DIA 3 | 3 hrs

Circuitos e Integracion con Hardware

ESP32-S3, microfono I2S, matriz LED, buzzer, PlatformIO y compilacion

¿Que es un microcontrolador?

Definicion simple

Una computadora completa del tamano de un sello postal. Tiene procesador, memoria y conexiones para sensores y actuadores.

A diferencia de tu laptop:
- Consume 200 mA (vs 3A de una laptop)
- Cuesta $10 USD (vs $500+)
- No tiene pantalla ni teclado
- Pero ejecuta IA en tiempo real

Analogia: el cerebro del robot

Tu cerebro recibe senales de los sentidos (ojos, oidos), las procesa y envia ordenes (mover mano, hablar).

Un microcontrolador hace lo mismo:
- Sentidos: microfono, sensor temp, camara
- Cerebro: procesador con IA
- Ordenes: encender LED, activar buzzer

Su programa se ejecuta en un LOOP infinito: escuchar → procesar → responder → repetir.

Ejemplos de microcontroladores en tu vida diaria:

Control remoto TV	Horno microondas	Lavadora	Arduino	ESP32 (nuestro)
Boton → infrarrojo	Timer, potencia	Ciclos, nivel agua	Proyectos makers	WiFi, BT, IA

ESP32-S3-N16R8: nuestro chip

Especificaciones tecnicas

Procesador	Dual-core Xtensa LX7, 240 MHz
Flash (N16)	16 MB — almacena firmware + modelo IA
PSRAM (R8)	8 MB Octal — memoria de trabajo rapida
WiFi	802.11 b/g/n 2.4 GHz (lo apagamos)
Bluetooth	BLE 5.0 (lo apagamos)
GPIO	45 pines programables
I2S	Bus de audio digital (para microfono)
SPI	Bus serial rapido (para matriz LED)
USB	USB-C nativo (programar + debug)
Costo	~$10 USD

¿Que significa N16R8?

N16 = 16 MB de Flash
(memoria no volatil — como el disco duro)
Aqui se guarda el programa y el modelo.

R8 = 8 MB de PSRAM
(memoria volatil rapida — como la RAM)
Aqui se hacen los calculos de MFCC y CNN.

¿Por que apagamos WiFi y Bluetooth?
Porque consumen energia y memoria que necesitamos para la IA. Nuestro sistema es 100% local.

Nuestro modelo MicroLightCNN pesa 22 KB. La Flash tiene 16 MB. Es como guardar un documento de 1 pagina en un disco de 1 TB — sobra espacio. El arena TFLite usa 100 KB de los 8 MB de PSRAM.

Memoria: Flash vs PSRAM

Flash (16 MB) — El 'disco duro'

No volatil: los datos se conservan al apagar.

Contenido:
- firmware.bin (~1.5 MB) — tu programa
- g_model[] (~22 KB) — pesos de la CNN
- Tablas de particiones

Analogía: como la memoria de tu celular donde guardas apps y fotos.

Velocidad: 80 MHz en modo QIO
(bastante rapida para un micro).

PSRAM (8 MB) — La 'RAM de trabajo'

Volatil: se borra al apagar.

Contenido en ejecucion:
- Arena TFLite (~100 KB)
- Buffers de audio (32,000 × 4 bytes)
- Matrices MFCC (20 × 124 × 4 bytes)
- Variables del programa

Analogía: como la mesa de trabajo donde pones los papeles que estas usando AHORA.

Velocidad: Octal SPI (OPI) — mas rapida que la Flash.

`heap_caps_malloc(size, MALLOC_CAP_SPIRAM)` — asi le pedimos memoria a la PSRAM desde el codigo C++.

Protocolo I2S y microfono INMP441

¿Que es I2S?

Inter-IC Sound: un protocolo de comunicacion DIGITAL para audio de alta calidad.

Usa 3 cables:
- BCLK (bit clock): marca el ritmo
- LRCLK (word select): izq/derecho
- DATA: los bits de audio

Ventaja vs analogico: sin ruido electrico, sin conversion ADC externa, calidad de estudio.

El ESP32-S3 tiene hardware I2S integrado con DMA (acceso directo a memoria).

Microfono INMP441 ($3 USD)

VDD	3.3V	Alimentacion
GND	GND	Tierra
SD (data)	GPIO 1	Datos de audio
SCK (clock)	GPIO 2	Reloj de bits
WS (word sel)	GPIO 7	Seleccion L/R
L/R	GND	Canal izquierdo

Configuracion I2S en el codigo (main.cpp)

sample_rate = 16000 Hz | bits_per_sample = 32 bits | channel = MONO (solo derecho)
dma_buf_count = 8 | dma_buf_len = 512 → DMA captura audio sin interrumpir al procesador
i2s_read() captura 32,000 muestras (2 seg) en un solo bloque → listo para procesar MFCCs

Matriz LED MAX7219 y buzzer

Matriz LED 8×8 con MAX7219

64 LEDs individuales controlados por SPI.
El chip MAX7219 maneja la multiplexion.

Conexion (SPI):
MOSI (datos): GPIO 13
SCK (reloj): GPIO 14
CS (chip sel): GPIO 10

Usamos libreria MD_MAX72XX.
Tipo hardware: FC16_HW (modulos comunes).

Muestra iconos:
Cara neutral: cuando no hay enojo
Cara enojada: cuando enojo >= 70%
Puntos: cuando no detecta voz

Buzzer piezoelectrico

Un buzzer convierte electricidad en sonido.

Conexion ultra simple:
Pin positivo: GPIO 21
Pin negativo: GND

Control:
digitalWrite(21, HIGH) → suena
digitalWrite(21, LOW) → silencio

Se activa cuando la probabilidad de enojo >= 30%
(BUZZER_ANGRY_THRESHOLD).

Es la alarma del sistema.

Diagrama de conexion completo

- INMP441 → I2S (GPIO 1,2,7)
- MAX7219 → SPI (GPIO 10,13,14)
- Buzzer → GPIO 21
- Todo alimentado por USB-C (5V)

PlatformIO: el IDE para ESP32

¿Que es PlatformIO?

Un IDE (entorno de desarrollo) que se instala como extension de VS Code.

Ventajas sobre Arduino IDE:
- Manejo automatico de librerias
- Compilacion mas rapida
- IntelliSense (autocompletado)
- Soporte para cientos de boards
- Configuracion por archivo (platformio.ini)

Gratis y open source.

Instalacion (5 minutos)

1. Abrir VS Code
2. Ir a Extensions (Ctrl+Shift+X)
3. Buscar 'PlatformIO IDE'
4. Click 'Install' y esperar
5. Reiniciar VS Code
6. Aparece icono de hormiga en sidebar

Flujo de trabajo: Abrir proyecto → Compilar (✓) → BOOT+RESET en ESP32 → Upload (→) → Monitor Serial

La barra inferior de VS Code tiene todos los botones: ✓ Build | → Upload | 🔌 Serial Monitor | 🗑 Clean

Si el upload falla con 'Unable to verify flash chip': mantener BOOT, presionar RESET, soltar BOOT, luego Upload.

Estructura del proyecto y archivos src/

Arbol del proyecto

```
AngryVoiceRecognition/
├─ platformio.ini    Config board+mem
├─ lib/
│   ├─ tfmicro/        TFLite Micro
│   └─ MD_MAX72XX/     Libreria LED
└─ src/
    ├─ main.cpp        Programa principal
    ├─ model_data.h    Header del modelo
    ├─ model_data.cpp Modelo INT8 (bytes)
    ├─ NeuralNetwork.h Motor CNN header
    ├─ NeuralNetwork.cpp Motor CNN impl
    ├─ SpeakerNetwork.h Hablantes (stub)
    ├─ MFCC.h/cpp      Pipeline MFCC C++
    ├─ FFT.h           FFT optimizada
    ├─ emotions.h/cpp Iconos LED
    └─ normalizacion.h Normaliz. xvector
```

main.cpp

El director de orquesta. setup() inicializa todo una vez. loop() repite infinitamente: leer mic → MFCC → CNN → LED/buzzer.

NeuralNetwork.cpp

Carga el modelo TFLite, reserva arena en PSRAM, registra ops, cuantiza entrada INT8, ejecuta inferencia, dequantiza salida.

MFCC.cpp + FFT.h

Pipeline MFCC identico al notebook: pre-enfasis → Hamming → FFT → Mel → log → DCT. FFT.h tiene la FFT optimizada para ESP32.

model_data.h/cpp

El modelo MicroLightCNN como array de 25,064 bytes. Lo genera el notebook automaticamente. model_data.h declara extern g_model[].

platformio.ini explicado linea por linea

`[env:esp32s3]`	Nombre del entorno de compilacion
`platform = espressif32`	Plataforma Espressif (fabricante del ESP32)
`board = esp32-s3-devkitc-1`	Placa especifica (ESP32-S3 DevKit)
`framework = arduino`	Usar Arduino framework (no ESP-IDF puro)
`board_build.arduino.memory_type = qio_opi`	Flash en QIO + PSRAM en Octal (OPI). CRITICO para N16R8.
`board_build.partitions = huge_app.csv`	Particion de ~3.1 MB para app. Sin esto: 'region overflowed'.
`board_upload.flash_size = 16MB`	Tamano real de la Flash (N16).
`-DBOARD_HAS_PSRAM`	Habilita los 8 MB de PSRAM. Sin esto: heap_caps_malloc falla.
`-DARDUINO_USB_CDC_ON_BOOT=1`	Habilita USB-C para Serial Monitor.
`monitor_speed = 115200`	Velocidad del Serial Monitor (debe coincidir con Serial.begin).

Compilar, flashear y monitorear

1

Compilar (Build)

Click ✓ en barra inferior (o Ctrl+Alt+B). Compila todo el codigo C++ y genera firmware.bin. Si hay errores, los muestra en Terminal.

~30 seg

2

Modo Bootloader

Mantener BOOT presionado → presionar y soltar RESET → soltar BOOT. El ESP32 entra en modo de programacion. Necesario SOLO para el primer upload.

3 seg

3

Upload (Flash)

Click → en barra inferior (o Ctrl+Alt+U). Envia el firmware al ESP32 por USB-C. Si dice 'Unable to verify': repetir paso 2.

~10 seg

4

Reset

Presionar RESET una vez. El ESP32 reinicia y ejecuta tu programa. Si no reacciona, desconectar y reconectar USB.

1 seg

5

Serial Monitor

Click icono enchufe (o Ctrl+Alt+S). Muestra los mensajes Serial.println() del codigo. Aqui ves las predicciones de enojo en tiempo real.

continuo

Errores comunes y como resolverlos

'Unknown board ID' Nombre del board mal escrito en platformio.ini	Fix: Usar: board = esp32-s3-devkitc-1
'region overflowed' Particion muy pequena para firmware+TFLite	Fix: Agregar: board_build.partitions = huge_app.csv
'Unable to verify flash' ESP32 no esta en modo bootloader	Fix: Mantener BOOT → presionar RESET → soltar BOOT → Upload
'undefined reference g_model' model_data tiene extension .cc (no la compila) o falta #include	Fix: Renombrar a .cpp y agregar #include "model_data.h"
'PSRAM not found' Falta -DBOARD_HAS_PSRAM en build_flags	Fix: Agregar: -DBOARD_HAS_PSRAM en platformio.ini
Todos los LEDs encendidos Tipo de hardware LED incorrecto	Fix: Cambiar GENERIC_HW por FC16_HW en main.cpp

Ejercicio: armar circuito y primer upload

1 **Conectar microfono INMP441** — *10 min*

SD→GPIO1, SCK→GPIO2, WS→GPIO7, VDD→3.3V, GND→GND, L/R→GND. Usar jumpers macho-hembra.

2 **Conectar matriz LED MAX7219** — *5 min*

DIN→GPIO13, CLK→GPIO14, CS→GPIO10, VCC→5V, GND→GND. Cuidado: VCC va a 5V, no 3.3V.

3 **Conectar buzzer** — *2 min*

Pin positivo (+) → GPIO21, Pin negativo (-) → GND. El positivo suele ser el pin mas largo.

4 **Conectar ESP32 por USB-C** — *1 min*

Usar cable de DATOS (no solo de carga). Si no aparece puerto: instalar driver CP210x.

5 **Compilar + Upload + Probar** — *15 min*

Build → BOOT+RESET → Upload → Monitor Serial. Hablar al microfono y observar predicciones.