

Pull-up, pull-down and bounce: wiring a button properly



Online guide,
always up to date

Why an input left unconnected reads random values, which resistor to use, when the internal one is enough and how to stop one press from being counted three times.

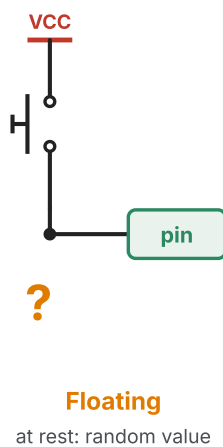
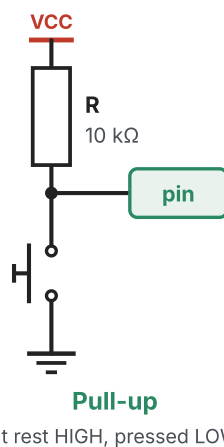
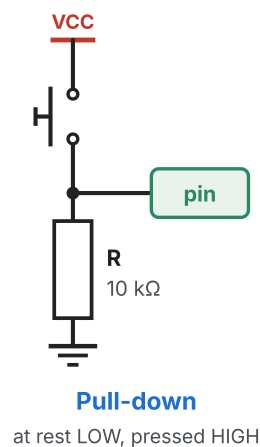
By **Ada Intelisano** · informaticait.it technical team

IN SHORT

- A digital input connected to nothing is **floating**: it picks up noise and reads HIGH and LOW at random.
- A resistor holds it at a known level: **pull-up** to VCC (HIGH at rest) or **pull-down** to GND (LOW at rest). Arduino suggests **10 kΩ** as a good value.
- Often the resistor is already inside the chip: INPUT_PULLUP on the UNO and on the ESP32, which also has INPUT_PULLDOWN. The exceptions are the ESP32's **GPIO 34–39**.
- A mechanical contact **bounces** for a few milliseconds: the sketch has to wait for the signal to settle.

Three ways to wire a button

Three ways to wire a button



informaticait.it

In the first case, when the button is released the pin is connected to nothing: Arduino warns that an input like this reports random changes, because it picks up the electrical noise around it. With the pull-up, the pin at rest sees VCC through the resistor and the button pulls it to GND: **pressed = LOW**, and in the sketch you check for `== LOW`. The pull-down does the opposite. The resistor also limits the current when the button is pressed: with 10 kΩ at 5 V, 0.5 mA flows.

The internal resistors

	Arduino UNO (ATmega328P)	ESP32
Value	20 to 50 kΩ	about 45 kΩ
How to enable it	<code>pinMode(pin, INPUT_PULLUP)</code>	<code>INPUT_PULLUP</code> or <code>INPUT_PULLDOWN</code>
Exceptions	pin 13: the board's LED pulls the voltage down to about 1.7 V; Arduino recommends INPUT with an external pull-down	GPIO 34–39 are input only and have no pull-up or pull-down: you need an external resistor
Good to know	during reset the internal pull-ups are off	some pins decide how the chip boots: see below

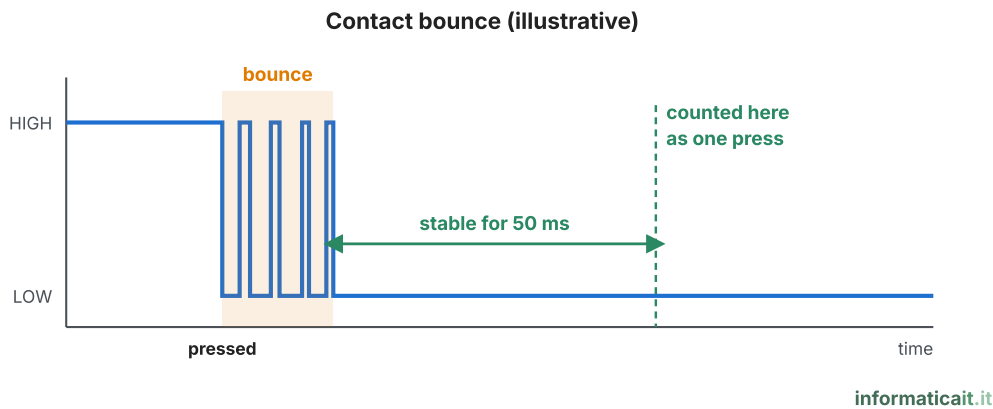
The internal ones are fine for a button close to the board. With long wires or in noisy environments, use a lower external resistor, for example 10 kΩ, which holds the level more firmly.

ESP32: the pins that decide how it boots

At power-on the ESP32 reads the level of five pins (strapping pins) to decide how to start. Each one has an internal resistor that holds it at the right value: a button, or an external pull-up or pull-down on these pins, can change the boot. Choose other pins for buttons.

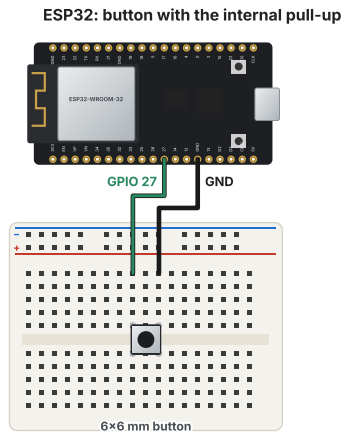
Pin	At rest	What it decides at boot
GPIO 0	pull-up	at 0 (together with GPIO 2 at 0) the chip enters upload mode instead of running the sketch
GPIO 2	pull-down	must be at 0 for upload mode
GPIO 12 (MTDI)	pull-down	at 1 it sets the flash memory supply to 1.8 V
GPIO 15 (MTDO)	pull-up	at 0 it turns off the boot messages on the serial port
GPIO 5	pull-up	together with GPIO 15, an SDIO interface timing setting

Bounce



When the contacts of a button touch, they don't close cleanly: they bounce, and the signal switches several times between HIGH and LOW. For the Omron B3F tactile switch the manufacturer states a bounce of **5 ms maximum**. The microcontroller reads thousands of times per second and sees every jump as a press. The simplest fix is in the sketch: accept the change only when the pin has been steady for a certain time. The official Arduino example uses **50 ms**.

Wiring with the ESP32



informaticait.it

The button straddles the center channel: one side goes to **GPIO 27**, the other to **GND**. No resistor: we use the internal pull-up. With an Arduino UNO the same wiring goes to pin 7.

The sketch

```
// Button with the internal pull-up and debounce: counts the presses without counting fake ones.
// The button connects the pin to GND: pressed = LOW. Same sketch for Arduino UNO and ESP32.
#include <Arduino.h>

#ifdef ARDUINO_ARCH_ESP32
const int BUTTON_PIN = 27;    // do not use GPIO 34-39 (no internal pull-up) or the boot pins
#else
const int BUTTON_PIN = 7;
#endif
const unsigned long DEBOUNCE_MS = 50;    // debounce time: increase it if it still counts double

int stableState = HIGH;    // last accepted state
int lastReading = HIGH;    // last raw reading
unsigned long changedAt = 0;
unsigned long presses = 0;

void setup() {
  Serial.begin(115200);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop() {
  int reading = digitalRead(BUTTON_PIN);
  if (reading != lastReading) {
    changedAt = millis();    // the contact is still bouncing: start over
    lastReading = reading;
  }
  if (millis() - changedAt > DEBOUNCE_MS && reading != stableState) {
    stableState = reading;    // steady for long enough: it's a real change
    if (stableState == LOW) {
      presses++;
      Serial.print("Presses: ");
      Serial.println(presses);
    }
  }
}
```

The sketch counts the presses without ever stopping with `delay()`, so it can do other things in the meantime. If a press is still counted twice, increase `DEBOUNCE_MS`.

Common mistakes

What happens	Why	Fix
The pin changes by itself, without touching anything	floating input	INPUT_PULLUP or an external resistor
One press counted two or three times	contact bounce	debounce in the sketch; increase the wait if needed
The button works backwards	with the pull-up, pressed reads LOW	check for == LOW
Button on GPIO 34 that does not respond	GPIO 34–39 have no internal pull-up	10 kΩ resistor to 3.3 V, or another pin
The ESP32 does not start or stays waiting for an upload	button or resistor on a boot pin	move the button to a free pin, for example 27
UNO pin 13 reading wrong with INPUT_PULLUP	the board's LED is on the same pin	use another pin or an external pull-down

Sources: Arduino, “Digital Pins” documentation and official “Debounce” example · Microchip, ATmega328P datasheet (DS40002061B), §14.2.6 and Table 30-1 · Espressif, ESP32 Series Datasheet v5.3, chapter 3 (strapping pins), Tables 5-3 and 6-1; ESP32 Hardware Design Guidelines · Espressif, ESP-IDF and Arduino-ESP32 documentation, GPIO pages · Omron, B3F tactile switch catalog (Cat. No. A070-E1-08)