



ESCUADERÍA BORREGOS CCM

MICROCONTROLADORES, SENsoRES Y PLATFORMIO (CON ESP32)

Campus Ciudad de México
Por Arturo Cesar Morales Montaño

Temporada 2025-2026



SIEMENS

GENERAC®

SANDVIK
coromant

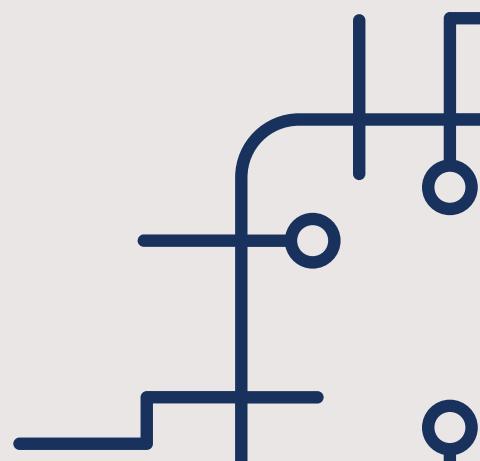
PPG

JCC SEGUROS
Tu tranquilidad lo vale todo

ÍNDICE

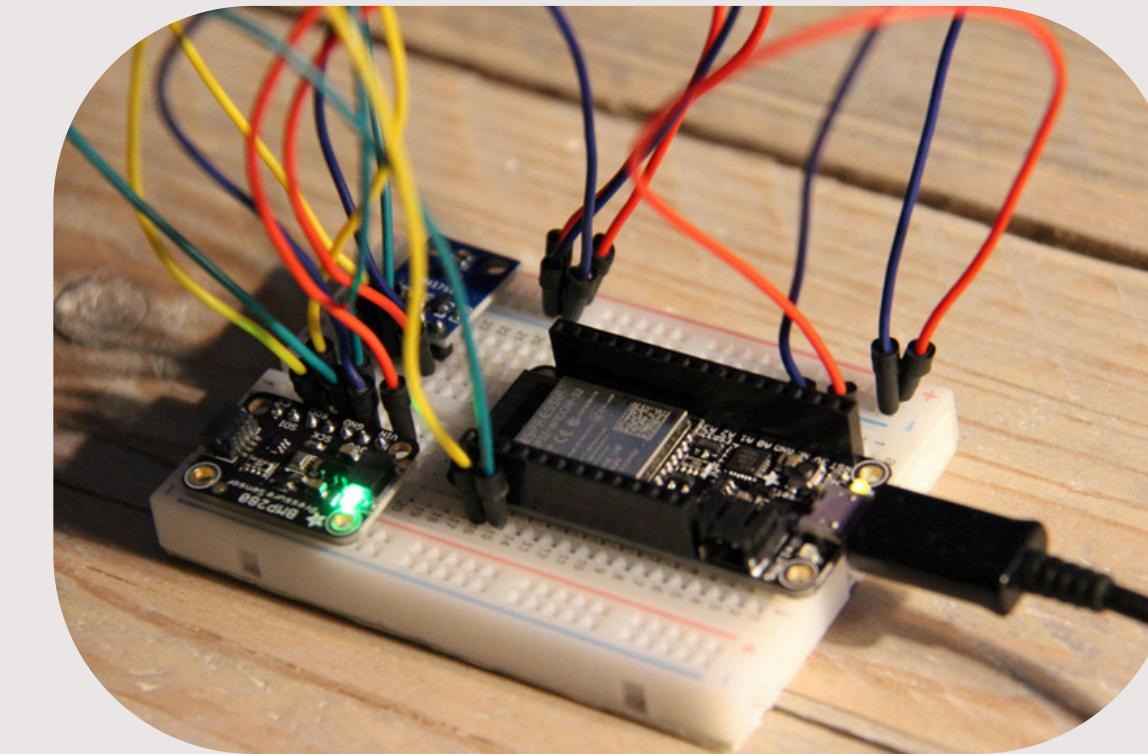
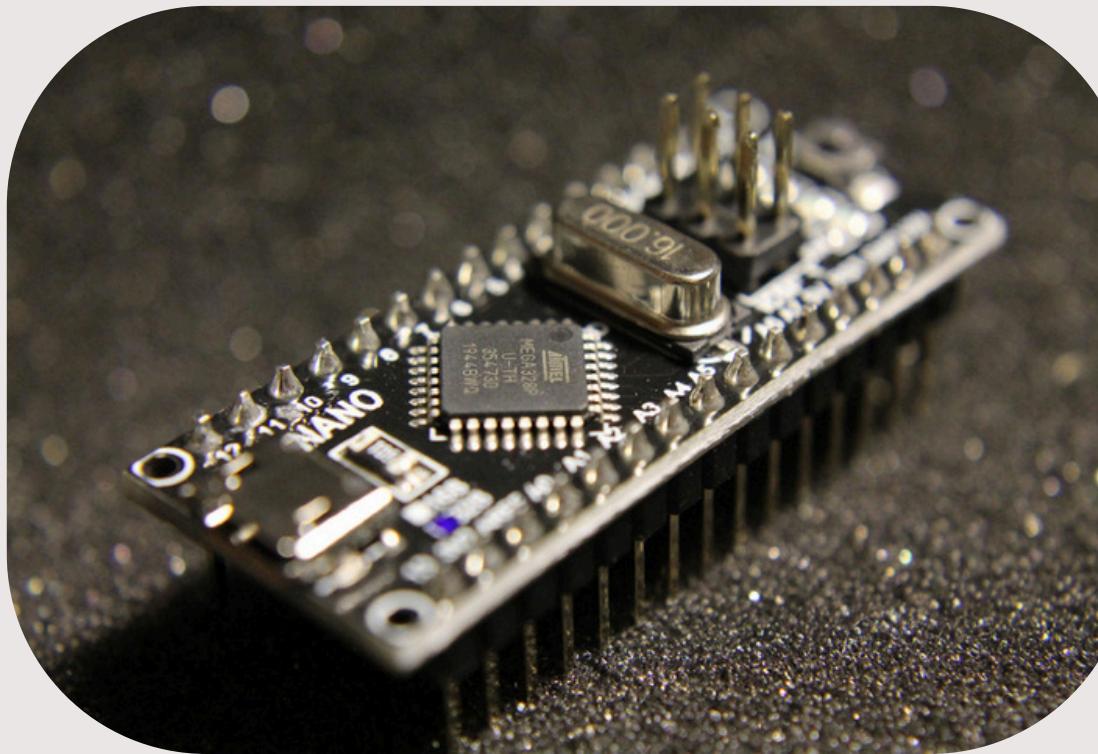


1. ¿Qué es un microcontrolador?
2. Arquitectura básica
3. Conceptos eléctricos básicos
4. Interfaces digitales
5. Tipos de sensores y cómo funcionan
6. Proceso paso a paso para proyectos con microcontroladores (GUÍA PRÁCTICA)
7. ESP32 - ¿qué es y por qué usarlo?
8. ¿Cómo usar la ESP32? (hardware y pines)
9. ¿Qué es PlatformIO, su estructura y cómo se usa?
10. ¿En qué se programa PlatformIO? (C/C++) y ventajas
11. Programación y estructura: C vs C++, archivos .h, librerías, platformio.ini
12. Caso práctico: medidor de distancia con ESP32 + LCD I2C + buzzer + HC-SR04



¿QUÉ ES UN MICROCONTROLADOR?

Un microcontrolador es un pequeño computador en un solo chip diseñado para controlar dispositivos electrónicos: tiene CPU, memoria (flash y RAM), entradas/salidas (pines), y periféricos (timers, ADC, comunicaciones).



MICROCONTROLADOR VS. MICROPROCESADOR

Microcontrolador (MCU)

Un sistema completo en un solo chip.

Incluye CPU + memoria + periféricos
(GPIO, ADC, PWM, I2C, SPI).

- Controla hardware directamente
- Muy bajo consumo
- No necesita sistema operativo
- Ideal para IoT, sensores, robótica,
automatización
- Ejemplo: ESP32

Microprocesador (MPU)

Solo es la CPU, necesita componentes
externos:

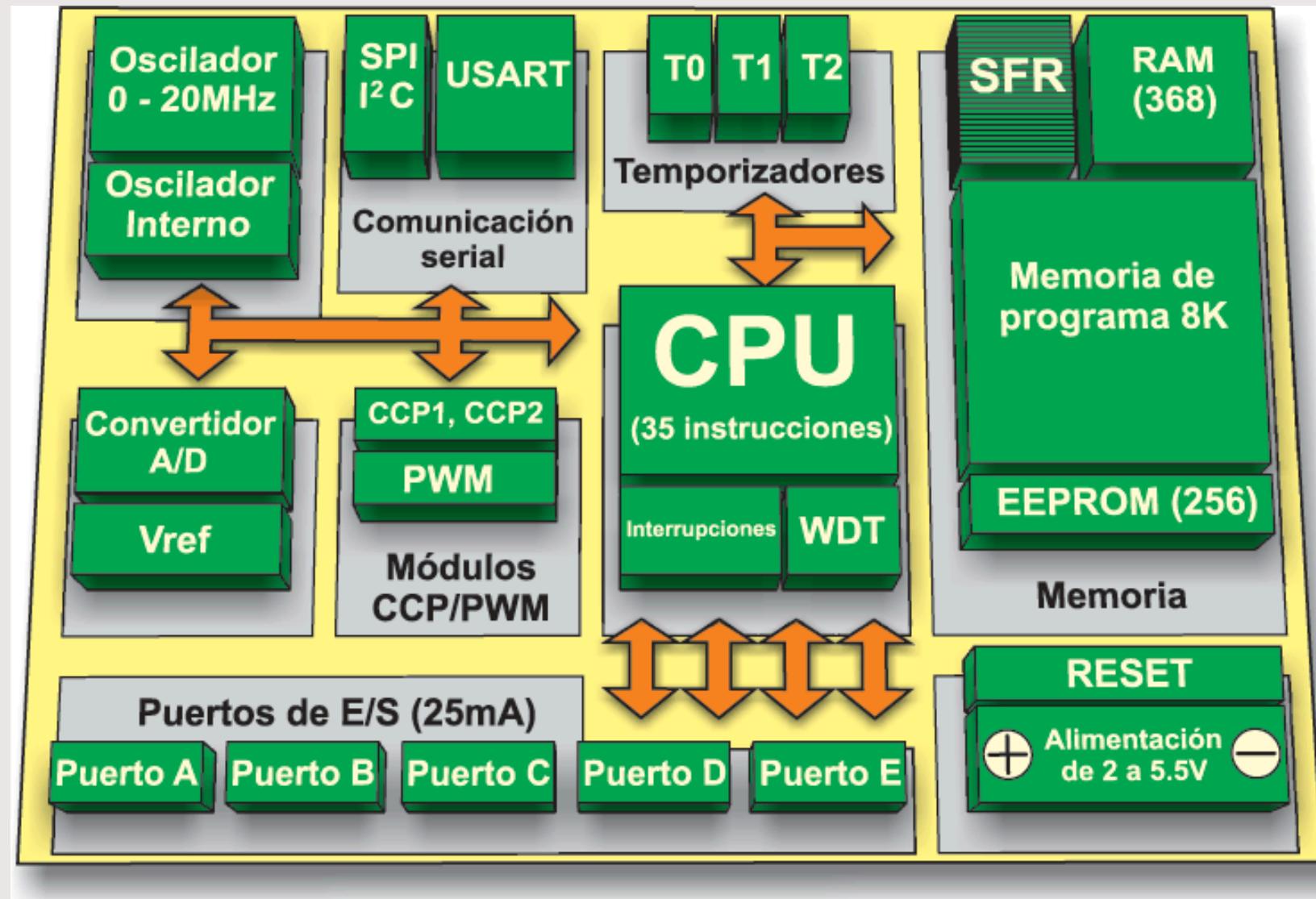
RAM, almacenamiento, controladores,
sistema operativo.

- Diseñado para alto rendimiento
- Mayor consumo de energía
- Necesita Linux/Windows para
funcionar
- Ideal para aplicaciones complejas
(visión, multimedia, servidores)
- Ejemplo: Raspberry Pi

Microcontrolador = controla el mundo físico.

Microprocesador = ejecuta sistemas y aplicaciones complejas.

ARQUITECTURA BÁSICA



[https://www.mikroe.com/ebooks/microcontroladores-pic-programacion-en-c-con-ejemplos/caracteristicas-basicas-del-pic16f887?
srsltid=AfmBOoqMKJQNiFvuJqkO6fJAnV3q45gEr0th-fisEII6kL7gp7xNetw](https://www.mikroe.com/ebooks/microcontroladores-pic-programacion-en-c-con-ejemplos/caracteristicas-basicas-del-pic16f887?srsltid=AfmBOoqMKJQNiFvuJqkO6fJAnV3q45gEr0th-fisEII6kL7gp7xNetw)

- **CPU (core(s)):** ejecuta el programa.
- **Memoria Flash:** dónde está el programa (persistente).
- **RAM:** datos temporales.
- **Periféricos:** USART, SPI, I²C, ADC, DAC, PWM, Timers, RTC.
- **Pines GPIO:** entradas/salidas digitales con funciones multiplexadas.
- **Reguladores y dominio eléctrico** (Vcc, GND).

ARQUITECTURA: CICLO DE EJECUCIÓN Y MODOS DE ENERGÍA



¿Qué es el *ciclo de ejecución*?

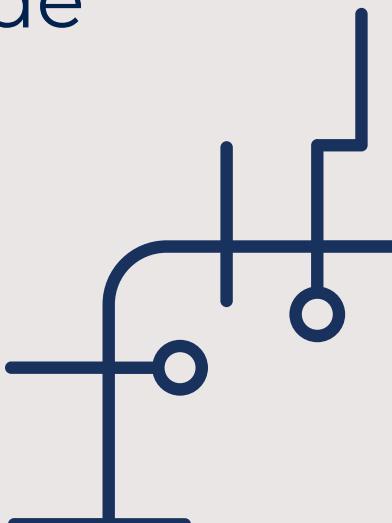
Proceso interno que sigue la CPU para ejecutar una instrucción del programa. Cada instrucción, por simple que parezca, pasa por tres etapas principales:

1

Fetch (Búsqueda)

La CPU obtiene (lee) la siguiente instrucción desde la memoria Flash o RAM.

- El program counter (PC) indica la dirección de memoria donde se encuentra la próxima instrucción.
- La instrucción se copia del programa hacia el registro de instrucción de la CPU.



ARQUITECTURA: CICLO DE EJECUCIÓN Y MODOS DE ENERGÍA



2

Decode (Decodificación)

La CPU interpreta qué debe hacer esa instrucción.

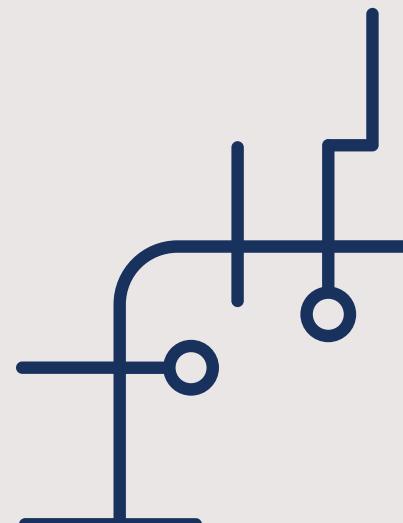
- Se analiza el código de operación (opcode).
- El procesador determina si debe realizar una suma, comparación, salto, escritura en un registro, lectura de un puerto GPIO, etc.

3

Execute (Ejecución)

La CPU realiza la acción definida:

- Operaciones aritméticas o lógicas (sumar, comparar, desplazar bits).
- Acceso a memoria (leer/escribir variables).
- Interactuar con periféricos (GPIO, ADC, I2C...).



ARQUITECTURA: INTERRUPT SERVICE ROUTINES (ISR)



¿Qué son y cómo funcionan?

Una interrupción es un evento que detiene temporalmente el flujo normal del programa para ejecutar una rutina especial llamada ISR (Interrupt Service Routine).

- Un pin cambia de estado (botón presionado)
- Un temporizador interno llega a cero
- Un periférico (UART, I2C) recibió datos



¿Cómo funciona el sistema de interrupciones?

- 1.Ocurre un evento (GPIO de entrada).
- 2.La CPU pausa el programa y guarda su estado.
- 3.Ejecuta la ISR correspondiente.
- 4.Regresa al punto donde se quedó.

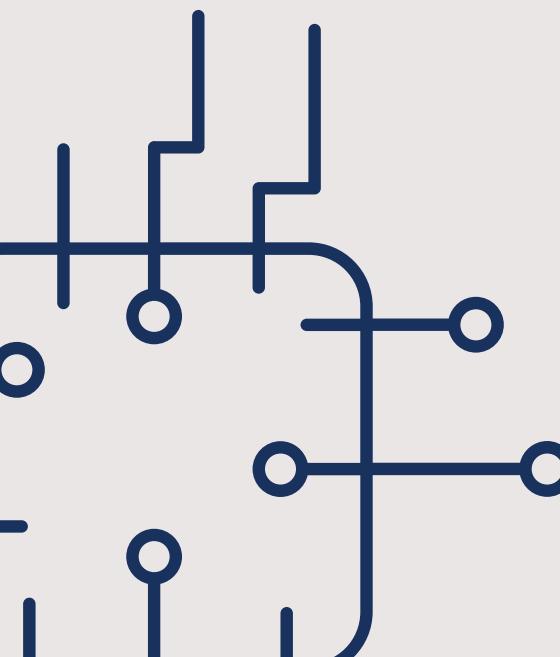
ARQUITECTURA: INTERRUPT SERVICE ROUTINES (ISR)



Prioridades:

Algunos microcontroladores permiten asignar prioridades (quién interrumpe a quién). Las interrupciones críticas (como temporizadores de precisión) pueden tener prioridad más alta que otras.

- Eventos que requieren respuesta inmediata.
- Lectura precisa de tiempos, pulsos o eventos rápidos.
- NO debe incluir código pesado: nada de delay(), comunicación I2C prolongada, ni impresiones seriales.



ARQUITECTURA: MODOS DE BAJO CONSUMO



Los microcontroladores pueden reducir su consumo energético usando distintos modos:

Sleep (sueño ligero)

- La CPU se detiene temporalmente.
- Algunos periféricos siguen funcionando (timers, UART...).
- Sale del modo por interrupciones o eventos.

Deep Sleep (sueño profundo)

- Se apaga gran parte del chip.
- Consumo extremadamente bajo.
- El programa se reinicia al despertar (similar a un reset).

Uso práctico con sensores:

- Dispositivos IoT usan deep sleep entre mediciones para ahorrar batería.
- Ejemplo: medir temperatura cada 10 minutos >>> dormir >>> despertar >>> medir >>> enviar >>> volver a dormir.

CONCEPTOS ELÉCTRICOS BÁSICOS



- 1 **VCC / 3.3V / 5V:** tensión positiva de alimentación. Muchos sensores y módulos usan 3.3V en proyectos con ESP32.
- 2 **GND (Ground):** referencia común. Siempre conectar GND compartido entre módulos y ESP32.
- 3 **Nivel lógico:** la tensión que representa 0 o 1 (p. ej. 0V = LOW, 3.3V = HIGH para ESP32). No aplicar >3.3V a pines de ESP32 sin conversor.
- 4 **Corriente:** alimentar sensores con la capacidad de corriente del regulador: algunos displays/antenas/LED grandes consumen corriente significativa.

No conectar 5V a entradas GPIO del ESP32 directamente.



INTERFACES DIGITALES

Conjunto de circuitos, pines y protocolos (como SPI, I2C, UART) que actúan como puente para que el microcontrolador se comunique de manera confiable con otros dispositivos digitales (sensores, pantallas, memorias, otros chips).

- **Pines de E/S (Entrada/Salida):** Los "puertos" físicos donde se conectan los cables.
- **Circuitos de Conexión:** Hardware que adapta señales (voltajes, corrientes).
- **Protocolos:** Reglas de comunicación (SPI, I2C, UART, USB) que definen cómo se envían y reciben los datos, incluyendo el orden y el ritmo (sincronización).

INTERFACES DIGITALES

- GPIO (entrada / salida)
- UART (serial): TX/RX, baud rates, conversores TTL >>> <<<USB
- I2C (bus compartido, direcciones, pull-ups)
- SPI (master/slave, líneas SCK/MOSI/MISO/CS)
- PWM (control de brillo, motores)
- ADC (lectura analógica) y resolución (8/10/12/16 bits)
- Interrupciones y señales por hardware

I2C y SPI: diferencias prácticas

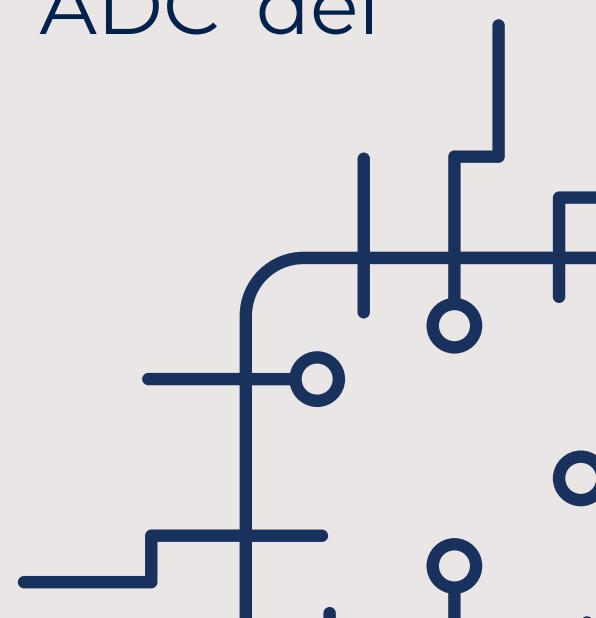
- I2C: 2 líneas, multidrop, direcciones, lento vs simple (ideal para sensores y pantallas I2C)
- SPI: más rápido, full-duplex, requiere línea CS por esclavo (ideal para memoria, pantallas rápidas)

PINES Y SEÑALES COMUNES



- **SDO / SDI:** A veces son sinónimos de MISO/MOSI según nomenclatura del fabricante (SDO = Serial Data Out = equivalente a MISO, SDI = Serial Data In = MOSI). Revisar datasheet.
- **LED:** Pin de control o salida para encender/backlight de pantallas; puede requerir transistor si consume mucha corriente.
- **SDA / SDA0 / SDA1:** Línea de datos del bus I²C (bidireccional).
- **SCL:** Línea de reloj del bus I²C.
- **INT / DIO / IRQ:** Pin de interrupción: el sensor/módulo notifica evento al microcontrolador.
- **TX / RX:** Pines de transmisión y recepción de UART (serial).
- **A0 / A1 / Vout:** Señal analógica de salida de un sensor (se conecta a ADC del microcontrolador).

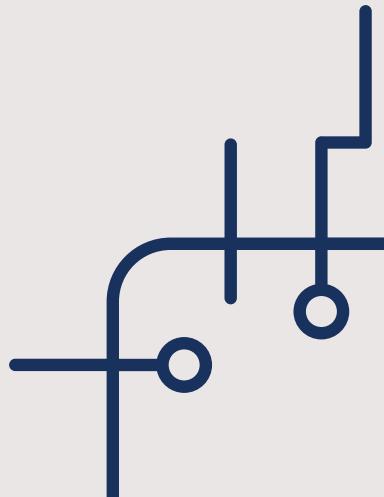
Las etiquetas a veces cambian según el fabricante; cuando haya duda, buscar el datasheet o la hoja de pines del módulo.



TIPOS DE SENSORES

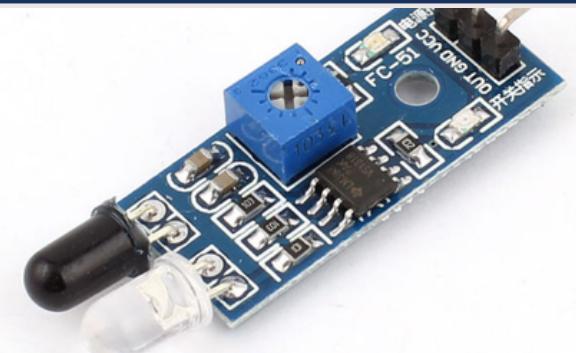


- **Magnitud medida:** posición, proximidad, presión, temperatura, humedad, luz, sonido, gas, aceleración, giroscopio, magnetómetro, imagen
- **Salida:** digital (I2C, SPI, 1-Wire, UART, PWM) y analógica (salida Vout proporcional)
- **Tecnología:** resistivos, capacitivos, piezoeléctricos, ópticos, inductivos, ultrasónicos, MEMS



SENSORES: TIPOS Y CÓMO CONECTARLOS

Clasificación por salida



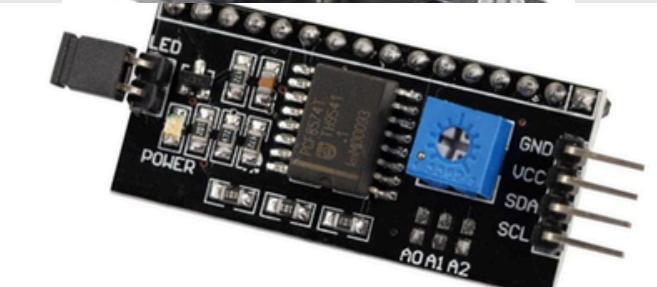
Digital (I/O)

Descripción

Sensores que devuelven HIGH/LOW o usan bus (I²C/SPI/UART).
Ej.: sensores de presencia IR con salida digital.



Analógicos (voltaje)



Bus serial (I²C/SPI/UART)

Potenciómetros, joysticks: conectan a ADC.

Sensores con interrupción

Acelerómetros, magnetómetros, pantallas, LoRa.

Avisan al MCU mediante un pin INT para ahorrar CPU.

TIPOS DE SENSORES



Sensores digitales y buses

- Sensores I2C: TMP102, BME280, MPU6050 (IMU), pantallas LCD I2C (expander PCF8574)
- Sensores SPI: acelerómetros rápidos, memoria flash, LCDs SPI
- Sensores UART/Serial: módulos GPS, modems
- Sensores 1-Wire: DS18B20 temperatura

Sensores híbridos y módulos comunes

- Módulos con acondicionamiento: DHT22 (temp/hum), HC-SR04 (ultrasonido), MQ series (gases)
- Ventaja: nivel lógico y alimentación integrada, pero verificar compatibilidad de 3.3V

Sensores ultrasónicos

- Emite un pulso ultrasónico; mide tiempo de ida y vuelta (ToF)
- Distancia = $(\text{ToF} \times \text{velocidad del sonido}) / 2$
- Consideraciones: ángulo de detección, materiales reflectantes, temperatura y calibración



GUÍA PRÁCTICA

PROCESO PARA HACER PROYECTOS CON MICROCONTROLADORES

GUÍA PRÁCTICA: Proceso para hacer proyectos con microcontroladores

1. Definir objetivo y requerimientos (qué medir/controlar, precisión, respuesta, interfaz)
2. Seleccionar MCU adecuado (pines, ADC, comunicaciones, consumo, conectividad)
3. Revisar hoja de datos y pinout del MCU
4. Seleccionar sensores y periféricos compatibles (niveles lógicos, alimentación)
5. Diseñar la conexión eléctrica (wiring) y alimentación segura
6. Elegir entorno de desarrollo y librerías (PlatformIO / Arduino / Espressif SDK)
7. Implementar firmware por módulos (lectura sensor >>> tratamiento >>> salida)
8. Probar incrementalmente (prueba de cada componente por separado)
9. Integrar y validar (tests de integración)
10. Documentar y empaquetar (PCB, carcasa, versión del firmware)

Checklist y consideraciones al usar sensores

1. ¿Sensor requiere 5V o 3.3V? Si 5V, usar nivelador o circuitería adaptadora.
2. Consumo del sensor: ¿requiere fuente independiente? ¿ruido en la línea?
3. ¿Salida es analógica o digital? ¿Necesito ADC con resolución específica?
4. Tiempo de muestreo y frecuencia: ¿cuánto tarda el sensor en estabilizarse?
5. Precisión y calibración: ¿necesito calibrarlo contra referencia?
6. Protección eléctrica: aislamiento, diodos, filtros

Selección de pines en ESP32

- Consultar pinout y evitar pines reservados/ligados a la memoria flash (GPIO6–GPIO11).
- Evitar usar pines de strapping sin entender su efecto (GPIO0, 2, 12, 15, 5). Usarlos con precaución.
- Tenga en cuenta pines "input-only" (GPIO34–39) no sirven como salidas.
- Para I2C: por convención SDA = GPIO21, SCL = GPIO22 (se pueden reconfigurar).
- Para PWM y ADC, verificar la resolución y canales disponibles para el pin elegido.

Tipos de GPIO y recomendaciones

- GPIO normales: entrada/salida digitales con pull-up/down
- GPIO con ADC: usar pines ADC para señales analógicas (ver resolución y canales)
- GPIO con DAC: pines con salida analógica (si aplica)
- GPIO con capacidades especiales (touch, capacitive, RTC) — consultar doc del MCU

Software: cómo identificar librerías necesarias

- Identifica el protocolo/hardware del periférico (*I2C, SPI, UART, digital simple*).
- Busca en el registrador de PlatformIO (*PIO Home → Libraries*) o en GitHub "*<nombre_sensor> Arduino library*".
- Revisa: última actualización, issues abiertas, compatibilidad con *ESP32/Arduino framework*.
- Añadela a *lib_deps* de *platformio.ini* para reproducibilidad.
- Lee ejemplos incluidos en la librería: son una gran guía de uso.

Cómo saber qué programar y qué probar primero

- Implementa pruebas unitarias por componente: primero cargar ejemplo de la librería del sensor y verificar lectura en serial.
- Luego crear módulo que convierta lectura cruda a unidades físicas y filtre ruido (media móvil, median filter).
- Implementa salidas simples (LED, buzzer) para pruebas físicas.
- Integra en el loop principal y agrega manejo de errores/timeouts.



PROYECTO PRACTICO

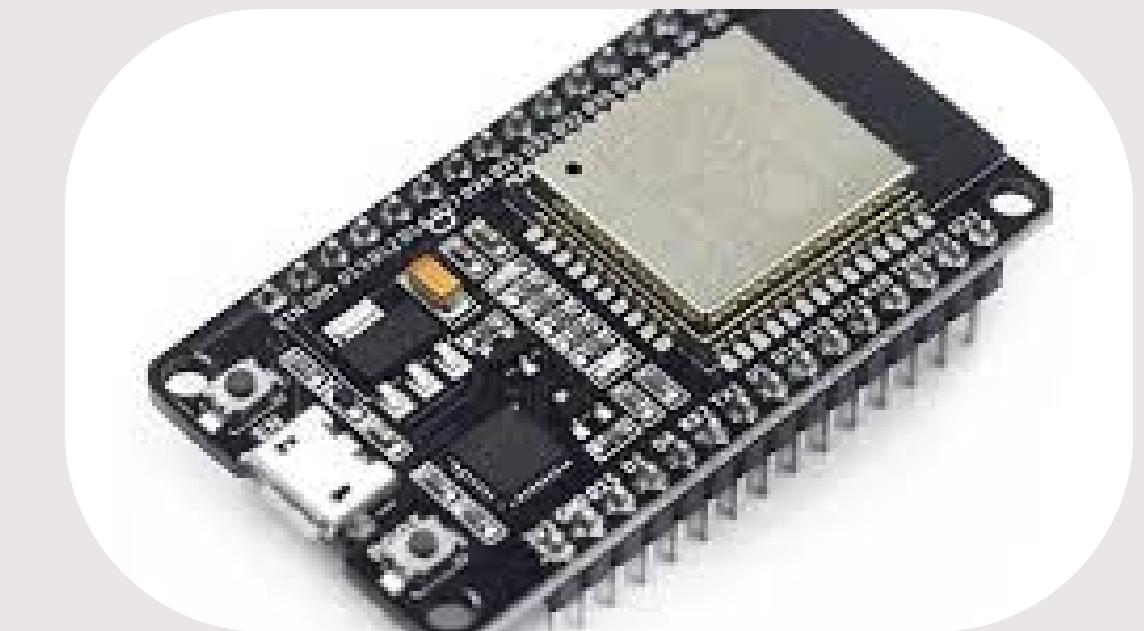
MEDIDOR DE DISTANCIA CON BUZZER Y LCD

ESP32 - ¿QUÉ ES Y POR QUÉ USARLO?



El ESP32 es una familia de microcontroladores de Espressif con Wi-Fi y Bluetooth integrados, doble core (algunos modelos), ADC, DAC, múltiples UART/SPI/I²C, touch sensors, y amplio ecosistema. Ideal para proyectos IoT y prototipos.

- Alimentación típica: 3.3V.
- Soporta Wi-Fi 802.11 b/g/n y Bluetooth/BLE.
- Varios pines ADC, PWM, SPI, I²C, UART, TOUCH.
- Programación con PlatformIO/Arduno/ESP-IDF.



Existen muchas variantes físicas (DevKit, WROOM, WROVER con PSRAM, etc.)



ESP32 PINOUT

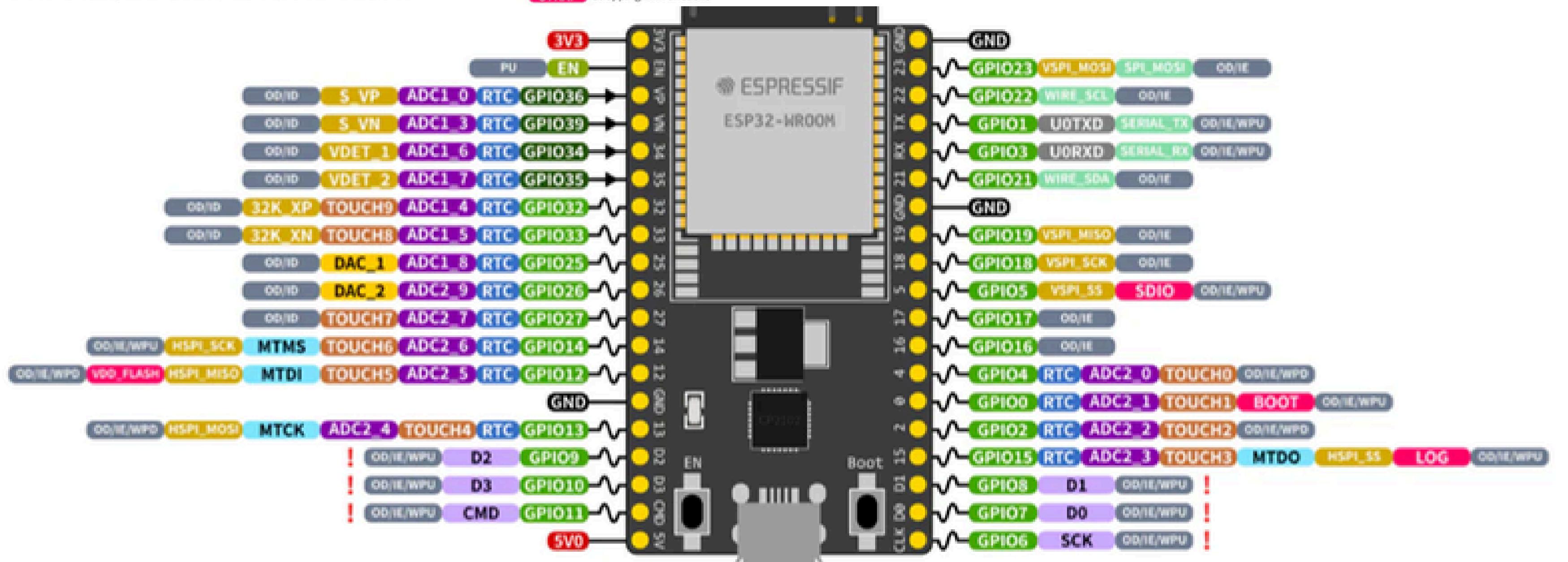
ESP32 Specs

32-bit Xtensa dual-core @240MHz
 Wi-Fi IEEE 802.11 b/g/n 2.4GHz
 Bluetooth 4.2 BR/EDR and BLE
 520 KB SRAM (16 KB for cache)
 448 KB ROM
 34 GPIOs, 4x SPI, 3x UART, 2x I2C,
 2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
 1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

	PWM Capable Pin
	GPIO Input Only
	GPIO Input and Output
	Digital-to-Analog Converter
	JTAG for Debugging
	External Flash Memory (SPI)
	Analog-to-Digital Converter
	Touch Sensor Input Channel
	Other Related Functions
	Serial for Debug/Programming
	Arduino Related Functions
	Strapping Pin Functions

	RTC Power Domain (VDD3P3_RTC)
	Ground
	Power Rails (3V3 and 5V)
	Pin Shared with the Flash Memory
	Can't be used as regular GPIO

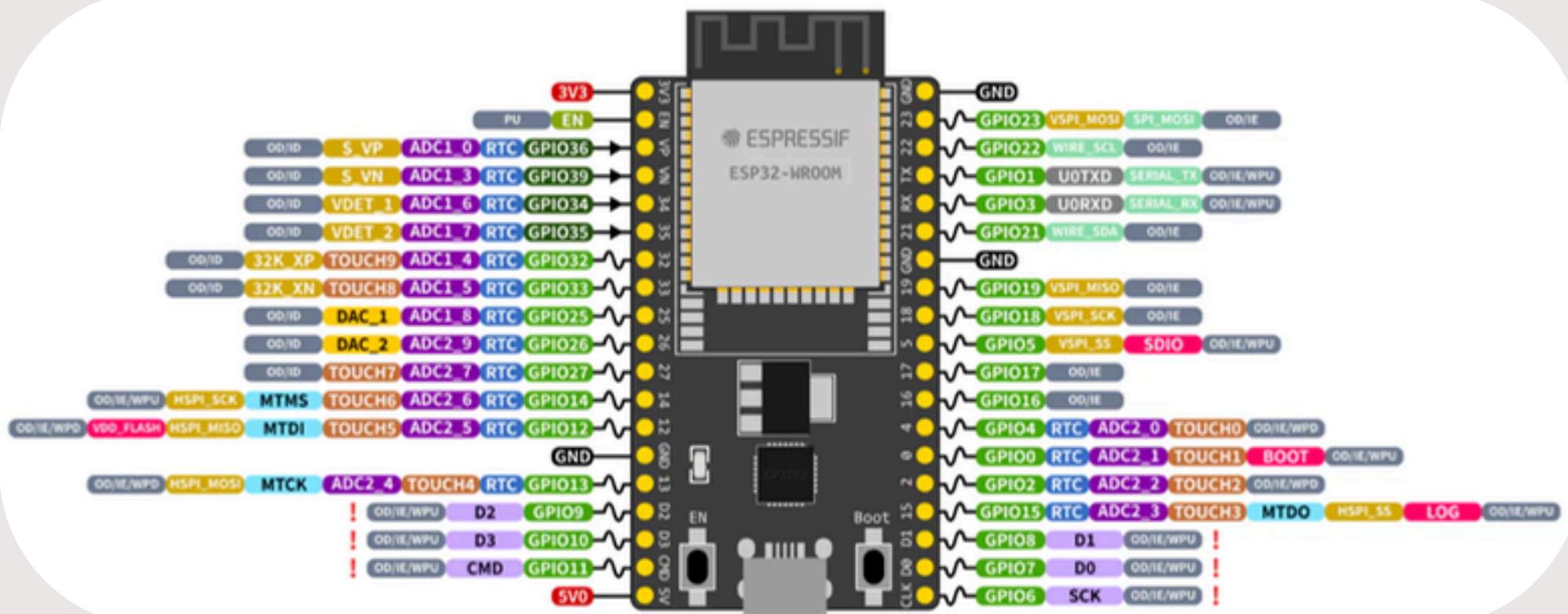
	GPIO STATE
	WPU: Weak Pull-up (Internal)
	WPD: Weak Pull-down (Internal)
	PU: Pull-up (External)
	ID: Input Enable (After Reset)
	ID: Input Disabled (After Reset)
	OE: Output Enable (After Reset)
	OE: Output Disabled (After Reset)



CONSIDERACIONES DEL ESP32 (PINES)



- **Pines de boot y EN:** algunos pines determinan modo de arranque (boot). Por ejemplo, IO0 se usa para entrar en modo de programación en algunos flashheadores; EN o RST reinician el chip. (Nota: la asignación concreta puede depender del módulo/placa).
- **Pines que evitar para salidas/entradas especiales:** algunos pines se usan en arranque o son reservados en ciertas placas; consultar la hoja del devboard.



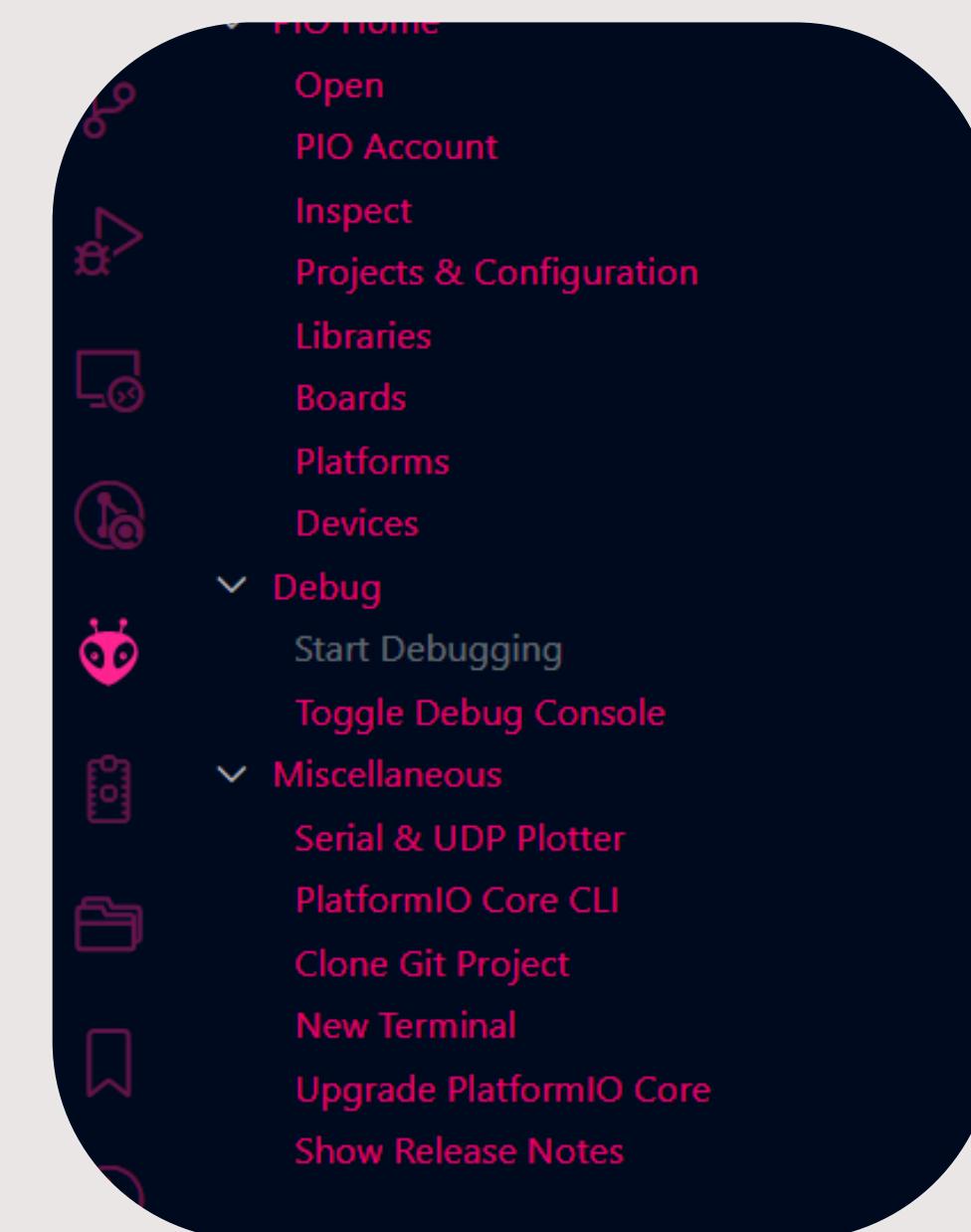
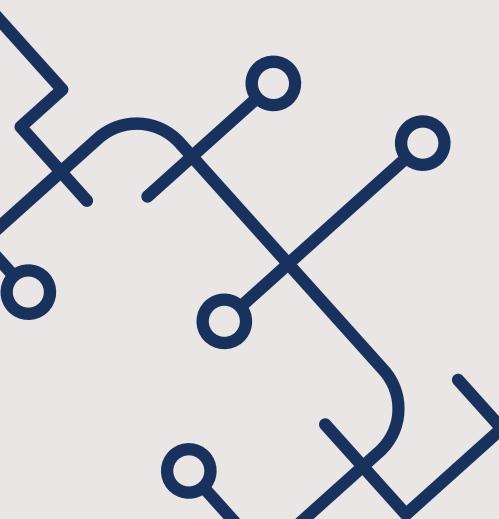
PLATFORMIO EN VISUAL STUDIO CODE

¿QUÉ ES Y POR QUÉ USARLO?



Entorno de desarrollo (IDE) y gestor de proyectos para microcontroladores integrado en VSCode. Maneja toolchains, bibliotecas, frameworks (Arduino, ESP-IDF), y facilita compilación, carga y depuración multiplataforma.

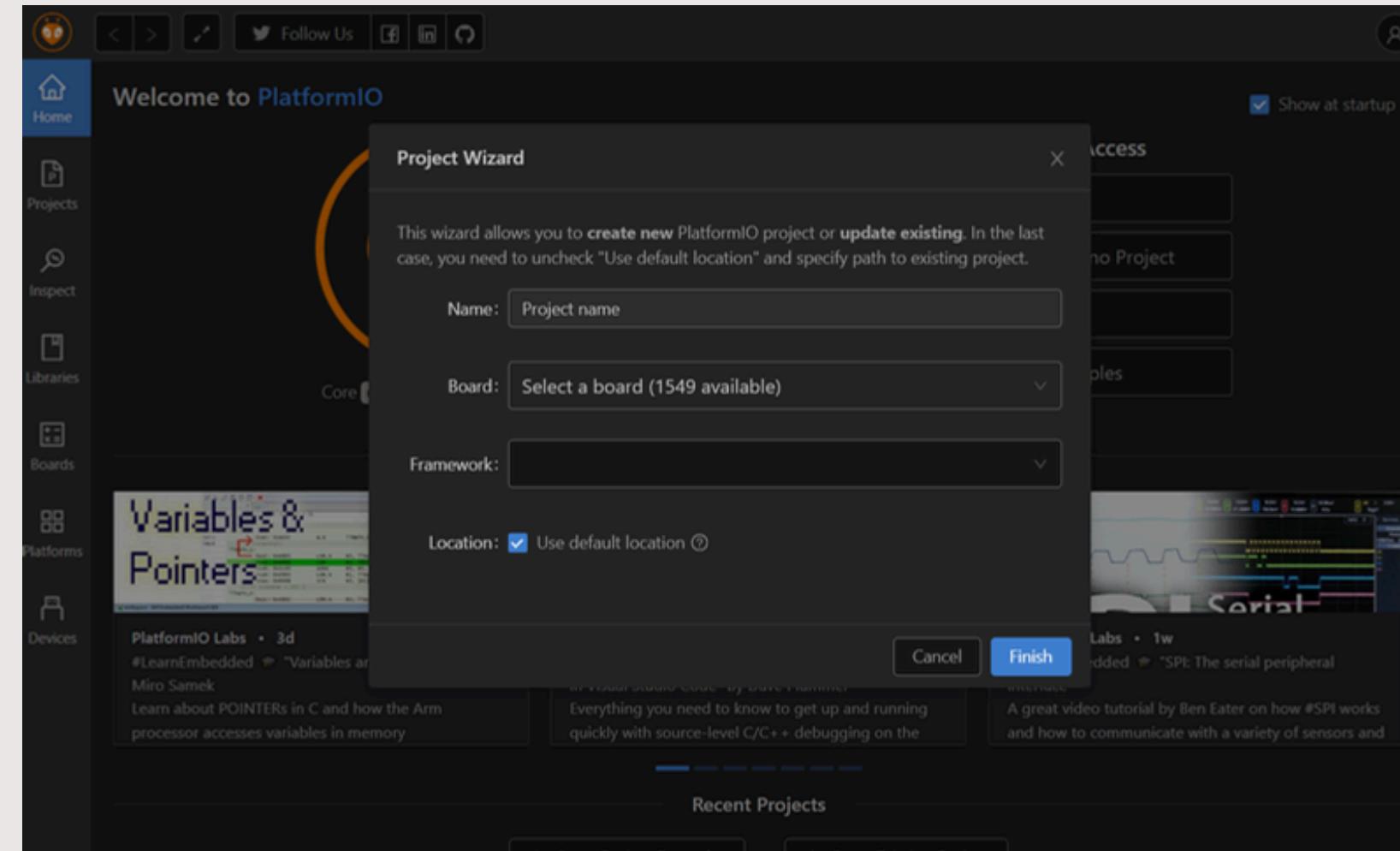
- Proyectos reproducibles
- Dependencias declaradas
- Múltiples entornos de compilación (p. ej. distintas placas/flags)
- Integración con test automático.



INSTALACIÓN RÁPIDA



- Instalar Visual Studio Code.
- Abrir VSCode → Extensions → buscar PlatformIO IDE e instalar.
- Reiniciar VSCode.
- Conectar ESP32 por USB.
- Crear nuevo proyecto PlatformIO (menú PlatformIO → New Project) → elegir board (ej. esp32dev) y framework (Arduino o ESP-IDF).
- Abrir platformio.ini y configurar upload_port o dejar automático.



ESTRUCTURA DE UN PROYECTO PLATFORMIO



- **.pio:** Carpeta generada automáticamente donde PlatformIO guarda compilaciones, archivos temporales y dependencias internas.
- **.vscode:** Configuraciones específicas de Visual Studio Code para el proyecto (atajos, IntelliSense, rutas).
- **include:** Carpeta para colocar archivos .h (cabeceras) que contienen declaraciones de funciones, constantes y variables globales.
- **lib:** Carpeta donde van librerías personalizadas creadas por el usuario para organizar mejor el código.
- **src:** Carpeta principal donde se colocan los archivos .cpp del programa, incluyendo main.cpp.
- **test:** Carpeta usada para pruebas unitarias mediante PlatformIO Unit Testing.
- **.gitignore:** Archivo que indica qué elementos deben ignorarse al usar Git (por ejemplo, la carpeta .pio).
- **platformio.ini:** Archivo de configuración del proyecto: placa a usar, librerías, velocidad del puerto, frameworks, etc.

PROGRAMACIÓN Y ESTRUCTURA



- **¿Qué es C?:** lenguaje procedural de bajo nivel cercano al hardware.
- **¿Qué es C++?:** superset de C con programación orientada a objetos y abstractions.
- **Archivos .h (headers):** declaraciones de funciones, clases y constantes que se usan en varios archivos.
- **Cómo organizar:** include/mi_sensor.h + src/mi_sensor.cpp + src/main.cpp
- **platformio.ini:** explicar entradas clave (board, platform, framework, lib_deps, monitor_speed)

PLATFORMIO.INI (EJEMPLO)



```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
monitor_speed = 115200
lib_deps =
    teckelI2C/NewPing
    LiquidCrystal_I2C
```

Contacto

FRIDA SOPHIA CHÁVEZ JUÁREZ

Capitána

A01665457@tec.mx



@fri-ch

ARTURO CESAR MORALES MONTAÑO

Lider de Electronica

A01659412@tec.mx

artmoram.com



@Arturo-Cesar-Morales-Montaño



Escudería Borregos CCM



electraton_ccm



electratonccm



Escuderia-Borregos-CCM