LIBRARY · REFERENCE GUIDE

THE ON-CHIP COMMS PERIPHERALS

The UART, SPI, and I2C buses live inside a microcontroller as hardware peripherals that clock the bits out for you. Why that beats bit-banging, and how pin muxing works.

ONE THOUSAND DRONES ENGINEERING TEAM · VERIFIED 2026-07

The serial buses a board uses to talk to sensors and other chips, UART, SPI, and I2C, live inside the microcontroller as hardware peripherals. You configure a block, hand it your data, and it clocks the bits out with exact timing. The chip does the hard, fast part so your code does not have to.

HARDWARE PERIPHERAL VERSUS BIT-BANGING

You could toggle a plain GPIO up and down in software to imitate a bus, which is called bit-banging. It works, but it ties up the CPU for every bit and the timing is only ever as steady as your code. A hardware peripheral is a dedicated block that generates the clock and shifts the bits itself, so it frees the CPU and gets the timing exactly right, and far faster than software could.

THE THREE COMMON BUSES

UART is a simple two-wire link (transmit and receive) with no shared clock, used for consoles and many modules. SPI is a fast bus with a shared clock, data in and out, and a chip-select line per device, used for displays and precision ADCs. I2C is a two-wire bus (a clock and a data line) that addresses many chips on the same pair, used for small sensors. The ESP32 has hardware blocks for all three.

- Espressif. ESP-IDF Programming Guide: Peripherals API (UART, SPI, I2C blocks). docs.espressif.com

PIN MUXING: ROUTE A BLOCK TO YOUR PINS

The ESP32 does not hard-wire each peripheral to fixed pins. A pin mux, short for multiplexer, lets you route a peripheral's signals out to almost any GPIO you choose, so you pick pins that suit your board layout and then assign the bus to them in firmware. Watch the usual caveats when you choose: keep a bus off the strapping pins and off the pins reserved for USB or the on-module flash.

DEEP DIVE · WHY HARDWARE TIMING WINS FOR A FAST BUS

At a few megahertz, an SPI clock edge comes every few hundred nanoseconds, faster than a software loop can reliably toggle a pin while doing anything else at all. The hardware peripheral clocks it out of a buffer with no jitter, and on the ESP32 it can hand off to DMA so a whole block of data moves between memory and the bus with the CPU barely involved. Bit-banging simply cannot keep that pace, which is why the hardware blocks exist. (Espressif ESP-IDF peripherals)

A HARDWARE BUS BLOCK, ROUTED THROUGH THE PIN MUX TO WHICHEVER GPIO YOU PICK.

Choosing which pins carry a bus ties straight into reading the pinout, which is the next lesson: some pins can take a peripheral cleanly, and some carry reset-time or USB duties you must not disturb.

CHECKPOINT

1. Why use a hardware SPI peripheral instead of bit-banging the bus in code?

a. It needs fewer wires than software

b. The hardware handles the exact timing and frees the CPU

c. It works without using any pins at all

ANSWER · B

A dedicated block clocks the bits out precisely and fast, leaving the CPU free for other work.

2. What does the ESP32's pin mux let you do?

a. Raise the supply voltage to the peripheral

b. Add more flash memory

c. Route a peripheral's signals out to pins of your choosing

ANSWER · C

The mux maps a peripheral onto most any GPIO, so you pick pins that fit your layout.

3. Bit-banging a bus in software has what drawback?

a. It ties up the CPU and the timing is only as steady as the code

b. It cannot use GPIO pins

c. It always requires an external ADC

ANSWER · A

Software toggling costs CPU time per bit and jitters, which a hardware peripheral avoids.

- Related: reading the ESP32 pinout
- Next: power modes and sleep