LIBRARY · REFERENCE GUIDE

INTERRUPTS

Instead of polling a pin over and over, an interrupt lets the hardware call your code the instant something happens. Why the handler must be short, and what can trigger one.

ONE THOUSAND DRONES ENGINEERING TEAM · VERIFIED 2026-07

Instead of constantly asking has it happened yet, an interrupt lets the hardware call your code the instant something happens. It is how a board reacts to a button press or a sensor's data-ready line right away, without wasting cycles endlessly checking a pin that has not changed.

POLLING VERSUS INTERRUPTS

Polling means looping and reading a pin over and over, waiting for it to change. You catch the event only as often as the loop comes back around, and between checks you burn the CPU on nothing useful. An interrupt flips that around: the hardware watches for the event, and the moment it fires, it jumps the CPU straight into a handler. No wasted checking, and a near-instant response.

THE ISR: SHORT AND FAST

The handler is an interrupt service routine, an ISR. It runs immediately, cutting in on whatever the CPU was doing, so it has to be short. Set a flag, grab one reading, nudge a counter, then return and let the main code do the slow work when it gets to it. A long ISR blocks everything else while it runs, including other interrupts, which is how a board starts dropping events or stuttering.

WHAT CAN INTERRUPT

The common sources are a GPIO edge (a button, or a sensor's data-ready line), a timer reaching its target count (the last lesson), or a peripheral finishing a transfer. On the ESP32 you attach an interrupt to a GPIO and name the function to run when that pin sees a rising or a falling edge, and the chip handles the rest.

- Espressif. ESP-IDF Programming Guide: GPIO (interrupt service, edge triggers). `docs.espressif.com`
- SparkFun. Processor Interrupts tutorial (interrupts versus polling). `learn.sparkfun.com`

DEEP DIVE · DEBOUNCING: KEEP THE MESSY DECISION OUT OF THE ISR

A mechanical button does not make one clean edge when you press it; its contacts bounce, throwing several fast edges from a single press. A raw GPIO interrupt will fire on each one, so one press looks like five. The standard fix is to have the ISR do almost nothing: just record a flag or a timestamp and return. The main loop then debounces it, ignoring repeat triggers that land within a few milliseconds of the first. The button still interrupts instantly, but the timing judgement lives in slow, ordinary code where it belongs, and the ISR stays short.

POLLING CHECKS OVER AND OVER; AN INTERRUPT CALLS THE HANDLER THE MOMENT THE EVENT FIRES.

CHECKPOINT

1. What is the advantage of an interrupt over polling?

a. It uses less flash memory

b. It makes the system clock faster

c. It reacts instantly without wasting cycles checking

ANSWER · C

The hardware calls your code on the event, so there is no wasted polling and the response is immediate.

2. Why must an interrupt service routine (ISR) be short?

a. It cuts in on everything else, so a long ISR blocks the rest of the system

b. Long functions will not compile inside an ISR

c. It runs on a separate chip with little memory

ANSWER · A

An ISR pre-empts the running code and other interrupts, so it must finish fast and defer slow work.

3. A bouncing mechanical button is usually handled how?

a. By making the ISR longer so it waits out the bounce

b. By having the ISR set a flag and letting the main loop debounce it

c. By removing the pull-up resistor

ANSWER · B

The ISR stays short (flag or timestamp); the main loop ignores repeats within a few milliseconds.

- Prerequisite: GPIO, reading and driving pins
- Related: clocks and timers
- Next: the on-chip comms peripherals