

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

**Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa**

MICRÒFON AMB FILTRES

Processadors Digitals

ALUMNES:

Laia March Cervantes

David Hernández Morales



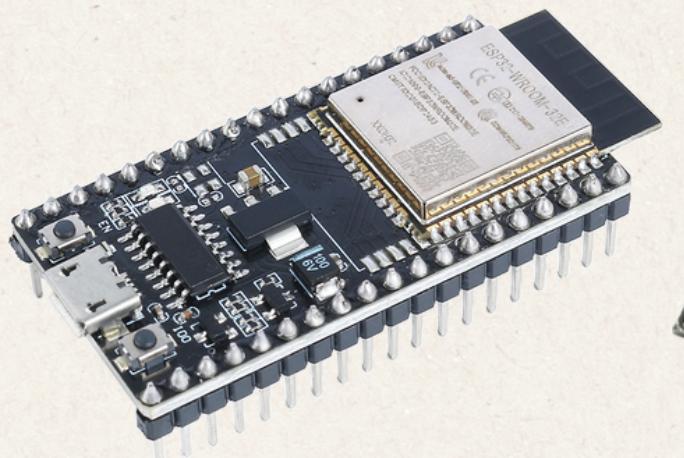
Index

03	Introducció i Materials
04	Costos
05	Diagrama de blocs
06	Diagrama de flux
07	Esquema de muntatge
08	Pàgina Web
09	Explicació del codi
10	Demostració

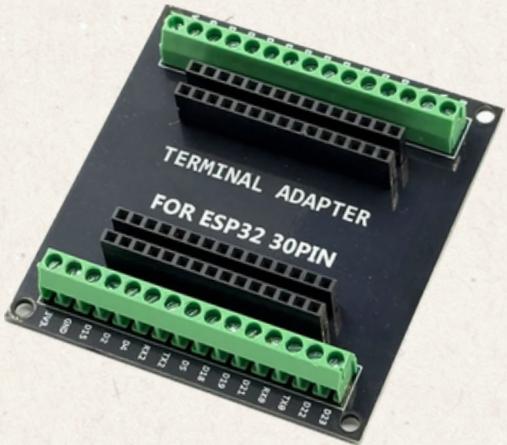
Introducció i Materials

Micròfon amb efecte de pitch

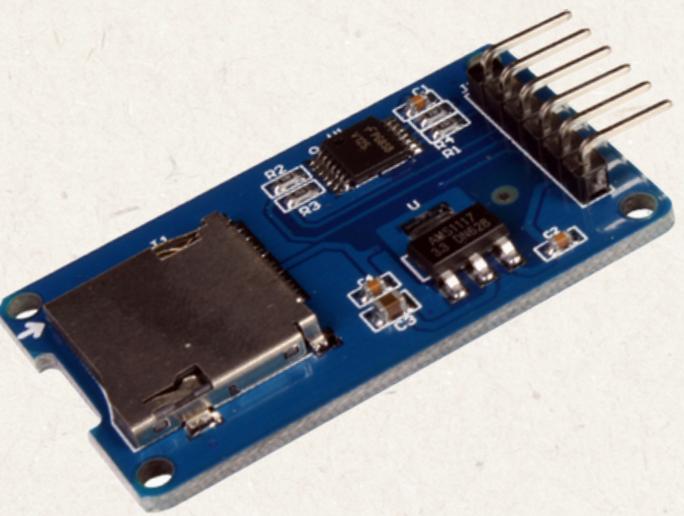
Desenvolupat amb una placa ESP32-S3, aquest sistema aplica un efecte de veu, permet gravacions i gestiona fitxers d'àudio des d'una interfície web, emmagatzemant-los en una targeta SD.



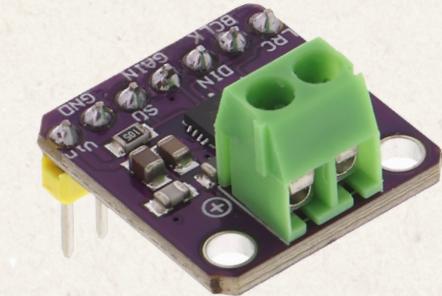
ESP32-S3



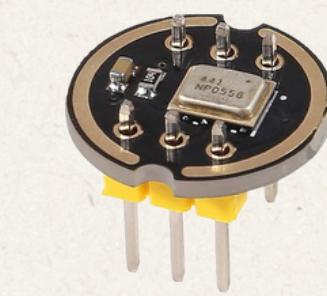
Suport



Mòdul
microSD



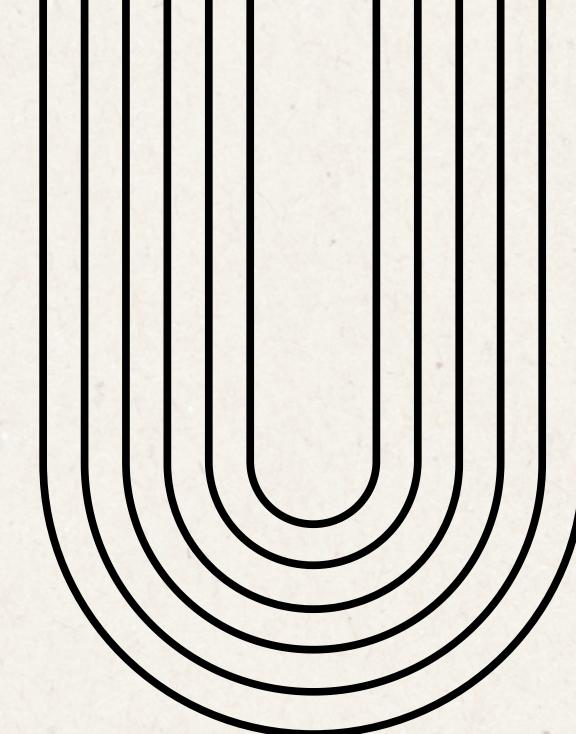
Aplificador
MAX98357



Mòdul
micròfon I2S



Cables
mini-jack



Component	Preu orientatiu (€)
ESP32-S3	5,17
Mòdul tarjeta microSD	1,08
Aplificador MAX98357	1,75
Adaptador Jack	1,25
Micròfon I2S	3,19
Cost total	12,44

Diagrama de blocs

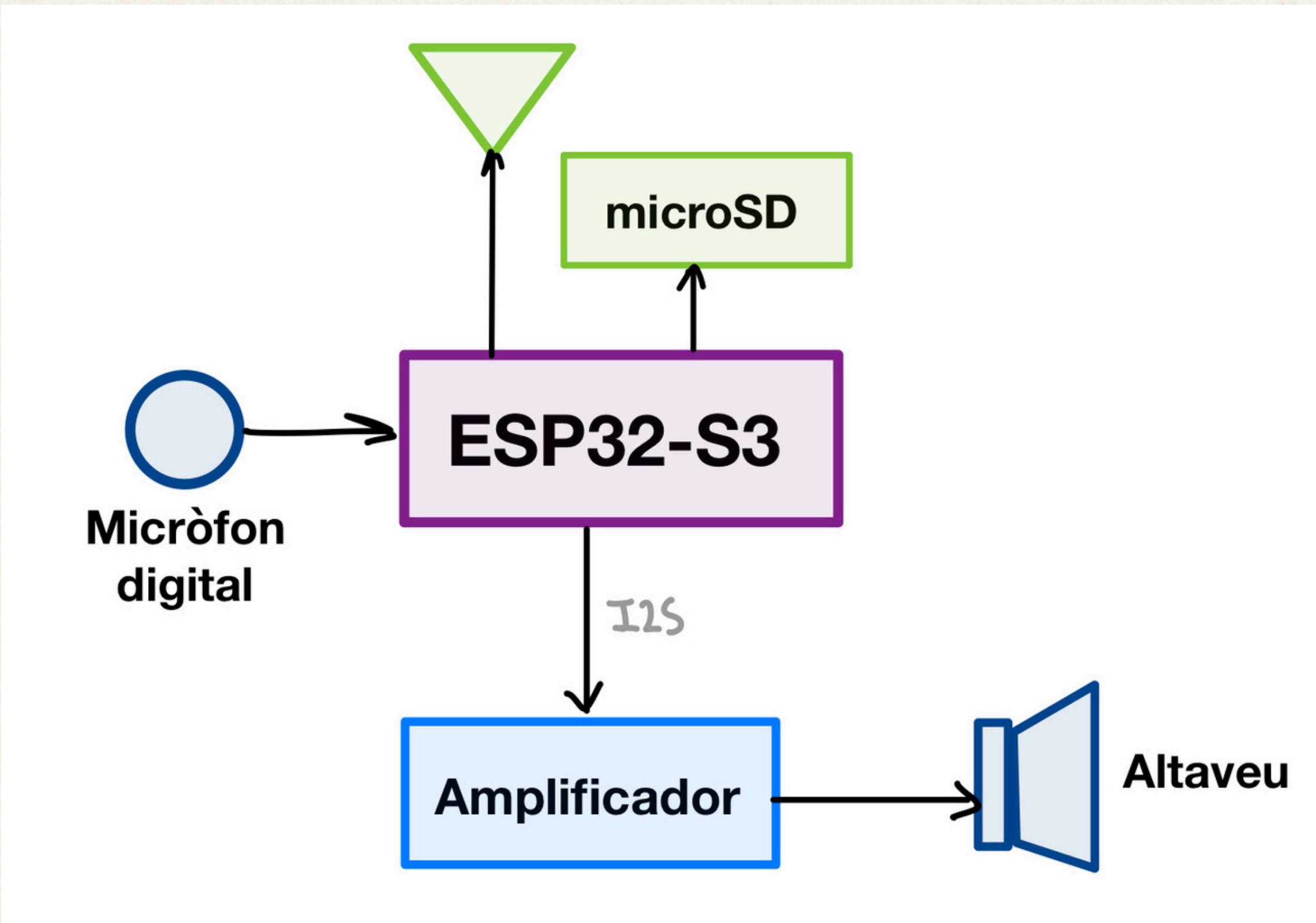
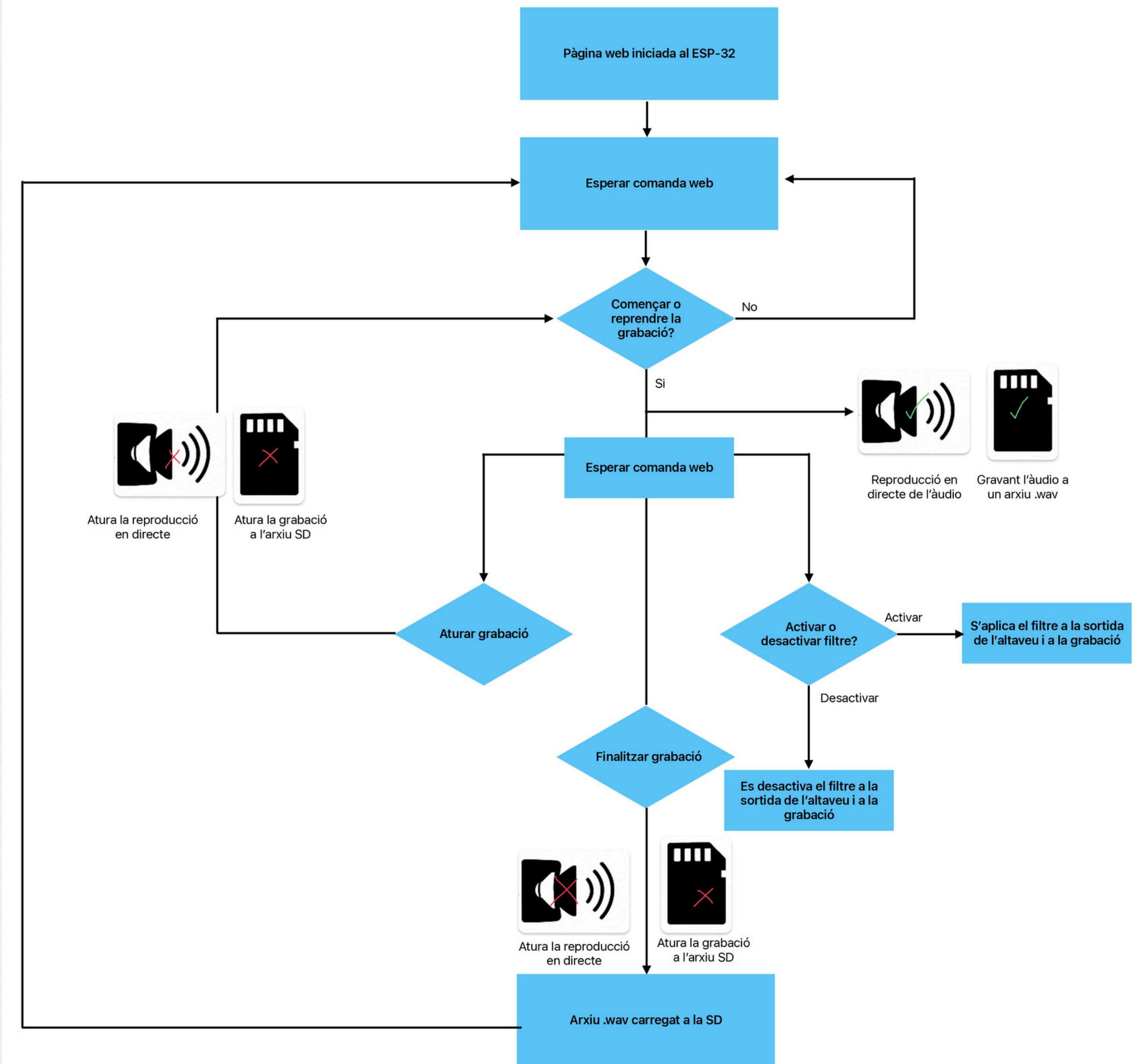
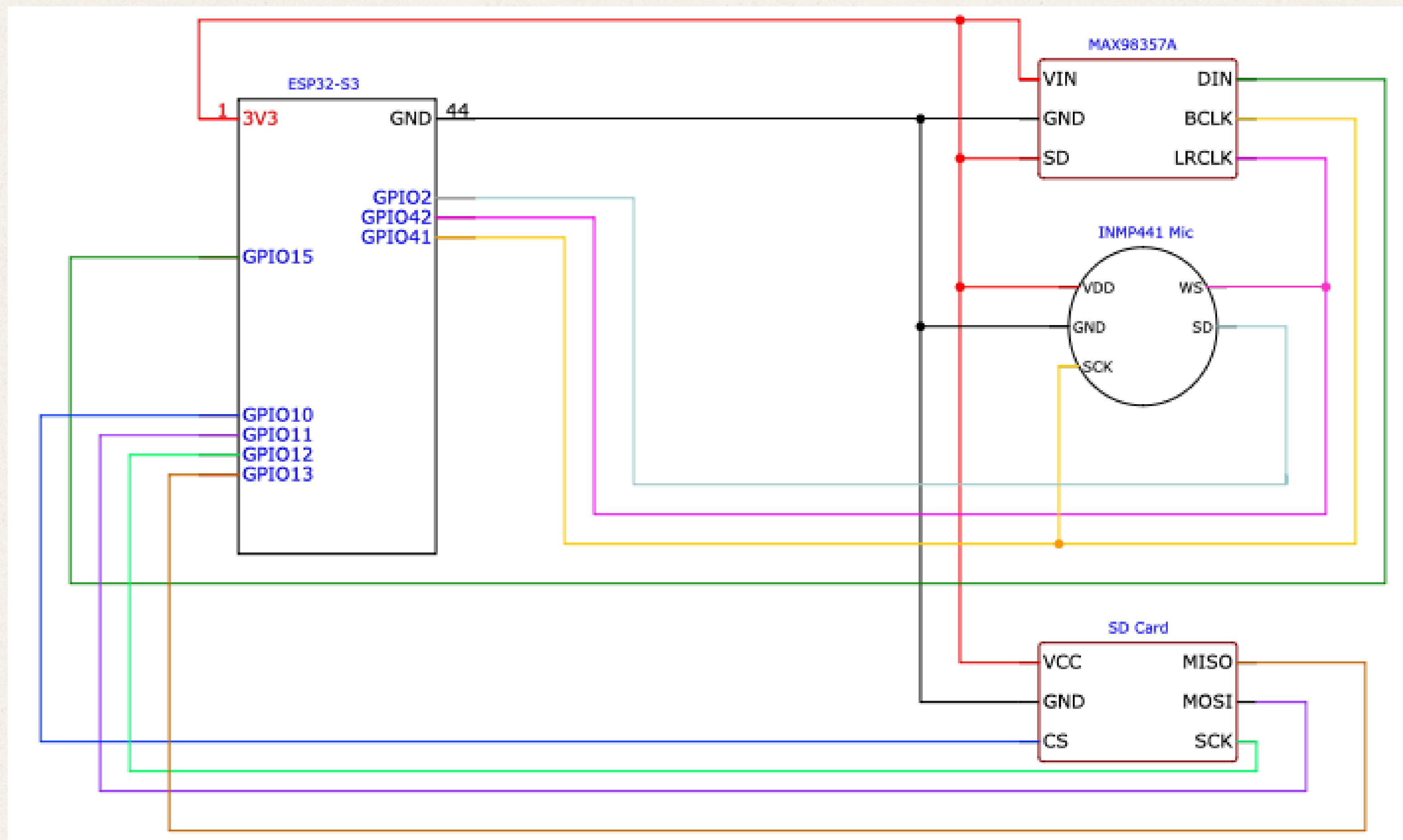


Diagrama de flux

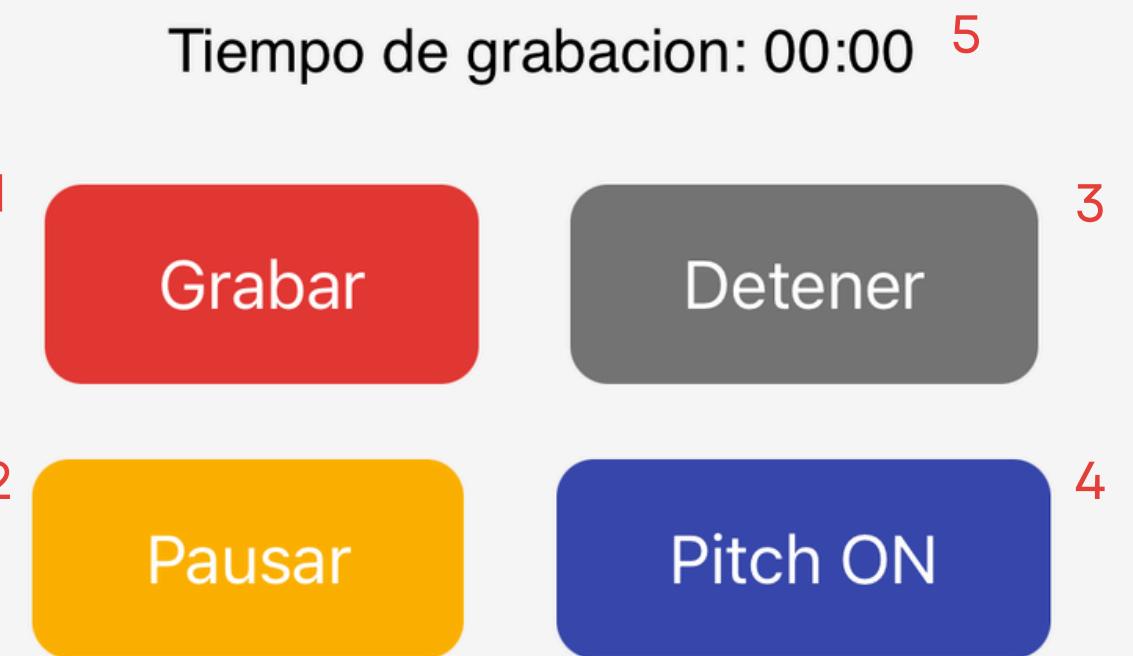


Esquema de muntatge



Control de Grabacion

- 01** Començar a grabar
- 02** Aturar /Reprendre la grabació
- 03** Aturar la grabació
- 04** Activar / Desactivar filtre
- 05** Temps de grabació



Pàgina Web

Una vegada executat el codi a la ESP32, haurás de connectarte al wifi Projecte_PD amb la contraseña 12345678. Després haurás d'introduir al teu navegador la IP que surt pel monitor serial.

Codi

```
1 // Librerías necesarias para WiFi, SD, SPI e I2S
2 #include <Arduino.h>
3 #include <WiFi.h>
4 #include <SPI.h>
5 #include <SD.h>
6 #include <driver/i2s.h>
7
8 // Definición de pines para I2S (interfaz de audio digital)
9 #define I2S_WS      42
10 #define I2S_SD_IN   2
11 #define I2S_SD_OUT  15
12 #define I2S_SCK     41
13
14 // Definición de pines para SPI y SD
15 #define SD_CS      10
16 #define SPI_MOSI   11
17 #define SPI_MISO   13
18 #define SPI_SCK    12
19
20 // Datos de la red WiFi en modo punto de acceso
21 const char* ssid = "ESP32-David_Laia";
22 const char* password = "123456789";
23 WiFiServer server(80); // Puerto 80 para HTTP
24
25 // Variables globales de audio
26 File wavFile;
27 bool grabando = false;
28 bool pausado = false;
29 bool pitchActivo = false; // Efecto pitch activado o no
30
31 uint32_t dataChunkSize = 0;
32 unsigned long lastFlushTime = 0;
33 const unsigned long flushInterval = 1000;
34 const uint32_t sampleRate = 16000;
35 const uint16_t bitsPerSample = 16;
36 const uint16_t numChannels = 1;
37 unsigned long grabacionStartTime = 0;
38 unsigned long tiempoPausado = 0;
39 unsigned long pausaInicio = 0;
```

```
41 // Configuración del bus I2S
42 void setupI2S() {
43     i2s_config_t i2s_config = {
44         .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX | I2S_MODE_TX),
45         .sample_rate = sampleRate,
46         .bits_per_sample = I2S_BITS_PER_SAMPLE_16BIT,
47         .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
48         .communication_format = I2S_COMM_FORMAT_I2S_MSB,
49         .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
50         .dma_buf_count = 8,
51         .dma_buf_len = 512,
52         .use_apll = false,
53         .tx_desc_auto_clear = true,
54         .fixed_mclk = 0
55     };
56
57     i2s_pin_config_t pin_config = {
58         .bck_io_num = I2S_SCK,
59         .ws_io_num = I2S_WS,
60         .data_out_num = I2S_SD_OUT,
61         .data_in_num = I2S_SD_IN
62     };
63
64 // Instalación del controlador I2S
65 i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
66 i2s_set_pin(I2S_NUM_0, &pin_config);
67 i2s_zero_dma_buffer(I2S_NUM_0); // Limpiar buffer DMA
68 }
69
70 // Función para escribir el encabezado WAV en el archivo
71 void writeWavHeader(File &file, uint32_t dataLength) {
72     uint32_t chunkSize = 36 + dataLength;
73     uint16_t audioFormat = 1;
74     uint32_t byteRate = sampleRate * numChannels * bitsPerSample / 8;
75     uint16_t blockAlign = numChannels * bitsPerSample / 8;
```

```

77  file.seek(0); // Inicio del archivo
78  file.write((const uint8_t*)"RIFF", 4);
79  file.write((uint8_t*)&chunkSize, 4);
80  file.write((const uint8_t*)"WAVEfmt ", 8);
81  uint32_t subchunk1Size = 16;
82  file.write((uint8_t*)&subchunk1Size, 4);
83  file.write((uint8_t*)&audioFormat, 2);
84  file.write((uint8_t*)&numChannels, 2);
85  file.write((uint8_t*)&sampleRate, 4);
86  file.write((uint8_t*)&byteRate, 4);
87  file.write((uint8_t*)&blockAlign, 2);
88  file.write((uint8_t*)&bitsPerSample, 2);
89  file.write((const uint8_t*)"data", 4);
90  file.write((uint8_t*)&dataLength, 4);
91 }
92
93 // Buscar un nombre de archivo disponible tipo /audioX.wav
94 String getNextAvailableFilename() {
95  int fileIndex = 1;
96  String filename;
97  while (true) {
98    filename = "/audio" + String(fileIndex) + ".wav";
99    if (!SD.exists(filename)) return filename;
100   fileIndex++;
101 }

```

```

152 // Configuración inicial del sistema
153 void setup() {
154   Serial.begin(115200);
155   delay(1000);
156
157   SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
158   if (!SD.begin(SD_CS)) {
159     Serial.println("✗ Error inicializando SD.");
160     while (true);
161   }
162
163   Serial.println("✓ SD OK");
164   setupI2S();
165   Serial.println("✓ I2S OK");
166
167   WiFi.softAP(ssid, password);
168   server.begin();
169   Serial.print("● IP Web Server: ");
170   Serial.println(WiFi.softAPIP());
171 }
172
173 // Procesar comandos de teclado (serie)
174 void procesarComandoTeclado(char comando) {
175   switch (comando) {
176     case 'r':
177       // Iniciar grabación
178       Serial.println("● [Serial] Iniciando grabación");
179       if (!grabando) {
180         String filename = getNextAvailableFilename();
181         wavFile = SD.open(filename, FILE_WRITE);
182         if (!wavFile) {
183           Serial.println("✗ [Serial] Error al crear archivo WAV");
184           return;
185         }
186         for (int i = 0; i < 44; i++) wavFile.write((uint8_t)0);
187         dataChunkSize = 0;
188         grabando = true;
189         pausado = false;
190         grabacionStartTime = millis();
191         tiempoPausado = 0;
192       }
193     break;

```

```

193     break;
194
195 case 's':
196 // Detener grabación
197 Serial.println("▣ [Serial] Deteniendo grabación");
198 if (grabando) {
199     writeWavHeader(wavFile, dataChunkSize);
200     wavFile.flush();
201     wavFile.close();
202     grabando = false;
203     pausado = false;
204 }
205 break;
206
207 case 'p':
208 // Pausar o reanudar grabación
209 if (grabando) {
210     pausado = !pausado;
211     if (pausado) {
212         pausaInicio = millis();
213         Serial.println("■ [Serial] Grabación pausada");
214     } else {
215         tiempoPausado += millis() - pausaInicio;
216         Serial.println("▶ [Serial] Grabación reanudada");
217     }
218 }
219 break;
220
221 case 'e':
222 // Activar o desactivar efecto pitch
223 pitchActivo = !pitchActivo;
224 Serial.println(pitchActivo ? "▣ [Serial] Pitch activado" : "▢ [Serial] Pitch desactivado");
225 break;
226
227 default:
228 Serial.println("?] [Serial] Comando no reconocido. Usa: r = grabar, s = stop, p = pausa, e = pitch");
229 break;
230 }

```

```

233 // Calcular duración real de la grabación
234 unsigned long calcularTiempoGrabacion() {
235     if (!grabando) return 0;
236
237     if (pausado) {
238         return pausaInicio - grabacionStartTime - tiempoPausado;
239     } else {
240         return millis() - grabacionStartTime - tiempoPausado;
241     }
242 }
243
244 // Bucle principal
245 void loop() {
246     WiFiClient client = server.available();
247
248     // Comando desde puerto serie
249     if (Serial.available()) {
250         char c = Serial.read();
251         procesarComandoTeclado(c);
252     }
253
254     // Procesar petición web
255     if (client) {
256         String req = "";
257         while (client.connected() && client.available()) {
258             req += (char)client.read();
259         }
260
261         // Analizar comandos por URL: r, s, p, e
262         if (req.indexOf("GET /?cmd=r") != -1) {
263             Serial.println("🔴 Iniciando grabación");
264             String filename = getNextAvailableFilename();
265             wavFile = SD.open(filename, FILE_WRITE);
266             if (!wavFile) {
267                 Serial.println("🔴 Error al abrir archivo WAV.");
268                 return;
269             }
270             for (int i = 0; i < 44; i++) wavFile.write((uint8_t)0);
271             dataChunkSize = 0;
272             grabando = true;
273             pausado = false;
274             grabacionStartTime = millis();
275             tiempoPausado = 0;

```

```
275     tiempoPausado = 0;
276 }
277
278 if (req.indexOf("GET /?cmd=s") != -1 && grabando) {
279     Serial.println("▣ Deteniendo grabación");
280     writeWavHeader(wavFile, dataChunkSize);
281     wavFile.flush();
282     wavFile.close();
283     grabando = false;
284     pausado = false;
285 }
286
287 if (req.indexOf("GET /?cmd=p") != -1 && grabando) {
288     pausado = !pausado;
289     if (pausado) {
290         pausaInicio = millis();
291         Serial.println("■ Grabación pausada");
292     } else {
293         tiempoPausado += millis() - pausaInicio;
294         Serial.println("▶ Grabación reanudada");
295     }
296 }
297
298 if (req.indexOf("GET /?cmd=e") != -1) {
299     pitchActivo = !pitchActivo;
300     Serial.println(pitchActivo ? "🎛️ Pitch activado" : "🔕 Pitch desactivado");
301 }
302
303 // Enviar HTML con botones y temporizador
304 unsigned long tiempo = calcularTiempoGrabacion();
305 unsigned int minutos = tiempo / 60000;
306 unsigned int segundos = (tiempo % 60000) / 1000;
307 char buffer[10];
308 sprintf(buffer, "%02d:%02d", minutos, segundos);
```

```
310    client.println("HTTP/1.1 200 OK");
311    client.println("Content-type:text/html\r\n");
312    client.println("<!DOCTYPE html><html><head><meta name='viewport' content='width=device-width, initial-scale=1'>");
313    client.println("<style>");
314    client.println("body { font-family: sans-serif; text-align: center; background-color: #f5f5f5; margin: 0; padding: 20px; }");
315    client.println("h1 { color: #333; }");
316    client.println(".button { display: inline-block; margin: 10px; padding: 15px 30px; font-size: 18px; border: none; border-radius: 10px; cursor: pointer; transition: background 0.3s ease; }");
317    client.println(".record { background-color: #e53935; color: white; }");
318    client.println(".record:hover { background-color: #c62828; }");
319    client.println(".stop { background-color: #757575; color: white; }");
320    client.println(".stop:hover { background-color: #616161; }");
321    client.println(".pause { background-color: #ffb300; color: white; }");
322    client.println(".pause:hover { background-color: #ffa000; }");
323    client.println(".effect { background-color: #3949ab; color: white; }");
324    client.println(".effect:hover { background-color: #303f9f; }");
325    client.println("</style></head><body>");
326    client.println("<h1>Control de Grabacion</h1>");
327    client.println("<p>Tiempo de grabacion: <span id='tiempo'>" + String(buffer) + "</span></p>");
328    client.println("<script>");
329    client.println("let tiempoMs = " + String(tiempo) + ";");
330    client.println("let grabando = " + String(grabando ? "true" : "false") + ";");
331    client.println("let pausado = " + String(pausado ? "true" : "false") + ";");
332    client.println("setInterval(() => {");
333    client.println("  if (grabando && !pausado) {");
334    client.println("    tiempoMs += 1000;");
335    client.println("    let mins = Math.floor(tiempoMs / 60000);");
336    client.println("    let secs = Math.floor((tiempoMs % 60000) / 1000);");
337    client.println("    document.getElementById('tiempo').textContent = String(mins).padStart(2, '0') + ':' + String(secs).padStart(2, '0');");
338    client.println("  }");
339    client.println("}, 1000);");
340    client.println("</script>");
341    client.println("<button class='button record' onclick=\"location.href='/?cmd=r'\">Grabar</button>");
342    client.println("<button class='button stop' onclick=\"location.href='/?cmd=s'\">Detener</button>");
343    client.println("<button class='button pause' onclick=\"location.href='/?cmd=p'\">" + String(pausado ? "Reanudar" : "Pausar") + "</button>");
344    client.println("<button class='button effect' onclick=\"location.href='/?cmd=e'\">" + String(pitchActivo ? "Pitch OFF" : "Pitch ON") + "</button>");
345    client.println(listAudioFiles());
346    client.println("</body></html>");
347    client.stop();
348 }
```

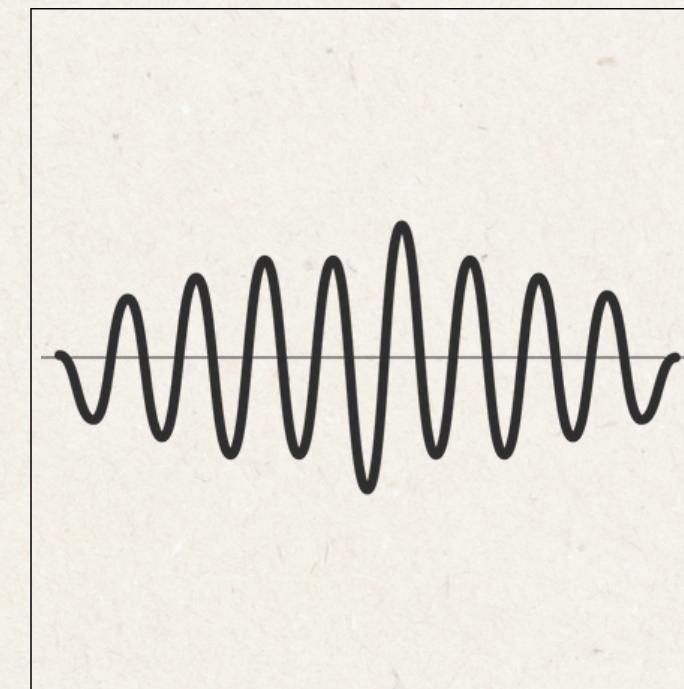
```
350 // Capturar y guardar audio en tiempo real
351 if (grabando && !pausado) {
352     int16_t samples[512];
353     size_t bytesRead;
354     esp_err_t result = i2s_read(I2S_NUM_0, &samples, sizeof(samples), &bytesRead, portMAX_DELAY);
355     if (result == ESP_OK && bytesRead > 0) {
356         size_t bytesWritten;
357         if (pitchActivo) {
358             // Procesar pitch agudo suavizado (interpolación)
359             int16_t processed[256];
360             int outIndex = 0;
361             for (int i = 0; i < (bytesRead / 2) - 1; i += 2) {
362                 processed[outIndex++] = (samples[i] + samples[i + 1]) / 2;
363             }
364             int outBytes = outIndex * 2;
365             wavFile.write((uint8_t*)processed, outBytes);
366             dataChunkSize += outBytes;
367             i2s_write(I2S_NUM_0, processed, outBytes, &bytesWritten, portMAX_DELAY);
368         } else {
369             wavFile.write((uint8_t*)samples, bytesRead);
370             dataChunkSize += bytesRead;
371             i2s_write(I2S_NUM_0, samples, bytesRead, &bytesWritten, portMAX_DELAY);
372         }
373
374         // Guardar a SD cada cierto intervalo
375         if (millis() - lastFlushTime >= flushInterval) {
376             wavFile.flush();
377             lastFlushTime = millis();
378         }
379     }
380 }
381 }
```

Demostració



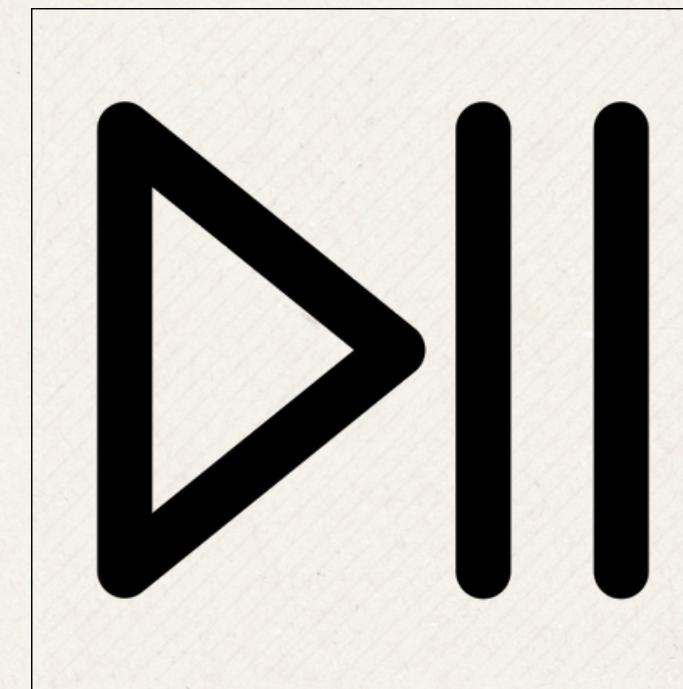
Grabar sense filtre

1



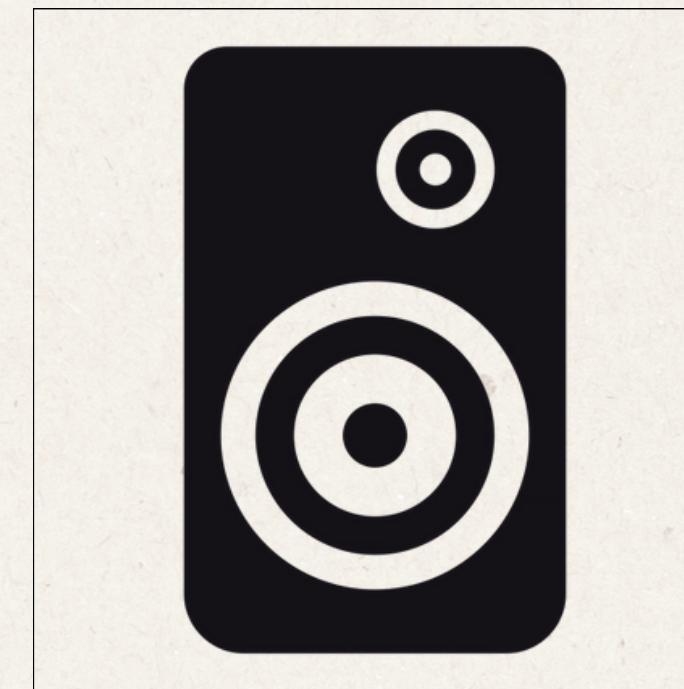
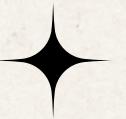
Grabar amb filtre

2



Grabar audio per parts

3



Escoltar els audios

4

