) pins will output the boot message during booting. Read more about the ESP32 bootloader here.

Pull-Up & Pull-Down

All GPIO pins support internal pull-up and pull-down configurations, as well as a high-impedance state. This makes the pin **tristate** compatible.

LED

The onboard LED is connected to `GPIO2` which can be used for debugging. In the Arduino environment, you can invoke this pin as `LED_BUILTIN`.

UART

ESP32 has three UARTs inside (asynchronous only) with hardware and software flow control. The UARTs are named **`UART0`**, **`UART1`**, and **`UART2`**. `UART0` is used for serial programming and to print debug messages. This is the UART we use with the USB serial port to print messages from the ESP32. `UART0` is the default `Serial` instance in the Arduino environment.

`Serial1` is `UART1` and its default pins are mixed with the QSPI interface (GPIO pins **6-11**). So it is not recommended to use `Serial1` without remapping the pins. Remapping of the pins can be done by passing the new pins to the `begin()` function. That said, you can still use `Serial1` with the default GPIOs **9** (RX1, SD2) and **10** (TX1, SD3) but you will only be able to transmit. The receive function won't work. Only a few types of ESP32 boards have GPIOs 9 and 10 broken out.

`UART2` is assigned to `Serial2` in the Arduino sketch. All UART functions can be assigned to any GPIO pins you like. The default ones are listed below.

Arduino Instance	UART	RX Pin	TX Pin	CTS	RTS
`Serial`	UART0	GPIO 3 (RX0)	GPIO 1 (TX0)	N/A	N/A
`Serial1`	UART1	GPIO 9 (RX1)	GPIO 10 (TX1)	GPIO 6	GPIO 11
`Serial2`	UART2	GPIO 16 (RX2)	GPIO 17 (TX2)	GPIO 8	GPIO 7

ESP32 Arduino Serial UART pins

SPI

There are four SPI peripheral blocks inside the ESP32. **`SPI0`** and **`SPI1`** have special functions including communicating with the flash memory and therefore we don't use them. **`SPI2`** and **`SPI3`** are general-purpose SPI interfaces called **HSPI** and **VSPI** respectively. Similar to UART, SPI functions can be mapped to any GPIO pins. Below we have the default pins and their respective Arduino instances. Only one SPI is defined in the Arduino framework, but you can easily add the second one.

Arduino Instance	SPI	COPI	CIPO	SCK	CS
`SPI`	VSPI	GPIO 23	GPIO 19	GPIO 18	GPIO 5

Arduino Instance	SPI	COPI	CIPO	SCK	CS
–	HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

ESP32 Arduino SPI pins

For some reason, the Arduino environment and the ESP32 HAL driver assign HSPI's default pins to VSPI.

ADC

Analog to Digital Converters (ADC) are used to convert analog voltages to digital values. There are two **12-bit** SAR ADCs available on ESP32 with 18 input channels. But only 16 channels are available on Arduino. Following is the list of ADC channels and their Arduino instances. ADC channels `ADC1_CH1` and `ADC1_CH2` are not used.

Arduino Pin	ADC Channel	GPIO	Usable?
`A0`	ADC1_CH0	36	YES
`A3`	ADC1_CH3	39	YES
`A4`	ADC1_CH4	32	YES
`A5`	ADC1_CH5	33	YES
`A6`	ADC1_CH6	34	YES
`A7`	ADC1_CH7	35	YES
`A10`	ADC2_CH0	4	YES
`A11`	ADC2_CH1	0	NO
`A12`	ADC2_CH2	2	NO (LED Connected)
`A13`	ADC2_CH3	15	YES
`A14`	ADC2_CH4	13	YES

Arduino Pin	ADC Channel	GPIO	Usable?
A15	ADC2_CH5	12	NO
A16	ADC2_CH6	14	YES
A17	ADC2_CH7	27	YES
A18	ADC2_CH8	25	YES
A19	ADC2_CH9	26	YES

ESP32 Arduino ADC pins

DAC

ESP32 has two 8-bit **Digital to Analog Converters (DAC)**.

Arduino Pin	GPIO
DAC1	25
DAC2	26

ESP32 Arduino DAC pins

Touch Sensor

There are 10 capacitive touch channels in ESP32.

Arduino Pin	Touch Channel	GPIO	Usable?
T0	TOUCH0	4	YES
T1	TOUCH1	0	NO
T2	TOUCH2	2	NO (LED Connected)
T3	TOUCH3	15	YES

Arduino Pin	Touch Channel	GPIO	Usable?
`T4`	TOUCH4	13	YES
`T5`	TOUCH5	12	NO
`T6`	TOUCH6	14	YES
`T7`	TOUCH7	27	YES
`T8`	TOUCH8	33	YES
`T9`	TOUCH9	32	YES

ESP32 Arduino Touch Sensor pins

I2C

There are two I2C peripherals inside ESP32. I2C is also called **Two Wire Interface (TWI)**. Similar to UART and SPI, I2C pins can also be mapped to any GPIO pins. There are two I2C interfaces defined for Arduino; `Wire` (I2C0) and `Wire1` (I2C1) but only Wire has the pins defined. You need to manually set the pins for `Wire1`.

Arduino Instance	I2C	SDA	SCL
`Wire`	I2C0	GPIO 21	GPIO 22
`Wire1`	I2C1	–	–

ESP32 Arduino I2C pins

PWM

ESP32 supports up to **16** independent **PWM (Pulse Width Modulation)** channels with 16-bit precision. PWM outputs can be mapped to any GPIO pins that support output mode.

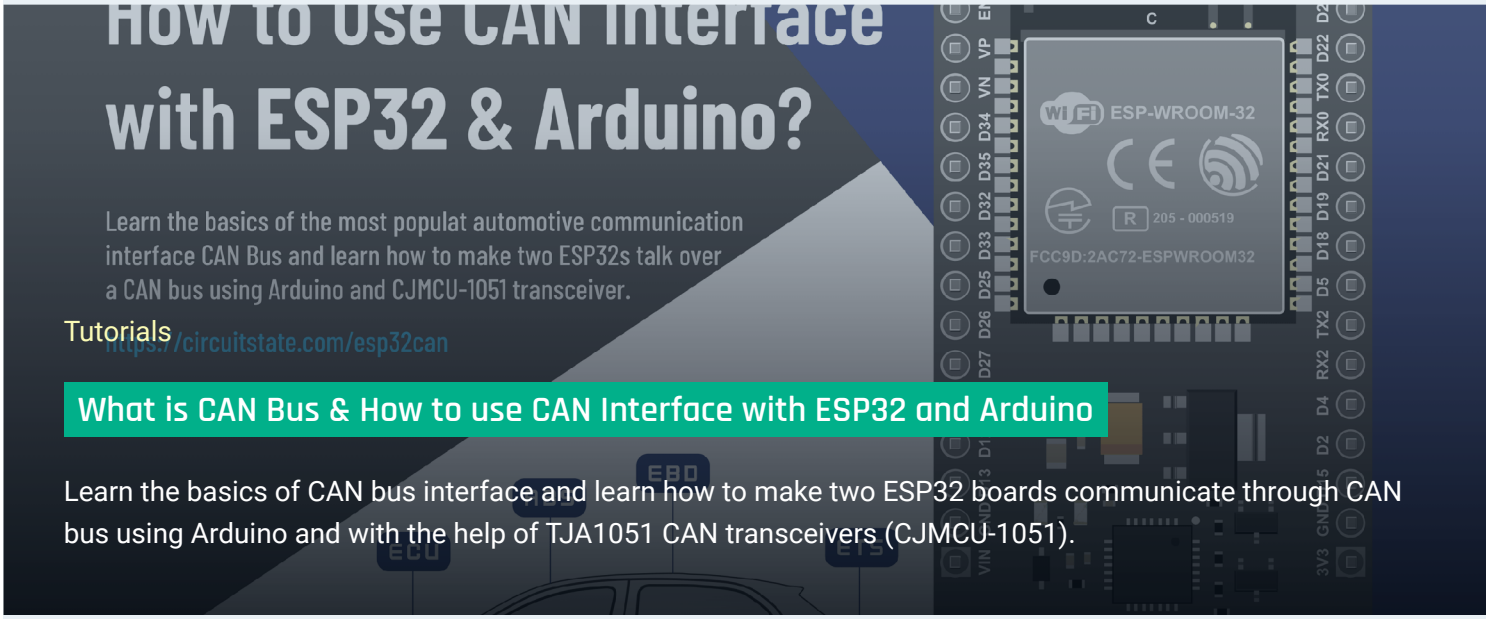
I2S

Inter-Integrated Sound (I2S) is a digital audio interface supported by ESP32. I2S pin functions can also be mapped to any GPIO pins except the clock pins `I2S0_CLK` and `I2S1_CLK` which can only

be mapped to either `GPIO0`, `U0RXD` or `U0TXD`.

CAN/TWAI

Controller Area Network (CAN) or **Two Wire Automotive Interface (TWAI)** is a two-wire communication interface that mainly finds application in the automotive industry. CAN functions can be mapped to any GPIO pins. We have a complete tutorial for using CAN interface with ESP32 which you can find below.



JTAG

JTAG (Joint Test Action Group) is a standard interface used for programming and debugging microcontrollers. ESP32 supports JTAG programming and debugging. JTAG has four main signals and an optional reset line. You can use supported debuggers like the **ESP-Prog** to debug your ESP chips.

Pin Name	GPIO	Function
MTDI	12	Test Data In
MTCK	13	Test Clock
MTMS	14	Test Mode Select

Pin Name	GPIO	Function
MTDO	15	Test Data Out

ESP32 JTAG pins

If you want to learn more about ESP32 debugging through JTAG using the ESP-Prog, we have a complete tutorial on it.

External Interrupts

ESP32 supports external interrupts on all GPIO pins. The interrupt types can be level-triggered, edge-triggered, or state change.

Ethernet MAC

ESP32 has a single **Ethernet MAC** (Medium Access Control) controller. This controller can be combined with an external **Ethernet PHY** (Physical Layer) to accomplish Ethernet communication. Ethernet pins can only be mapped to their assigned ones shown in the GPIO function matrix.