



Universidad
de Cádiz

Escuela Superior
de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA EN ELECTRÓNICA INDUSTRIAL

DISEÑO DE ESTACIÓN ACÚSTICA INTELIGENTE (ESAI)

AUTOR: DANIEL PECCI SÁNCHEZ

Cádiz, 4 de Diciembre de 2023



Universidad
de Cádiz

Escuela Superior
de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA EN ELECTRÓNICA INDUSTRIAL

DISEÑO DE ESTACIÓN ACÚSTICA INTELIGENTE (ESAI)

DIRECTOR: NESTOR MORA NUÑEZ

AUTOR: DANIEL PECCI SÁNCHEZ

Cádiz, 4 de Diciembre de 2023

Tabla de contenido

1	INTRODUCCIÓN	11
1.1	Alcance, objetivo y motivaciones	12
1.2	Estado del arte.	14
2	TEORÍA, MATERIALES USADOS Y PROCEDIMIENTOS	16
2.1	Sistema de captación acústico	17
2.1.1	Introducción teórica. Unidades de medida	17
2.1.2	Micrófonos. Clases y características	23
2.1.3	Protocolo I2S	28
2.1.4	Material utilizado para el proyecto	29
2.2	Unidad de control de la ESAI	31
2.2.1	Placa y módulo	31
2.2.2	Características del chip	33
2.2.3	Configuración del IDE	36
2.2.4	Gestión de memoria	38
2.3	Protocolos de comunicación	43
2.3.1	Introducción al internet de las cosas	43
2.3.2	Protocolo HTTP	45
2.3.3	Protocolo MQTT	50
2.4	SSL/TLS	55
2.4.1	Guía para generar certificados con OpenSSL	56
2.5	Configuración del servidor	59
2.5.1	Django	59
2.5.2	Bróker Mosquitto local	63
2.6	Modelo de red neuronal profundo YAMNet	65
3	FIRMWARE	69
3.1	ESP32	70
3.1.1	Estado de portal captivo	71
3.1.2	Estado de escucha/chivato	73
3.2	DJANGO	84
4	DISEÑO ELECTRÓNICO	88
4.1	Diagrama de bloques del sistema	89

4.2	Esquemático	91
4.2.1	MCU	91
4.2.2	MÓDULO DE ALIMENTACIÓN	102
4.2.3	MÓDULO DE CARGA	106
4.2.4	MEMS	109
4.2.5	MÓDULO ADAPTADOR USB-UART.....	111
4.2.6	BUS USB.....	113
4.2.7	CIRCUITO DE AUTO-ARRANQUE (AUTO-BOOTLOADER)	117
4.2.8	MONITOR DE BATERÍA	118
4.2.9	PINES DE SALIDA DE LA PLACA	122
4.3	Diseño de la PCB	123
4.3.1	Consideraciones sobre EMI. Teoría y métodos de mejora prácticos.....	123
4.3.2	Vista de capas.....	134
4.3.3	Información técnica de la PCB:.....	135
4.4	Consumo:.....	136
4.4.1	Consumo de corriente teórico aproximado	136
4.4.2	Consideraciones adicionales de ahorro energético.....	136
5	RESULTADOS	138
6	Bibliografía.....	146
7	ANEXO. Incidencias con el protocolo I2S.....	151

ÍNDICE DE FIGURAS

Figura 1. Respuesta en frecuencia de las ponderaciones A, C y Z [3].....	18
Figura 2. Comparativa de indicadores en un micrófono digital. Extraído de "Application Note AN-1112", TDK Invensense.....	22
Figura 3. Esquema electrónico y construcción física de un ECM. [10].....	24
Figura 4. Estructura de un micrófono MEMS. [11].....	25
Figura 5. Modulación PDM de una señal senoidal [12]	26
Figura 6. Interfaz de comunicación PDM en un formato estéreo. [13].....	26
Figura 7. Diagrama de bloques de MEMS con interfaz I2S. "SPH0645LM4H-1 datasheet"	27
Figura 8. Trama I2S en modo estándar. "INMP441 datasheet".....	29
Figura 9. Vista de la capa inferior del módulo INMP441. Imagen extraída del producto en Amazon	30
Figura 10. Vista de la placa ESP32 DOIT DEVKIT V1. [17]	31
Figura 11. Diagrama esquemático del módulo WROOM-32, "ESP32 datasheet"	32
Figura 12. Diagrama funcional de bloques del chip, "ESP32 Series datasheet".....	33
Figura 13. Consumo del chip para los distintos modos."ESP32 Series datasheet"	34
Figura 14. Vista del programa VS Code. Creada por el autor	37
Figura 15. Organización de la SRAM y ampliación de la DRAM. [19].....	39
Figura 16. Código añadido en el archivo main.cpp para aumentar el tamaño de la pila de la tarea loopTask. Creado por el autor.....	40
Figura 17. Partición de la memoria FLASH del ESP32. [20]	42
Figura 18. Capas del modelo OSI con ejemplos. [21].....	44
Figura 19. Mensaje de la consola por un desbordamiento del módulo criptográfico. Creado por el autor	48
Figura 20. Formato de reconocimiento de comandos de MQTT. [25]	50
Figura 21. Estructura de un mensaje MQTT. [25].....	51
Figura 22. Parte baja del byte de control, correspondiente a las banderas de control. [25].....	51
Figura 23. Esquema con los pasos para generar un certificado firmado localmente.	55
Figura 24. Guía de instalación de OpenSSL. Creado por el autor	56
Figura 25. Espectrograma realizado para un segmento de 2,4MHz de la banda FM. Creado por el autor.	66
Figura 26. Estructura de un espectrograma. Ventana temporal e intervalos de FFT. [29].....	66
Figura 27. Diagrama de flujos del firmware de la ESAI. Creación del autor.....	70
Figura 28. Desbordamiento del buffer DMA. [30].....	77
Figura 29. Representación gráfica de la distribución del procesamiento y almacenamiento entre varios buffers. [30].....	78
Figura 30. Forma canónica del formato WAVE. [31].....	81
Figura 31. Diagrama de bloques de la placa ESAI	89
Figura 32. Ampliación del módulo ESP32-S3-MINI-1	90
Figura 33. Resultados del buscador de productos de Espressif.....	93
Figura 34. Esquemático de la datasheet del módulo ESP32-S3 MINI-1	94
Figura 35. Diagrama de bloques funcional del chip, "ESP32-S3 Series".....	95
Figura 36. Sobreimpulsos y subimpulsos generados por una incorrecta adaptación de impedancia [35].....	96
Figura 37. MCU. Fragmento extraído del esquemático de la ESAI.....	98
Figura 38. Comparativa entre los parámetros de medida de una fuente luminosa.	99
Figura 39. Diagrama con el timing de los parámetros de Power-Up y Reset. ESP32-S3 Series datasheet.....	100

Figura 40. Módulo de alimentación. Fragmento de esquemático de la ESAI	102
Figura 41. Gráfica PSRR frente a frecuencia. Datasheet del AP2112	103
Figura 42. Módulo gestor de carga. Fragmento del esquemático de la ESAI	106
Figura 43. Gráfica del voltaje y corriente de carga frente al tiempo (batería de 1000 mAh). Extraído de la datasheet del MCP73831/2.....	108
Figura 44. Módulo de captación acústico. Fragmento del esquemático de la ESAI.....	109
Figura 45. Módulo adaptador USB-UART. Fragmento del esquemático de la ESAI.....	111
Figura 46. USB-OTG y USB-UART. Fragmento del esquemático de la ESAI	113
Figura 47. Características del diodo LESD5D5.0CT1G [43].....	115
Figura 48. Modelado de un diodo TVS [44]	116
Figura 49. Circuito auto-bootloader. Fragmento del esquemático de la ESAI.....	117
Figura 50. Monitor de batería. Fragmento del esquemático de la ESAI.....	118
Figura 51. Comparativa de linealidad del ADC para una resolución de 12 bits	119
Figura 52. Comparativa de mitigación de ruido del ADC. Análisis ofrecido por Espressif [48]	120
Figura 53. Botones de control y pines de salida de la placa.....	122
Figura 54. Trazado de pistas de peor a mejor. [49].....	124
Figura 55. Ejemplos de bucles de área malos(a,c) y mejores(b,d). [49]	125
Figura 56. Corriente generada por diferencia de potencial en las tierras [50].....	126
Figura 57. Circuito equivalente de diafonía [51].....	127
Figura 58. Buena praxis de diseño para evitar diafonía entre señales. Fragmento de PCB de la ESAI	127
Figura 59. ICS-43434 datasheet. Distribución de los pads y dimensiones en mm	128
Figura 60. Modelo equivalente de un condensador [53].....	130
Figura 61. Impedancia característica de un arreglo de distintos MLCC [54].....	131
Figura 62. Resultado de usar un arreglo de condensadores distintos frente a una arreglo de condensadores distintos repetidos múltiples veces.	132
Figura 63. Guía de posicionamiento del módulo con antena PCB. Guía de diseño del ESP32-S3	133
Figura 64. PCB de la ESAI. Capa superior	134
Figura 65. PCB de la ESAI. Capa inferior.....	134
Figura 66. Conexión desde móvil android con el SoftAP generado por la ESAI. Creado por el autor.	138
Figura 67. Portal captivo, configuración de red. Creado por el autor	139
Figura 68. Plantilla html con mensaje de éxito de la configuración de red. Creado por el autor....	139
Figura 69. Página web del portal para la configuración de la ESAI. Creado por el autor.....	140
Figura 70. Plantilla html final del portal. Creado por el autor	140
Figura 71. Vista de la terminal serial de la ESAI. Creado por el autor.....	141
Figura 72. Configuración de la estación en modo chivato. Creado por el autor.....	141
Figura 73. Pruebas realizadas para mostrar el funcionamiento total del sistema ESAI. Creado por el autor.	142
Figura 74. Plantilla html de Django para la configuración de la ESAI. Creado por el autor	143
Figura 75. Solicitud de OTA enviada desde el servidor, proceso de OTA y ejecución del nuevo programa. Creado por el autor.	144
Figura 76. Pruebas realizadas con el INMP441 y la placa ESP32 DEV montados en una protoboard. Creado por el autor	152
Figura 77. Resultados de la prueba con 16 bits por muestra y 32 bits de canal. Creado por el autor	154
Figura 78. Código Python para guardar datos en formato binario. Creado por el autor.....	155
Figura 79. Pestaña de importación de archivos de Audacity. Creado por el autor.....	155

<i>Figura 80. Visualización de los datos de audio en bruto. Creado por el autor</i>	<i>155</i>
<i>Figura 81. Respuesta ante la incidencia publicada en GitHub. Creado por el autor.....</i>	<i>156</i>
<i>Figura 82. Fragmento de código del proyecto de comunicación i2s con el framework ESP-IDF. Creado por el autor</i>	<i>157</i>
<i>Figura 83. Prueba con la librería i2s_std.h. Creado por el autor</i>	<i>158</i>
<i>Figura 84. Prueba realizada con la versión 1.8.18 de Arduino. Creado por el autor.....</i>	<i>159</i>
<i>Figura 85. Audio siendo reproducido de forma exitosa en audacity. Creado por el autor.....</i>	<i>160</i>
<i>Figura 86. Representación hexadecimal de los 4 bytes trasmitidos por cada muestra. Creado por el autor</i>	<i>161</i>

ÍNDICE DE TABLAS

<i>Tabla 1. Valores por defecto de los "strapping pins" _____</i>	<i>¡Error! Marcador no definido.</i>
<i>Tabla 2. Comparativa de características de los reguladores AMS1117 y AP2112. Datos extraídos de las datasheets _____</i>	<i>104</i>
<i>Tabla 3. Comparativa de MEMS digitales _____</i>	<i>110</i>
<i>Tabla 4. "ADC Characteristics" extraído del documento "ESP32-S3 series" _____</i>	<i>119</i>

1 INTRODUCCIÓN

1.1 Alcance, objetivo y motivaciones

La motivación de este proyecto surge con la idea de desarrollar un sistema que esté a la altura y a la vanguardia del mundo actual que vivimos, cada vez más cerca de alcanzar la singularidad tecnológica.

Los dispositivos inteligentes, la interconectividad, las comunicaciones inalámbricas, las redes neuronales de aprendizaje profundo, la nube e Internet están tomando el control de la industria y de nuestras vidas cotidianas. Los avances en fotolitografía han permitido la miniaturización extrema de los microchips, hasta tal punto de que se empieza a hablar ya de ángstroms en vez de nanómetros para definir la separación del canal de los MOSFETs que lo componen. Tamaños de transistores más pequeños ha llevado a una mayor velocidad de propagación de las señales, menor consumo de potencia para conmutar y una mayor densidad de transistores para una misma superficie. Como consecuencia de esto último, chips mucho más pequeños tienen capacidades de cómputo impresionantes. La tecnología CMOS ha reducido aún más el consumo energético de los microchips (el consumo estático es virtualmente cero) y encima de todo lo anterior la tecnología de baterías de ion y polímero de litio mejora anualmente, permitiendo mayor densidad de energía por mismo volumen de celda y a menor coste. Este escenario está llevando irremisiblemente a la integración de sistemas inteligentes e interconectados, completamente independientes, en espacios cada vez más reducidos.

El proyecto de la ESAI (Estación Acústica Inteligente) no pretende desarrollar un concepto tecnológico nuevo sino aunar lo mejor del marco existente, para crear una aplicación única y eficiente.

El objetivo será lograr diseñar e implementar un sistema de detección acústica autónomo que estará formado por un conjunto de nodos inteligentes interconectados con un servidor central que adquiere, almacena y procesa los datos transmitidos por estos nodos. Este procesamiento se llevará en última instancia por un modelo de Red Neuronal Convolucional (CNN) entrenado en la identificación y clasificación de sonido. Los nodos, denominados de ahora en adelante como estaciones o ESAIs, aprovecharán al máximo la capacidad de cómputo de bajo consumo y coste reducido de un microcontrolador. El sistema potencialmente puede ser capaz de detectar más de 500 tipos de sonidos, desde roturas de cristales, encendido de motores, sonido de maquinaria hasta decenas de sonidos asociados a los humanos y animales pasando por la clasificación de géneros musicales. Para la muestra del proyecto se ha limitado al alcance a la detección de 4 eventos. Cada evento está formado por la agrupación de un conjunto de clases similares que describen un mismo fenómeno. Los eventos a detectar incluyen la rotura de cristales (para reconocer intrusiones), el llanto de bebé (como monitor de bebé), la detección de pájaros y, finalmente, la detección de presencia humana (respiración, tos, habla, pisadas, silbido, etc.). La adición de más eventos no cambia la estructura interna del sistema, sino que no supone más complicación que la de repetir bloques existentes.

Otra parte importante del proyecto será la flexibilidad, personalización y escalabilidad:

-Flexibilidad: La función o modo de operación de una estación puede transformarse en otra en cualquier instante.

-Personalización: Otro objetivo de este proyecto es la de ofrecer una experiencia óptima al usuario. Aunque los nodos operen de forma autónoma, se ofrecerá la posibilidad de escuchar de forma remota y en tiempo real el sonido capturado por la estación, así como visualizar la clasificación del sonido realizado por la red neuronal. Por otro lado, el usuario podrá configurar el modo de operación, los eventos a detectar o incluso actualizar el firmware remotamente de cualquier estación.

-Escalabilidad: El sistema estará preparado para la adición o combinación de nuevos módulos que se integren fácilmente sin necesidad de rediseñar o cambiar lo existente.

La estación podrá operar de dos formas principales:

- Modo escucha: Captará el sonido de su entorno a través del micrófono integrado y transmitirá los paquetes de audio al servidor. Recibirá y procesará la predicción del modelo para detectar eventos. Si el modo silencio no está activado, activará una salida que puede ser por ejemplo un led indicador, un zumbador o un driver para controlar una alarma acústica de alto nivel sonoro.

- Modo chivato: No captará el sonido ambiente. Se limitará a recibir y procesar la inferencia para la detección de eventos del sonido captado por otra ESAI. Este modo surge como forma de avisar de forma instantánea y efectiva al usuario que un evento ha ocurrido, aunque se encuentre a miles de kilómetros del lugar donde está la estación escucha, siempre y cuando tenga conexión a un punto de acceso conectado a su vez a internet. De esta manera, solventamos de forma elegante el hecho de que el módulo ESP32 usado no tenga capacidad GSM o LTE poder enviar mensajes “sms” o hacer llamadas a un número de teléfono. También se podría implementar esta funcionalidad en el servidor, pero en cualquier caso aumenta complejidad y supone implicar el servicio de terceras partes lo cual incrementa el coste también.

Para la experiencia de este trabajo, se utilizarán dos estaciones (una en escucha y otra en modo chivato) y el servidor se ejecutará en la red local. Este proyecto debe entenderse como el desarrollo de un prototipo, aún en fase experimental pero que sea capaz de cumplir con todos los objetivos anteriores propuestos.

1.2 Estado del arte.

Dado que la ESAI es un sistema polifacético puede realizar la tarea equivalente de distintos dispositivos a la vez.

Sistemas de detección acústica clásicos:

Se caracterizan por constar de un módulo microfónico seguido de una etapa de amplificación y otra etapa de filtrado analógico de la señal. La etapa de filtrado suele ser un grupo de filtros pasabanda especializados para captar una banda de frecuencias muy concreta asociada al fenómeno que se quiere detectar como el de la rotura de vidrios. Sin embargo, la complejidad y diversidad espectral del fenómeno subyacente es mucho mayor que simplemente la activación de ciertas bandas de frecuencia por lo que estos sistemas no son muy robustos y tienden a generar falsas alarmas de forma habitual. En adición a lo anterior, no son capaces de aportar información específica del sonido y sólo sirven para detectar un único tipo de evento. En versiones más modernas de este tipo de detectores, se integran unidades microcontroladoras capaces de realizar procesamiento de señal como la transformada de Fourier o aplicar ciertos algoritmos de reconocimiento de patrones utilizando la capacidad de cómputo del dispositivo. Esto puede ayudar a reducir las falsas alarmas y mejorar la precisión en la detección, aunque seguirá siendo limitada y poco flexible.

Sistemas de detección acústica basados en aprendizaje profundo:

Hasta hace relativamente poco tiempo los métodos de detección de eventos de sonido (SED, "Sound Event Detection") solían emplear una serie de redes neuronales convolucionales (CNN) para extraer características útiles de la señal de audio de entrada y luego utilizaban redes neuronales recurrentes (RNN) para modelar un contexto temporal más largo en las características extraídas. El número de canales de las CNN y el tamaño de las matrices de peso de las RNN tenían un efecto directo en la cantidad total de parámetros del método SED, que podía ser del orden de varios millones. El inconveniente de este sistema es que tiene un mayor tiempo de entrenamiento y dificultad en el flujo de gradiente. La arquitectura de convolución separable en profundidad ha permitido reducir de forma drástica el número de parámetros necesarios para obtener iguales o incluso mejores resultados que el modelo anterior. El modelo YAMNet que es el que utilizamos en este proyecto, está basado en esta arquitectura. Desde que fue lanzado al público de forma gratuita por la empresa Google a finales de 2018 han surgido artículos académicos y proyectos de distintas Universidades e instituciones a lo largo de todo el mundo. Un ejemplo es el artículo titulado "A Sound Events Detection and Localization System based on YAMNet Model and BLE Beacons" publicado a raíz de la decimonovena conferencia internacional sobre comunicaciones móviles e inalámbricas. Este artículo se trata de una propuesta para "smart cities" o ciudades inteligentes. Está basado en el modelo YAMNet y varias Raspberry Pi 3 para la detección y localización espacial de sonidos con el objetivo de alertar de posibles peligros cercanos a personas con discapacidad.

Sin embargo, no existe ningún proyecto a fecha de envío de esta memoria que sea un sistema de captación y detección de eventos personalizables, basado en un microcontrolador de bajo consumo y coste como el ESP32 y con la capacidad de crear una red escalable integrada gracias a que se utilizará un método de comunicación ligero como es el MQTT.

Ver referencias: [1], [2]

2 TEORÍA, MATERIALES USADOS Y PROCEDIMIENTOS.

2.1 Sistema de captación acústico.

En este capítulo se abordarán los tipos de sistemas transductores de sonido existentes para integración en circuitos impresos. Además, se realizará una introducción de conceptos, parámetros y unidades de medida relacionados con el mundo del sonido.

2.1.1 Introducción teórica. Unidades de medida.

El sonido es una onda mecánica que se propaga a través de un medio producida por una fuente sonora que crea variaciones u oscilaciones de presión del medio circundante. El estudio del sonido se realiza desde una perspectiva antropocéntrica. Catalogamos como sonido aquellas frecuencias que se encuentran dentro del rango de nuestra audición (20-20KHz) y aquellas ondas cuyas frecuencias esté por encima o por debajo de dicho rango se denominan ultrasonido e infrasonido respectivamente. Continuando la línea antropocéntrica la manera de analizar y modelar el sonido es siguiendo la escala logarítmica ya que es así como los humanos percibimos el sonido y no de forma lineal. Además, las unidades para medir el sonido también están orientadas en torno al umbral de audición humano y a las bandas de frecuencia dentro de nuestro espectro auditivo que mejor captamos con nuestro transductor biológico, el oído.

-Unidades de medida:

- Nivel de presión sonora SPL (“Sound Pressure Level”): Es una unidad relativa para cuantificar la intensidad de una onda de sonido respecto a un valor de referencia estándar P_0 que representa el umbral mínimo de audición humana correspondiente a una onda de presión acústica de 20 μ Pa. La unidad SPL se calcula de la siguiente manera:

$$SPL(dB \text{ SPL}) = 10 \log_{10} \left(\frac{P}{P_0} \right)$$

Como se comentó previamente, el oído humano no percibe de igual manera todas las frecuencias del espectro auditivo a lo igual que un amplificador operacional no amplifica todas las frecuencias de entrada de igual forma. Como resultado surgió la necesidad de establecer una nueva unidad de medida que adaptase el valor de presión sonora a cómo sería percibido o captado por nuestros sensores naturales:

- Db(A), dB(C), dB(Z): Estas unidades se construyen a partir del valor de un nivel de presión sonora al que se le establece una “ponderación” la cual sirve como factor de corrección que puede aumentar o disminuir el valor del parámetro SPL en función de la frecuencia que corresponda al sonido. La ponderación más usada es la de tipo A y está diseñada para reflejar la respuesta real del oído humano medio, atenuando las

frecuencias que sean bajas o altas y amplificando el rango de frecuencias intermedias comprendidas entre los 1-4 KHz (curiosamente corresponde con el rango de mayor densidad espectral del habla humana). La ponderación C tiene una respuesta más plana, atenuando y amplificando en mucha menor medida y se utiliza para sonidos de alto nivel de presión sonora ya que la respuesta del oído humano se vuelve también más plana en presencia de ondas por encima de los 100 dB SPL. Finalmente, la ponderación Z ofrece una respuesta plana (sin ponderar) para las frecuencias comprendidas entre 10 y 20KHz. A continuación, se muestra en una gráfica de frecuencia versus ganancia los 3 tipos de ponderaciones:

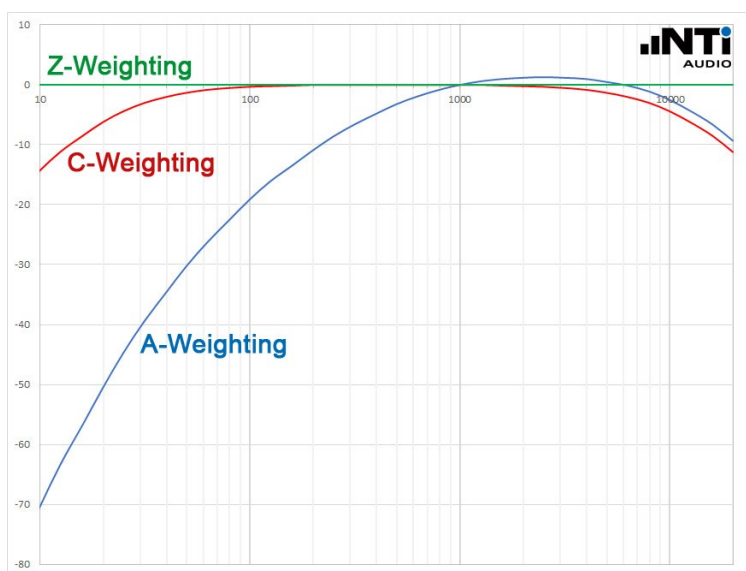


Figura 1. Respuesta en frecuencia de las ponderaciones A, C y Z [3]

Generalmente las hojas de datos de los fabricantes de micrófonos utilizan la unidad dB SPL, dB FS. En caso de mostrar un parámetro con ponderación casi siempre utilizan la de tipo A. Generalmente lo usan para parámetros relacionado con la cantidad de ruido que el micrófono introduce, en parte como estrategia de marketing, ya que al ponderar el valor en dB se verá menor.

-Indicadores más importantes de transductores acústicos

A continuación, se muestran algunos de los parámetros más importantes que se utilizan en la industria del sonido para cuantificar la calidad y eficiencia de un sistema de captación acústico y poder ser comparado con otros. Dado que el lenguaje universal de las hojas de datos es el inglés, se hará en todo momento mención a la versión en inglés.

- Sensibilidad (“sensitivity”):

Indica la capacidad de un micrófono para captar una onda de presión sonora. Para determinar la sensibilidad de los micrófonos se establece una referencia marcada por el estándar internacional de una onda senoidal de 1KHz con una presión sonora medida en el encapsulado de 94 dB SPL (equivalente a 1Pa). La magnitud de salida del micrófono para dicha onda es lo que se cuantifica como la sensibilidad del micrófono. En micrófonos analógicos la sensibilidad se especifica en dBV, es decir la tensión a la salida del micrófono ante la onda de presión sonora (dada en mV/Pa) de 1 Pa estándar utilizada en la prueba de sensibilidad, pero referenciado frente a 1 voltio. Si el fabricante diera el valor de sensibilidad en dB sin más información deberíamos dudar, pues podrían estar dando el dato para una referencia desconocida que podría ser un nivel de referencia inferior (menor a 1 voltio) para maquillar el valor real de sensibilidad y que parezca mayor. La expresión para la sensibilidad referenciada a 1 voltio quedaría de la siguiente manera:

$$Sensibilidad(dBV) = 20 \log_{10} \left(\frac{Sensibilidad \left(\frac{mV}{Pa} \right)}{VRef \left(\frac{1000mV}{Pa} \right)} \right)$$

Por lo tanto, con el dato de sensibilidad en dBV de la datasheet, podríamos calcular la tensión de salida del micrófono ante una onda sonora conocida (1 Pa) simplemente despejando la Sensibilidad (mV/Pa) de la expresión anterior. Esto puede ser de utilidad para tener una idea del rango de tensiones de salida del micrófono.

Nota: El valor de tensión que se utiliza para medir la sensibilidad corresponde al valor eficaz (RMS) de la señal producida por la onda de presión sonora de la prueba.

En cuanto a los micrófonos digitales el concepto de sensibilidad no es tan directo ya que la salida no es una señal de tensión analógica sino un valor digital cuantificado con una determinada resolución que dependerá del ADC integrado. La referencia que se suele utilizar para medir la sensibilidad es el valor de fondo de escala o FS del inglés “Full Scale”, esto es, el máximo valor digital que puede alcanzar el micrófono. Por ejemplo, si tiene una resolución de 12 bits el valor de FS será $2^{12} - 1 = 4095$.

En el caso de que el micrófono tenga mapeado o asociado de forma precisa el valor palabra digital máximo frente a la mayor entrada acústica medible por el sensor, la sensibilidad podría determinarse como la diferencia entre el valor SPL de referencia de 94 dB (1 Pa) y el valor SPL de la entrada acústica máxima. Suponiendo un SPL máximo de 120 dB la sensibilidad sería: $94 \text{ dB} - 120 \text{ dB} = -26 \text{ dB FS}$. Sin embargo, la mayor parte de las veces esto no es así puesto que es común que el punto acústico máximo esté a un valor de palabra digital inferior al máximo por lo que generalmente se presenta la sensibilidad como el porcentaje de FS que es generado por la entrada de 94 dB-SPL.

$$Sensibilidad(dB FS) = 20 \cdot \log_{10} \left(\frac{Sensibilidad(\% FS)}{DRef(FS)} \right)$$

Donde Dref sería el valor porcentual del fondo de escala, el 100%.

Nota: Una confusión muy frecuente es el hecho de que los valores de FS de salida de un micrófono digital son valores **pico** y no **eficaces**. Esta diferencia es importante tenerla en cuenta de cara a un futuro procesamiento digital que requiera niveles de señal precisos. Por ejemplo, un procesador de señal de rango dinámico como un compresor establece los umbrales en valores eficaces por lo que habría que hacer la conversión de nuestro valor de pico al valor eficaz (el cual varía dependiendo de la forma de la señal). Para convertir valor de pico a valor eficaz de una señal senoidal:

$$V_{ef} = \frac{FS}{\sqrt{2}}$$

La expresión anterior supondría que el valor RMS está 3dB por debajo del valor de pico. Si tenemos micrófono con una sensibilidad de -26 dB FS, para una entrada de 94 dB SPL tendremos una señal de salida de -26dB FS de pico o -29 dB FS eficaces.

- Relación señal/ruido (SNR):

Conocido por las siglas anglosajonas SNR ("Signal to Noise Ratio") se define como la relación existente entre la potencia de la señal de sonido de referencia (de nuevo la de 94 dB SPL de 1KHz) con la potencia de la señal de ruido superpuesta en la salida del micrófono. El valor de SNR se expresa normalmente aplicando la ponderación "A" entre los 20-20KHz. Para comparar micrófonos es importante que ambos valoren sigan la misma ponderación aplicados también a la misma banda.

$$SNR(dB) = 10 \cdot \log_{10} \left(\frac{Potencia_{Señal}}{Potencia_{Ruido}} \right) \quad [4]$$

- Distorsión armónica total (THD):

La Distorsión Armónica Total (THD por las siglas "Total Harmonic Distortion") es una medida porcentual del nivel de distorsión en la señal de salida para una señal de tono puro (ej. senoidal o cuadrada pura) dada. Este porcentaje es la relación entre la suma de las potencias de todas los armónicos y la potencia de la frecuencia fundamental. Cuanto mayor sea el valor de THD más presencia tendrán los armónicos y la calidad de sonido será peor. El valor de THD se calcula tomando los primeros cinco armónicos de una señal senoidal de 105 dB SPL:

$$THD(\%) = \frac{\sum_{n=1}^5 P_n}{P_0}$$

Es posible que algunos fabricantes usen otro nivel de presión sonora, pero como regla general el THD se triplica con cada aumento en 10 dB de SPL.

- Punto de sobrecarga acústico (AOP):

La AOP (“Acoustic Overload Point”) se define como el nivel de presión sonora que produce una THD del 10%.

- Nivel de ruido equivalente (EIN):

El parámetro EIN (“Equivalent Input Noise”) es el nivel de ruido por defecto a la salida del micrófono expresado en dB SPL. Niveles de presión sonora de entrada inferiores al nivel de EIN estarán por debajo del umbral de ruido del micrófono y fuera del rango dinámico de las señales para las cuales el micrófono produce una salida. El EIN se puede derivar tanto del SNR como del rango dinámico:

EIN = AOP – Rango dinámico

EIN = 94 dB – SNR

- Rango dinámico (“dynamic range”):

El rango dinámico de un micrófono mide la diferencia entre los niveles más altos y bajos de SPL a los que el micrófono responde de manera lineal. A diferencia de los convertidores de audio (ADC o DAC), el rango dinámico de un micrófono no es equivalente a su relación señal-ruido. La SNR del micrófono mide la diferencia entre el nivel de ruido y una referencia de 94 dB SPL, pero el micrófono sigue respondiendo de manera útil por encima de este nivel. El rango dinámico de un micrófono MEMS es la diferencia entre su punto de sobrecarga acústica (AOP) y su ruido equivalente de entrada (EIN). Por ejemplo, el INMP441, tiene un OAP de 120 dB SPL y EIN de 33 dB SPL, por lo tanto, tiene un rango dinámico de 87 dB.

En la figura de abajo se muestra visualmente las diferencias entre los parámetros que relacionan la dB SPL de entrada con la dB FS de salida en un micrófono digital:

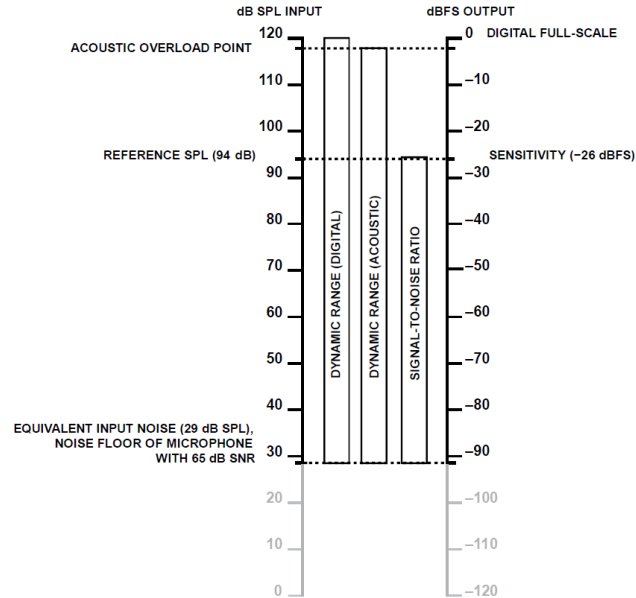


Figura 2. Comparativa de indicadores en un micrófono digital. Extraído de "Application Note AN-1112", TDK Invensense.

- PSR y PSRR:

Existe algo de confusión y caos respecto a estos dos parámetros. El factor de rechazo de la fuente o PSRR ("Power Supply Rejection Ratio") es un indicador que se utiliza de forma similar (aunque no igual) en amplificadores operacionales y en reguladores LDO ("Low DropOut voltage"). En pocas palabras, el PSRR se cuantifica, de forma decibélica, en la manera en la que el componente se perturba a su salida por culpa de una señal de ruido de valor conocido introducido en la alimentación:

$$PSRR(dB) = 20 \log_{10}\left(\frac{V_{in}}{V_{out}} \cdot A_v\right)$$

Cuanto mayor sea el PSRR más estable será la salida del dispositivo. Sin embargo, en el mundo de la electrónica acústica, aunque la base sigue siendo medir la manera en la que se rechaza el ruido de alimentación, algunos fabricantes establecen criterios para medirlos que poco tienen que ver:

PSR: Para calcularlo se superpone a propósito una señal de 100mVp-p y 217 Hz en el pin de alimentación y se mide el valor de salida del micrófono. La prueba se realiza en una cámara anecoica en ausencia de cualquier sonido. Se aplica el factor de ponderación A. El valor de PSR será siempre negativo y en caso de que el micrófono tuviera un factor de

rechazo total, el valor de PSR sería el nivel de ruido EIN (una vez ponderado). En micrófonos analógicos vendrá dado en dBV y en digitales en dB FS.

PSRR: La señal introducida es una cuadrada de 100 o 200 mVp-p cuya frecuencia puede ser todo un rango o muchos otros casos una señal de 1KHz dependiendo del fabricante. Su valor será también negativo y se expresa en dB o dBV/ dBFS.

Es común no encontrar consenso entre los fabricantes en cuanto a unidades a usar.

- Direccionalidad:

Este parámetro indica el patrón de dirección espacial que determina la mejor recepción acústica del micrófono. Existen 3 tipos:

- a) Unidireccional: Capta el sonido principalmente desde una sola dirección, siendo más sensible en esa dirección y mucho menos en otras. Útil para situaciones en las que se quiere aislar una fuente de sonido específica y minimizar la captación de ruido ambiental.
- b) Bidireccional: Es eficaz para grabaciones de entrevistas cara a cara o situaciones en las que se desea capturar el sonido desde dos fuentes ubicadas en lados opuestos del micrófono.
- c) Omnidireccional: Capta el sonido de manera uniforme desde todas las direcciones.

Este dato se presenta normalmente con gráficos de patrones polares similar a cuando se indica la direccionalidad de emisión/recepción de una antena de RF.

Ver referencias: [5]–[9]

2.1.2 Micrófonos. Clases y características.

-Micrófono de electreto: También conocido como ECM (“Electret Condenser Microphone”) es un tipo de micrófono muy común y ampliamente utilizado por la industria desde hace décadas. Tal como se muestra en la figura 3, consiste en una estructura de dos placas o electrodos separadas a una cierta distancia, como un condensador, salvo que una de las placas se le añade una película de polímero para insertar una carga fija determinada que permanecerá inmutable (en la práctica se pierde carga con el tiempo). El material de electreto suele ser el propio diafragma, aunque varía dependiendo del modelo, pero el principio es el mismo, el movimiento del diafragma por las ondas sonoras produce una variación en la capacidad y como la carga es fija provocará que la tensión de salida varíe. La placa de electreto se encuentra internamente conectada a la puerta de un JFET el cual servirá de buffer y preamplificador. Requiere de componentes pasivos adicionales: la resistencia de drenador y un condensador para desacoplar la tensión

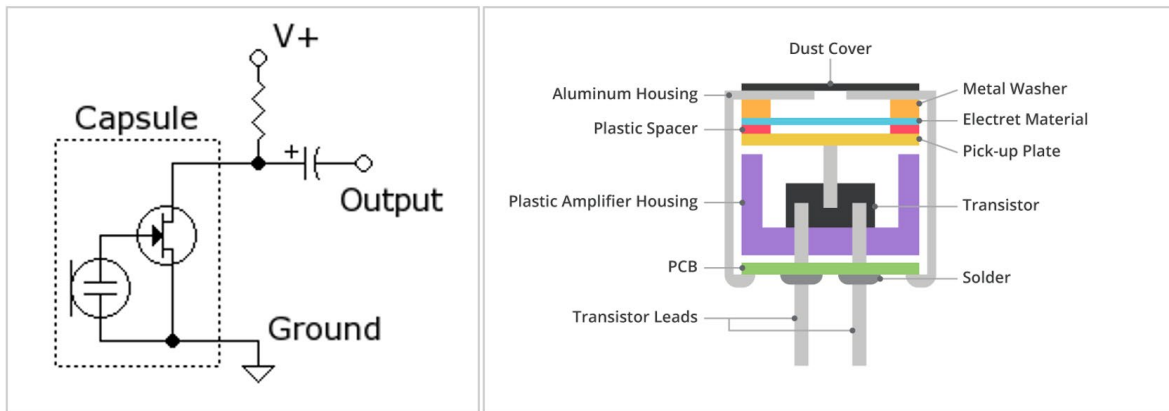


Figura 3. Esquema electrónico y construcción física de un ECM. [10]

continua de polarización del transistor FET. Dado que la tensión de salida del FET es muy baja (del orden de mV) es necesaria una posterior etapa amplificadora, normalmente basada en un amplificador operacional siguiendo una topología inversora (ya que se reduce el problema del modo común frente al montaje no inversor).

Destaca por su bajo precio, tener un ángulo de captación omnidireccional y tener una buena sensibilidad. Presentan el inconveniente de tener una respuesta en frecuencia limitada (alrededor de los 14-16 KHz de banda) y un mayor tamaño que los micrófonos MEMS. Como consecuencia de esto los ECM introducen mayor ruido eléctrico y son bastante sensibles al polvo.

-Micrófono MEMS ("Microelectromechanical System"):

La tecnología de sistemas microelectromecánicos (MEMS) ha permitido el desarrollo de micrófonos con características como tamaño reducido, bajo consumo de energía, costos reducidos, alta calidad de señal y buena estabilidad frente a variaciones de temperatura y humedad. Los micrófonos MEMS son dispositivos compuestos por un transductor acústico conectado a un circuito integrado de aplicación específica (ASIC). Generalmente, el ancho de banda de audio tiene un rango de frecuencia de 20 Hz a 20 kHz, aunque algunos pueden llegar incluso más allá de los 30KHz. Al igual que con los micrófonos de condensador electret, la presión de la onda acústica interactúa con el diafragma del micrófono, pero en el caso de los MEMS estas deflexiones se pueden medir mediante diferentes principios de transducción: piezoeléctrico, piezorresistivo, óptico, electromagnético y electrostático (capacitivo). Los micrófonos MEMS con detección capacitiva pueden utilizar una o dos placas traseras rígidas y un diafragma que oscila debido a la presión incidente. Comúnmente, en el encapsulado del micrófono, existe un orificio por el cual fluye el aire y permite las deflexiones del diafragma. Si este orificio se

encuentra en la parte superior, diremos que el micrófono es de puerto superior (“top ported”) y si está en la parte inferior es de puerto inferior (“bottom ported”). La posición del orificio influye en el desempeño del micrófono. Si es “bottom ported” la cámara de diafragma podrá ser de mayor tamaño, tendrá mejor SNR y una respuesta en frecuencia más plana, sin embargo, montar y soldar componentes en la capa inferior es más complicado y aumenta los costes de producción.

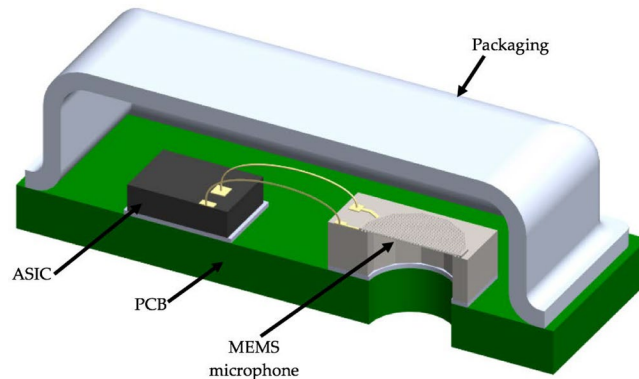


Figura 4. Estructura de un micrófono MEMS. [11]

Entre los micrófonos MEMS podemos distinguir 2 tipos:

- **Analógicos:** Estos micrófonos constan del elemento transductor, un convertidor de impedancia y una etapa amplificadora de salida. Pueden conectarse directamente a un ADC, a un “codec” de audio o a una etapa de mayor amplificación basada en un amplificador operacional como en el micrófono ECM. Sin embargo, en contraste con este último, los MEMS analógicos tienen mejores prestaciones en absolutamente todos los parámetros (especialmente en el SNR y THD), incluso hasta el precio.
- **Digitales:** Son algo más caros que los analógicos, pero lo compensan notablemente con la ventaja de que no necesitan de una posterior etapa amplificadora ni de un convertidor analógico-digital externo. En su interior la señal del transductor se amplifica y se introduce en la entrada de un ADC modulador sigma-delta(Σ - Δ), con una resolución de 16-24 bits generalmente. Este tipo de convertidor utiliza la modulación PDM (“Pulse Density Modulation”) de 1 bit. Este tipo de modulación consiste en muestrear a muy alta frecuencia (típicamente 3MHz). En función del valor de la señal analógica de entrada la densidad de pulsos de la señal portadora será mayor o menor.

Ventajas: Mucho más resiliente al ruido eléctrico respecto a los micrófonos analógicos. Además, dado que los datos se transmiten con una densidad de un número muy alto de pulsos, también tiene una gran tolerancia a los errores de bits cuando hay degradación de la señal. Otra ventaja es que sólo se necesitan 2 hilos para la interfaz de comunicación para un formato de audio tanto mono como estéreo, aprovechando los flancos de subida y bajada de la señal de reloj. Para indicar que ha finalizado el envío de datos y dejar paso a

la otra línea la salida se pone en estado de alta impedancia(Z).

Inconvenientes: La frecuencia en el muestreo e interfaz PDM es muy alta, por lo que es sensible en comunicaciones a larga distancias debido a la degradación que producen los efectos parásitos capacitivos inherentes a las pistas. Necesita de un DPS o un microcontrolador con el codec adecuado con una etapa de diezmado o submuestreo. Este proceso consiste en reducir la tasa de muestreo y que pueda ser utilizado por otros dispositivos.

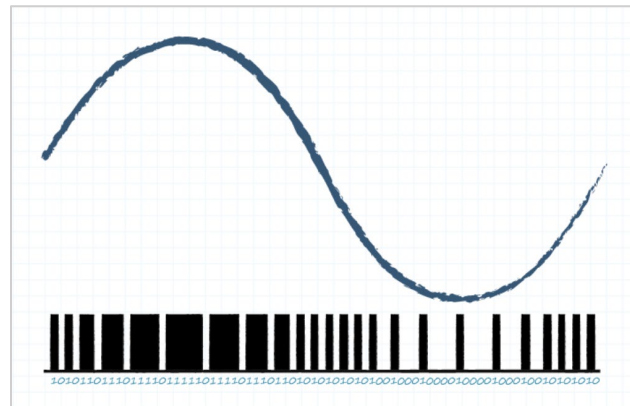


Figura 5. Modulación PDM de una señal senoidal [12]

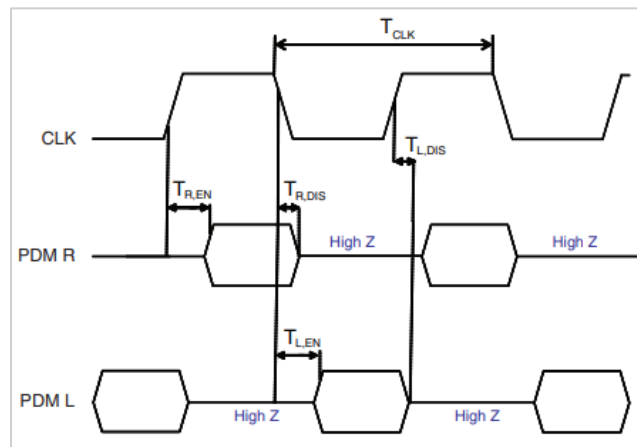


Figura 6. Interfaz de comunicación PDM en un formato estéreo. [13]

Otra modalidad de micrófonos MEMS son aquellos que tienen interfaz de comunicación I2S. La figura 7 muestra el diagrama de bloques de este tipo de MEMS. Tras la amplificación y modulación Σ - Δ se integra de forma adicional dentro del dispositivo la etapa de diezmado de la señal(amarillo), un filtro paso bajo anti-aliasing (azul), el control de tri-estado y el puerto de comunicación i2s.

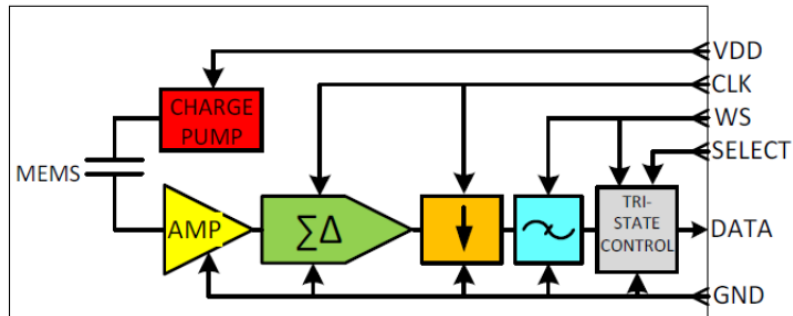


Figura 7. Diagrama de bloques de MEMS con interfaz I2S. "SPH0645LM4H-1 datasheet"

La ventaja de que incorpore I2S es que no se requiere un DPS o codec de audio externo. En el siguiente apartado se abordará con un poco más de profundidad en protocolo I2S.

Conclusión:

Para un dispositivo basado en la captación de audio que aspire a tener buena calidad, la elección de un MEMS es casi obligatoria.

Como extensión al análisis anterior, se puede encontrar en la plataforma de "Youtube" experimentos realizados para determinar la diferencia de calidad con sonido real. Se realiza una comparación de la calidad de sonido entre:

- El módulo MAX9814: Está basado en un micrófono electret y una posterior etapa de amplificación con la interesante funcionalidad de control automático de ganancia (AGC)
- Los micrófonos MEMS INMP441 y SPH0645.

Los resultados son bastante claros. Incluso hasta con el AGC, la calidad de audio del micrófono de electret es bastante mediocre en comparación con la de los dos micrófonos MEMS. Para mejorar las condiciones en la adquisición de sonido del MAX9814 podría utilizarse el puerto I2S0 del ESP32 para comunicarse internamente con el ADC. Esta funcionalidad sólo existe para el puerto 0 de la serie ESP32.

El enlace al video es el siguiente:

<https://youtu.be/3g7l5bm7fZ8?t=277>

2.1.3 Protocolo I2S

Es un método de comunicación serial síncrono desarrollado por Phillips (ahora NXP) en 1980 en respuesta al avance de la electrónica de consumo de la época. Aunque las siglas puedan tener cierta semejanza con el protocolo I2C su funcionamiento es completamente distinto e incluso se podría decir que tiene más semejanza con la comunicación SPI.

Un bus I2S consta de las siguientes líneas:

-Señal de reloj SCK o BCLK (“Bit Clock”): Es la línea que sincroniza la transmisión de datos desde el transmisor al receptor. También sirve para sincronizar la línea de selección de palabra (WS) con la de datos. Cada ciclo BCLK está asociado a la transferencia de un 1 bit de datos. Dependiendo del modo de comunicación I2S, la señal de datos SD puede estar desfasada un ciclo de reloj respecto a WS.

-Línea de selección WS (“Word Select”): Llamado también LRCLK (“Left Right Clock”) se utiliza para seleccionar el canal que se va a transmitir. Un nivel lógico alto indica que los datos que se van a transferir corresponden al canal derecho y un nivel bajo indica la transmisión del canal izquierdo.

-Línea de datos SD (“Serial Data”): Es la línea por donde se transmiten los datos. El transmisor y receptor no necesitan acordar el tamaño de palabra ya que en I2S el ancho del canal está predeterminado y es tarea del receptor gestionar los datos recibidos en la forma que ha sido configurado. El primer bit enviado es el más significativo (MSB) de la secuencia de datos. Por norma general el dato se cambia durante el flanco de bajada BCLK y se registra en los flancos de subida.

Formato y modos de la comunicación I2S:

-Formato de canal: Mono o estéreo. Cuando un canal termina de transmitir los datos pone su salida en estado de alta impedancia hasta que termine la transmisión del canal (“slot”). De igual forma si el formato es mono todos los bits serán de alta-Z durante la transmisión del otro canal (*ver nota final*).

-Formatos de comunicación estándar:

- Phillips: La señal de datos está desfasada un ciclo de reloj respecto a WS. El primer bit de dato es el MSB. Es el más usado por los micrófonos MEMS.
- Formato MSB: Igual que el anterior salvo que no existe desfase entre SD y WS.
- Formato PCM short: Este formato tiene desfase en 1 bit de la línea SD y el primer bit de datos es el más significativo, pero ahora la señal WS es un pulso que dura 1 ciclo SCK.

La siguiente figura extraída de la hoja de datos del micrófono INMP441 muestra un ejemplo de una trama I2S para un formato de canal estéreo y un formato de comunicación Phillips estándar:

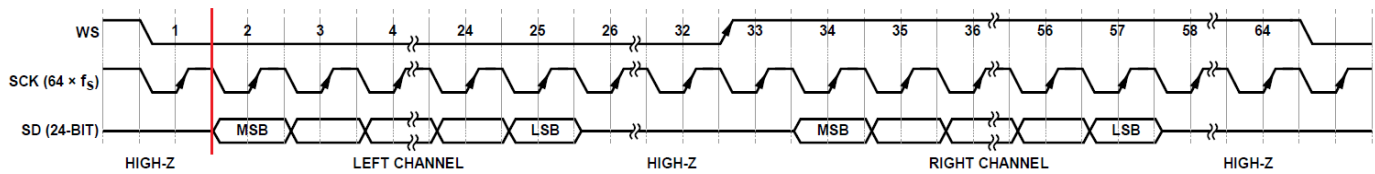


Figura 8. Trama I2S en modo estándar. "INMP441 datasheet"

Nota importante:

En una trama I2S, **independientemente del número de bits de datos a enviar o del formato de canal**, por cada ciclo WS debe haber **64 ciclos** de reloj BCLK. El número de bits destinados para cada canal se denomina "slot width" o ancho de canal y es algo que de forma general está predeterminado a 32 bits. Por ejemplo, para una aplicación en la que se quiera guardar en un buffer los datos de audio en formato mono con una resolución de datos de 16 bits la manera en adquirir los datos deberá ser la misma que para una aplicación en estéreo y 32 bits de datos. Lo único que cambiará será que, en el primer caso, del primer "slot" se descartará los últimos 16 bits y se ignorarán todos los bits del segundo canal.

Esto es una fuente común de confusiones tanto por parte de programadores como incluso de los propios desarrolladores de librerías para la interfaz I2S. En el anexo se verá este problema con mejor detalle.

2.1.4 Material utilizado para el proyecto

El componente SMD de la empresa "TDK Invensense" utilizado para el desarrollo del proyecto, ha sido el micrófono INMP441. Se trata de un micrófono MEMS (bottom ported) digital omnidireccional de bajo consumo que cuenta con interfaz I2S. Sus características principales son las siguientes:

- Resolución de 24 bits de datos.
- Valor de SNR de 61 dBA.
- Nivel de ruido estático EIN de 33 dBA SPL
- Buen valor de sensibilidad de -26 dB FS.
- Respuesta en frecuencia plana desde los 60 Hz hasta los 15 kHz.
- Consumo de corriente de 2,4 mA en modo normal.

- PSR de -75 dBFS. No se aporta el dato de PSRR
- Tamaño reducido $4.72 \times 3.76 \times 1$ mm.

Aunque es un micrófono bastante bueno en relación calidad-precio, actualmente no se suele utilizar para nuevos diseños ya que ha sido superado en todas sus características por los nuevos modelos que han llegado al mercado. Esto se refleja en la tabla 3 del apartado “[MEMS](#)” de la sección de Diseño electrónico en la comparativa con otros modelos.

Para la experiencia se ha utilizado el módulo de la empresa “ARCELI” disponible en Amazon [14]. El módulo basado en el INMP441 incluye un condensador de bypass, la resistencia para descargar la capacidad parásita de la línea de datos y los cabezales para poder conectarse a una placa de prototipado o con cables “jumpers” hembra.



Figura 9. Vista de la capa inferior del módulo INMP441. Imagen extraída del producto en Amazon

2.2 Unidad de control de la ESAI

2.2.1 Placa y módulo

Para el desarrollo del proyecto se han utilizado las siguientes placas de desarrollo:

- Placa de desarrollo ESP32 DOIT DEVKIT V1 adquirido en Aliexpress [15]
- Placa de desarrollo ESP32 ESP-WROOM-32 NodeMCU de AZDelivery adquirido en Amazon [16]

Salvo por la diferencia en algunos pines de salida y calidad del diseño, ambas placas de desarrollo están basadas en el módulo ESP32-WROOM-32. Utilizan el integrado CP2102N como puente USB-UART, el famoso AMS1117 como regulador LDO y tienen un conector USB MICRO-B. A continuación, se realiza una descripción de la placa ESP32 DOIT DEVKIT V1:

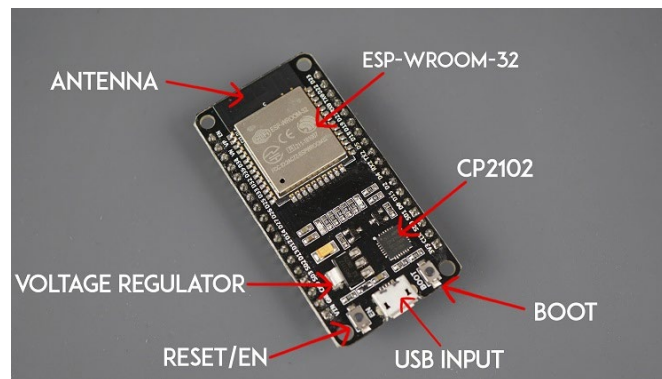


Figura 10. Vista de la placa ESP32 DOIT DEVKIT V1. [17]

-Botones “Boot” y “EN”: Se utilizan para hacer entrar en modo arranque (“bootmode”) o resetear el microcontrolador.

-CP2102: El chip dentro del módulo ESP32 carece de transceptor USB. Este componente permite la conexión del dispositivo con la computadora para cargar el firmware e intercambiar información a través de la consola serial.

-AMS1117: Es un popular regulador de voltaje lineal utilizado en la mayoría de las placas de desarrollo de Arduino y similares para regular la tensión de entrada (de hasta 15V) para entregar a la salida 3.3V o 5V dependiendo del modelo exacto. En la sección de Diseño electrónico se encuentra más información al respecto.

-Finalmente tenemos el módulo ESP32-WROOM-32:

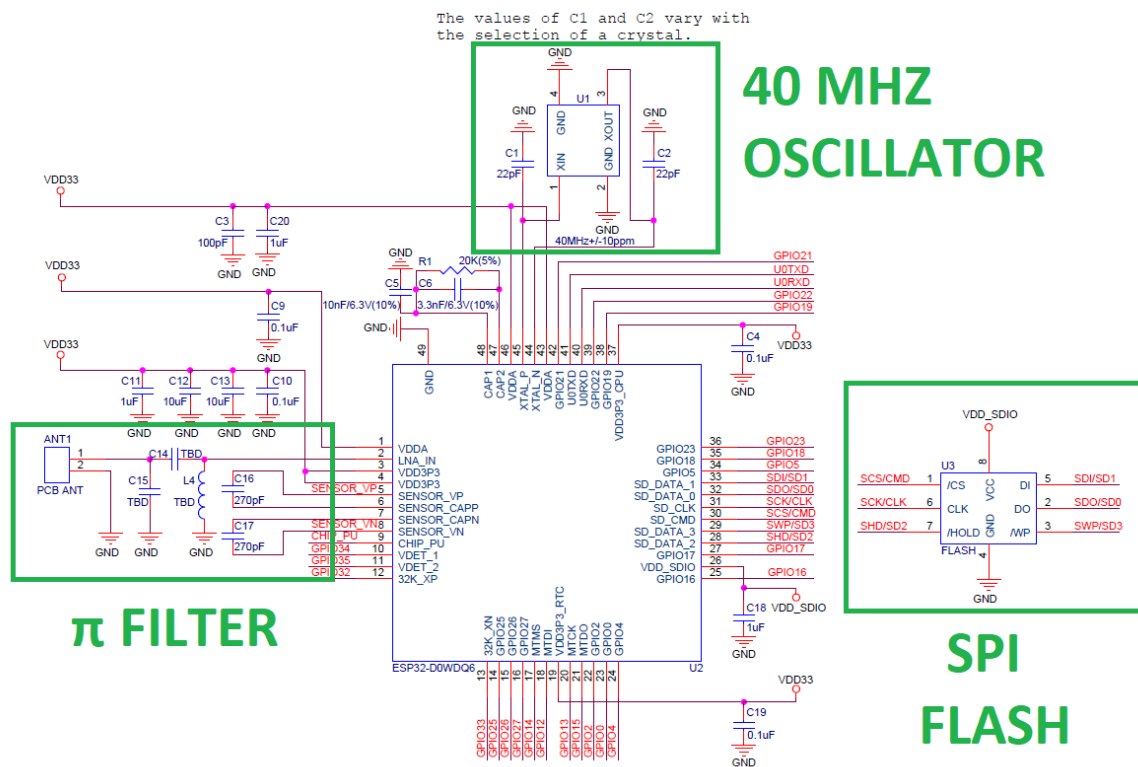


Figura 11. Diagrama esquemático del módulo WROOM-32, "ESP32 datasheet"

Características integradas:

- Antena PCB de 50 ohms de impedancia característica más el circuito adaptador de impedancias (filtro π formado por un arreglo CLC), conocido como "matching circuit". Este circuito se utiliza para mejorar la recepción y transmisión de la señal de RF con la antena. Ver apartado [MCU](#) en la sección de diseño electrónico para más información.
- Memoria SPI FLASH de 4MB.
- Oscilador de 40 MHz de cristal.

En el núcleo del módulo encontramos el chip ESP32-D0WDQ6. Este chip de bajo consumo está diseñado para lograr la mejor eficiencia energética y rendimiento de radiofrecuencia (RF), mostrando robustez, versatilidad y confiabilidad en una amplia variedad de aplicaciones.

2.2.2 Características del chip

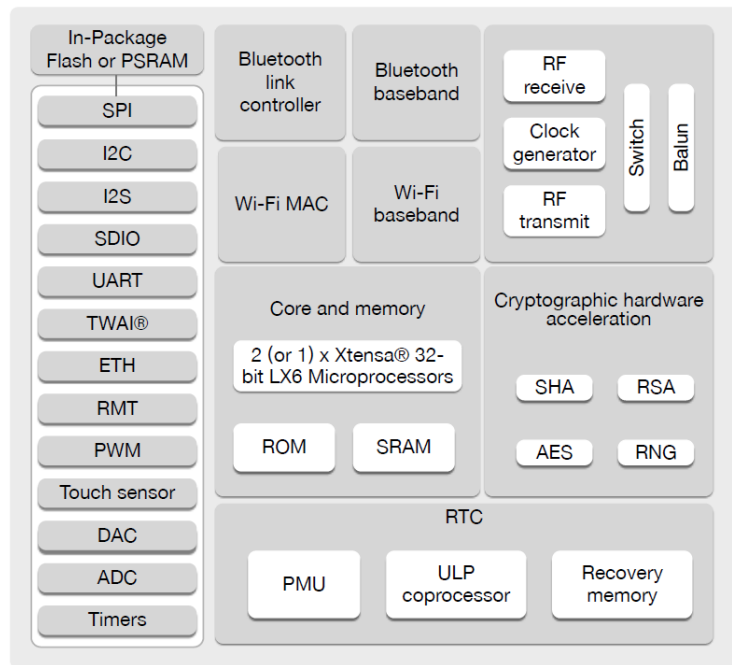


Figura 12. Diagrama funcional de bloques del chip, "ESP32 Series datasheet"

-CPU y memoria:

- Basado en el microprocesador Xtensa LX6 de 32 bits y arquitectura de doble núcleo de la empresa Tensilica. La frecuencia de reloj de la CPU puede ajustarse desde los 80MHz hasta 240 MHz.
- 448 KB de ROM, 520KB de SRAM ("Static RAM"), 16KB SRAM para RTC

-Módulo RF:

- WiFi 2.4 GHz, 802.11 b/g/n (802.11n hasta los 150 Mbps)
- Bluetooth v4.2 BR/EDR y Bluetooth LE
- Puede operar como estación o punto de acceso("SoftAP")

-Gestión de energía:

- Dispone de un coprocesador de ultra bajo consumo (ULP) que puede operar cuando la CPU central está desactivada. Sólo puede ser programado en lenguaje ensamblador.

- Distintos modos para el ahorro de consumo energético: Activo, “modem-sleep”, “light-sleep”, “deep-sleep”, hibernación.

-Seguridad:

- Arranque seguro (“secure boot”) y encriptación de Flash.
- Módulo coprocesador de aceleración criptográfica. Soporta los siguientes algoritmos: Hash (SHA-2), AES, RSA, ECC, RNG.

-Periféricos destacados:

- 34 GPIOs programables (6 utilizados por la SPI FLASH).
- SAR ADC de 12 bits con hasta 18 canales.
- 2 x 8 bits DAC.
- SPI (4), I2S (2), I2C (2), UART (3).
- Interfaz de tarjeta SD.

-Consumo:

Work Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	—	240	—	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	—	190	—	mA
Transmit 802.11n, OFDM MCS7, POUT = +14 dBm	—	180	—	mA
Receive 802.11b/g/n	—	95 ~ 100	—	mA
Transmit BT/BLE, POUT = 0 dBm	—	130	—	mA
Receive BT/BLE	—	95 ~ 100	—	mA

Power mode	Description			Power Consumption
Active (RF working)	Wi-Fi Tx packet			Please refer to Table 4-4 for details.
	Wi-Fi/BT Tx packet			
	Wi-Fi/BT Rx and listening			
Modem-sleep	The CPU is powered up.	240 MHz *	Dual-core chip(s)	30 mA ~ 68 mA
			Single-core chip(s)	N/A
		160 MHz *	Dual-core chip(s)	27 mA ~ 44 mA
			Single-core chip(s)	27 mA ~ 34 mA
		Normal speed: 80 MHz	Dual-core chip(s)	20 mA ~ 31 mA
Single-core chip(s)	20 mA ~ 25 mA			
Light-sleep	-			0.8 mA
Deep-sleep	The ULP coprocessor is powered up.			150 μ A
	ULP sensor-monitored pattern			100 μ A @1% duty
	RTC timer + RTC memory			10 μ A
Hibernation	RTC timer only			5 μ A
Power off	CHIP_PU is set to low level, the chip is powered down.			1 μ A

Figura 13. Consumo del chip para los distintos modos. "ESP32 Series datasheet"

Los valores de consumo con el módulo de radiofrecuencia activo (tabla azul) son valores pico, no valores medio.

Para nuestra aplicación el ESP32 estará la mayor parte del tiempo capturando el sonido y almacenándolo y entre medio se producirán los envíos que, dependiendo del tamaño del paquete de audio y el protocolo de comunicación usado serán más o menos tiempo, pero en cualquier caso siempre será considerablemente más pequeño que el tiempo en el que esté recibiendo solo. Por lo tanto, es de esperar un consumo ligeramente superior a los 100 mA por el consumo del periférico I2S principalmente. Para obtener resultados más precisos, habría que realizar pruebas de consumo con herramientas específicas. Se tienen las siguientes alternativas:

- *Mejor opción*: Usar un módulo analizador de potencia de rango dinámico, con resistencias shunt bien adaptadas y que además se pueda conectar a un ordenador y mediante software se obtienen representaciones en gráficas con herramientas de análisis y cálculo. Ej. El módulo “Oti” y el “Power Profiler kit II”.

- Joulescopio.

- Osciloscopio usando sondas especializadas de tipo pinzas para medir corrientes intermedias del orden de 10-800 mA.

- *Opción utilizada*: Utilizar un multímetro para obtener una estimación del valor medio de corriente consumida. Es una opción humilde, pero sirve para obtener una primera aproximación con la que trabajar. Los resultados se midieron para una frecuencia de reloj de 160 MHz y 240MHz.

Resultados de la prueba:

- Condiciones: Modo de operación habitual usando todos los recursos, capturando sonido, recibiendo y transmitiendo a través del módulo RF.

- Multímetro utilizado: Aneng DM850.

- Consumo medio para 160 MHz: 115-125 mA

- Consumo medio para 240 MHz: 130-140 mA

El aumento del consumo no es tan proporcional como se podría pensar. Sin embargo, es importante destacar que al disminuir la frecuencia de reloj si aumenta de forma más proporcional el tiempo que se tarda en realizar las operaciones por lo que realmente hay que considerar si merece la pena. El objetivo es que la transmisión dure lo menos posible ya que es el punto donde habrá mayor consumo.

2.2.3 Configuración del IDE

Para el desarrollo del firmware se ha utilizado principalmente el entorno de desarrollo de Visual Studio Code con la extensión de PlatformIO. Esta potentísima herramienta se podría considerar literalmente como una “extensión” del entorno de desarrollo al que se integra para mejorar la experiencia de desarrollo. PlatformIO se encarga de descargar todas las librerías, drivers y la “toolchain” que ofrecen los fabricantes de microcontroladores para la programación de sus dispositivos. Además, permite la posibilidad de integrar múltiples “frameworks” o marcos de referencia, por ejemplo, permite integrar el framework de ESP-IDF con el de Arduino. A la hora de hacer esto hay que prestar atención a las versiones que utiliza PlatformIO de cada marco porque pueden no coincidir con la versión más actual de dicho framework. Esto se verá más adelante en la sección ya mencionada de “Incidencias con el protocolo I2S”. PlatformIO soporta actualmente miles de placas basadas tanto en microcontroladores como en microprocesadores más avanzados (ej. RaspBerry Pi). A fecha de escritura de este documento platformIO ofrece 209 placas basadas en el ESP32.

-Guía de configuración:

1. Ir a la página oficial “code.visualstudio.com” y descargar el instalador para la versión del S.O en el que se vaya a ejecutar, Windows 10 en nuestro caso.
2. Una vez instalado abrimos VS Code y accedemos a la pestaña de extensiones situada a la izquierda de la pantalla. Buscamos PlatformIO y lo instalamos.
3. Aparecerá una nueva pestaña con el logo de PlatformIO (la cara de un insecto con 2 antenas). Pinchamos y se abrirá una nueva ventana con distintas pestañas. En “Libraries” PlatformIO nos mostrará todas las librerías existentes y más actuales (cargadas directamente de sus respectivos repositorios de GitHub) disponibles para el ESP32. En “Home” podemos crear o abrir un nuevo proyecto. Pinchamos en “new project”. Escribimos el nombre de nuestro proyecto, seleccionamos la placa de desarrollo con la que estemos trabajando y el framework (Arduino en nuestro caso). Si deseccionamos la casilla del final podemos guardar el proyecto
4. Al terminar el proceso nos aparecerá en la ruta que hayamos guardado una carpeta con una serie de archivos creados de forma automática para la compilación y la carga del firmware en el microcontrolador.
5. El archivo “platformio.ini” nos permite modificar, añadir o eliminar parámetros como la velocidad de transmisión de la UART, la frecuencia por defecto de la CPU, la placa, plataforma, framework o librerías. También podemos modificar la versión que queremos que se ejecute de la librería, plataforma o marco de referencia usado.

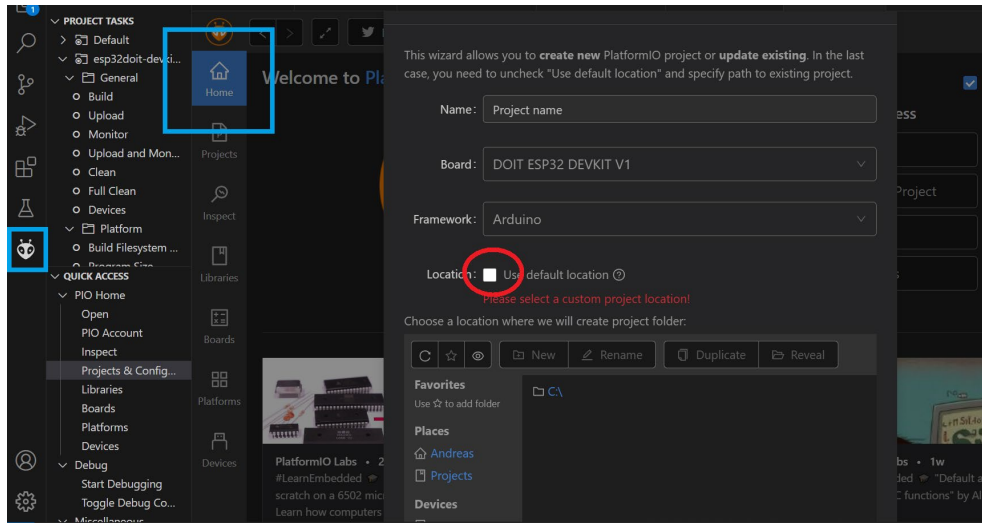


Figura 14. Vista del programa VS Code. Creada por el autor

Adicional:

Se ha utilizado también el IDE de Arduino, versión 1.8.18 principalmente para el uso del graficador serial (“Serial Plotter”).

Para descargar una versión más antigua tenemos que irnos a la web oficial de Arduino y buscar el apartado de “OldSoftwareReleases”:

<https://www.arduino.cc/en/software/OldSoftwareReleases>

Una vez instalados hay que descargar la librería de ESP32 desde el gestor de librerías.

Posteriormente vamos a preferencias y en el campo “URLs adicionales de gestor de tarjetas”, añadimos la siguiente URL:

https://espressif.github.io/arduino-esp32/package_esp32_index.json

2.2.4 Gestión de memoria

En nuestro programa la memoria será un factor a tener fuertemente en cuenta. La estación trabajará almacenando y enviando paquetes de audio y realizando otro número de tareas al mismo tiempo. Los paquetes de audio serán bastante exigentes en cuanto a memoria.

Para calcular cuánta memoria será requerida por los buffers de audio deberemos tener en cuenta la resolución de las muestras, canales utilizados (izquierdo, derecho o ambos), longitud y número buffers usados. En nuestro proyecto, uno de los buffers principales encargado de almacenar los datos transmitidos por el micrófono tendrá que almacenar 10-16 kilo-muestras, con un ancho de canal de 32 bits para una muestra de sonido en formato mono. Utilizando la nomenclatura de las etiquetas usadas por la biblioteca “driver/i2s.h” la memoria ocupada por dicho buffer en el peor de los casos sería:

$$\text{Memoria (KB)} = \text{samples} \cdot \frac{\text{slot}_{width}}{8} \cdot n^{\circ}_{channels} = 16K \cdot \frac{32}{8} = 64 \text{ KB}$$

Si tenemos en cuenta que las librerías usadas para I2S y MQTT realizan internamente copias de los buffers que introducimos como argumento y que además tenemos más buffers, variables y plantillas “html” en forma de cadenas de texto existe una gran posibilidad de obtener problemas de memoria.

Si declarásemos un vector en nuestra función principal de forma común como en nuestro caso el buffer de i2s teniendo en cuenta los parámetros anteriores:

`int32_t buffer[16000]`

Al compilar tendríamos un problema de desbordamiento de pila (**stack overflow**) en la salida del monitor serial. Esto es así porque la función loop() no deja de ser una tarea RTOS(“Real Time Operating System”) y como tal tiene asignado un trozo de memoria llamado pila(stack). Esta pila se utiliza para almacenar datos de las variables locales (además de almacenar los argumentos de la función y la dirección de retorno). La tarea loopTask() así como su tamaño de pila son predefinidas de base por el framework de Arduino a un valor de **8192 bytes**. Este tamaño de memoria resulta insuficiente para nuestra aplicación por lo que debemos recurrir a varias posibles soluciones. Pero antes hay que conocer la organización de la memoria.

El ESP32 cuenta con 520 KB de memoria SRAM interna distribuida en tres bloques

SRAM0, SRAM1 y SRAM2. Esta SRAM se usa con dos finalidades, una para la memoria de instrucciones (IRAM), utilizada para la ejecución del código y la otra para la memoria de datos (DRAM) usada para datos [18]. Se cuenta teóricamente con 328 KB de SDRAM disponible para el almacenamiento de datos repartidos entre la SRAM1 y la SRAM 2.

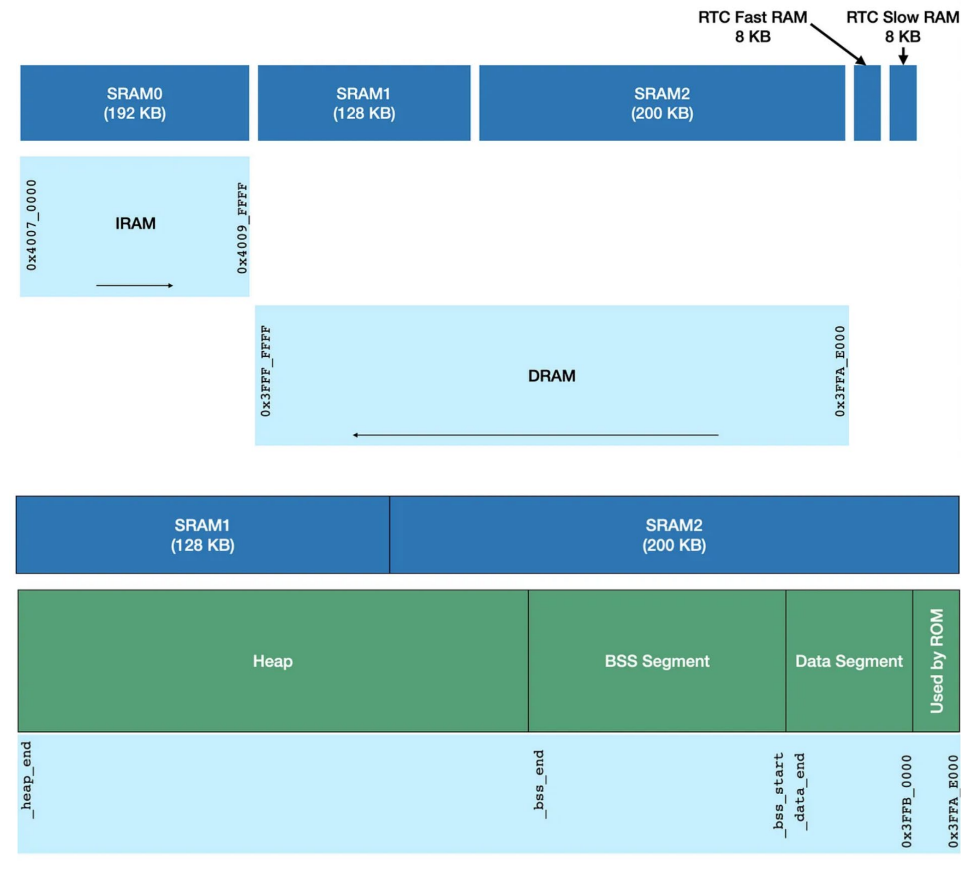


Figura 15. Organización de la SRAM y ampliación de la DRAM. [19]

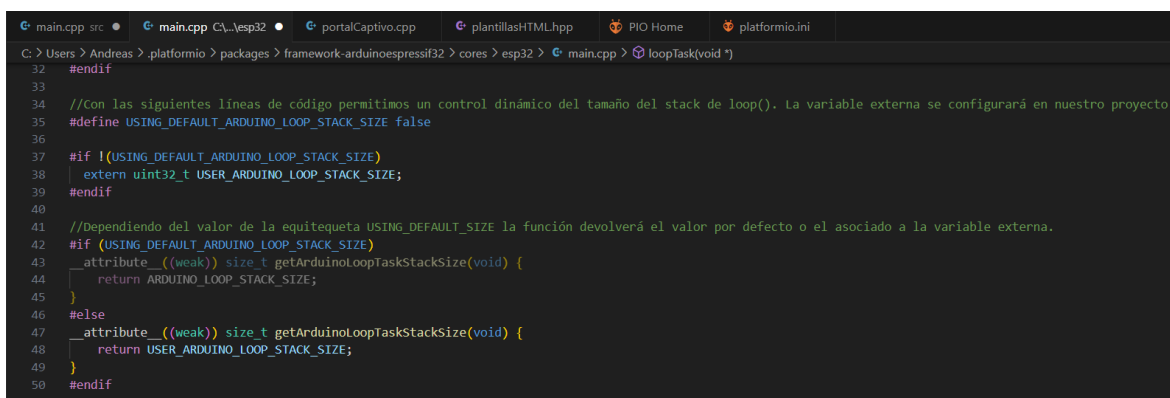
Aunque parte de la SRAM1 se puede usar como IRAM y la IRAM a su vez se puede llegar a usar como memoria de datos no lo vamos a cubrir ya que presenta inconvenientes y limitaciones que hacen que su uso no sea de interés. Por un lado, será más lenta que la memoria de la DRAM y tendrá limitaciones de acceso en términos de alineación de dirección y tamaño. Por otro lado, no es seguro de usar con multinúcleo y no sirve como memoria DMA.

Dentro de la DRAM encontramos la “call stack” o pila de llamada y el almacenamiento libre o “heap”:

-La heap a diferencia de la pila, permite reservar memoria dinámicamente, pudiendo ocupar y liberar bloques de memoria de tamaño variable en cualquier momento. Dentro de

esta memoria encontramos también las variables globales y estáticas. Para asignar memoria dinámicamente usamos las funciones `malloc()/calloc()` y `free()` para liberar bloques de memoria del almacenamiento respectivamente. Para declarar una variable como memoria estática sólo tenemos que añadir “static” antes, pero hay que tener en cuenta que ésta memoria se asignará antes de iniciar la ejecución del programa, durante la compilación, y no podremos liberar su memoria cuando no se esté utilizando.

-La stack es la memoria encargada de almacenar información sobre las subrutinas o tareas activas durante la ejecución de un programa. Al llamarse una función se crea un nuevo bloque con las variables locales la subrutina, los parámetros y la dirección de retorno. Este tipo de memoria sigue la estructura LIFO (“Last In First Out”) es decir el bloque de memoria de la última subrutina llamada será el primero en liberarse del espacio de la stack, manteniéndose así el orden correcto de ejecución del programa. Para solucionar la falta de memoria de la tarea “loopTask” podemos crear una tarea RTOS con una pila de un tamaño suficiente grande o incluso modificar la tarea “loopTask” con un tamaño de pila superior al valor por defecto. Se puede crear un tamaño de pila modificable desde cualquier nuevo proyecto añadiendo código adicional en el archivo `main.cpp` (el archivo principal completo que se encuentra en el caso de este trabajo para la asignada por platformio: `C:\Users\Usuario\.platformio\packages\framework-arduinoespressif32\cores\esp32`):



```
32 #endif
33
34 //Con las siguientes líneas de código permitimos un control dinámico del tamaño del stack de loop(). La variable externa se configurará en nuestro proyecto.
35 #define USING_DEFAULT_ARDUINO_LOOP_STACK_SIZE false
36
37 #if ! (USING_DEFAULT_ARDUINO_LOOP_STACK_SIZE)
38     extern uint32_t USER_ARDUINO_LOOP_STACK_SIZE;
39 #endif
40
41 //Dependiendo del valor de la equitequeta USING_DEFAULT_SIZE la función devolverá el valor por defecto o el asociado a la variable externa.
42 #if (USING_DEFAULT_ARDUINO_LOOP_STACK_SIZE)
43     __attribute__((weak)) size_t getArduinoLoopTaskStackSize(void) {
44         return ARDUINO_LOOP_STACK_SIZE;
45     }
46 #else
47     __attribute__((weak)) size_t getArduinoLoopTaskStackSize(void) {
48         return USER_ARDUINO_LOOP_STACK_SIZE;
49     }
50 #endif
51
```

Figura 16. Código añadido en el archivo `main.cpp` para aumentar el tamaño de la pila de la tarea `loopTask`. Creado por el autor

En cada proyecto nuevo que hagamos deberemos definir la variable “USER_CONFIG_ARDUINO_LOOP_STACK_SIZE” al valor deseado.

Esta solución puede ser útil durante el proceso de creación del firmware del dispositivo, ya que estará modularizado en numerosos fragmentos separados y de esta manera nos ahorramos la tarea de asignar memoria manualmente cada vez que vayamos a trabajar con un buffer. La solución óptima para el firmware final pasará por asignar dinámicamente la memoria de los buffers y liberadla cuando sea posible para ahorrar espacio.

Comprobar espacio de memoria real

Si revisamos la figura 15 observamos que de los 328 KB que en principio parecíamos tener como memoria para datos, parte de la memoria es usada por la ROM y por el segmento “bss” y de datos. Estos dos últimos segmentos no tienen una extensión fija, sino que dependerán del programa. Para tener una noción de la memoria total que podemos llegar a tener se ha creado un archivo de prueba en el que se han utilizado ciertas funciones para devolver el valor de la heap libre y utilizada.

Se utiliza un tamaño de pila de la función “loopTask” mínima de 1548 bytes. Para obtener los bytes y bloques disponibles/ocupados se ha hecho uso de dos funciones de distintas bibliotecas. Las funciones de la biblioteca “esp_heap_caps.h” arrojan valores superiores a los que ofrecen las funciones de la biblioteca “esp_systems.h”. La segunda librería es más limitada pues sólo muestra el espacio libre actual y el mínimo alcanzado. Los resultados son los siguientes:

-Placa: ESP32DEV

-Framework: Arduino

-Heap total (bytes): 382128 / (no aporta)

-Heap libre (bytes): 355472 / 283708

-Mayor bloque libre (bytes): 114676 / (no aporta)

-Bytes almacenados por defecto: Heap total – Heap libre = 26656

Hay que tener en cuenta que la memoria Heap total disponible será siempre superior al mayor bloque que podamos guardar. Por lo tanto, cuanto mayor sea el bloque que queremos guardar menos de estos podremos poner antes de que haya un error al asignar la memoria y se guarde un puntero nulo.

Algo a destacar es que la memoria que se muestra por la primera biblioteca es mayor que los 328 KB de la memoria DRAM. Esto puede deberse a los aproximadamente 50 KB de la memoria IRAM que puede utilizarse

Almacenamiento no volátil (“NVS”)

Nuestra estación necesita recordar los datos del punto de acceso y de los modos de operación con el que ha sido configurada. La memoria SRAM vista hasta ahora no es capaz de retener información una vez se pierda la fuente de energía. Para guardar toda esta información de vital importancia hacemos uso de la memoria de almacenamiento no volátil o NVS (“Non Volatil Storage”).

Existen dos formas de hacer uso de este tipo de memoria en el ESP32:

-Usando las “preferences”: Es un tipo de memoria pensada para guardar pares de clave-valor. Podemos acceder a esta memoria usando la librería <Preferences.h>. Por defecto 2 KB de espacio de la Flash es reservado para la memoria de preferences por lo que se debe utilizar para almacenar variables de pequeño tamaño. Será la memoria NVS que usemos en nuestro proyecto.

-SPIFFS (“Serial Pheripheral Interface Flash File System”): Esta memoria no volátil se reserva para almacenar archivos de mayor tamaño como archivos de audio o video. Puede considerarse como una tarjeta SD integrada. Se puede asignar hasta 1,5 MB para la memoria SPIFFS. Es algo más complicada de usar.

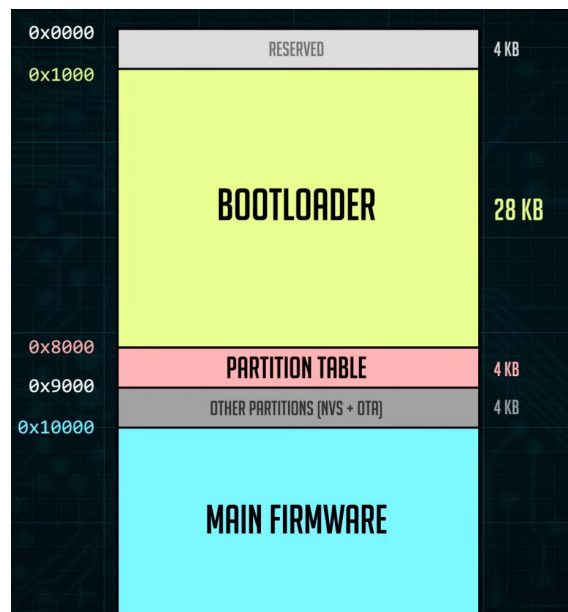


Figura 17. Partición de la memoria FLASH del ESP32. [20]

2.3 Protocolos de comunicación

2.3.1 Introducción al internet de las cosas

El avance de la tecnología de fotolitografía en las últimas décadas ha permitido la creación de chips con circuitos integrados cada vez más complejos, más pequeños y mucho más eficientes energéticamente. Lo anterior, junto a la mejora de las comunicaciones inalámbricas que permiten la transmisión de datos con un mayor ancho de banda, ha llevado a la posibilidad de integrar sistemas inteligentes y autónomos en espacios altamente reducidos.

El internet de las cosas, oficialmente llamado IoT de las siglas en inglés “Internet of Things” es un término para describir la interconectividad de dispositivos cotidianos o industriales con el Internet, desde un reloj inteligente, un automóvil o hasta incluso una bombilla. Los dispositivos, llamados “cosas” operan como nodos inteligentes, capaces de adquirir datos, procesar información localmente y comunicarse de manera inalámbrica con otros dispositivos o con los servidores de la nube. La integración en espacios pequeños y en muchos casos, alimentados sólo con batería ha llevado a que los microcontroladores tengan una presencia importante en el mundo IoT.

Los protocolos de red más antiguos se desarrollaron con una perspectiva distinta a la actual, que era la de servir la comunicación entre una computadora y un servidor conectados a la red informática mundial o WWW (“World Wide Web”). Por ende, estos protocolos no fueron diseñados con un enfoque en la eficiencia, lo que los hace relativamente pesados y lentos. Esta situación común de ausencia de recursos disponibles es lo que ha llevado a lo largo de los últimos años a la creación y mejora de nuevos protocolos de comunicación más ligeros y eficientes, tanto de consumo energético como de memoria.

Actualmente existe una amplia variedad de protocolos usados: HTTP, MQTT, CoAP, LoRAWAN, XMPP, DDS, AMQP, LwM2M ...etc. Cada protocolo tiene sus ventajas y características propias. En nuestro proyecto se ha seleccionado el protocolo MQTT y se comparará con el protocolo HTTPS. El objetivo principal será determinar cuál de los dos es capaz de transmitir los paquetes de audio de forma segura en el menor tiempo posible.

Uno de los parámetros que más debemos cuidar es el de la **latencia**. En electrónica digital se denomina latencia al tiempo que transcurre desde que la señal entra en el circuito hasta que se obtiene la salida tras el procesamiento. Entre medio existen retardos debido a la propagación de las señales, la conmutación de los transistores MOSFET, el tiempo para cambiar las salidas de las puertas lógicas (y por ende de biestables) etc. En caso de que el sistema realice tareas de cómputo complejas, es decir, que implique un procesador que ejecute una serie de operaciones a medida que lee sus instrucciones, los tiempos de latencia aumentan significativamente. En nuestro caso el mayor factor de latencia lo introduce el modelo de Red Neuronal Yamnet al imponer un tamaño de archivo

de audio de al menos 964ms (aunque posteriormente se verá que puede funcionar con menos) para que la respuesta sea coherente. A lo anterior hay que sumarle el tiempo de cómputo de realizar el espectrograma, codificación del archivo de audio para adaptarlo como entrada de la CNN, realizar una masiva cantidad de operaciones a través de todas las capas convolucionales de la red utilizando de por medio funciones no lineales como la ReLU hasta obtener la inferencia. Esa inferencia a su vez es “leída” por el servidor y finalmente se envía de vuelta al ESP32 a través de un protocolo de comunicación de red, el cual tendrá su propia latencia. Es ahí donde nos vamos a centrar, en lograr disminuir al máximo la latencia en los envíos de datos.

Antes de entrar en materia, es importante tener una noción básica de la estructura de las pilas de protocolos. Esto no es más que la definición de una serie de niveles de abstracción para definir y organizar el funcionamiento y ámbito de operación de los protocolos de red y los procesos que lo hacen posible. Los niveles más alto de la capa estarán más alejados del mundo físico subyacente y se referirán a la aplicación o producto final. Las capas inferiores tendrán un menor nivel de abstracción hasta llegar a las especificaciones eléctricas y mecánicas, para activar, mantener o desactivar el enlace físico entre sistemas finales. El modelo más utilizado es el modelo de capas OSI, inventado a mediados de los 80 por la Organización Internacional de Estandarización (ISO) en respuesta a la necesidad de integrar la ingente cantidad de distintos protocolos de red que estaban surgiendo con sus reglas propias según el fabricante. A continuación, se muestra un esquema del modelo OSI con ejemplos para cada capa una:

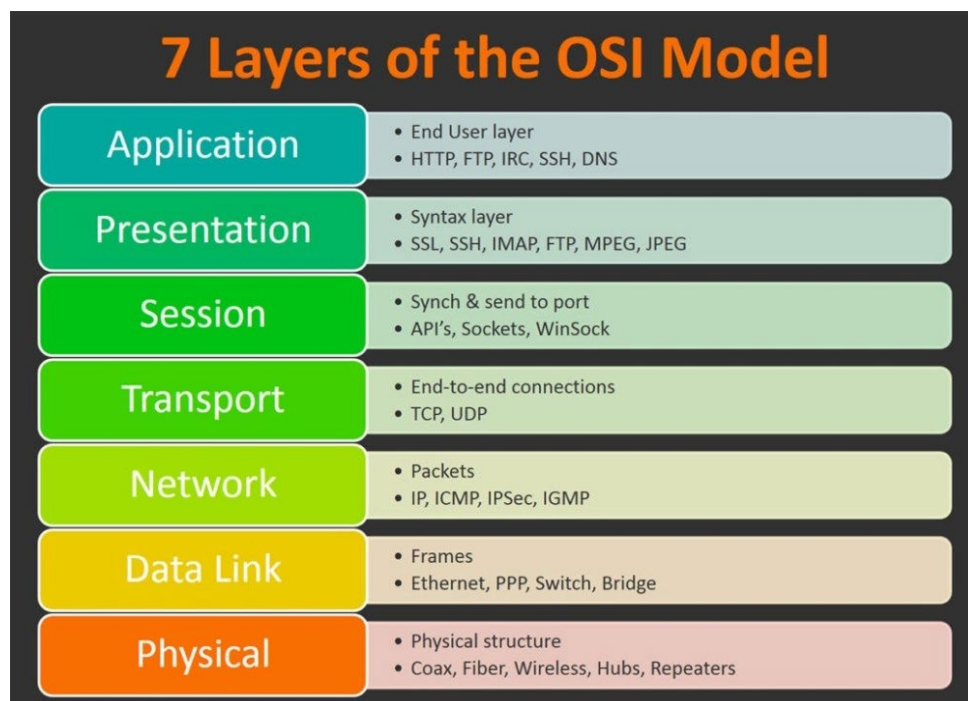


Figura 18. Capas del modelo OSI con ejemplos. [21]

2.3.2 Protocolo HTTP

El Protocolo de Transferencia de Hipertexto (HTTP, por sus siglas en inglés) es posiblemente el protocolo más famoso que se utiliza para la transmisión de información en la WWW. Es un protocolo de la capa de aplicación que se utiliza para la transferencia de datos en forma de texto, sonido, video y otros archivos multimedia. Utiliza el protocolo TCP e IP en la capa de transporte y red respectivamente. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. El cliente realiza una petición enviando un mensaje con cierto formato al servidor y el servidor (típicamente llamado servidor web) le envía un mensaje de respuesta (asociado a un código de estado). La estructura de una petición HTTP es la siguiente:

--Línea de petición ("request"): Esta primera línea especifica el tipo de petición y la versión HTTP a usar. Los métodos más comunes son GET y POST para realizar la solicitud a la obtención de un recurso o para transmitir información respectivamente. Tras el método de petición se añade la dirección URL ("Uniform Resource Locator") al que va dirigida la solicitud.

--Cabeceras ("headers"): Son metadatos que aportan información sobre la solicitud que se va a realizar al servidor. En las cabeceras se indica la dirección del host con el que se va a establecer la comunicación.

--Cuerpo("Body"): El cuerpo se añade en una petición de tipo POST. Básicamente contiene la información a transmitir.

-Código de respuesta (exclusivo del servidor): El código de respuesta o de estado es un número de 3 dígitos que indica lo que ha pasado con la petición. Se añade en la primera línea tras especificar la versión de HTTP. El resto del contenido de la respuesta dependerá del valor de este código. Cada número tiene un significado concreto y se agrupan en categorías. El primer dígito (la centena) es el que indica el tipo de respuesta:

- 1xx: Respuestas informativas. Indica que la petición ha sido recibida y se está procesando.
- 2xx: Respuestas correctas. Indica que la petición ha sido procesada correctamente.
- 3xx: Respuestas de redirección. Indica que el cliente necesita realizar más acciones para finalizar la petición.
- 4xx: Errores causados en la petición HTTP del cliente.
- 5xx: Indica que ha habido un error en el procesamiento de la petición a causa de un fallo en el servidor.

Los mensajes HTTP se envían como texto plano, lo que lo hace más legible y fácil de depurar, pero también más largos. Todas las líneas deben ir acabadas por un salto de

línea("\n") y un retorno de carro("\r"). Para declarar el fin de la petición se utilizan un "\r\n" como terminador.

Ver Referencias: [22]–[24] para más información.

A continuación, se muestra un ejemplo de solicitud y respuesta HTTP realizado por el ESP32 usando la librería "WiFi.h". A la vez, se añadirá más información técnica para la correcta comprensión del protocolo.

- Petición POST para enviar un paquete de audio en formato WAV al servidor Django:

```
const char* host="192.168.3.16";
String request="POST /datasave HTTP/1.1\r\n";
String headers="Host: " + String(host) + "\r\n" +
               "Connection: Close\r\n" +
               "Content-Type: application/octet-stream\r\n"+
               "Content-Length: " + String(buffer_size) + "\r\n" +
               "\r\n";
client.print(request+headers);
client.write((uint8_t*)buffer, buffer_size);
```

-El nombre del servidor es una secuencia numérica en vez de lo que se está comúnmente acostumbrado a ver cuando se navega por la web. Realmente la dirección real de todos los servidores es una secuencia numérica de similar estructura a la anterior pero dado que los humanos somos naturalmente mejores recordando palabras que secuencias largas de números se creó los servidores DNS. **Un servidor DNS** ("Domain Name Server") se utiliza para asociar a una dirección IP una identificación alfanumérica única característica. De manera inversa es también capaz de convertir una dirección web expresada como texto a su equivalente dirección IP. Por ejemplo, podría utilizar un servidor DNS y asociar la dirección de mi servidor a "miServer.com". Entonces en el lugar de host colocaría miServer.com en vez de 192.168.1.36.

-“Connection:Close/Keep Alive”: Sirve para indicar si se cierra o se mantiene la conexión con el servidor.

-Content-Type: Indica el tipo de contenido que se va a enviar en el cuerpo de la petición. El tipo más habitual es el de "url-encoded" es decir, transmitir parámetros usando la codificación url (ej. "&" para separar parámetros, "=" para indicar el valor del parámetro o "?" para separar la URL de los parámetros). Para enviar un dato como el de un paquete de audio, aunque dure milisegundos, es una pésima idea tener que convertirlo a string y enviarlo como una cadena. En su lugar es mejor utilizar el tipo "octect/stream" para indicar que se va a enviar una secuencia de bytes sin formato.

-La función “client.write(const uint8_t *payload, size_t size)” es un método de la clase WiFiClient definido en la librería “WiFi.h”. Sirve para enviar una cadena de bytes al host de destino.

•Ejemplo de respuesta HTTP. Se va a utilizar un fragmento de código de la ESAI cuando opera como punto de acceso y servidor para ser configurado. El cliente (el navegador de un móvil u ordenador) realiza una petición GET:

```
const char* form_modos =  
"HTTP/1.1 200 OK\r\n"  
"Content-Type: text/html\r\n"  
"Connection: Close\r\n"  
"\r\n"
```

El servidor estaría indicando que la petición ha sido correctamente procesada. Tras el terminador enviaría la plantilla “html” con la página web a mostrar al usuario. En concreto en este caso, enviaría la página de la configuración del modo de operación y los eventos a detectar de la ESAI. El usuario seleccionará una serie de botones que se enviarán como un formulario (petición POST).

2.3.2.1 Pruebas de rendimiento:

La forma de medir el rendimiento consistirá en realizar una serie de envíos de paquetes de audios de forma iterada y medir el tiempo que dura la transacción hasta que se recibe la respuesta del servidor. La idea es simular las condiciones reales en las que operará de forma habitual. La prueba se realizará primero utilizando HTTP de forma insegura y posteriormente creando un canal cifrado mediante el protocolo criptográfico SSL/TLS. Este formato seguro derivado del protocolo HTTP se le conoce comúnmente como protocolo HTTPS. En el apartado de SSL/TLS se encuentra más información.

Para la prueba se han realizado un total de 200 envíos de un buffer de 8000 elementos de tipo int16_t. El tamaño del paquete es de 16KB y se ha promediado el valor. La posición del dispositivo es a una distancia de 3 metros del punto de acceso y con dos paredes entre medio. Es importante destacar que el ESP32 usará una frecuencia de CPU de 240 MHz. Menores frecuencias arrojarán mayores tiempos de envío.

Resultados:

HTTP >> $T_{medio} = 152,18 \text{ ms}$

HTTPS >> $T_{medio} = 729,894 \text{ ms}$

Podríamos realizar un análisis estadístico para evaluar la dispersión de los datos, pero la idea es obtener una aproximación. Excepto el primer envío que siempre tiene un tiempo

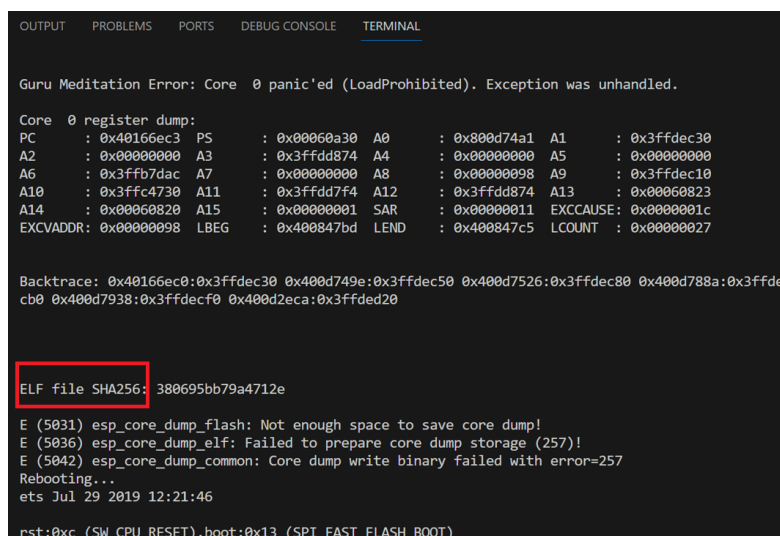
anómalo la mayor parte de los valores de tiempo de envío se han mantenido cercano a la media.

Conclusión:

Se trata de un resultado bastante tétrico. Un valor medio de 730 ms es un valor que simplemente no es aceptable si queremos enviar audio en tiempo real a nuestro servidor. Incluso aunque se repartiesen la actividad de lectura de audio y el envío de datos al servidor mediante 2 tareas en tiempo real ejecutadas en distintos núcleos, el sistema no va a funcionar de forma robusta y óptima. Simplemente, para los recursos limitados del EPS32, este método no es el adecuado. Además, con HTTP es mucho más difícil lograr uno de los objetivos del internet de las cosas: la flexibilidad y escalabilidad. Es decir, la posibilidad de añadir e integrar muchos dispositivos similares sin añadir a la vez complejidad y poder controlarlos a la vez.

A pesar de todo, el protocolo HTTP sigue siendo el más usado por los navegadores debido a su robustez y facilidad para intercambiar plantillas "html" o cualquier otro tipo de archivo. Además, a diferencia de otros modos de comunicación IoT, HTTP no tiene límite teórico para el tamaño de archivos a enviar, lo cual puede ser más interesante en dispositivos con alta capacidad de procesamiento como un móvil inteligente o un ordenador.

Nota importante: El módulo acelerador criptográfico que realiza las operaciones criptográficas del protocolo HTTPS tiene una limitación de 16 KB de datos. Si se intentan enviar un flujo de datos superior puede saturar al módulo y salirnos el siguiente mensaje de error en la consola antes de que la unidad se reinicie:



```
OUTPUT  PROBLEMS  PORTS  DEBUG CONSOLE  TERMINAL

Guru Meditation Error: Core  0 panic'ed (LoadProhibited). Exception was unhandled.

Core 0 register dump:
PC      : 0x40166ec3  PS      : 0x00060a30  A0      : 0x800d74a1  A1      : 0x3ffdec30
A2      : 0x00000000  A3      : 0x3ffdd874  A4      : 0x00000000  A5      : 0x00000000
A6      : 0x3ffb7dac  A7      : 0x00000000  A8      : 0x00000098  A9      : 0x3ffdec10
A10     : 0x3ffc4730  A11     : 0x3ffdd7f4  A12     : 0x3ffdd874  A13     : 0x00060823
A14     : 0x00060820  A15     : 0x00000001  SAR     : 0x00000011  EXCCAUSE: 0x0000001c
EXCVADDR: 0x00000098  LBEG    : 0x400847bd  LEND    : 0x400847c5  LCOUNT : 0x00000027

Backtrace: 0x40166ec0:0x3ffdec30 0x400d749e:0x3ffdec50 0x400d7526:0x3ffdec80 0x400d788a:0x3ffdec
b0 0x400d7938:0x3ffdecf0 0x400d2eca:0x3ffded20

ELF file SHA256: 380695bb79a4712e

E (5031) esp_core_dump_flash: Not enough space to save core dump!
E (5036) esp_core_dump_elf: Failed to prepare core dump storage (257)!
E (5042) esp_core_dump_common: Core dump write binary failed with error=257
Rebooting...
ets Jul 29 2019 12:21:46

rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
```

Figura 19. Mensaje de la consola por un desbordamiento del módulo criptográfico. Creado por el autor

El archivo ELF (“Executable and Linkable Format”) es aquel para representar archivos ejecutables y objetos compilados que contienen el código y los datos necesarios para programas y firmware que se ejecutan en el microcontrolador ESP32. También se encuentra la información de volcado del núcleo (“core dump”).

En dicho mensaje nos está indicando que el fallo está relacionado con la función SHA-256 del protocolo de encriptación SHA.

Información adicional de interés:

Hay una particularidad desconocida del ESP32, de la cual no existe ningún tipo de información oficial al respecto, relacionada con el protocolo HTTPS que ocurre con la librería “WiFiClientSecure”. Si la dirección del host viene dada en formato IP posiblemente fallará la conexión con el servidor. Podría pensarse que se trata de un problema del CN (“Common Name”) del certificado del servidor, pero lo cierto es que creando el certificado con el CN igual a la IP (no a su nombre DNS) el problema persiste. Parece ser que la librería no acepta el formato IP como dirección de host.

2.3.3 Protocolo MQTT

MQTT, de las siglas MQ (Message Queing) Telemetry Transport, es un protocolo de comunicación M2M ("machine-to-machine") de la capa de aplicación, ligero y que sigue un patrón de tipo publicador/suscriptor. A lo igual que HTTP, está basado en la pila TCP/IP como base para la comunicación, aunque también opera con el protocolo QUIC como alternativa para TCP en la capa de transporte. MQTT tiene una arquitectura completamente distinta a la de petición-respuesta entre cliente y servidor de HTTP. En su lugar, la comunicación se establece entre publicadores y suscriptores (pub-sub) con un servidor central conocido como "broker".

Este broker actúa como un intermediario que facilita la comunicación entre los publicadores y los suscriptores. Para filtrar los mensajes que son enviados a cada cliente los mensajes se disponen en temas ("topics") organizados jerárquicamente. Un cliente o varios clientes publican un mensaje en un determinado topic. Otros clientes pueden suscribirse a este tema y el broker le hará llegar los mensajes suscritos. La enorme ventaja de esta estructura es que múltiples dispositivos pueden recibir el mismo mensaje simultáneamente (alta escalabilidad).

En el caso de MQTT cada conexión entre clientes y el broker se mantiene abierta y se "reutiliza" en cada comunicación. Es una diferencia, por ejemplo, a una petición HTTP 1.0 donde cada transmisión se realiza a través de conexión.

MQTT es un protocolo donde los elementos de control son bytes binarios y no pesadas cadenas ("strings") de texto. Se utiliza un formato de comando y reconocimiento de comando. Eso significa que cada comando tiene un reconocimiento asociado.

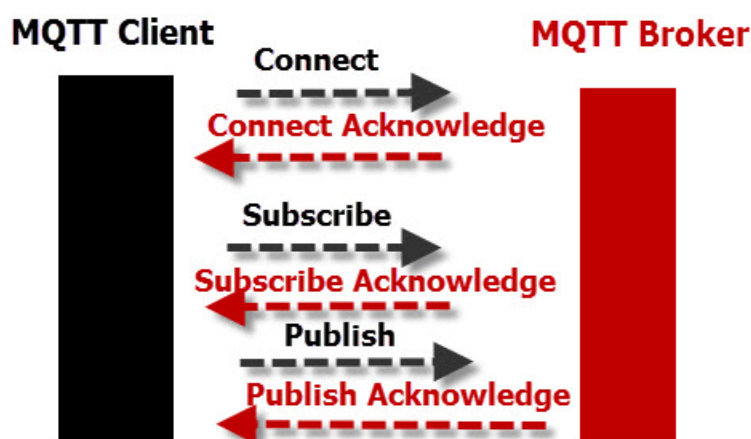
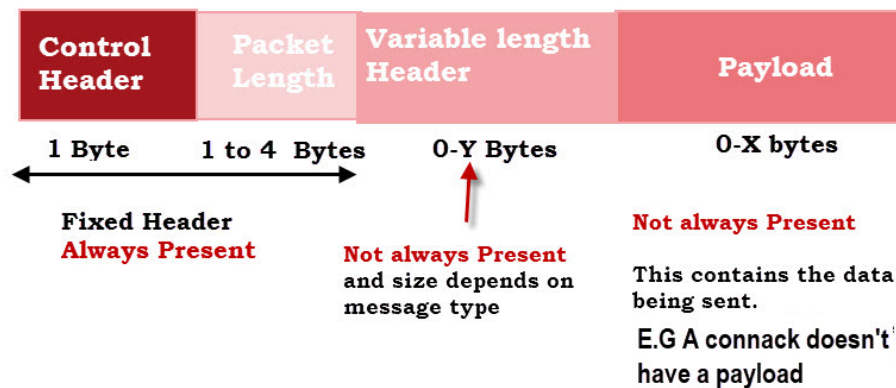


Figura 20. Formato de reconocimiento de comandos de MQTT. [25]

Excluyendo la información del protocolo MQTT como ID de cliente, los nombres de usuario y las contraseñas (para acceder al broker) que se codifican como cadenas UTF-8, todo el resto de datos, como la carga útil (“payload”) del mensaje, son datos binarios. El contenido y el formato son específicos de la aplicación.

El formato de paquete o mensaje MQTT consta de un encabezado fijo de 2 bytes (obligatorio) + encabezado de tamaño variable (depende del tipo de mensaje) + carga útil (opcional).



MQTT Standard Packet Structure

Figura 21. Estructura de un mensaje MQTT. [25]

El primer byte de control se estructura de la siguiente forma: Los 4 bits más significativos codifican en función de su valor (0001,0010...1111) el tipo de mensaje a enviar (CONNECT,CONNACK,PUBLISH,SUSBCRIBE,DISCONNECT etc). Los 4 bits menos significativos funcionan como banderas (“flags”) para indicar la QoS (“quality of service”), la retención, o si trata de un mensaje duplicado (QoS 1 o 2).

Control Packet	Fixed header flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	DUP ¹	QoS ²	QoS ²	RETAIN ³
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0

Annotations:

- Duplicate message**: Points to the DUP flag in the PUBLISH row.
- Quality of Service**: Points to the QoS flags in the PUBLISH row. 00,01,10 =QOS 0,1,2
- Retain Message**: Points to the RETAIN flag in the PUBLISH row.

Figura 22. Parte baja del byte de control, correspondiente a las banderas de control. [25]

Mediante esta estructura se puede lograr una eficiencia astronómica si se compara con una solicitud HTTP. Por ejemplo, un mensaje de desconexión constaría sólo de 2 bytes (1 byte fijo de control y otro byte para indicar la longitud del paquete).

El término calidad del servicio QoS, mencionado anteriormente, son tres medidas distintas de garantizar el envío y recepción del mensaje al destinatario:

-QoS 0, ("at most once"): En este nivel, el mensaje se entrega al receptor al máximo una vez. No se garantiza la entrega y no hay confirmación de recepción. Este nivel es adecuado para aplicaciones donde la pérdida ocasional de mensajes no es crítica.

-QoS 1, ("At least once"): En este nivel, se garantiza que el mensaje se entregará al menos una vez. El remitente envía el mensaje y espera una confirmación de recepción del receptor. Si no se recibe la confirmación, el mensaje se reenvía. Puede haber duplicación de mensajes, pero se asegura la entrega.

-QoS 2, ("Exactly once"): Este nivel garantiza que el mensaje se entregará exactamente una vez. Implica un proceso más complejo de 8 pasos, donde el remitente y el receptor se intercambian mensajes de confirmación para asegurar que el mensaje se entregue sin duplicados. Aunque proporciona la entrega exacta, supone una mayor sobrecarga de comunicación.

Dada la importancia de que nuestra estación acústica funcione de manera óptima y robusta para la detección de eventos, nuestra aplicación usará una QoS de 2, con el fin de ser un dispositivo que ofrezca las mejores prestaciones. En caso de que un QoS de 2 produjese latencias excesivas, podría optarse por usarse QoS de 1 aunque disminuyese la calidad (repetición del mismo paquete de sonido).

Existen muchas librerías para implementar MQTT en el ESP32. Se ha invertido un tiempo de investigación y análisis para encontrar la más adecuada. Las librerías principalmente analizadas son: PubSubClient.h de Nick O'Leary, AsyncMqttClient de Marvin Rogers y espMqttClient.h de Bert Melis. Para las pruebas de tiempo de envío se han utilizado paquetes de 16KB (excepto para la biblioteca de Marvin Rogers en la que se ha usado 10,2 KB), en condiciones similares a las pruebas HTTP.

El broker utilizado ha sido el mosquito ejecutado en la red local. También se ha probado a utilizar un bróker mosquito online gratuito con la dirección "test.mosquitto.org". Los resultados son sorprendentemente muy similares. Por favor vaya al subapartado "[Broker](#)" dentro el apartado "Servidor" para ver como configurar el broker mosquito local, con cifrado TLS.

2.3.3.1 Pruebas de rendimiento:

<PubSubClient>: Es la librería más famosa y descargada. Se trata de una librería de MQTT síncrona sencilla e intuitiva. Sin embargo, tiene una importante limitación y es que no puede publicar mensajes con una QoS (Quality of Service) de tipo 1 o 2, sólo de tipo 0.

Prueba de latencia:

MQTT >> $T_{medio} = 14,496 \text{ ms}$

MQTTs >> $T_{medio} = 56,360 \text{ ms}$ (128,4 ms con mosquitto online)

<AsyncMqttClient>: Permite la comunicación MQTT asíncrona. Tiene el problema de forzarte a enviar la payload como un array const char*. PubSubClient daba la posibilidad de enviarlo como un array "uint8_t". La conversión de un array int16_t a un array uint8_t es sencilla y directa usando el operador de cast (uint8_t *). Sin embargo, con esta librería tendríamos que convertir todo el paquete de audio en una cadena de texto usando bucles y la función "sprintf". Éste método es muy ineficiente. Un elemento int16 que fuese por ejemplo 10500(sonido fuerte) lo que estaba codificado con 2 bytes ahora necesitaría 5(1 char por cada dígito). Teniendo en cuenta el tamaño de nuestro buffer (15000-30000) tendríamos que crear otro buffer con un tamaño del doble o el triple, y eso acabaría con nuestra memoria SRAM. Además, la librería funciona de forma inestable (necesita añadir "delays" de al menos 10 ms y tiene problemas con el límite de carga (por eso se han usado 10.200 bytes para la prueba). Encima de todo no funciona bien con protocolo TLS. Las pruebas de cronometría arrojan un tiempo medio de alrededor 10-15 ms. Se descarta el uso.

<espMqttClient>: Esta librería fue desarrollada como mejora de la librería anterior. Ofrece una comunicación MQTT tanto síncrona como asíncrona, aunque de forma asíncrona no soporta TLS"). No requiere añadir retardos para su funcionamiento, soluciona el problema de la carga y las inestabilidades asociadas a los envíos con cifrado TLS. La "guinda del pastel" es que además los resultados de latencia son impresionantes.

MQTTs >> $T_{medio} = 301,41 \text{ } \mu\text{s}$

Incidencias con MQTT:

Cuando se utiliza cualquier librería MQTT junto con la librería "driver/i2s" se produce de forma inesperada cada cierto tiempo un retardo en el tiempo envío de mensajes. Este retardo puede llegar a ser un valor grosero (mayor a 400-1200 ms). Incluso aun desactivando el puerto I2S usando las funciones "i2s_stop()", i2s_uninstall() el conflicto

persistía. El incidente parece estar relacionado con el uso de recursos y liberación de semáforos y puede ser muy complejo de solucionar manualmente.

La solución más útil y elegante al problema fue la de separar la tarea de lectura i2s y envío mqtt en dos tareas en tiempo real, y ejecutarlas en núcleos distintos. Esta separación de tareas en tiempo real además disminuye la latencia ya que la tarea de lectura no tiene que esperar a que se realice el envío de datos al servidor.

Conclusión:

Si visualizamos el tiempo de envío de datos respecto al tiempo en el que no se transmite (dado por el resto del tiempo hasta que se complete la lectura del audio y su almacenamiento en un buffer) como si fuera el ciclo de trabajo de una señal digital obtendríamos una medida del consumo de corriente dada por la relación de tiempo que el módulo RF está transmitiendo/recibiendo.

Dado que el envío de los datos se realiza en paralelo a la recepción I2S el intervalo total por cada ciclo será equivalente a la tarea de mayor duración, en este caso la tarea I2S. El tiempo de lectura será de 0,75 segundos aproximadamente, resultado de realizar 12000 muestras a una frecuencia de muestreo de 16 KHz.

$$HTTPS \rightarrow D(\%) = \frac{729.894 \text{ ms}}{750 \text{ ms}} = 97,32 \%$$

$$MQTTS \rightarrow D(\%) = \frac{0.400 \text{ ms}}{750 \text{ ms}} = 0,053 \%$$

Esto significa que en con MQTT el ESP32 estará casi al 100% en modo de recepción. Como vimos anteriormente, el consumo es considerablemente más bajo. En el caso de HTTP sería todo lo contrario, la mayor parte del tiempo estaría transmitiendo.

Indiscutiblemente el protocolo MQTT supera en casi todos los sentidos al HTTP tanto en celeridad, consumo de recursos, pero también en las ventajas de escalabilidad y flexibilidad que ofrece el sistema de mensajería entre subscriptores, publicadores y el bróker.

La única “desventaja” que puede tener el protocolo MQTT es el tamaño de la carga que está limitado a 256 MB. Y se ha entrecomillado desventaja a propósito porque 256MB es muchísimo más que suficiente de lo que necesitan la inmensa mayoría de aplicaciones IoT.

2.4 SSL/TLS

En aras de simplificar, el enfoque de esta sección será más bien práctico. El funcionamiento subyacente del cifrado SSL/TLS, el intercambio de claves y el papel que juegan los diversos protocolos de encriptación (AES, SHA...) están fuera del alcance de este proyecto. El contenido se centrará en los pasos a seguir para crear certificados SSL/TLS autofirmados que puedan ser utilizados para asegurar las comunicaciones con el servidor Django y el bróker:

1. Creamos una Autoridad Certificadora local, compuesta por el certificado raíz CA ("root CA") y la clave privada CA.
2. Generamos la clave privada del servidor.
3. Con la clave del servidor se realiza una CSR ("Certificate Signing Request") o solicitud de firma de certificado.
4. A partir de la petición CSR la autoridad certificadora firmará la petición y como resultado se generará el certificado SSL/TLS del servidor.
5. El último paso será incluir en nuestro navegador del móvil u ordenador al CA local que hemos creado dentro de la lista de certificados de confianza. De lo contrario el navegador emitirá un mensaje de advertencia al ingresar la dirección del servidor.

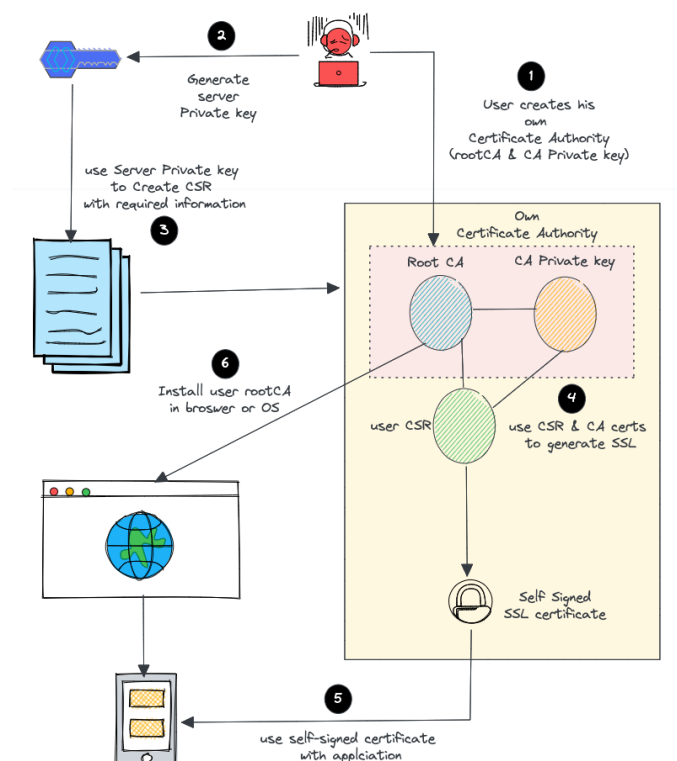


Figura 23. Esquema con los pasos para generar un certificado firmado localmente.

2.4.1 Guía para generar certificados con OpenSSL

a) Descargar el software: <https://slproweb.com/products/Win32OpenSSL.html>

Durante la instalación seleccionar cuando aparezca “Copy openSSL DLLs to” seleccionar la opción “The OpenSSL binaries /bin directory”:

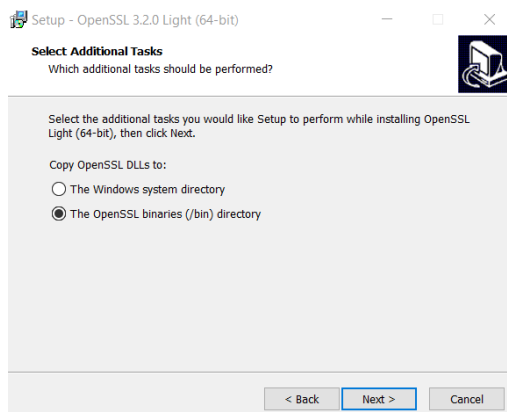


Figura 24. Guía de instalación de OpenSSL. Creado por el autor

b) Abrir cmd como administrador e ir a la dirección de la carpeta donde vayamos a crear nuestros certificados.

c) Comandos a introducir explicados:

1.Crear la llave privada del root CA:

```
openssl genrsa -des3 -out ca.key 2048
```

Nota: “rsa -des3” se refiere al tipo de encriptación, en concreto triple encriptación con contraseña(“passphrase”) que deberemos de introducir. 2048 se refiere a la longitud en bits del módulo de la llave.

2.Con la anterior llevamos creamos el certificado CA:

```
openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
```

Explicación:

-“days” 1826 es el tiempo que durará el certificado en días.

-“x509” es un estándar referido al formato del certificado digital

-Por lo tanto, en el anterior comando decimos: solicitamos(req) un nuevo(new) certificado con formato -x509 que dure 1826 días llamado ca.crt usando la llave ca.key

-Se pedirá la clave de la llave y se presentará un formulario a rellenar. Ejemplo:

Country Name (2 letter code) [AU]:ES

State or Province Name (full name) [Some-State]:Cádiz

Locality Name (eg, city) []:San Fernando

Organization Name (eg, company) [Internet Widgits Pty Ltd]: PECCI

Organizational Unit Name (eg, section) []:TFGTEST

Common Name (e.g. server FQDN or YOUR name) []: 192.168.X.X

Email Address []:dapecsan@gmail.com

Aviso:

Es importante que el common name(CN) **sea la dirección en formato IP donde se vaya a ejecutar el servidor**, es decir la IP del punto de acceso al que esté conectado nuestro ordenador que gestiona el servidor. No es recomendable poner el nombre **DNS** asociado al servidor, en caso de estar usándolo (al menos durante los experimentos realizados hacer eso causó errores en la identificación de los certificados durante el “handshake”).

3.Crear la llave privada del servidor:

```
openssl genrsa -out server.key 2048
```

No añadimos el comando -des3 dado que no queremos una contraseña cada vez que se vaya a realizar una conexión con el servidor. Si la clave tiene contraseña puede usarse la siguiente línea para borrarla: clave: `openssl rsa -in server.key -out server-nopass.key`

4.Generar la CSR a partir de la llave creada del servidor:

```
openssl req -new -out server.csr -key server.key
```

5.Usar la llave del CA raíz para firmar la CSR y generar el certificado del servidor:

```
openssl x509 -req -in server.csr -CA ca.crt -Cakey ca.key -Ccreateserial -out server.crt -days 360
```

-Aparecerá de nuevo un formulario. Debemos asegurarnos que el CN siga siendo la IP de nuestro punto de acceso. Una vez completado nos pedirá introducir la contraseña. Al finalizar deberemos tener 5 archivos: ca.key, ca.crt, server.key, server.crs y server.crt

Los archivos server.key y server.crt serán utilizados por el servidor. Los clientes deberán tener el certificado de autoridad raíz, ca.crt para poder establecer una conexión con el servidor. En el caso del navegador lo añadimos a la lista de certificados de confianza. En un dispositivo como el ESP32, deberemos copiar y pegar el certificado en una cadena de texto: Ej:

```
char* ca_cert = R"(... )" ;
```

Los certificados son un galimatías de caracteres seguidos. La “R” se utiliza para ignorar las secuencias de escape que pueden causar la conjunción de ciertos caracteres. De no hacerlo aparecerán con casi total seguridad problemas para validar el certificado en algún punto.

Algunos servidores prefieren ciertos formatos a otros. Si necesitamos cambiarlo sólo tenemos que ejecutar la siguiente línea de comando:

Transformar el formato del certificado “ca.crt” a “ca.pem” para que funcione en Django:

```
*openssl x509 -in ca.crt -out ca.pem*
```

2.5 Configuración del servidor.

2.5.1 Django

Django es un “framework” basado en el lenguaje de programación de alto nivel Python que en los últimos años ha ido ganando gran popularidad entre los desarrolladores de servidores y aplicaciones web por ser una herramienta sencilla de usar, potente y eficaz. Fue creado por programadores experimentados que conocían de primera mano el calvario que supone el desarrollo de aplicaciones web. Django permite al usuario el diseño con un nivel de abstracción superior para que se pueda concentrar en crear cualquier aplicación sin necesidad de reinventar la rueda, además es gratuito y de código abierto. Django está enfocado principalmente en el diseño “**backend**”, es decir la parte del lado del servidor que no es visible ni accesible por el cliente, que se encarga de gestionar y procesar los datos, interactuar con las bases de datos y manejar la lógica de la aplicación. No obstante, Django se integra bien con las tecnologías “**frontend**” (HTML, CSS, JavaScript) ya que cuenta con un motor de plantillas HTML que se puede usar para renderizar contenido dinámico.

Conda y miniconda:

Anaconda o Conda es un sistema de gestión de paquetes multiplataforma que puede instalar, actualizar y gestionar paquetes y entornos de software en varios sistemas operativos. Miniconda es una pequeña versión de arranque de Anaconda minimalista que incluye solo conda, Python, los paquetes de los que ambos dependen y una pequeña cantidad de otros paquetes útiles (como pip, zlib y algunos otros). Pip será el gestor que nos permitirá instalar todas las librerías y frameworks que necesitemos para nuestro proyecto. Esta herramienta permite crear un entorno virtual (“venv”), que no es más que un espacio donde almacenaremos todas las librerías y herramientas que vayamos instalando con unas versiones determinadas. Un problema común cuando se desarrolla con Python y sus librerías es que a veces ocurren conflictos entre ellas y sus respectivas versiones. Además, cada proyecto puede tener unas necesidades concretas de librerías. El entorno evita que haya estos conflictos y ofrece las librerías específicas a las necesidades del proyecto.

Guía de configuración para windows:

1. Descargar e instalar el software de miniConda de la web:

<https://docs.conda.io/projects/miniconda/en/latest/>

2. Ir al inicio de Windows y acceder a la terminal de miniconda. Para crear y activar el venv en una carpeta personalizada escribimos:

```
cd C:\dirección\al\la\carpeta  
conda create --name mi_venv  
conda activate mi_venv
```

3. Instalar python y comprobar si se ha instalado correctamente:

```
(mi_venv) conda install -c anaconda django  
(mi_venv) python --version
```

Debe de aparecer Python junto con la versión. Ej: Python 3.10.10

4. Crear un nuevo proyecto:

```
django-admin startproject nombre_del_proyecto
```

Se creará una carpeta con varios archivos.

5. Crear la carpeta con la aplicación y sus archivos:

```
python manage.py startapp nombre_app
```

6. Abrir el proyecto en VS code. Podemos ir a la carpeta donde está el proyecto y arriba en la barra de dirección escribimos “cmd”. Se nos abrirá la la consola en la dirección de dicha carpeta. Escribimos “code .” y se abrirá VS Code cargando el proyecto.

7. Ir al archivo settings.py:

```
# Añadir la IP del punto de acceso al que está conectado nuestra computadora
```

```
ALLOWED_HOSTS = ['localhost', '192.168.1.XX']
```

```
#Añadir nuestra aplicación
```

```
INSTALLED_APPS = [... 'nombre_app']
```

```
#False si quieres que la dirección acabe sin barra.
```

```
APPEND_SLASH=True
```

```
#Importante para que no de fallos durante la validación de certificados
```

```
TIME_ZONE = 'Europe/Madrid'
```

#-PARA utilizar la sesión en Django (y poder guardar variables globalmente añade el middleware de sesión “django.contrib.sessions.middleware.SessionMiddleware” en la lista de MIDDLEWARE. Por lo general, se recomienda colocar este middleware después de django.middleware.security.SecurityMiddleware y antes de django.middleware.common.CommonMiddleware.

#Opcionalmente puede especificarse aquí el puerto para no tener que añadirlo en la consola cada vez que queramos ejecutar el servidor.

#PORT = 9000

8. Seleccionar el intérprete apropiado (el entorno virtual que hemos creado). Para ello pulsamos CTRL+SHIFT+P y arriba nos saldrá una barra con la opción Python: Select interpreter >> Enter interpreter path >> Buscamos la dirección donde creamos nuestro entorno y le damos a “Select interpreter”.

9. Ejecutar el servidor con el siguiente comando:

```
python manage.py runserver 192.168.1.XX:port    #port típicamente suele ser 8000
```

Nos saldrá un mensaje de que nuestro servidor se está ejecutando en la dirección:

<http://192.168.X.X:8000/>

En el archivo “views.py” definiremos todas nuestras funciones o “vistas” que realicen las tareas de nuestro proyecto. Cada vista podrá ser potencialmente un recurso de nuestro servidor que pueda servir plantillas html, variables, archivos multimedia etc. Si queremos que pueda ser accedida deberemos añadir su dirección en el archivo “urls.py” dentro de nuestra app

Guía para MQTT en Django:

Se procede a describir los pasos a seguir para lograr que django gestione la conexión a un broker y se suscriba a un tema:

1. Instalar la librería paho-mqtt e importarla:

```
conda install -c conda-forge paho-mqtt
```

#Añadir las siguientes directivas en la parte superior de views.py

```
import paho.mqtt.client as mqtt
```

```
from django.conf import settings (en caso de configurar los parámetros en settings.py)
```

2. Definimos en views la función mqtt_init() que configura e inicializa todos los parámetros: dirección del broker, puerto, tiempo para mantener la conexión(“keepalive”), usuario y

clave si la hubiera). Dentro de `mqtt_init()` asignamos la dirección de las funciones de llamada (callback) de los eventos mqtt (`on_message`, `on_connect`...) que hemos definido previamente en “views.py”. Finalmente establecemos la conexión con el broker y retorna el objeto “`mqtt.Client`” para que pueda ser usado por las demás funciones de forma global.

Como crear hilos separados:

Para que el servidor django pueda gestionar la comunicación MQTT y además atender a las solicitudes HTTP entrantes se crea una estructura multihilo. Si no hacemos esto, django atenderá sólo a las peticiones mqtt dado que la función “`mqttClient.loop_forever()`” es bloqueante y las peticiones HTTP de cualquier cliente serán rechazadas automáticamente.

1. Importamos la librería `threading` y `time`.
2. Configuramos la tarea en bucle de gestión mqtt en un hilo paralelo al principal donde se ejecutarán todas nuestras demás funciones. Añadimos una pausa en dicha tarea de 1-100ms. Hay que tener en cuenta que este retardo se añadirá a la latencia existente.

```
hilo_mqtt = threading.Thread(target=mqtt_loop)
```

```
hilo_mqtt.daemon = True # El hilo se detendrá cuando el programa principal lo haga
```

```
hilo_mqtt.start()
```

Nota: Esta librería no crea una estructura multihilo real, sino que funciona como un planificador de tareas. Quedaría pendiente de mejorar otra manera de lograr el multihilo de forma más óptima. También podría ejecutarse un servidor django en paralelo o utilizar alguna librería mqtt asíncrona disponible para Django y que sea compatible con TLS.

Guía para utilizar seguridad TLS:

En este apartado se asume que ya se tiene el certificado TLS creado y firmado por la CA local.

1. El comando predeterminado “`manage.py runserver`” de Django no admite SSL; por lo tanto, necesitamos usar el comando alternativo “`manage.py runserver_plus`”, que es parte del paquete Django Extensions:

```
conda install -c conda-forge django-extensions werkzeug
```

2. Añadimos “`django_extensions`” dentro de “`INSTALLED_APPS`” en `settings.py`.
3. Importar la librería “`ssl`”
4. Para ejecutar el servidor tenemos que utilizar el siguiente comando:

```
python manage.py runserver_plus --cert-file djangoServer.crt --key-file djangServer.key  
192.168.X.X:443
```

-443 es el puerto típico para HTTPS

Ver referencias: [26]–[28]

2.5.2 Bróker Mosquitto local

Esta sección se ha creado en caso de que se utiliza el bróker mosquitto de forma local en vez de la versión online gratuita. Puede ser de interés utilizar el bróker local si queremos tener un mejor seguimiento de cómo se está llevando a cabo la comunicación entre los clientes y el bróker. Vamos a explicar todos los pasos para hacer que funcione utilizando TLS.

Lo primero será descargar “mosquitto” de la página “mosquitto.org” e instalarlo.

Crear credenciales de usuario:

1. Abrir el cmd en modo administrador e ir a la carpeta donde esté instalando, por defecto en “C:\Archivos de programa\mosquitto”.

2. Introducir el siguiente comando:

```
mosquitto_passwd -c claves Cliente1
```

Siendo “claves” el nombre que tendrá el archivo con las contraseñas asociada al usuario Cliente1

3. Nos pedirá después que introduzcamos la clave. Al ingresarla se nos creará el archivo.

4. Nos vamos al archivo mosquitto.conf dentro de la carpeta mosquitto:

-Cambiamos “allow_anonymous” a false

-Agregamos el archivo con las credenciales de usuarios permitidos:

```
password_file claves
```

Si se quiere añadir más usuarios utilizando el comando anterior se sobrescribirá el archivo en vez de añadir un nuevo par usuario-clave. La solución es crear un archivo de texto “claves.txt”, con el mismo formato:

```
usuario1:pass1
```

```
usuario2:pass2
```

Después en el cmd como administrador se usa el siguiente comando:

```
mosquitto_passwd -U claves.txt
```

De esta manera podremos añadir tantos usuarios como queramos.

Nota: Recuerda quitar la extensión “.txt” y añadirlo al archivo “mosquitto.conf”

SSL/TLS en mosquitto:

Se asume que se ha creado ya el certificado raíz de la CA y el del servidor.

1. Ir al archivo “mosquitto.conf”y añadir:

```
listener 8883 0.0.0.0  
allow_anonymous false  
protocol mqtt
```

-Especificar las rutas de la clave y certificado del servidor y del certificado root CA. Finalmente indicamos la versión de TLS. No debemos escribir nada en “capath”.

```
certfile C:\Program Files\mosquitto\certs\brokerCert.crt
```

```
keyfile C:\Program Files\mosquitto\certs\mosquitto.key
```

```
cafile C:\Program Files\mosquitto\certs\rootCA.crt
```

```
tls_version tlsv1.2
```

Iniciar bróker mosquitto

Dentro de la carpeta donde está instalado mosquitto abrimos el cmd y ejecutamos el siguiente comando:

```
mosquitto -c mosquitto.conf -v
```

Al añadir “-v” indicamos que se inicie en modo “verbose” para poder visualizar todo el intercambio de mensajes entre los clientes y el bróker.

2.6 Modelo de red neuronal profundo YAMNet

YAMNet ("Yet Another Multilayer Perceptron Network") es un modelo de red neuronal desarrollado por Google en 2019. El modelo está basado en la arquitectura de convolución separable en profundidad de MobileNet v1. Funciona como un clasificador de eventos de audio que toma la forma de onda de sonido como entrada y realiza predicciones independientes para cada uno de los 521 eventos de audio de la ontología de AudioSet. Ésta ontología es un conjunto de datos ("dataset") mediante el cual se ha entrenado al modelo que está formada por 5.800 horas de sonido tomadas de más de 2,1 millones de videos de youtube. YAMNet tiene 3.7 millones de pesos y realiza 69.2 millones de multiplicaciones para cada "frame" de entrada de 960 ms.

Explicar cómo funciona la nueva arquitectura de MobileNet v1 estaría algo fuera del alcance, pero de forma simplificada logra mantener las ventajas de las redes convolucionales clásicas (que son muy potentes para el análisis y clasificación de imágenes) pero reduciendo significativamente la cantidad de parámetros y operaciones de cómputo. Como consecuencia de ser más ligero y rápido se ha logrado integrar el modelo YAMNet en muchos sistemas embebidos con recursos más limitados (teléfonos inteligentes principalmente).

Sin embargo, MobileNet v1 es una arquitectura diseñada realmente para la visión por computadora en dispositivos móviles. ¿Cómo se puede utilizar entonces para la detección de sonido? La respuesta está en el espectrograma. Un espectrograma se genera aplicando la Transformada Rápida de Fourier (FFT) a pequeños segmentos o ventanas temporales de la señal de audio en intervalos regulares a lo largo del tiempo. El desplazamiento o salto temporal que existe entre los segmentos ("hop length") puede ser menor que la duración de las ventanas temporales ("window length"), lo cual significa que puede existir solapamiento entre segmentos. El espectrograma convierte la señal de audio en el dominio del tiempo al dominio de la frecuencia, mostrando cómo varía la energía de diferentes frecuencias a lo largo del tiempo. En la figura 25 se encuentra un ejemplo de espectrograma realizado por el software "sdrsharp" para el procesamiento de la señal de radiofrecuencia entrante. En este caso para una visualización más intuitiva el eje de abscisas se corresponde al de frecuencia y el eje vertical con la dimensión temporal, aunque normalmente es al contrario. La intensidad de la señal para la frecuencia que nos da la FFT se asocia al color con el que se representa. También se puede realizar el espectrograma en blanco y negro mediante escala de grises.

A la hora de diseñar las ventanas temporales de un espectrograma hay que llegar a un compromiso entre la resolución temporal y el solapamiento. Si el salto temporal o "hop" es muy pequeño lograremos una mayor resolución, pero también un mayor solapamiento lo que se traduce en datos redundantes (ver figura 26).

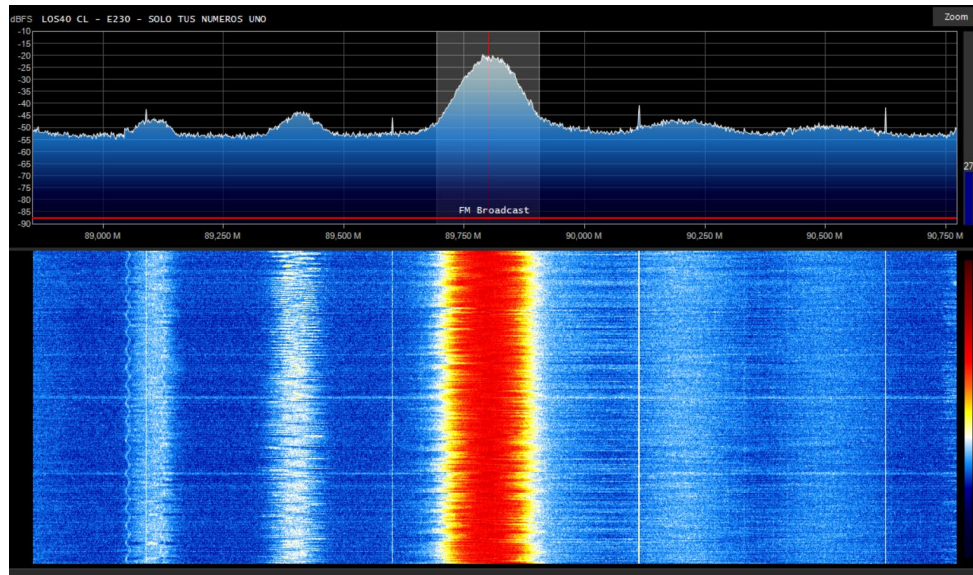


Figura 25. Espectrograma realizado para un segmento de 2,4MHz de la banda FM. Creado por el autor.

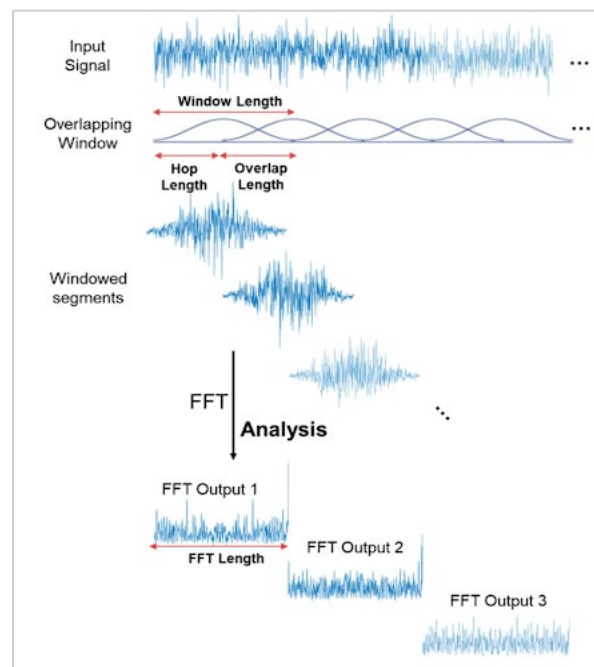


Figura 26. Estructura de un espectrograma. Ventana temporal e intervalos de FFT. [29]

A partir del espectrograma original se construyen otros tipos de parámetros los cuales pueden ayudar a obtener otro tipo de información de la señal de audio. Los más comunes

son el espectrograma logarítmico de mel y los coeficientes Cepstrales de las frecuencias de Mel (MFCC).

YAMNet está entrenado con fragmentos de espectrograma log-mel de 64 canales (“mel bins”) y 96 cuadros temporales. El desplazamiento de los cuadros del espectrograma (“hop length”) es de 10 ms, por lo que 96 cuadros abarcan 960 ms. Debido a que la ventana temporal del espectrograma es de 25 ms, se tarda 975 ms en obtener una salida de clasificación. Esto no quiere decir que la entrada de nuestro modelo no pueda ser menor o mayor a 975 ms. Gracias a las capas de “pooling” se puede reducir la dimensionalidad de la información conservando las características más relevantes mediante operaciones de promediación de regiones. Esto ayuda a capturar características importantes independientemente de su ubicación exacta en el tiempo, permitiendo al modelo ser más robusto a variaciones temporales y, por ende, generalizar eficazmente a longitudes de entrada que difieran ligeramente de la de entrenamiento. Además, el modelo convertirá la entrada igualmente en un archivo de 960 ms, lo único que sucederá es que habrá una región sin sonido en caso de que el audio introducido no llegue a 960 ms.

Experimentalmente se ha podido comprobar a lo largo de este proyecto, que el modelo es capaz de responder eficazmente incluso con entradas de audio de la mitad a la entrada teórica. En el proyecto de la ESAI tenemos un gran interés en reducir esta entrada para lograr reducir la latencia al máximo posible. Si una entrada que genere menor latencia sigue arrojando buenos resultados entonces se aprovechará la oportunidad.

Entradas/salidas del modelo:

YAMNet acepta como entrada un vector unidimensional de tipo NumPy o Tensor. Este vector de entrada representa una muestra de audio normalizada (valores comprendidos entre -1 y 1) en formato mono y con una frecuencia de muestreo de 16 KHz.

El modelo devuelve una tupla de tres elementos:

- **Scores:** es un Tensor float32 de forma (N, 521) que contiene las puntuaciones predichas por cuadro para cada una de las 521 clases en la ontología de AudioSet que son compatibles con YAMNet.
- **Embeddings:** es un Tensor float32 de forma (N, 1024) que contiene los embeddings o incrustamientos por cada frame, donde el vector de embedding es la salida promediada (“average-pooled”) que se introduce en la última capa del clasificador.
- **Log mel spectrogram:** es un Tensor float32 que representa el espectrograma logarítmico de mel de toda la forma de onda. Estos son los rasgos de audio pasados al modelo y tienen forma (num_spectrogram_frames, 64), donde “num_spectrogram_frames” es el número de frames producidos a partir de la forma de onda al deslizar una ventana de análisis de espectrograma de longitud 0.025 segundos con un paso de 0.01 segundos, y 64 representa el número de bins de mel.

Nuestro proyecto hará uso principalmente de la salida de scores o puntuaciones para la detección de eventos. El espectrograma puede utilizarse en el servidor Django para ofrecer más información visual, pero meramente a título de puesta en escena.

3 FIRMWARE

3.1 ESP32

En esta sección se asume que se han completado los pasos para programación del firmware del ESP32 y del servidor Django. Sólo El firmware del ESP32 tiene en total una extensión de casi 1000 líneas de código. Dada la inmensidad y complejidad del firmware total, se ha preferido hacer una descripción general del código por partes según funcionalidad, incluyéndose fragmentos del código para ejemplificar. Los fragmentos del código estarán ya parcialmente explicados mediante los comentarios.

El firmware del ESP32 hace un intenso uso de clases, objetos, Strings, funciones y librerías exclusivas del lenguaje C++ por lo que se aconseja tener cierto conocimiento del lenguaje para la correcta comprensión del código. La ESAI se puede entender como una máquina de estados finita, en cuyos estados se realiza una gran cantidad de funciones. La MCU trabajará siguiendo una estructura multihilo, aprovechando la arquitectura de doble núcleo. La actividad principal, la lectura de audio I2S y transmisión de los datos será realizada por dos tareas en tiempo real ejecutadas en distintos núcleos. De fondo, habrá otro cierto número de tareas con menor prioridad. A continuación, se muestra un diagrama que recoge de forma muy simplificada el funcionamiento de la ESAI:

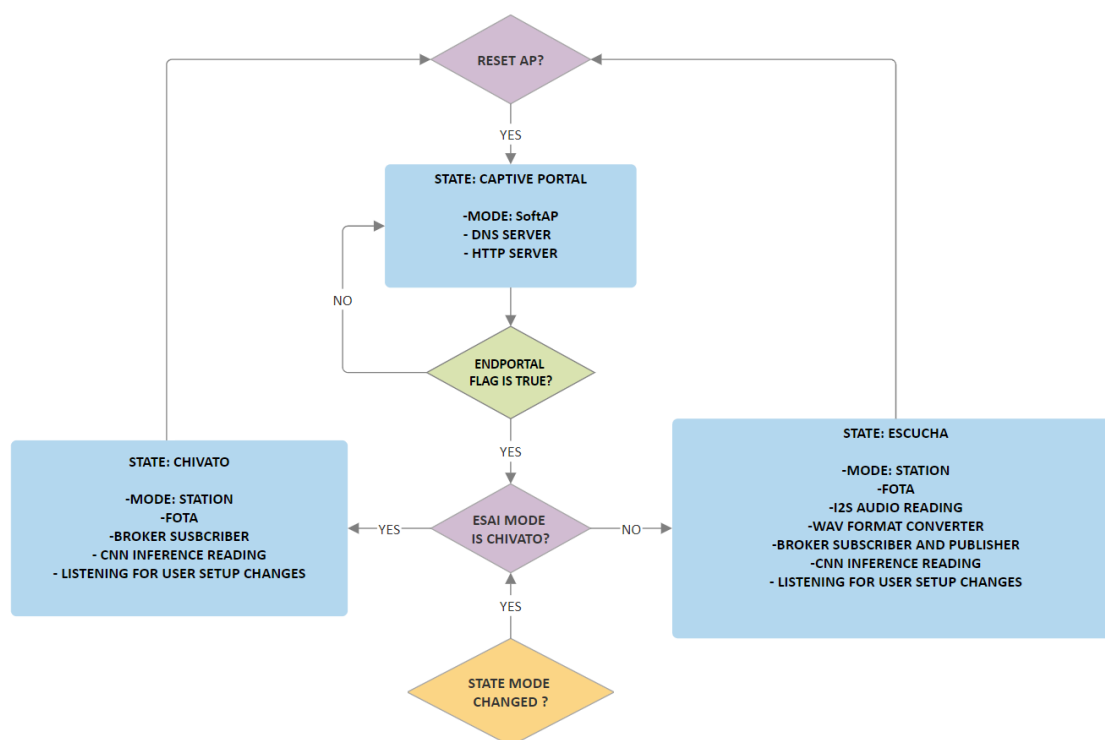


Figura 27. Diagrama de flujos del firmware de la ESAI. Creación del autor

3.1.1 Estado de portal captivo

Descripción general:

Estado inicial de la ESAI. En el modo portal captivo la estación operará como SoftAP ("Software-enabled Access Point"), o punto de acceso habilitado por software.

Paralelamente se configurará e iniciará un servidor DNS y un servidor web HTTP en el puerto 80. El servidor DNS redirigirá cualquier dirección entrante a la dirección IP de nuestro punto de acceso virtual (8.8.4.4).

El usuario deberá ver aparecer una nueva red WiFi llamada "ESAI_SSID". Al conectarse le saldrá una notificación de inicio de sesión o de acción requerida, o si el cliente es un Windows 10 con Chrome, aparecerá automáticamente una nueva ventana.

La primera web será para introducir los datos del punto de acceso al que la ESAI se conectará. Si las credenciales WiFi son correctas saldrá un mensaje de éxito con un enlace para seleccionar el modo de operación. La ESAI habrá entrado en el siguiente subestado y ya no se podrá volver a la página anterior, aunque se retroceda en el navegador. Tras pulsar en el enlace o recargar la página se verá un nuevo formulario con botones con los distintos modos de la ESAI (escucha/chivato y modo silencio) y los eventos a detectar. La última casilla será para buscar actualizaciones del firmware de la ESAI remotamente (FOTA).

Se procede a explicar las principales subrutinas que conforman este estado, a saber portalSetup, portal_captivo() y server handler:

-”portalSetup”: Configura el AP(dirección IP, SSID, clave, número de conexiones etc.) e inicia el servidor HTTP y DNS. Tiene como argumentos el puntero a la dirección del objeto de preferencias donde se almacenan los datos de la ESAI.

```
void portalSetup(Preferences* preferences_main){
    preferencias = preferences_main;
    Serial.println("Configurando el punto de acceso(AP)");
    WiFi.softAPConfig(apIP, apIP, netMsk);
    //Iniciamos el SoftAP. Con los argumentos (.1,0,1) transmitimos en el canal 1, hacemos visible la red y limitamos a 1 sola conexión.
    WiFi.softAP(softAP_SSID, softAP_PASS, 1, 0, 1); //Por defecto la contraseña está inicializada a nula "const char *passphrase = (const char *)_null"
    Serial.print("IP del punto de acceso: ");
    Serial.println(WiFi.softAPIP());
    //Si el DNSServer comienza con "*" para el nombre del dominio, responderá con la IP proveída(en el tercer argumento) a todas las peticiones DNS
    dnsServer.start(dnsPort, "*", apIP);
    //Iniciamos el servidor TCP(HTTP)
    server.begin(80);
}
```

- **"portal_captivo"**: Es la función principal. Se encarga de procesar nuevas peticiones del servidor DNS y buscar si hay clientes conectados. Al conectarse, almacenará la solicitud HTTP y llamará a la función `server_handler()` para que gestione la petición. La bandera `endPortal` hará terminar el bucle y finalizar los servidores.

```
void portal_captivo(){
    while(!endPortal){
        uint32_t tactual,tprev,timeout=2000;

        //El servidor DNS se ejecuta primero para transformar todas las direcciones IP entrantes a la de nuestro portal captivo
        dnsServer.processNextRequest();
        client=server.available(); //Verificar si un nuevo cliente se ha conectado.

        if(client){
            tactual=millis();
            tprev=tactual;
            String currentline="";
            while(client.connected() && (tactual-tprev<=timeout)){ //Esperar a que los datos del cliente estén disponibles.Timeout de 2 seg
                tactual=millis();
                if(client.available()){ //Si el cliente está disponible para transmitir almacenamos los bytes entrantes
                    char c= client.read();
                    headers+=c;
                    if(c=='\n'){
                        if(currentline.length()==0){ //Si el nuevo byte es '\n' tras acabar una línea entonces es el terminador de las cabeceras.
                            Serial.println("CLIENTE <:");
                            Serial.print(headers);
                            server_handler(); //Gestionar la petición GET o POST
                            break;
                        }
                        else{
                            currentline=""; //Encontrar un '\n' indica el fin de la línea. Por eso se reinicia.
                        }
                    }
                }
                else if(c!='\r')
                    currentline+=c; //Todos los bytes entrantes que no sean \n o \r se almacenan.
            }
            client.stop(); //Se cierra la conexión con el cliente.
            headers="";
        }
    }
    delay(7000); //En el ordenador no ocurre pero en el móvil la pestaña se cerrará automáticamente y no se podrá ver la plantilla de respuesta.
    server.end();
    dnsServer.stop();
    WiFi.mode(WIFI_MODE_STA);
    return;
}
```

- **"server_handler"**: Función: Gestionar la petición HTTP entrante. Distingue entre GET y POST. Tras procesar la petición transmite la respuesta HTTP con el código de estado y las plantillas HTML. La bandera `savedCredentials` permite avanzar al siguiente estado interno para configurar la ESAI. Primero mostrará la plantilla para la configuración del punto de acceso y una vez se hayan introducidos las credenciales válidas, se mostrará la página web con la setup de la ESAI. Cuando la bandera cambie de estado ya no se volverá a mostrar la web anterior (a menos que se reinicie el dispositivo). Todas las plantillas html de las respuestas HTTP se toman del archivo `"plantillasHTML.hpp"`. A continuación, se muestra el código comentado:

```

void server_handler(){
    if(headers.startsWith("GET")){
        if(!savedCredentials) respuesta = formulario_credenciales; //Dependiendo del estado de la bandera, ofrecemos distintas webs al cliente
        else respuesta = form_modos;
    }

    else if(headers.startsWith("POST")){
        //Obtener tamaño del cuerpo de la petición POST
        int16_t length_param = headers.indexOf("length:") + 7 ; //Obviamos la "L" por si viniese tanto en minúsculas como mayúsculas
        if(length_param==-1){ respuesta = err_400_1; return;} //Error 400. Missing Content-Length header
        int16_t length_param_end = headers.indexOf("\r",length_param);
        if(length_param_end==-1){ respuesta = err_400_2; return;} //Error 400. Malformed HTTP request
        String body_length_str = headers.substring(length_param,length_param_end); //Creamos un substring que empiece en "Length:" y termine en "\r"
        if(body_length_str.length()>6){ respuesta = err_413; return;} //payload too large

        uint16_t body_len= uint16_t(body_length_str.toInt()); //convertir el tamaño del body a un número entero
        String body="";
        while(body.length()<body_len){ //Leer los bytes enviados por el cliente mientras hasta completar el cuerpo.
            if(client.available()){
                char c=client.read(); //Alternativamente body += client.read();
                body+=c;
            }
        }

        Serial.println(body); //Imprimir por consola el cuerpo de la petición POST

        if(!savedCredentials) get_credentials(body); //Dependiendo de la bandera se llaman distintas funciones para procesar el body
        else{ get_modos(body); endPortal=true; } //Si se ha terminado de configurar la ESAI terminamos el portal captivo
    }

    else respuesta = err_404; //Sólo se admiten peticiones GET o POST

    client.println(respuesta); //Enviar respuesta con el código de estado y plantilla html al cliente
}

```

3.1.2 Estado de escucha/chivato

Descripción general:

Tras salir del estado de portal captivo, la ESAI configurará la interfaz I2S, WiFi y MQTT. Una vez se haya sincronizado el RTC procederá a intentar la conexión con el WiFi que fue configurado en el anterior estado. En caso de lograrlo, comprobará si hay actualizaciones OTA y comenzará la conexión con el broker. Cada intento de conexión tiene un timeout de 8 segundos. Si falla la conexión intentará reconectarse después de un segundo de espera. Si pasan 2 minutos y no se puede establecer la conexión la ESAI volverá al estado de portal captivo.

Si la ESAI ha sido configurada en modo escucha:

Una vez se tenga conexión con el broker la tarea de lectura I2S comenzará automáticamente. Cuando se tenga un paquete de audio procesado y convertido a formato "wav" la otra tarea en tiempo real saldrá de su estado de reposo y enviará el archivo al broker. Esta operación se realiza en paralelo a la de lectura para disminuir la latencia. Si hay un mensaje del broker entrante la ESAI realizará una operación u otra en función del tema ("topic"). Si el tema está relacionado con la inferencia del modelo YAMNet entonces se llamará a la función encargada de analizar la predicción del modelo en busca de los eventos con los que la ESAI ha sido configurada para detectar. Si el tema está relacionado con el modo de operación, la OTA o el punto de acceso se llamará a otra función que buscará y hará efectivo los nuevos cambios.

Si la ESAI ha sido configurada en modo chivato:

Igual que el estado anterior con la diferencia de que la estación ya no realiza la lectura I2S y el envío del paquete de datos. Sólo se limitará a “escuchar” los mensajes del broker asociados a la inferencia o cambios de configuración.

Opcional:

El siguiente código puede añadirse para el prototipo como funciones experimentales:

-Tarea adicional encargada de realizar el monitoreo de la batería mediante la lectura de un canal del ADC1 usando una atenuación de 6dB para obtener una mejor respuesta lineal entorno al rango de 0-1750 mV para el ESP32 ó 0-1600mV para el ESP32-S3. El procedimiento consiste en tomar al menos 64 muestras y calcular el valor promedio. Sería conveniente realizar la medida cuando el módulo RF no está transmitiendo, o idílicamente cuando estuviera apagado.

- Estado adicional de bajo consumo ([Ver consideraciones de ahorro energético](#)).

Módulos del firmware explicados:

I2S

La parte más importante y que requiere de más explicación es la estructura donde se realiza la configuración I2S. Primero se definen globalmente una serie de parámetros y variables para la interfaz I2S y el guardado de datos:

```
//Variables y parámetros para I2S
#define I2SPORT I2S_NUM_0
uint32_t SAMPLE_RATE = 16000;           //Tasa de muestreo
uint16_t samples = 12000;               //Nº de muestras
uint8_t headers_size=44;                //Nº de bytes de las cabeceras Wav
size_t buff_size = samples * sizeof(int32_t) + headers_size;
size_t buff16_size = samples * sizeof(int16_t) + headers_size;
int32_t *buff= (int32_t*)malloc(buff_size); //Asignación dinámica de memoria para el buffer donde se almacenará el slot entero
int16_t *buff16= (int16_t*)malloc(buff16_size); //Alojamiento de memoria para el buffer que almacenará los datos de audio
size_t bytes_read=0;
```

Tenemos el vector “buff” que guarda el canal de audio entrante de 32 bits. Por otro lado, tenemos buff16, que se deriva del primer buffer tras ser recortado y convertido para guardar el audio con una “resolución de 20 bits” en un vector de tipo int16. La función que realiza esto tiene precaución de evitar el desbordamiento de los valores máximos asociados a un entero de 16 bits para que no generen valores incongruentes y en su lugar saturar al valor máximo. La variable “headers_size” es el valor del tamaño en bytes de todas las cabeceras que se añaden al archivo para convertirlo a formato wav.

Para realizar la configuración de la comunicación I2S y del pinout se crea una estructura de tipo “i2s_config_t” e “i2s_pin_config_t” respectivamente:

```
void i2s_init(){
    //La configuración i2s depende de 2 estructuras de datos
    i2s_config_t i2s_config={
        .mode = i2s_mode_t (I2S_MODE_MASTER | I2S_MODE_RX ),
        .sample_rate = SAMPLE_RATE,
        .bits_per_sample = I2S_BITS_PER_SAMPLE_32BIT,
        .channel_format = I2S_CHANNEL_FMT_ONLY_RIGHT,
        .communication_format = I2S_COMM_FORMAT_STAND_I2S,
        .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
        .dma_buf_count = 16,
        .dma_buf_len = 64,
        .use_apll = false,
        .tx_desc_auto_clear = false,
        .mclk_multiple = I2S_MCLK_MULTIPLE_256,
        .bits_per_chan = I2S_BITS_PER_CHAN_32BIT
    };
    i2s_pin_config_t i2s_pinout={
        .bck_io_num = 32,
        .ws_io_num = 25,
        .data_out_num = -1,
        .data_in_num = 33,
    };
    i2s_driver_install(I2SPORT, &i2s_config,0,NULL);
    i2s_set_pin(I2SPORT,&i2s_pinout);
}
```

Es recomendable ver el apartado del anexo “Incidencias con el protocolo I2S” antes de continuar.

-La variable “dma_buf_len” no es el tamaño real del buffer DMA sino del frame. El frame representa los datos de los dos canales en un ciclo de WS, es decir equivale al número total de muestras, en el caso de que el número de bits y el ancho de canal coincidan. El tamaño real del buffer DMA será:

$$\text{dma_buf_size} = \text{dma_buf_len} * \text{chan_num} * \text{bits_per_chan} / 8.$$

En el análisis DMA vamos a profundizar más en el motivo de elección de los valores de buf_len y buf_count.

-“use_apll”: Si lo activamos podemos usar el APLL como reloj principal y lograr más precisión (pero también más consumo).

-“mclk_multiple”: Indicar el múltiplo del reloj MCLK. Para 32 o 16 bits su valor será unos 256 la frecuencia del reloj BCLK. El reloj MCLK sirve opcionalmente como referencia para sincronizar los esclavos.

-“bits_per_chan”: Representa el ancho de canal o slot. Como se ve en el anexo, estamos casi forzados a que sea siempre igual que sea de 32 bits.

La lectura I2S se realiza mediante una tarea en tiempo real ejecutada en el núcleo 0:

```

void i2sReadTask(void *pvparameter){
    for(;;){
        vTaskDelay( 1 / portTICK_PERIOD_MS);
        while(!mqttclient.connected())vTaskDelay(1 / portTICK_RATE_MS); //Esto impide que la tarea de lectura comience mientras no haya cone
        tsi=millis();

        esp_err_t i2s_err= i2s_read(I2SPORT, buff + headers_size/sizeof(int32_t), samples*sizeof(int32_t), &bytes_read, portMAX_DELAY); //Ex
        if(i2s_err != ESP_OK) {
            Serial.print("\nSe ha producido un fallo desconocido durante la comunicación i2s.\n");
            vTaskDelay(2000 / portTICK_RATE_MS);
        }
        tsf=millis();
        //Convertimos y adaptamos el buffer de 32 con los datos de audio para el de 16 bits que se enviará vía MQTT
        audioDataCorrector(buff,buff16,buff_size);
        publishAllowed=true;
        //Serial.printf("t_i2s=%u ms\n", (tsf-tsi));
    }
}

```

La tarea estará en espera mientras no haya conexión con el bróker. Los datos I2S se almacenan en el buffer de `int32_t` y posteriormente se procesa para adaptarlo a un array tipo `int16_t`. Esta operación se detalla también con más detalle en el anexo.

-Análisis de la configuración de la DMA (“Direct Memory Access”)

El acceso directo a la memoria permite a los periféricos acceder a la memoria del sistema sin involucrar a la CPU, la cual, sólo se encargará de iniciar la transferencia entre la memoria y el periférico. El resto del tiempo el controlador DMA se encargará de gestionar el intercambio de datos y la CPU estará disponible para realizar otras tareas. Una vez se completa la transferencia DMA la CPU recibe una interrupción y podrá gestionar el archivo de audio obtenido. Teniendo en cuenta cómo funciona, hay que prestar atención en el tamaño de nuestro buffer DMA y el número de buffers que se utilizarán.

Si el tamaño del buffer es pequeño, la CPU recibirá interrupciones con más frecuencia, y por lo tanto se verá más comprometido el desarrollo de otras tareas y cuanto mayor sea, menor será la frecuencia con que se interrumpe la CPU. El tamaño del buffer se establece al inicializar el valor de “`dma_buf_len`” en la estructura “`i2s_config_t`”.

Si jugamos a poner valores muy pequeños o grandes veremos que la consola nos devolverá el siguiente mensaje:

```

i2s.c:1073 (i2s_driver_install):I2S buffer length at most 1024 and more than 8
Guru Meditation Error: Core  1 panic'ed (LoadProhibited). Exception was unhandled.
Core 1 register dump:
PC      : 0x400d6701  PS      : 0x00060830  A0      : 0x800d14f8  A1      : 0x3ffb1ec0
A2      : 0x00000000  A3      : 0x3ffb850c  A4      : 0xffffffff  A5      : 0x00000000
A6      : 0x3f404434  A7      : 0x3f4012d8  A8      : 0x80105038  A9      : 0x3ffb1e70
A10     : 0x40143360  A11     : 0x3ffc456c  A12     : 0x3f401344  A13     : 0x00000010
A14     : 0x00000003  A15     : 0x00000001  SAR     : 0x00000004  EXCCAUSE: 0x0000001c
EXCVADDR: 0x0000002c  LBEG    : 0x4008bdad  LEND    : 0x4008bdbd  LCOUNT   : 0xffffffff

```

Figura 28. Desbordamiento del buffer DMA. [30]

Por lo tanto, los valores mínimos y máximos de muestras serán de 8 y 1024 respectivamente. Para una frecuencia de muestreo de 44,1 KHz. El tiempo entre interrupciones t_r será:

$$t_r = \frac{8 \text{ muestras}}{44,1 \text{ KHz}} \approx 181 \mu s$$

Para 1024 muestras:

$$t_r = \frac{1024 \text{ muestras}}{44,1 \text{ KHz}} \approx 23,22 \text{ ms}$$

A priori parece que deberíamos de usar el mayor tamaño de buffer posible para que la CPU sea interrumpida lo menos posible sin embargo un mayor tamaño de buffer implicaría una mayor latencia. Los datos transmitidos no serán procesados hasta que se complete la transferencia y se envíe la interrupción. Es decir, para el caso anterior de 1024 muestras, tendríamos un retardo de 23,22 ms desde que el sonido es captado hasta que pueda procesarse. Teniendo al menos 2 buffers, podemos mejorar la latencia ya que mientras el controlador DMA se encarga de almacenar en memoria los datos de entrada en un buffer, la CPU estará procesando el otro buffer ya completo como se ve en la figura 29. Por lo tanto, podemos concluir que la mejor solución pasa por crear varios buffers de tamaño medio, en vez de uno de un tamaño muy grande o muchos de tamaño reducido.

Experimentalmente valores de "dma_buf_count" de 16-8 y "dma_buf_len" de 64-128 ha dado buenos resultados. El tiempo entre interrupciones o latencia sería:

$$t_r = \frac{64 \text{ muestras}}{16 \text{ KHz}} = 4 \text{ ms}$$

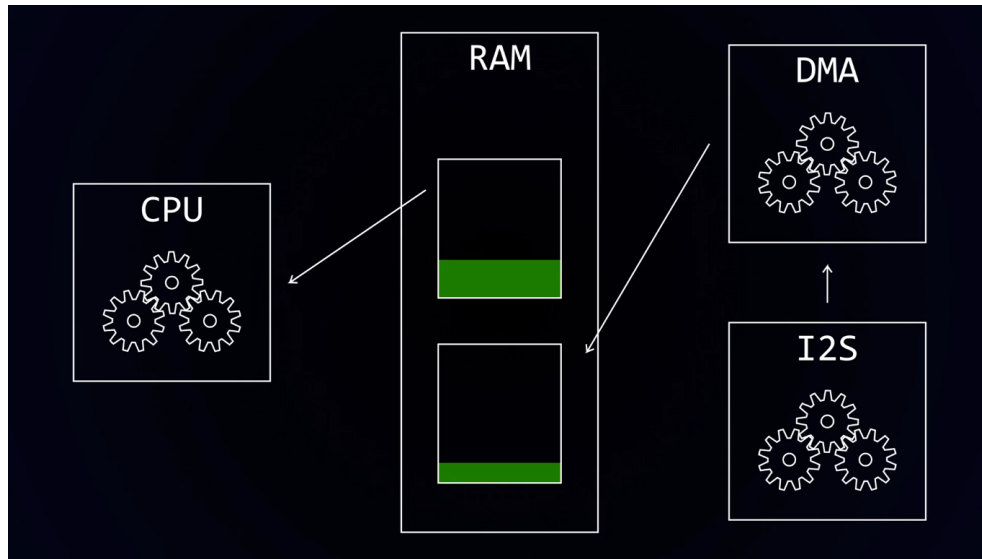


Figura 29. Representación gráfica de la distribución del procesamiento y almacenamiento entre varios buffers. [30]

WIFI

La estación intentará conectarse automáticamente a la red que ha sido guardada en la memoria NVS ("Non Volatil Storage") mediante la función "WiFiConnect". La subrutina "WiFiEvent" se encargará de las gestiones a realizar cuando se produzcan algún evento Wifi como la conexión y desconexión con la red. Al producirse la conexión, intentará realizar una FOTA en caso de que la variable NVS asociada sea cierta (es decir el usuario haya seleccionado la casilla del formulario web que ofrece el servidor Django). En caso de que no haya continuará y se intentará conectar al bróker. Ante el evento de pérdida de conexión con la red, se activará el timer para la reconexión WiFi.

En total existen 2 timers de 1000 ms con sus respectivas llamadas a las funciones de callback al producirse el evento "match". El timer de reconexión WiFi llama a la función WiFiConnect para intentar conectarse de nuevo a la red guardada cada 1 segundo.

Reiniciar AP:

Como mecanismo de seguridad, la estación no podrá estar en el estado de escucha o chivato por más de 2 minutos si no tiene conexión con la red configurada. Durante este tiempo, mediante el timer de reconexión wifi, la ESAI intentará reconectarse a la red guardada. Pasado más de 2 minutos si no se logró una conexión la ESAI asumirá que el punto de acceso fue introducido de forma incorrecta, o bien que la red no existe en este momento y volverá al estado de portal captivo para que se pueda configurar una nueva red.

RTC

```
//Parámetros para la sincronización del RTC vía NTP
uint16_t gmtoffset=3600; //España es UTC+1
uint16_t daylightoffset=0; //Horario de invierno. En verano habría que añadirle una hora(expresada en s).
const char* ntpServer = "pool.ntp.org";
//Ajustar el reloj interno del ESP32 a la hora UTC+1 horario de invierno
configTime(gmtoffset,daylightoffset,ntpServer);
```

Es de gran importancia mantener el reloj en tiempo real o RTC (“Real Time Clock”) del ESP32 sincronizado en todo momento. Si estuviera desfasado, el “handshake” realizado durante la creación del canal cifrado con el bróker o el servidor, podría fallar. Para el ajuste se utiliza el protocolo NTP (“Network Time Protocol”). Es un protocolo de Internet para sincronizar los relojes de los sistemas informáticos a través del enrutamiento de paquetes. NTP utiliza UDP en la capa de transporte, usando el puerto 123. Los servidores que utilizan este protocolo para ofrecer sincronización a los dispositivos se calibran y obtienen sus datos directamente de los relojes atómicos.

FOTA

La FOTA proviene del inglés “Firmware Update Over The Air”. Es posiblemente uno de los inventos más útiles y que más dinero ha ahorrado a los desarrolladores de los últimos tiempos. Antiguamente, tras desarrollar y lanzar a producción y venta un producto había que asegurarse de que el firmware funcionase perfectamente porque de lo contrario supondría perder toda la línea de producción ya distribuida y vendida y volver a sacar el dispositivo corregido. Los hechos demuestran que casi nunca un dispositivo que tenga un firmware mínimamente complejo ha funcionado a lo largo del tiempo sin que ocurran nuevos eventos imprevistos que le hayan llevado a fallos y un mal funcionamiento. La FOTA literalmente puede salvar la “vida” a los dispositivos que necesiten una reparación de algún error, siempre y cuando tengan conexión con a un punto de acceso conectado a internet. También se puede utilizar para añadir nuevas funcionalidades y mejoras.

En nuestro proyecto, la ESAI hará uso de la biblioteca <HttpsOTAUpdate.h> para efectuar la OTA. La librería utiliza el protocolo HTTPS para hacer el procedimiento de forma segura y evitar ataques externos para corromper el funcionamiento del dispositivo. La solicitud HTTP va dirigida al Servidor Django, el cual responde con el firmware almacenado en formato binario. El ESP32 deberá contar con el certificado raíz CA que firmó el certificado del servidor para que funcione el proceso. Dentro del esquema de partición de la memoria FLASH, se reserva un espacio para la OTA, donde se guarda los datos binarios del nuevo código. Una vez terminada la transferencia lo se reiniciará. Esta vez la dirección de memoria donde se almacena el programa a cargar no será la habitual, sino que será la dirección del segmento de memoria de la OTA.

MQTT

No requiere presentación. Principales subrutinas:

-“**mqtt_init**” : Establece las funciones de callback, el ID del ESP32 y el certificado, dirección y puerto del servidor. Se crea la tarea en tiempo real “mqttLoopTask” que a su vez llama a otra tarea (MqttClient::loop) para controlar el flujo de comunicaciones MQTT. Esta tarea se suspende y reanuda automáticamente cuando el esp32 se desconecte o reconecte al broker.

-Funciones **callback** explicadas:

```
//Realizar conexión con el servidor. Si falla y hay conexión a la red reintentar la operación cada segundo.
void mqttConnect(){
    if(!mqttclient.connect()){
        Serial.println("Falló la conexión con el broker.Verifica las credenciales,conectividad a la red y el estado del broker.");
        if(WiFi.status()==WL_CONNECTED){
            xTimerStart(mqttReconnectTimer,0);
        }
    }
}

//Evento de conexión con broker. Si la tarea de lectura i2s estaba suspendida, reanudarla. Suscribirse al tema de la inferencia de la red neuronal
//y al tema con cambios en la configuración de la ESAI y/o petición de FOTA.
void mqttOnConnect(bool sessionPresent){
    Serial.println("Conexión con el broker realizada con éxito.");
    mqttclient.subscribe("cnn",2);
    mqttclient.subscribe("ESAI_1_CONF",2);
    if(i2s_task_on)vTaskResume(i2sreadTaskHandler);
}

//Evento de desconexión con broker. Suspender tarea de lectura i2s. Si hay conexión a la red reintentar conexión con broker cada segundo.
void mqttOnDisconnect(espMqttClientTypes::DisconnectReason reason){
    Serial.print("Se ha perdido la conexión con el broker.Reintentando conexión. ");
    i2s_task_on = false;
    vTaskSuspend(i2sreadTaskHandler);
    if(WiFi.status()==WL_CONNECTED) xTimerStart(mqttReconnectTimer,0);
}

//Almacenar mensaje en una cadena. En función del topic se llamará a la función de detección de eventos o la de cambiar la conf de la ESAI.
void onMqttMessage(const espMqttClientTypes::MessageProperties& properties, const char* topic, const uint8_t* payload, size_t len, size_t index, size_t total){
    String mensaje="";
    for(uint32_t i=0; i<len; i++){
        mensaje += (char) payload[i];
    }
    if(String(topic) == "cnn") process_cnn_prediction(mensaje);
    else if(String(topic) == "ESAI_1_CONF") change_events(mensaje);
}
}
```

Nota: La función más interesante posiblemente sea la de “onMqttMessage”. Es la que responde con la llamada a dos funciones de gran importancia en función del “topic” al que pertenece el mensaje. Una de ellas permitirá la detección de los eventos(“process_cnn_prediction”) y la otra es la encargada de hacer efectivos los nuevos cambios de modo de operación de la ESAI(“change_events”).

-“**publishTask**”: Finalmente tenemos la tarea principal en tiempo real ejecutada en el núcleo 1:

```
//Envío de paquete de audio convertido a formato wav. El archivo se envía como una array de bytes. La tarea se ejecutará si hay conexión con el
//broker y si la tarea i2sReadTask cambia la bandera publishAllowed a true.
void publishTask(void *pvparameter){
    for(;;){
        if(mqttclient.connected() && publishAllowed){
            ti=micros();
            tf=tf;
            addWavHeaders((uint8_t*)buff16, samples*sizeof(int16_t)); //Añadir cabeceras WAV
            mqttclient.publish("django1",2,false,(const uint8_t*)buff16, buff16_size); //Enviar archivo de audio a Django a través del broker
            publishAllowed=false;
            tf=micros();
            Serial.printf("T_mqtt: %u us\n",(tf-ti));
        }
        vTaskDelay( 1 / portTICK_PERIOD_MS);
    }
}
```

Convertidor de formato WAV (WAVE)

El objetivo es convertir una serie de datos de audio en bruto sin formato a un archivo de audio con formato sin compresión, que pueda ser reproducido por cualquier reproductor. Estos datos proceden del muestreo y cuantificación del ADC del MEMS y que han sido posteriormente transmitidos por I2S a nuestro ESP32. Como hemos visto ya, antes de enviar el audio lo único que tenemos que hacer para que tenga extensión WAVE es añadirle una serie de cabeceras que consisten en metadatos que ofrecen información técnica para que el reproductor sepa cómo tiene que interpretar el archivo exactamente.

El formato de archivo WAVE es un subconjunto de la especificación RIFF de Microsoft. Un archivo RIFF comienza con un encabezado de archivo ("file header") seguido por una secuencia de bloques de datos ("data chunks"). La estructura más típica de un archivo WAV es la de un archivo RIFF con un único chunk "WAVE" que consta de dos sub-chunks: un subchunk "fmt" que especifica el formato de los datos y otro llamado "data" que contiene los datos muestreados. A esta forma se le denomina "forma canónica" y es la que se va a utilizar.

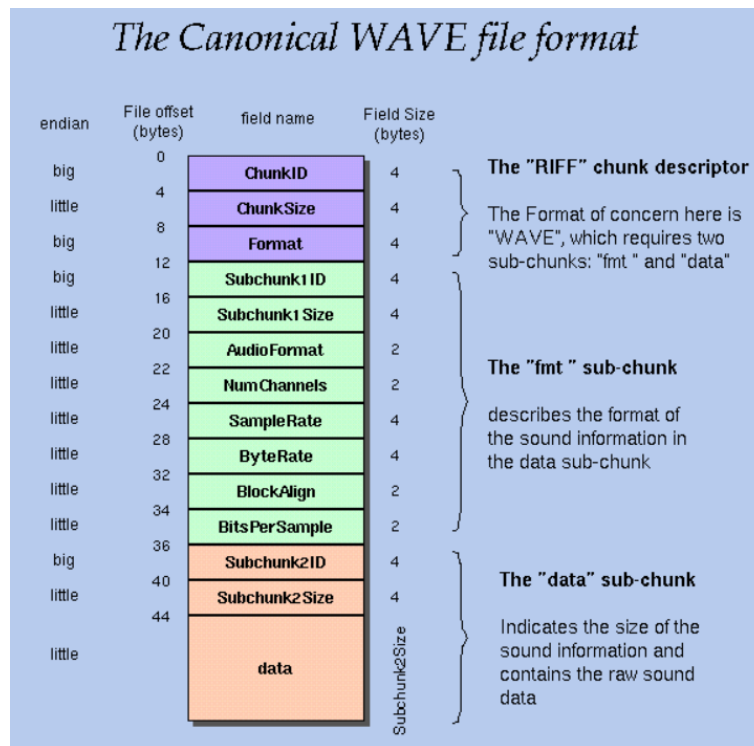


Figura 30. Forma canónica del formato WAVE. [31]

Lo único que debemos hacer es crear una función que tome como argumento nuestro buffer de datos de audio junto con su tamaño ("Subchunk2Size") y vaya construyendo byte a byte las cabeceras WAV. Hay que prestar especial atención al intercalado de

formato Little endian con big endian. La función “addWavHeaders” queda de la siguiente forma:

```
void addWavHeaders(uint8_t *headers, uint32_t SubChunk2Size){
    headers[0]='R'; //4 ChunkID
    headers[1]='I';
    headers[2]='F';
    headers[3]='F';
    uint32_t chunkSize= 36 + SubChunk2Size;
    headers[4]=(uint8_t)(chunkSize & 0xFF); //4 ChunkSize
    headers[5]=(uint8_t)((chunkSize>>8) & 0xFF);
    headers[6]=(uint8_t)((chunkSize>>16) & 0xFF);
    headers[7]=(uint8_t)((chunkSize>>24) & 0xFF);
    headers[8]='W'; //4 FORMAT
    headers[9]='A';
    headers[10]='V';
    headers[11]='E';
    headers[12]='f'; //4 SubChunk1ID
    headers[13]='m';
    headers[14]='t';
    headers[15]=' ';
    headers[16]=0x10; //4 Subchunk1Size PCM=16
    headers[17]=0x00;
    headers[18]=0x00;
    headers[19]=0x00;
    headers[20]=0x01; //2 AudioFormat PCM=1
    headers[21]=0x00;
    headers[22]=0x01; //2 NumChannels
    headers[23]=0x00;
    headers[24]=SAMPLE_RATE & 0xFF; //4 SampleRate
    headers[25]=(SAMPLE_RATE>>8) & 0xFF;
    headers[26]=(SAMPLE_RATE>>16) & 0xFF;
    headers[27]=(SAMPLE_RATE>>24) & 0xFF;
    uint32_t byteRate= SAMPLE_RATE * 1U * sizeof(int16_t); //4 ByteRate=SampleRate * NumChannels * BitsPerSample/8
    headers[28]=byteRate & 0xFF;
    headers[29]=(byteRate>>8) & 0xFF;
    headers[30]=(byteRate>>16) & 0xFF;
    headers[31]=(byteRate>>24) & 0xFF;
    headers[32]=0x02; // 2 BlockAlign== NumChannels * BitsPerSample/8
    headers[33]=0x00;
    headers[34]=0x10; //2 Bits per sample
    headers[35]=0x00;
    headers[36]='d'; //4 Subchunk2ID
    headers[37]='a';
    headers[38]='t';
    headers[39]='a';
    headers[40]= SubChunk2Size & 0xFF; //4 SubChunk2Size== NumSamples * NumChannels * BitsPerSample/8
    headers[41]= (SubChunk2Size>>8) & 0xFF; //En nuestro caso SubChunk2Size es argumento de la función
    headers[42]= (SubChunk2Size>>16) & 0xFF;
    headers[43]= (SubChunk2Size>>24) & 0xFF;
}
```

Procesamiento de la Inferencia. Detección de eventos.

Tarea imprescindible del firmware de la ESAI. Es la parte que permite que la estación pueda responder ante los eventos capturados por el módulo acústico. Primero vamos a ver la función “process_cnn_prediction” y la subrutina asociada “get_event” y después se dará una breve explicación:

```
//Función detectora de eventos
void process_cnn_prediction(const String& pred){
    if(preferences.getBool("silencio")) return; //Si está en modo silencio salimos.

    bool glass = preferences.getBool("glass");
    bool baby = preferences.getBool("baby");
    bool birds = preferences.getBool("birds");
    bool human = preferences.getBool("human");

    //Buscamos si en la inferencia del modelo se encuentran los eventos de interés y si es así activamos la salida para alarma
    if(glass && ( get_class(pred,"glass") || get_class(pred,"Chink, clink") || get_class(pred,"Breaking") || get_class(pred,"Tools") || get_class(pred,"Shatter") || get
    else if(birds && ( get_class(pred,"Bird") || get_class(pred,"Bird vocalization") )) digitalWrite(LED_PIN,HIGH);
    else if(human && ( get_class(pred,"Speech") || get_class(pred,"Whistling") || get_class(pred,"Wheeze") || get_class(pred,"Cough") || get_class(pred,"Walk, footsteps
    else if (baby && ( get_class(pred,"Baby cry, infant cry") || get_class(pred,"Crying, sobbing") || get_class(pred,"Wail, moan") )) digitalWrite(LED_PIN,HIGH);
    else digitalWrite(LED_PIN,LOW);
}

bool get_class(const String& pred, const String &CNNclass){
    int16_t Index = pred.indexOf(CNNclass); //La función devuelve -1 si no encuentra el string a buscar.
    if(Index==-1) return false;
    //Adquirir valor(score) asociado al evento y establecer un umbral para descartar ruido. La inferencia tiene la siguiente estructura: (n, 'class', score(float)).
    //Entre el fin del nombre del evento y la puntuación asociada van 3 caracteres(', ')
    int16_t posInicio = Index + CNNclass.length() + 3 /*, */;
    int16_t finalPos = pred.indexOf(" ",posInicio);
    String scoreStr = pred.substring(posInicio,finalPos);
    float score = scoreStr.toFloat();
    //Definir umbral y comprobar si el evento lo supera
    float umbral = 0.15;
    if(score > umbral) return true;
    else return false;
}
```

- “process_cnn_prediction”: Función que conmuta el valor del actuador de salida en función del evento que haya ocurrido y la configuración de los modos de la ESAI. Si el modo silencio está activado directamente sale de la función. Carga las variables NVS booleanas para almacenar su estado actual. Estas variables funcionan como interruptor para el siguiente paso. A continuación, hay una estructura condicional que llamará a la función “get_class” para buscar la clase especificada en el argumento dentro del mensaje con el resultado de la inferencia de la CNN Yamnet. Nótese que la primera condición es que la bandera de evento esté activa. Si se encuentra el evento la salida se activa (una alarma acústica, led etc).

- “get_class”: Encargada de buscar en el mensaje de la inferencia (predicción) de YAMNet la clase especificada en el segundo argumento. Después extrae la puntuación asociada (“score”) y la convierte a un número decimal. Si la puntuación supera un cierto umbral definido entonces la función devolverá TRUE.

Nota: No confundir los eventos de la ESAI con las clases. Una clase es uno de los 521 sonidos que puede inferir el modelo CNN YAMNet. Los eventos de la ESAI están formados por la agrupación de varias de estas clases. En la mayor parte de los casos basta con que se dé una de las clases del conjunto para que se active la detección de un evento.

3.2 DJANGO

Descripción general:

El servidor Django operará como cliente subscriptor/publicador del bróker mosquitto y a su vez ofrecerá un servicio web para mejorar la experiencia del usuario y poder comunicarse con las estaciones. Django deberá obtener los archivos de audio de las estaciones, procesarlos e introducirlos en el modelo YAMNet. La predicción resultante se enviará de vuelta a la estación o al par de estaciones escucha/chivato vinculadas.

Paralelamente el servidor Django dispondrá de un servicio web al usuario para que pueda acceder a los resultados de la inferencia del modelo de red neuronal y poder escuchar el sonido captado en tiempo real. También se permitirá al usuario volver a configurar de nuevo el modo de operación, los eventos a detectar o el punto de acceso y realizar una actualización de firmware remota (FOTA), si estuviera disponible. Como funcionalidad opcional Django puede ser capaz de recibir un archivo de audio segmentado en dos o más paquetes y almacenarlo como un único archivo. Otras características en fase experimental y no implementadas son: mostrar el estado de las baterías de las ESAIs, en caso de contar con el prototipo diseñado (ver siguiente sección “Diseño Electrónico”); mostrar el espectrograma en tiempo real; página web que permita buscar, por fecha y por ID de ESAI, el archivo de audio específico almacenado en el servidor. Aunque estas características son añadidos no relevantes para el funcionamiento del sistema, pueden mejorar aún más la experiencia.

Importaciones necesarias explicadas:

```
#Respuestas HTTP y envío de contexto para vistas y plantillas html
from django.shortcuts import render
from django.http import HttpResponse
#csrf_exempt es para poder evitar el certificado de seguridad csrf
from django.views.decorators.csrf import csrf_exempt
#Mqtt y SSL/TLS
import ssl
import paho.mqtt.client as mqtt                    #Librería MQTT síncrona más usada
from django.conf import settings                    #Importar datos del broker
# Para poder trabajar en modo multihilo y crear retardos
import threading
import time
#Librerías para YAMNet y trabajar con arrays:
import tensorflow as tf                            #Biblioteca para desarrollo de Redes Neuronales
import tensorflow_hub as hub                        #Biblioteca con el repositorio de modelos
import numpy as np                                  #Biblioteca para vectores y matrices
import csv                                          #Librería para leer archivos .csv
#Para escribir y leer datos en nuestro sistema
import os
#Librerías para extraer los datos de la señal de sonido de un archivo wav
import scipy
from scipy.io import wavfile

#Cargar modelo YAMNet
model = hub.load('https://tfhub.dev/google/yamnet/1')
```

Procesamiento del audio y del modelo YAMNet:

Esta tarea es realizada por 3 funciones:

-**"class_names_csvToArray"**: Como su nombre indica lee el mapa de clases en formato "csv", extrae la columna con el nombre de todas las clases y la devuelve como vector numpy. El archivo csv ("Comma-separated values") está formado por 3 columnas. La primera es un índice numérico, la segunda un identificador multimedia("mid"), que asocia cada clase con su correspondiente identificador usado por AudioSet y finalmente la tercera columna es el nombre de la clase.

-**"ensure_sample_rate"**: En caso de que la señal de entrada haya sido muestreada a una SR superior a 16 KHz se remuestrea para que se corresponda con una señal de 16KHz.

-**"procesar_audio"**: Lee el archivo wav almacenado en el sistema, extrae los datos de audio (se eliminan las cabeceras pero se analizan los metadatos) y se introduce en como entrada al modelo YAMNet que ha sido previamente cargado. Internamente se realiza el espectrograma (o los espectrogramas en caso de que el audio entrante sea mayor) para el frame de 960 ms. La función del modelo devuelve las puntuaciones de salida de la inferencia, las "embeddings" y el espectrograma. Aunque sólo se hace uso del array "scores" podría usarse como añadido visual el espectrograma, aunque no es relevante para nuestra aplicación. Se seleccionan las N mayores puntuaciones inferidas, se ordenan y se envían junto con el nombre de la clase asociado. Esto será lo que se envía al bróker y lo que en última instancia recibirá el ESP32 y que analizará con la función vista en el anterior apartado. A continuación, se muestran las 3 funciones con explicaciones:

```
def class_names_csvToArray(class_map_path):
    if tf.io.is_tensor(class_map_path):
        class_map_path_array=class_map_path.numpy()
    class_names=[]
    with tf.io.gfile.GFile(class_map_path_array) as csvfile:
        reader=csv.DictReader(csvfile)
        for row in reader:
            class_names.append(row['display_name'])
    return class_names

#En caso de que la señal de entrada haya sido muestreada a una SR superior a 16KHz
def ensure_sample_rate(original_sample_rate, waveform):
    desired_sample_rate=16000
    if original_sample_rate != desired_sample_rate:
        desired_length = int(round( float(len(waveform))*(desired_sample_rate/original_sample_rate) )) #La longitud del audio(en muestras) por tasa_deseada/tasa_original n
        waveform = scipy.signal.resample(waveform, desired_length)
    return waveform

def procesar_audio(audio_path):
    class_names=class_names_csvToArray(model.class_map_path()) #Convertir archivo .csv en un array con la columna de las clases
    sample_rate,wavedata=wavfile.read(audio_path) #Se extraen los datos de audio del archivo y se eliminan las cabeceras.
    wavedata16k=ensure_sample_rate(sample_rate,wavedata) #wavfile.read también devuelve la SR el archivo. Aseguramos que sea 16KHz
    waveform_norm=wavedata16k/tf.int16.max #YAMNet requiere una entrada de vector 1-D normalizado.
    scores,embeddings,espectrograma=model(waveform_norm) #Inferencia del modelo.Interiormente se realiza el espectrograma log-mel.
    scores=scores.numpy() #Scores devuelve todas las puntuaciones por cada ventanas de tiempo del espectrograma.Se debe visualizar
    scores_avg = np.mean(scores,axis=0) #como una matriz donde filas son las STFT y las columnas cada una de las clases. Axis=0 -> columnas.
    top_N=4
    top_scores=np.argsort(scores_avg)[::-1][:top_N] #argsort ordena de menor a mayor los valores del array argumento. Devuelve los INDICES del array asociado
    results=[] #a esos valores."::-1" es para invertir el orden(ahora de mayor a menor).Se toman los primeros N valores.
    for n in range(top_N):
        score_index=top_scores[n] #Indice del vector score asociado a la n de las N mayores puntuaciones de dicho vector.
        score_name=class_names[score_index] #Nombre de la clase asociado al indice anterior.
        score_value=scores_avg[score_index] #Valor de la score asociado al índice anterior.
        results.append((n,score_name,score_value)) #Array ordenado numéricamente con las mejores N scores y su valor. Será lo que la ESAI reciba.
    return results
```

MQTT

De forma similar al firmware del ESP32, tenemos una función de “setup”, las funciones callbacks y la función para mantener el flujo de comunicación mqtt de forma indefinida (mqtt_loop):

-**“single_packet”**: Guarda el archivo de audio byte a byte en una carpeta del proyecto; realiza la inferencia con las N primeras clases y su puntuación asociada; almacena la última inferencia de forma global para su uso por otras vistas; publica la inferencia al broker.

-**“on_connect”**: Indica que se ha realizado la conexión y se suscribe al tema “django1”.

-**“on_disconnect”**: Se indica que se ha perdido la conexión e intenta reconectarse.

-**“on_message”**: Convierte la “payload” del mensaje en un array de tipo int16 (tal como es el buffer original de las muestras de audio) y se introduce en la función “single_packet”. Se añade la posibilidad de discernir entre temas en caso de que se añada en el prototipo experimental el envío de los datos del monitor de batería.

-**“mqtt_init”**: Crea el objeto “mqtt.Client”; configura la dirección y puerto del bróker; asigna las funciones callback definidas previamente; establece la conexión con el broker; devuelve el objeto mqtt.Client para que sea utilizado por las demás funciones de “views.py”.

-**“mqtt_loop”**: Función ejecutada en un hilo paralelo al resto de funciones. Controla el flujo de mensajes entre el bróker y los clientes. Cuenta con un retardo del orden de milisegundos para permitir que el servidor Django atienda a las otras tareas. De lo contrario Django sólo podrá atender a la comunicación MQTT y no podrá trabajar como servidor web.

Funciones para ofrecer servicio web:

Las direcciones URL de todas las funciones estarán formadas por la dirección base del servidor, “<https://192.168.x.x:443>”, más la dirección del recurso específico que siempre será el nombre que tenga dicha función. Por ejemplo, si la función (o vista) de views.py se llama “vista1” la dirección para acceder a dicho recurso será: <https://192.168.x.x/vista1>. Nótese que se ha omitido el puerto ya que 443 es el puerto usado por defecto por el protocolo HTTPS.

-**“audio_web”**: Esta vista retorna como respuesta HTTP la plantilla HTML “audio_player.html”. Esta plantilla es capaz de presentar de forma dinámica la reproducción del último audio enviado por la ESAI y la inferencia, logrado mediante un bloque de código JavaScript que utiliza la técnica Ajax (“Asynchronous JavaScript and XML”). La principal característica de Ajax es la capacidad de realizar solicitudes asíncronas al servidor web que afecten a bloques específicos. Esto permite que la aplicación web actualice una parte de la página web, como el reproductor de audio o el

texto con la predicción del modelo, sin tener que recargar la página completa. Esto se lleva a cabo de forma separada por dos funciones JavaScript (“updateAudio” y “update_inference”) que realizan de forma asíncrona la petición HTTP a las funciones de “views.py” “get_last_audio” y “get_last_inference” para actualizar el estado del reproductor de audio y de la inferencia respectivamente.

El tiempo entre solicitudes HTTP viene dado por la función “setInterval” de JS, la cual pasado el tiempo especificado en milisegundos llamará automáticamente a la función introducida en el primer argumento (get_last_audio o get_last_inference por ejemplo). En ambos casos el tiempo se ha establecido en 765 ms, correspondiente al tiempo aproximado entre los 750 ms de duración del archivo de audio más los tiempos de envío del ESP32 al bróker y del bróker al servidor, más el posterior almacenamiento y procesamiento por el modelo. De todos los retardos anteriores el más restrictivo será el que pongamos en la función “mqtt_loop”. Sin embargo hay que tener en cuenta el tiempo que se toma la función AJAX en realizar la petición y obtener la respuesta del servidor.

-**“conf_form”**: Esta vista retorna como respuesta HTTP la plantilla HTML “conf_form.html”. En esta plantilla encontramos un formulario muy similar al que ofrecía el portal captivo del ESP32 cuando operaba como punto de acceso virtual para configurar la ESAI, con la diferencia de que aparecerá la opción de establecer un nuevo AP. Este formulario permite diferenciar quien será el objetivo de los nuevos cambios de configuración. Actualmente permite sólo elegir entre ESAI1 y ESAI2, asociado a las dos placas basadas en el ESP32 que se utilizan en el desarrollo del proyecto. La función “newConfPublish” recibirá la petición POST con los parámetros del formulario, extraerá el cuerpo de la petición y lo publicará en el bróker, en el tema correspondiente (ESAI1_CONF, ESAI2_CONF o en ambos).

FOTA:

-Si en el formulario anterior se añadió buscar actualizaciones, el ESP32 se reiniciará y comenzará el proceso de actualización OTA HTTPS. El host de destino es otro recurso de Django, la función de views llamada “get_OTA_binaries”, que lo que hace es devolver como respuesta HTTP el archivo binario con el firmware del ESP32.

4 DISEÑO ELECTRÓNICO

4.1 Diagrama de bloques del sistema

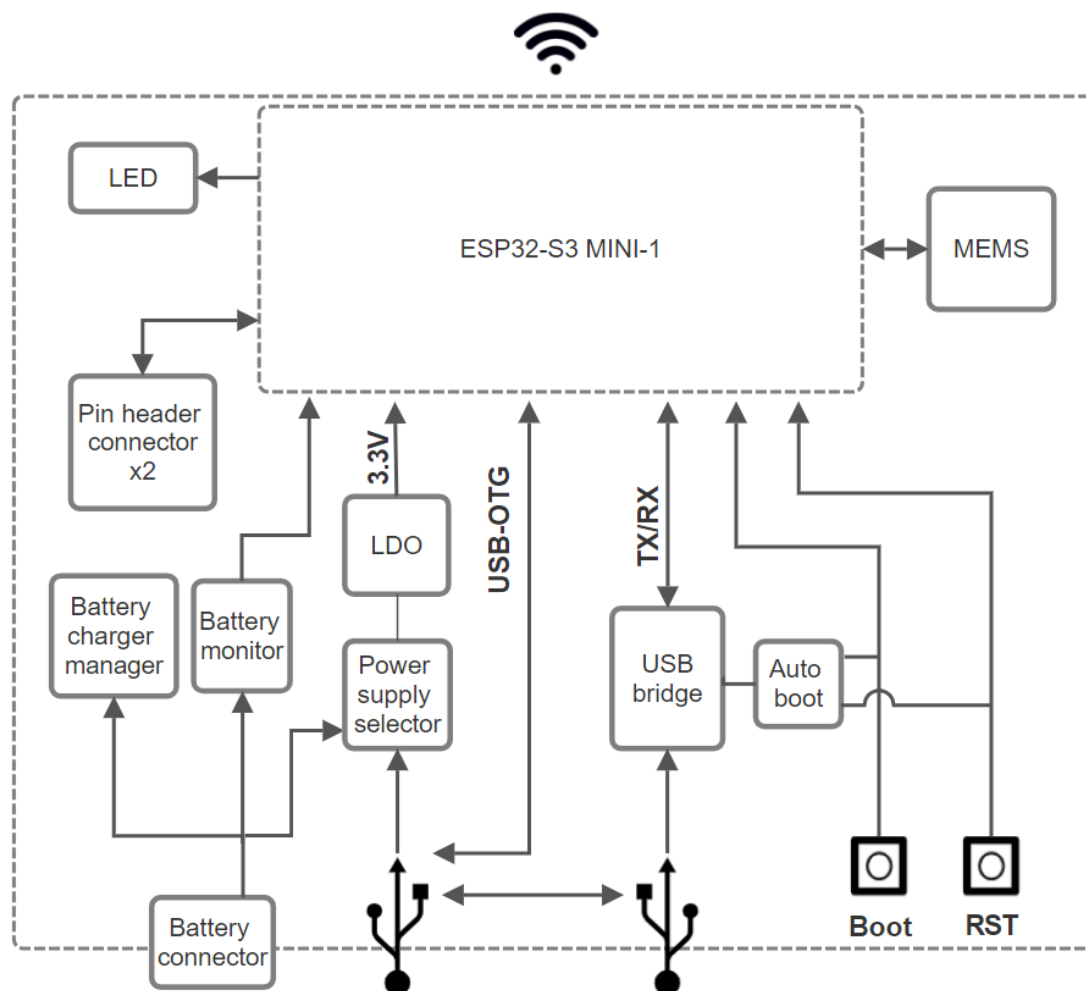


Figura 31. Diagrama de bloques de la placa ESAI

Nota: Esquema creado desde cero haciendo uso de la herramienta online de creación de diagramas “smartdraw”.

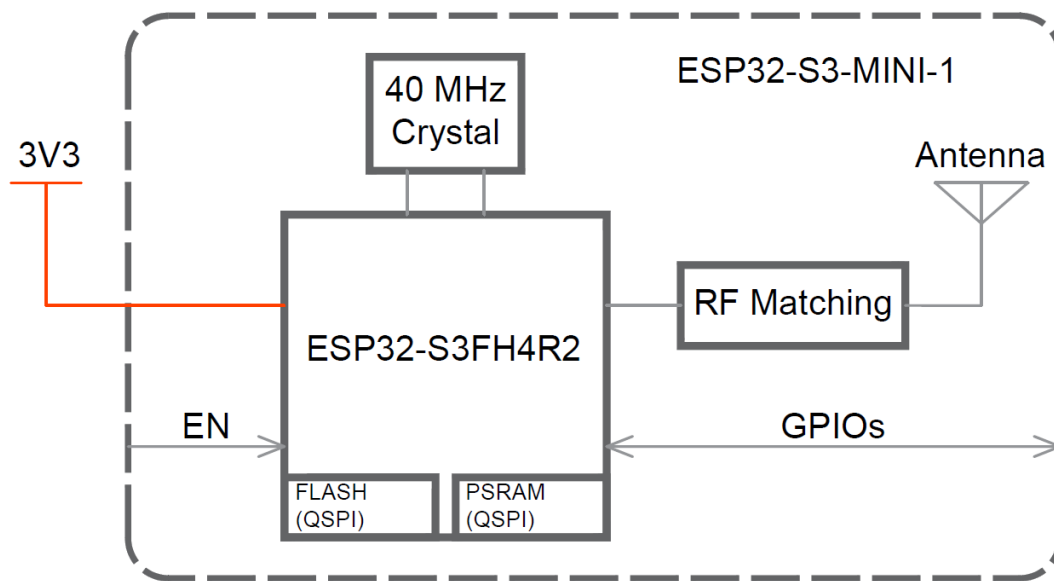


Figura 32. Ampliación del módulo ESP32-S3-MINI-1

Nota: Modificado a partir del diagrama ofrecido por Espressif en la hoja de datos.

4.2 Esquemático

Para el desarrollo del esquemático y la PCB se ha utilizado EasyEda. Se trata de un conjunto de herramienta de diseño EDA (“Electronic Design Automation”) que destaca por estar integrada en el catálogo de componentes de LCSC y el servicio de PCB JLCPCB lo que ayuda a los usuarios a ahorrar mucho tiempo. Además, es una herramienta sencilla de usar y tiene prácticamente las mismas prestaciones que otras herramientas de diseño más caras y complicadas de usar. EasyEda es gratuito, incluso en su versión profesional. EasyEda permite al diseñador seleccionar componentes en función del stock y el precio disponible. Cuando LCSC o JLCPCB suben un producto añaden también los datos asociados al mismo: la hoja de datos, el modelo para el esquemático, la huella para la PCB (“footprint”) y el modelo 3D. Adicionalmente el programa cuenta con una librería online, donde se pueden encontrar más componentes. En caso de no encontrar el componente, cuenta con una herramienta de creación de esquemático, footprint y modelo 3D. EasyEda permite exportar o importar un proyecto para que sea compatible con Altium y Eagle.

4.2.1 MCU

A la hora de elegir la unidad que controlará nuestro dispositivo hay muchas cosas a tener en cuenta y todo dependerá de nuestros objetivos y/o medios disponibles. Lo primero que debemos decidir es si queremos un SoC (System on Chip) o un módulo. La diferencia entre ambos es que el SoC es un integrado que consta únicamente del chip del micro ESP32 específico sin ningún otro añadido. Por otro lado, el módulo integra en un mismo componente la antena (PCB o IPEX) junto con ciertos elementos fundamentales como la memoria flash, la memoria PSRAM si la hubiera, el cristal de cuarzo (junto con los debidos condensadores de carga) y un circuito de adaptación de impedancia de la señal proveniente de la antena denominado matching circuit basado en un filtro pi. También se incluyen componentes pasivos adicionales para filtrar el ruido de alta frecuencia de la fuente de alimentación y diodos de protección para los transitorios de voltaje de la fuente. Encima de todo esto se añade un blindaje o shielding para proteger el módulo de cualquier interferencia electromagnética. Existe una línea de SoC bajo la denominación “PICO” el cual integra los mismos componentes que un módulo, pero con la diferencia de que carecen de antena integrada y del blindaje.

La elección de un SoC o un módulo como siempre es una cuestión que dependerá de las necesidades, recursos y tiempo. Los módulos tienen un mayor precio generalmente, pero sin embargo nos ahorrarán una importante parte del trabajo de diseño y sobre todo nos ahorrarán tiempo de pruebas en laboratorio. Estas pruebas están referidas para encontrar los valores exactos de los componentes de la red CLC (filtro PI) que conecta la salida de la antena con el pin del chip. Estos valores dependen de las características intrínsecas de la PCB y del tipo de antena, así como la forma en la que está integrada. Por ello es necesario realizar diversas pruebas de forma experimental hasta encontrar los valores de

los componentes que permitirán desarrollar el mejor rendimiento de la antena para la captación de la frecuencia deseada y una vez realizado este paso se comenzaría el proceso de obtención de certificados por las diferentes entidades certificadoras RF como el CE y ETSI de la Unión Europea o FCC de Estados Unidos con el objeto de valorar si nuestro dispositivo se ajusta a las respectivas normativas. Un caso en el que elegir el SoC como mejor opción podría ser para una empresa que quiera producir un producto en una gran cantidad y por lo tanto el tiempo-dinero de las pruebas pertinentes estará compensando por el ahorro en el coste unitario de cada PCB. Este fue uno de los problemas que enfrentó la empresa Humanox (en la que tuve el placer de realizar las prácticas de empresa) y que tuvo que subcontratar los servicios de una empresa externa para realizar las debidas pruebas descritas. Para nuestro proyecto, y teniendo en cuenta que es un prototipo de pruebas que no va a ser producido en masa la mejor opción, sin lugar a duda, es la del módulo.

Por otro lado, dentro de la familia ESP32 encontramos una gran cantidad de series de chips disponibles: ESP32, ESP32-S2/S3, ESP32-C2/C3/C6 y ESP32-H2. Para acotar la elección tendremos como requisitos un módulo que tenga:

- Arquitectura de doble núcleo (sólo la serie ESP32 y ESP32S3 tienen doble núcleo)
- Máxima memoria SRAM posible para evitar desbordamiento de la heap (sería interesante si el chip incorpora PSRAM)
- Frecuencia de 160-240 MHz para lograr un resultado óptimo con la menor latencia posible (no todas las series llegan a los 240MHz)
- Controlador y transceptor JTAG integrado eliminando la necesidad de un IC externo adaptador de USB-JTAG. Las series ESP32-S3, ESP32-C3 y ESP32-H2 lo cumplen.

Entre opciones similares se valorará el precio y el tamaño del encapsulado.

A priori buscaremos un módulo cuyo chip esté basado en el ESP32-S3 dado que es el único que cumple todos los requisitos anteriores. Espressif nos ofrece un selector de productos [32] que nos permitirá buscar entre todas las series filtrando por una gran cantidad de parámetros. Para nuestro caso seleccionamos como filtros Products->module, Series->ESP32-S3, Status->Mass production, Core->Dual Core, Antenna->PCB, Memory->SRAM max y obtenemos los siguientes resultados:

List: 14 items

IC/Module Development Board

Comparison Export

☐	Index	Name	Fi	Bluetooth	Thread/Zigbee	Temp (°C)	GPIO	Flash (MB)	SRAM (KB)	ROM (KB)	PSRAM (MB)	Freq. (MHz)	Size (mm)
☐	1	ESP32-S3-WROOM-1	2.11	Bluetooth	N/A	-40 ~ 85	36	4	512	384	0	240	18*25.5*3.1
☐	2	ESP32-S3-WROOM-1	2.11	Bluetooth	N/A	-40 ~ 85	36	8	512	384	0	240	18*25.5*3.1
☐	3	ESP32-S3-WROOM-1	2.11	Bluetooth	N/A	-40 ~ 85	36	16	512	384	0	240	18*25.5*3.1
☐	4	ESP32-S3-WROOM-1	2.11	Bluetooth	N/A	-40 ~ 65	36	4	512	384	8	240	18*25.5*3.1
☐	5	ESP32-S3-WROOM-1	2.11	Bluetooth	N/A	-40 ~ 65	36	8	512	384	8	240	18*25.5*3.1
☐	6	ESP32-S3-WROOM-1	2.11	Bluetooth	N/A	-40 ~ 65	36	16	512	384	8	240	18*25.5*3.1
☐	7	ESP32-S3-WROOM-1	2.11	Bluetooth	N/A	-40 ~ 85	36	4	512	384	2	240	18*25.5*3.1
☐	8	ESP32-S3-WROOM-1	2.11	Bluetooth	N/A	-40 ~ 85	36	8	512	384	2	240	18*25.5*3.1
☐	9	ESP32-S3-WROOM-1	2.11	Bluetooth	N/A	-40 ~ 85	36	16	512	384	2	240	18*25.5*3.1
☐	10	ESP32-S3-WROOM-1	2.11	Bluetooth	N/A	-40 ~ 105	36	4	512	384	0	240	18*25.5*3.1
☐	11	ESP32-S3-MINI-1	2.11	Bluetooth	N/A	-40 ~ 85	39	8	512	384	0	240	15.4*20.5*2.4
☐	12	ESP32-S3-MINI-1	2.11	Bluetooth	N/A	-40 ~ 85	39	4	512	384	2	237	15.4*20.5*2.4
☐	13	ESP32-S3-WROOM-2	2.11	Bluetooth	N/A	-40 ~ 85	39	16	512	384	0	240	18*25.5*3.1
☐	14	ESP32-S3-WROOM-2	2.11	Bluetooth	N/A	-40 ~ 65	33	32	512	384	8	240	18*25.5*3.1

Figura 33. Resultados del buscador de productos de Espressif.

Tenemos dos candidatos similares de 4MB de Flash y 2 MB de PSRAM. Usando la opción “comparison” observamos que solo difieren en el número de pin ligeramente, en las entidades certificadoras y en el tamaño, siendo el ESP32-S3-MINI-1 de menor tamaño ya que tiene integrada la flash de 4MB en el propio chip. En cuanto al precio acudimos a la página oficial del proveedor LCSC Electronics que usaremos para tomar los componentes de nuestro diseño electrónico en el software EasyEda [33]

ESP32-S3-WROOM-1 (N4R2) US\$3.9789

ESP32-S3-MINI-1 (FH4R2) US\$3.8198

Seleccionamos finalmente el ESP32-S3-MINI-1-N4R2 como candidato para el diseño de nuestro prototipo por ganar en tamaño y precio. Ambos cuentan con los certificados RF de las siguientes entidades FCC, CE-RED, IC, SRRC y TELEC. No obstante, sólo el WROOM-1 posee los certificados WiFi de Wifi-Aliance y BT de la compañía BT SIG.

Nota: El dato de 237 MHz de la tabla para el módulo ESP32-S3-MINI es una errata de la página.

Lo siguiente que vamos a hacer es revisar la hoja de datos del módulo seleccionado [34] para tener plena comprensión de lo que necesitamos añadir en nuestro diseño.

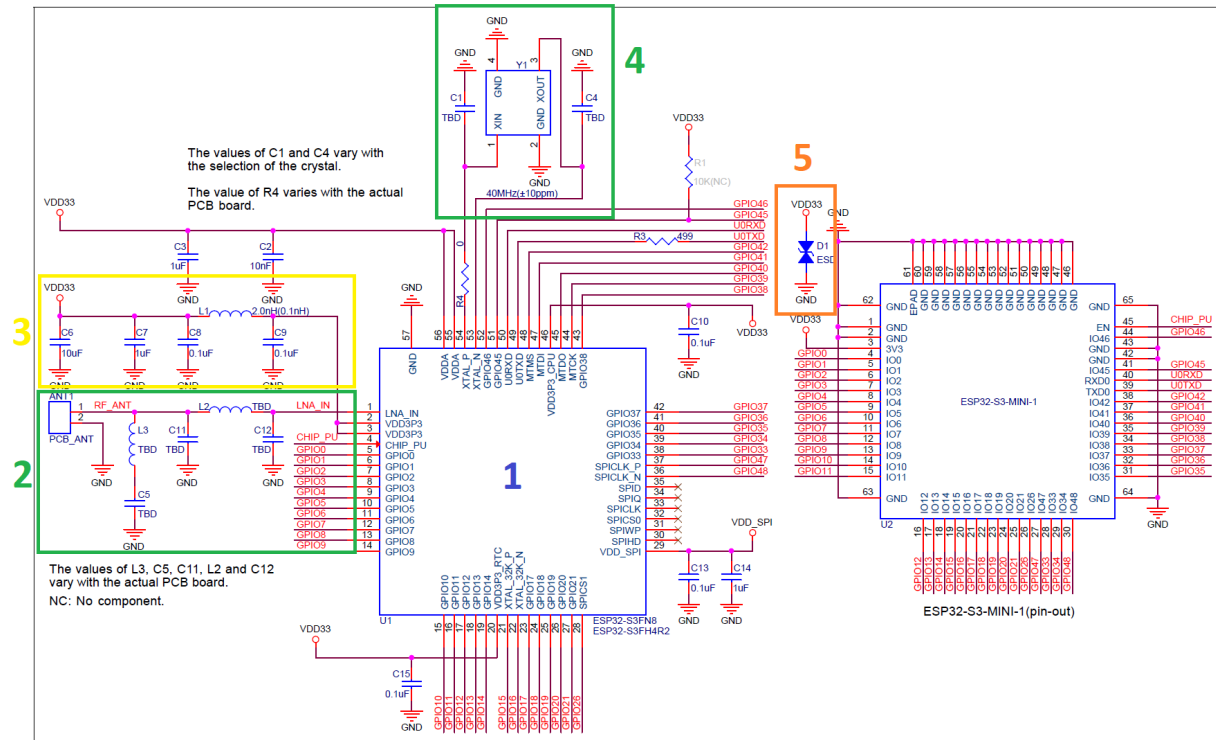


Figura 34. Esquemático de la datasheet del módulo ESP32-S3 MINI-1

1. Chip: El núcleo de nuestro sistema es el chip ESP32-S3FH4R2 el cual tiene integrado la memoria Flash de 4MB y una memoria PSRAM (PSeudo RAM) de 2 MB. Esta memoria aunque tenga en su nombre las siglas SRAM, no está formada por la típica estructura de celda de la memoria estática con 2 inversores más los 2 transistores de control de acceso sino más bien es una estructura de RAM dinámica pero con la particularidad de que incorpora un circuito de refresco de los condensadores de las celdas y un circuito de control de direccionamiento para que sea similar a una SRAM real (de ahí su nombre RAM pseudoestática). Por lo tanto, es una memoria que no será tan rápida como la SRAM pero tampoco tan lenta como una DRAM ya que no necesitará la atención constante de la CPU del sistema al que sirva. Los pines 30-35 del chip no estarán disponibles ya que son usados para la comunicación con ambas memorias pero es importante tener en cuenta que el chip nuestro utiliza el pin 28 (GPIO26) como segundo pin de selección (CS) del protocolo SPI para conectar con la PSRAM embebida. Aunque el diseño del módulo lo ofrezca como pad disponible debemos evitar su uso.

A continuación, se presenta el diagrama funcional de la serie EPS32-S3:

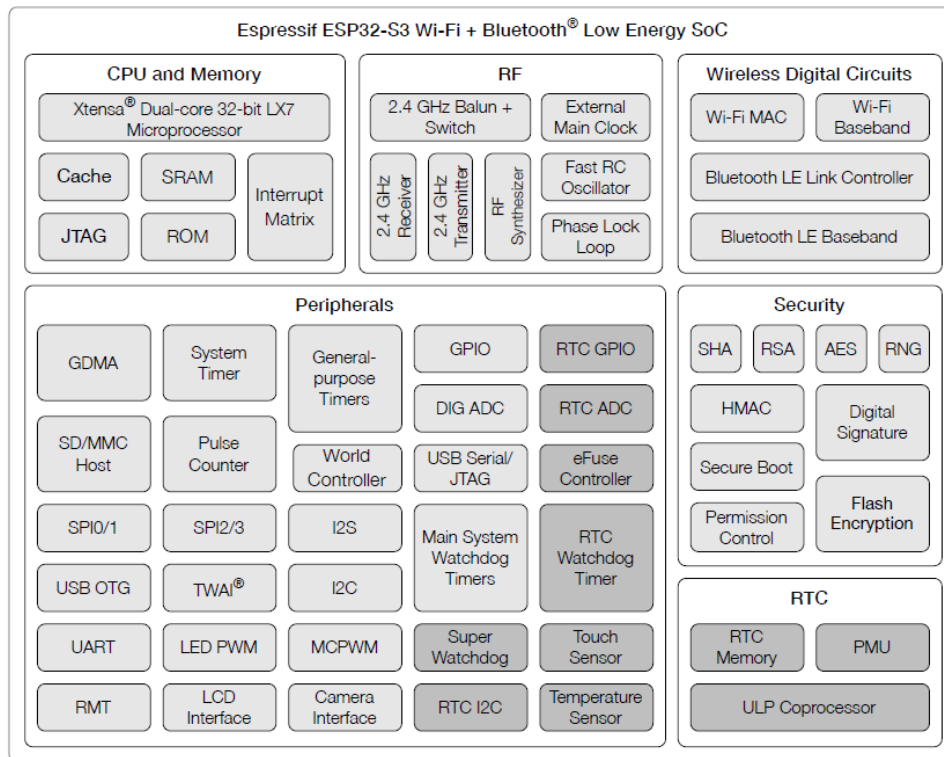


Figura 35. Diagrama de bloques funcional del chip, "ESP32-S3 Series"

Aparte de la PSRAM y FLASH integradas en el chip ya descritas la serie EPS32-S3 se procede a destacar algunas de las diferencias con la serie ESP32:

- Núcleo con una tecnología más avanzada (procesador Xtensa dual-core LX7 de 32 bits).
- Memoria: 512 KB de SRAM y 384 KB de ROM frente a 520KB y 448 KB de ROM. Aunque tenga algo menos de memoria en términos totales, los nuevos modelos de ESP32 se han refinado mediante mejores técnicas de fragmentación y son capaces de entregar la misma o incluso más memoria disponible para el usuario.
- Integra transceptor JTAG y funcionalidad USB-OTG.
- Nuevas funcionalidades del módulo de seguridad: HMAC, firma digital ("digital signatura") y control de permisos.
- Cambios en los periféricos. Se pierde el DAC y Ethernet pero se añaden GMDA, interfaz LCD y de Cámara, MCPWM y "world controller" entre otros.

2. Filtro Pi (Matching circuit): En el diseño de sistemas con señales de alta frecuencia como son los sistemas de radiofrecuencia un asunto de gran importancia es la adaptación de impedancias. Esta adaptación entre fuente-carga (sumidero) se hace para que la transmisión de potencia sea la máxima y para evitar el fenómeno de reflexión de la señal a

lo largo de la línea de transmisión. Esta onda reflejada puede superponerse de forma constructiva o destructiva lo cual causa que los valores de la señal real estén por encima o por debajo de la señal ideal. En la imagen siguiente se ilustra este efecto:

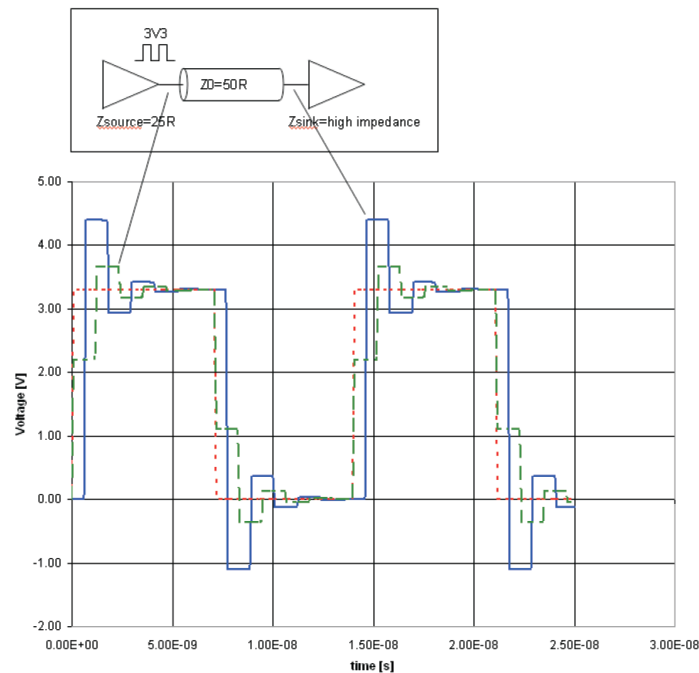


Figura 36. Sobreimpulsos y subimpulsos generados por una incorrecta adaptación de impedancia [35]

La onda representada en rojo corresponde a la señal de reloj de 3,3V de amplitud ideal, la verde es la señal real a la salida de la fuente y la azul es la señal al final de la línea de transmisión. En vez de variar entre 3,3V y 0V la señal alcanza los 4,4V y -1V lo cual podría llegar a causar daños en la carga.

Comúnmente el fenómeno de reflexión se expresa mediante los siguientes parámetros:

- Pérdida de retorno ("return loss"): Es simplemente la diferencia entre la potencia de la señal de la fuente frente a la potencia de la señal reflejada expresado en dB. Cuanto más similar al valor de potencia de la fuente mejor.
- Razón de onda estacionaria (VSWR por las siglas en inglés Voltage Standing Wave Ratio) es la relación entre el punto de tensión máximo y mínimo (V_{max}/V_{min}) que alcanza la cresta de la onda resultante constituida por la señal original más la superposición de la reflejada. Un valor de 1 o infinito supondría que no hay reflexión o que hay reflexión total respectivamente.

Para calcular matemáticamente la reflexión en la línea de transmisión se utiliza la expresión del coeficiente de reflexión que relaciona la impedancia de la fuente con la carga:

$$\Gamma = \frac{Z_L - Z_0}{Z_L + Z_0}$$

El valor de Γ será cero cuando no haya reflexiones y ± 1 si la reflexión es completa en los casos extremos en los que la impedancia de la carga sea infinita (extremo abierto) o cero (cortocircuito). Es por este motivo por el que está extremadamente desaconsejado transmitir con un transceptor de radio como un walkie-talkie sin la antena acoplada.

Para el caso que nos ocupa, la impedancia de salida (o entrada) de la antena parche es el valor estándar de 50Ω . Sin embargo, el circuito de entrada del pin LNA_IN tiene una impedancia con parte real e imaginaria (reactancia) por los efectos inductivos y capacitivos. El procedimiento de adaptación consiste en añadir filtros L-C de diversos tipos, siendo el recomendado en este caso un filtro π (CLC) con un filtro L-C previo para “cancelar” la parte reactiva y mejorar el acoplamiento entre la antena y el pin de alimentación RF del microcontrolador.

Por algún motivo en la guía de diseño del EPS32-S3 no se especifica el valor de impedancia del pin RF de los diversos chips de la serie como se especifica en las demás guías (por ejemplo $30/35 + 10j$ para los chips de la serie ESP32 ó $30 + 5j$ para los chips de la serie ESP32-S2). Es de esperar que el valor de impedancia sea similar a los otros chips aunque de todos modos los valores reales hay que determinarlo experimentalmente.

3. Filtrado de la alimentación del chip: Volvemos a encontrar un filtro (CLC) acompañado de varios condensadores de bypass para minimizar el rizado de la línea y estabilizar la tensión y corriente que recibe el pin. Externamente al módulo será igualmente necesario añadir más condensadores de desacoplo cerca de los pines del módulo.

4. Oscilador de cuarzo: Se incluye el integrado del oscilador de cristal de cuarzo junto a sus condensadores de carga conectados a los pines XTAL_P y XTAL_N del chip. Desconocemos el cristal concreto que se utiliza.

5. Diodo de protección: Este componente se sitúa cerca de la línea de alimentación de la antena y sirve para proteger al sistema de posibles descargas electrostáticas (ESD de las siglas en inglés Electrostatic Discharge). No se especifica el modelo utilizado. En el apartado “BUS-USB” dentro de esta sección se encuentra información más detallada sobre el funcionamiento y selección de este componente.

Una vez analizado el módulo veamos cómo se integra en nuestro circuito:

intensidad luminosa se refiere a la cantidad de luz emitida en una única dirección y la unidad en el SI es la candela (cd); el flujo luminoso depende de la intensidad luminosa y es un dato más completo pues realiza un cómputo de la intensidad luminosa a lo largo de todo el ángulo de emisión (120° para nuestro led) del haz luminoso y su unidad en el SI es el lumen (lm); finalmente la iluminancia que a su vez depende del flujo luminoso corresponde a la cantidad de luz reflejada en una superficie hipotética de 1m^2 a una distancia determinada de la fuente y su unidad es el lux ($1\text{ lux} = 1\text{lm} / \text{m}^2$). En la siguiente imagen se explica muy claramente de forma gráfica las diferencias entre los 3 parámetros:

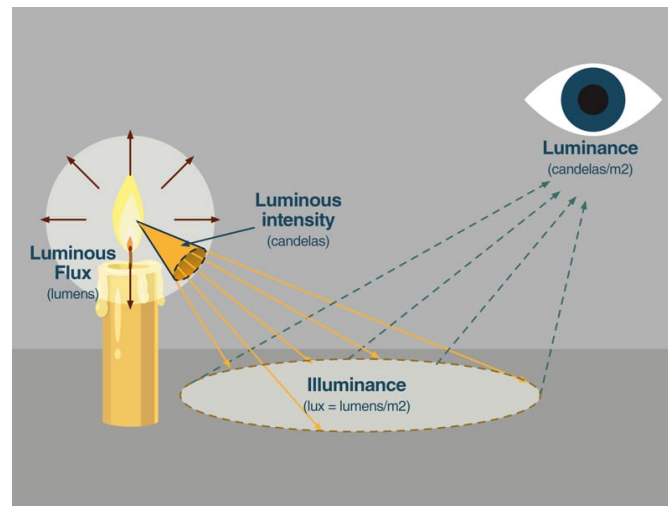


Figura 38. Comparativa entre los parámetros de medida de una fuente luminosa.

-Abajo a la izquierda de la figura 37 encontramos un circuito de retardo (delay) RC que es sugerido por las 3 hojas de datos principales (“ESP32-S3 MINI-1”, “ESP32-S3 Series”, “ESP32-S3 Hardware design guidelines”). En el encendido, una vez que se suministra una tensión de alimentación al chip se requiere una cierta cantidad de tiempo hasta que las diversas líneas de alimentación del chip se estabilicen antes de poner el pin CHIP_PU (definido como EN para nuestro esquemático) a nivel alto; a este tiempo se le denomina $t_{ST\ BL}$ (“ST” de Set Up o establecimiento). Recordemos que mientras este pin se encuentre a nivel bajo la unidad no se encenderá y permanecerá inactiva. De igual forma se requiere que el pin EN esté otro cierto tiempo (t_{RST}) por debajo de un umbral de tensión (V_{IL_nRST}) cercano al de tierra antes de que se produzca el reseteo. El circuito de retardo ayudará también a evitar que se produzca un reinicio indeseado en caso de que se pierda temporalmente la alimentación en el pin, pues mantendrá la tensión durante un cierto tiempo que vendrá impuesto por el valor de la red RC.

La figura siguiente es una representación de los parámetros de sincronización para el encendido y reinicio descritos anteriormente del documento ESP32-S3 Series:

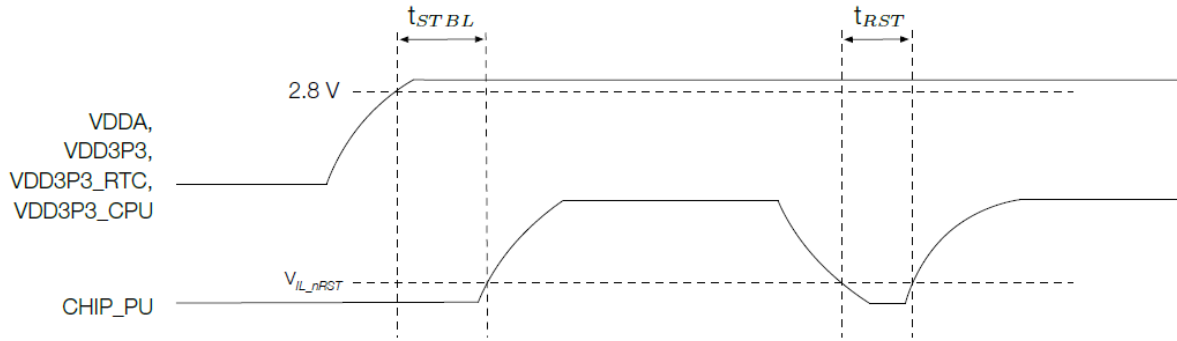


Figura 39. Diagrama con el timing de los parámetros de Power-Up y Reset. ESP32-S3 Series datasheet.

En la anterior figura habría que añadir como parámetro adicional el umbral de tensión por el cual se libera el reinicio, denominado V_{IH_nRST} el cual de acuerdo con la tabla de características en la misma datasheet indica que es del 75% de la VDD en el menor de los casos. Por otra parte, los valores de t_{STBL} y t_{RST} son de mínimo 50 μs para ambos parámetros. Para el dimensionamiento de los componentes recurrimos a la expresión de carga de un condensador.

$$V_C(t) = V_S (1 - e^{-\frac{t}{RC}})$$

Nuestro objetivo es que los valores de la red RC hagan que se alcance el valor de V_{IH_nRST} como mínimo en 50 μs , aunque realmente lo dimensionaremos para que sea mayor y estar sobre seguro. Espressif nos recomienda los valores de $R = 10\text{ k}\Omega$ y $C = 1\text{ }\mu F$. Para estos valores el tiempo transcurrido hasta llegar a la tensión umbral de RST será de:

$$t = -RC \ln\left(\frac{V_S - V_C}{V_S}\right) = -10k \cdot 1\mu \cdot \ln\left(\frac{3,3(1 - 0,75)}{3,3}\right) = 13,86\text{ ms}$$

Con los valores anteriores estamos seguros de cumplir con los requisitos mínimos junto a un amplio margen de seguridad.

-Finalmente, lo último a señalar son los denominados “strapping pins” lo cual no tiene una traducción exacta al español como en algunos otros casos dentro de la terminología de la electrónica y la computación. Estos funcionan como pines de ajuste o configuración del microcontrolador durante el encendido/reinicio dependiendo del valor que se les asigne. Estos pines tienen unos valores determinados por defecto y generalmente no hace falta añadir resistencias de pull-up/pull down pues estos pines ya están conectados a resistencias internas débiles (weak pull-up/down resistors), a no ser que queramos hacer

uso de alguna funcionalidad en concreto. La tabla siguiente recoge los strapping pins y sus valores por defecto:

Strapping Pin	Default Configuration	Bit Value
GPIO0	Pull-up	1
GPIO3	Floating	–
GPIO45	Pull-down	0
GPIO46	Pull-down	0

Tabla 1. Valores por defecto de los "strapping pins"

GPIO0 y GPIO46 son utilizados para hacer entrar al microcontrolador en "bootmode" o modo de arranque (ver el apartado de "Módulo de AUTO-BOOTMODE").

GPIO3 es un pin relacionado con la señal del JTAG y debe dejarse sin conectar.

GPIO45 es un pin para configurar la tensión VDD_SPI

GPIO46 también se utiliza como parámetro para configurar la impresión de los mensajes de la memoria ROM.

Consumo:

Igual que hicimos con el WROOM-32 la estimación de consumo se hará asumiendo que la mayor parte del tiempo estará en modo activo recibiendo sólo.

De acuerdo con la tabla 4-6 de la datasheet "ESP32-S3 Series" el consumo en modo activo RX es de 88-91 mA de pico máximo. Hay que tener en cuenta que las pruebas se han realizado con una frecuencia de CPU de 80 MHz y con el otro núcleo en reposo. Para nuestra aplicación a 160-240 MHz y usando los dos núcleos del procesador el consumo será algunos mA por encima. También, aunque pequeña en comparación, habría que añadir la contribución del resto de periféricos.

I = 100 mA (aproximación bastante hipotética)

4.2.2 MÓDULO DE ALIMENTACIÓN

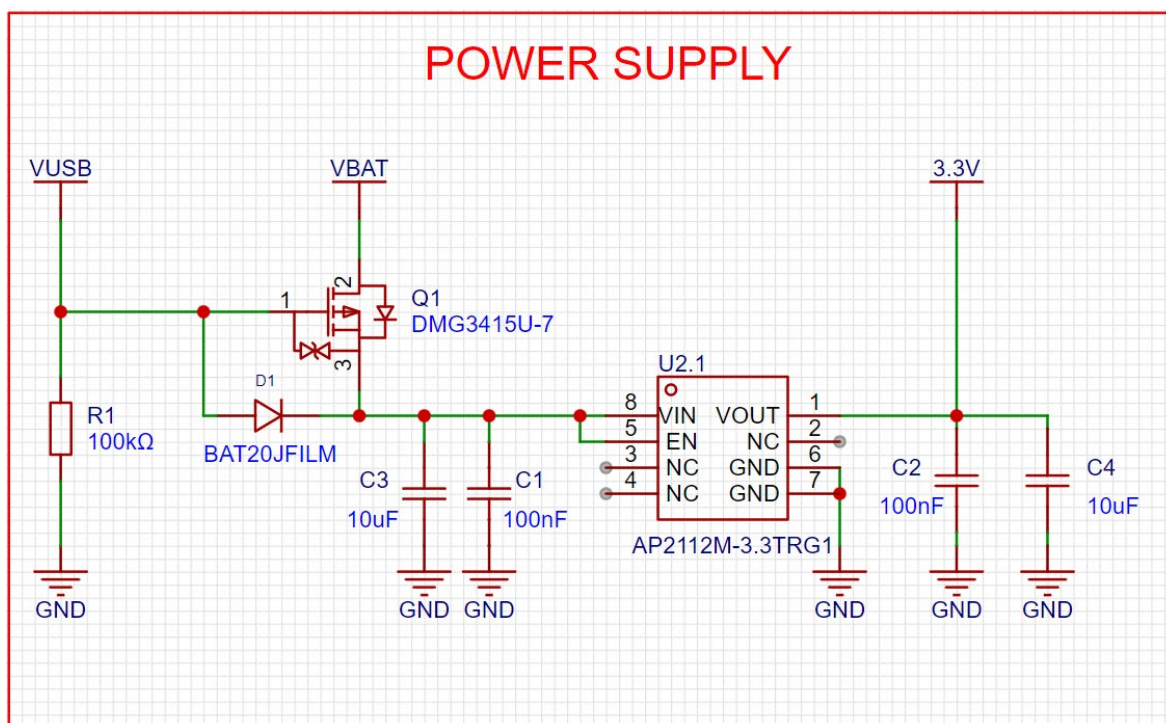


Figura 40. Módulo de alimentación. Fragmento de esquemático de la ESAI

Según el manual como norma general los microchips ESP32 pueden demandar durante el arranque corrientes de 500 mA y operan con una tensión nominal de 3,3V (con una ventana de margen de $\pm 0,3V$). Esta tensión deberá ser estable y con el menor ruido eléctrico posible, debido a que se trata de un microcontrolador que funciona como módulo transceptor RF de frecuencia ultraalta (banda 2.4 GHz), bastante sensible per se a cualquier tipo de interferencia electromagnética. Aunque una fuente conmutada DC-DC sería más eficiente queda descartada del diseño debido al rizado característico que emite a la salida y al coste de los componentes adicionales que hay que añadir. Además, la tensión de entrada tendrá en el peor de los casos alrededor de 5V y la corriente de salida no será excesivamente grande, por lo que las pérdidas de potencia serán comedidas. Nuestro objetivo será un regulador lineal de baja caída LDO (Low DropOut) con el máximo valor de PSRR y el menor valor de dropout o caída posible (la batería debe seguir funcionando aun cuando su tensión sea de 3,7V). Por otro lado, el circuito de alimentación debe de ser capaz de alternar autónomamente entre la alimentación del bus USB y de la batería externa.

-AP2112(U2): Es un LDO capaz de entregar una corriente de salida máxima de 600 mA y una tensión estable de 3.3V con un dropout de 250 mV típicos para una corriente de salida de 600 mA. Tiene un consumo de corriente (denominada corriente de quiescencia)

de 48-57 μA para 5V de entrada para el rango de temperaturas de trabajo. El fabricante asegura en la hoja de datos un PSRR de 65dB para frecuencias entre 100 y 1Khz y 0,5Vpp de rizado. Para tener una mejor comprensión la datasheet añade una gráfica más detallada del PSRR frente a la frecuencia:

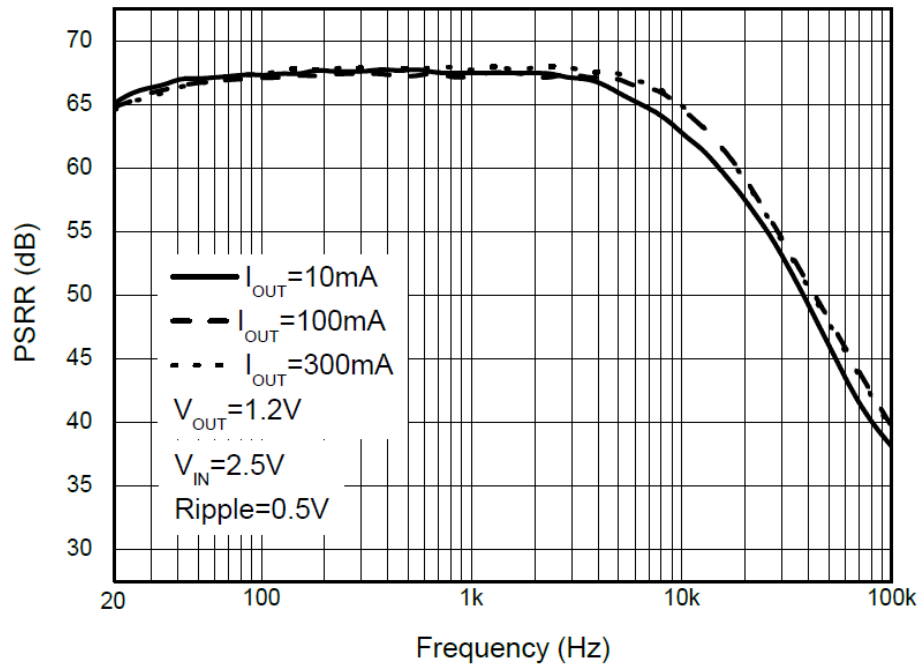


Figura 41. Gráfica PSRR frente a frecuencia. Datasheet del AP2112

Vemos que realmente entre las citadas frecuencias tenemos valores superiores de aproximadamente 67,5 dB. La cifra de los 65 dB se mantiene hasta los 6-10KHz dependiendo de la corriente de salida. El fabricante sólo ofrece la gráfica para el modelo con salida fija de 1,2V y una entrada de 2.5V.

Consideraciones térmicas:

La resistencia térmica(unión-ambiente) θ_{JA} para el encapsulado SOT-25 es de 184 $^{\circ}\text{C}/\text{W}$ y 114 $^{\circ}\text{C}/\text{W}$ para el SO-8. Es importante considerar el hecho de que estos valores son los que ofrece el fabricante mediante pruebas que ha realizado con las condiciones marcadas por el estándar JEDEC, que no son necesariamente las condiciones reales que tendrá el componente en nuestra PCB. Esta resistencia dependerá del tamaño de la PCB (cuanto mayor menor resistencia), grosor de las pistas, número de capas, espesor de estas y la posición que ocupará el componente en la placa. Sin embargo, el valor que nos da el fabricante nos puede servir de referencia y para comparar entre integrados similares.

La temperatura máxima en la unión es de 150 grados. Para una temperatura ambiente de 35 grados la potencia máxima que podría disipar el dado sería:

$$P_{max}^{SOT-25} = \frac{T_{J_{max}} - T_A}{\theta_{JA}} = \frac{150 - 35}{184} = 625 \text{ mW}$$

$$P_{max}^{SO-8} = \frac{T_{J_{max}} - T_A}{\theta_{JA}} = \frac{150 - 35}{114} = 1009 \text{ mW}$$

Como se mencionó anteriormente el microcontrolador puede, aunque de forma temporal, llegar a demandar corrientes de 500mA. La caída de tensión en el regulador máxima será cuando esté conectado el USB y habrá que restarle la caída en el diodo D1 (0,45V para una corriente de 500mA consultando la gráfica de la hoja de datos [38]. La corriente quiescente se despreciará debido a su bajo valor. Por lo tanto, la potencia máxima a disipar por el regulador en el peor de los casos en nuestra aplicación será:

$$P = I_{out} \cdot (V_{in} - V_{F,D1} - V_{reg}) = 0,5 \cdot (5 - 0,45 - 3,3) = 625 \text{ mW}$$

Si la temperatura ambiente o la resistencia térmica real fueran mayores estaríamos superando el límite recomendado en el encapsulado SOT-25. Dado que este proyecto se trata de un dispositivo que estará transmitiendo y recibiendo a través del módulo WiFi y que esto podía generar picos de corriente periódicos de hasta casi 400 mA en el peor de los casos (más el consumo además de los demás componentes) podría darse el caso de llegar a picos cercanos a los 500mA de forma regular. Es por esto que el encapsulado a elegir será el SO-8. No obstante también es cierto que, aunque de gran valor, estos picos duran poco tiempo y el consumo medio es menor (120-130 ms).

El regulador AP2112 se trata de un componente que a pesar de no ser caro ofrece unas prestaciones que exceden las del regulador AMS1117 (utilizado por las placas de desarrollo del proyecto) excepto por la resistencia térmica. A continuación, se realiza una comparativa:

	$V_{DROP}(V)$	I_Q	$\theta_{JA} (^{\circ}C/W)$	RMS Noise(μV)
AMS1117	1 - 1,3	5-10 mA	80/90	100
AP2112	0,250 - 0,4	55-80 μA	189/114/120	50

Tabla 2. Comparativa de características de los reguladores AMS1117 y AP2112. Datos extraídos de las datasheets

Nota:

El modelo del componente AP2112-M (nombre para el encapsulado SO-8) no existía. Haciendo uso de la herramienta de EasyEda de creación y en base a las medidas y pinout de la hoja de datos se creó el componente. Para hacerlo hay que ir a

File>>New>>Component para crear el modelo esquemático y luego
File>>New>>Footprint. Para vincular el esquemático con su huella (footprint) hay que ir a la opción "Package" dentro del componente y poner el nombre que le hayamos dado a la huella, en este caso "AP2112M-3.3TRG1".

-BAT20JFILM (D1): Este diodo Schottky de barrera (SBD) de muy baja tensión de umbral impide que la tensión de la batería se redirija a masa a través de la resistencia R1 de protección ESD (ElectroStatic Discharge) del transistor Q1 en ausencia de la línea USB y en caso de que esté presente protege al bus si la alimentación externa tuviese un valor superior a 5V.

-DMG3415U (Q1): Este componente es el encargado de conmutar de forma automática entre la tensión proveniente del bus USB o de la batería. Se trata de un transistor MOSFET de canal P de muy baja resistencia de conducción (R_{on}) el cual tiene conectado a su drenador la tensión de la batería y en su puerta la tensión del bus USB. En presencia del USB el transistor tendrá en su puerta una tensión superior a la de la fuente y por lo tanto estará en la región de corte. Sin embargo, la corriente podría fluir desde el drenador a la fuente a través del diodo parasitario del transistor Q1, que es independiente de la tensión en la puerta, pero esto no ocurre porque la tensión del bus USB, aún tras la caída en el SBD, es mayor que la de la batería que estará en el rango de 3,7-4,2 V, por lo que el diodo parasitario interno estará polarizado inversamente. Cuando se desconecte la alimentación del bus USB el transistor Q1 tendrá en su puerta 0V y por lo tanto entrará en conducción. La corriente pasará a circular por el nuevo canal formado, el cual tendrá una resistencia muy baja (36-50 mΩ) y dado que la corriente será del rango de 100-500 mA la caída de tensión será también muy baja (25 mV en el peor de los casos). Nótese que el MOSFET de canal P está invertido al sentido habitual de uso (que la corriente fluya de fuente a drenador), y esto no es ningún problema ya que los MOSFET conducen en ambas direcciones cuando el canal se ha formado [39]. El motivo de hacerlo conducir en sentido inverso al habitual es para que el diodo de cuerpo, que está polarizado inversamente, evite que la corriente del USB fluya de forma inversa hacia la batería y produzca una carga de esta de forma no controlada, lo cual podría causar graves daños. Podríamos posicionarlo en el sentido habitual, pero necesitaríamos un diodo adicional y estaríamos desperdiciando el diodo del MOSFET. Sin embargo, la topología seguida no sería óptima en caso de que el transistor Q1 estuviera trabajando en un régimen de conmutación a altas frecuencias (debido a que el tiempo de recuperación inversa del diodo t_{rr}) podría ser demasiado grande.

Consumo:

El consumo vendrá dado por la corriente de quiescencia del LDO más la corriente de fuga G-S del PMOS: $I = 80 \mu A + 10 \mu A = 90 \mu A$ (max)

4.2.3 MÓDULO DE CARGA

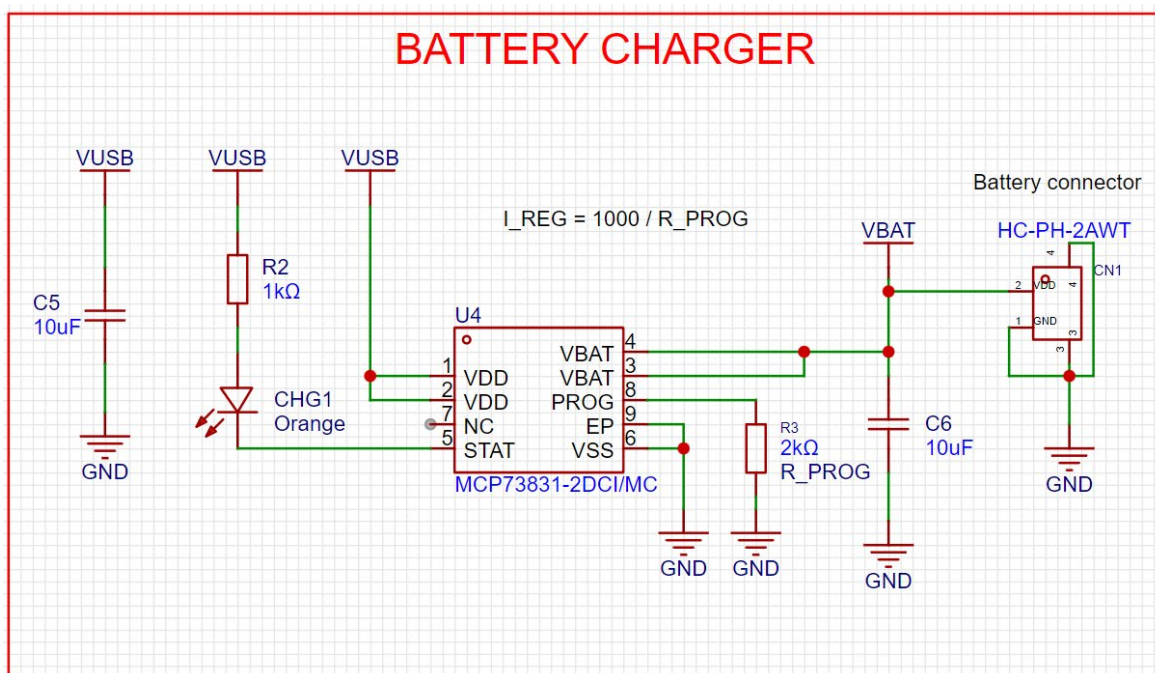


Figura 42. Módulo gestor de carga. Fragmento del esquemático de la ESAI

El componente MCP73831-2 es un controlador de gestor de carga autónomo para baterías de 1 celda de ion de litio o de polímero de litio. Es un dispositivo bastante completo, ajustable y con múltiples funcionalidades de gran interés. El integrado es una máquina de estados finitos dependiendo de las condiciones de diversos parámetros de entrada. Los estados son: Modo apagado, modo de preacondicionamiento, modo de carga rápida, modo de voltaje constante y modo de carga completa. A continuación, se describen las características más destacables del integrado:

-Corriente de carga programable: Mediante la resistencia de programación se puede ajustar la corriente de regulación de salida desde 15mA hasta un máximo de 500mA. Poner una señal lógica de nivel alto en este pin detiene el ciclo de carga por lo que este pin puede utilizarse como un apagado manual. Durante el estado desactivado consume típicamente 25uA y la corriente inversa de descarga de batería es de 2uA. La expresión que relaciona la corriente de carga rápida máxima con la resistencia de programación es la siguiente:

$$I_{REG} = \frac{1000}{R_{PROG}}$$

-Indicador de estado: El pin "STATUS" es un pin que ofrece información acerca de los 3 posibles estados de la carga: alta impedancia (Z) en ausencia de batería o en caso de estar en modo apagado (shutdown); nivel lógico bajo mientras esté en el proceso de carga y nivel alto cuando ésta se haya completado. Mediante la agregación del LED2 naranja conectado a la tensión USB obtenemos información del estado de carga.

-Detección de batería: El integrado suministra una pequeña corriente de 0,6uA en el pin VBAT y en función de si el voltaje sube o baja respecto a un umbral definido por VREG +100mV se considerará o no si hay ausencia de batería. En caso de ausencia, el integrado entrará en estado apagado.

-Under Voltage Lockout (UVLO): Un circuito interno monitoriza si la tensión de entrada supera cierto umbral (150mV por encima de VBAT) para salir del estado shutdown y comenzar el proceso de carga. El circuito tiene una histéresis incorporada de 100mV, de manera que se mantendrá el proceso de carga hasta que el voltaje de entrada baje el segundo umbral de 50mV por encima de VBAT.

-Recarga automática: El MCP73831 monitorea continuamente el voltaje en el pin VBAT en el modo de Carga Completa. Si el voltaje desciende por debajo del umbral de recarga, se inicia otro ciclo de carga y la corriente es suministrada nuevamente a la batería o carga. El umbral de recarga está preestablecido de fábrica a un 6% por debajo del voltaje regulado (4,2V).

-Apagado térmico: A medida que la temperatura de unión sube la corriente de carga máxima disminuye hasta suspenderse por completo la carga en caso de alcanzarse la temperatura máxima de la unión (150 °C).

La corriente de carga preferida por las celdas de las baterías de Li-Ion es la de 1C. Si la celda es de 250mAh, C1 correspondería a una corriente de 250mA. Cargar de esta manera proporciona los tiempos de ciclo de carga más cortos sin degradación en el rendimiento o vida útil del paquete de batería. No obstante a lo anterior, la mayoría de las hojas de datos de baterías sugieren que los valores de corriente de carga válidos varían entre 0,2-1C y en algunos casos se llega a 2C y 3C. En la siguiente imagen de la hoja de datos del MCP73831 se muestra el perfil de carga para una batería de 1000mAh y una corriente de 0,5C (500mA):

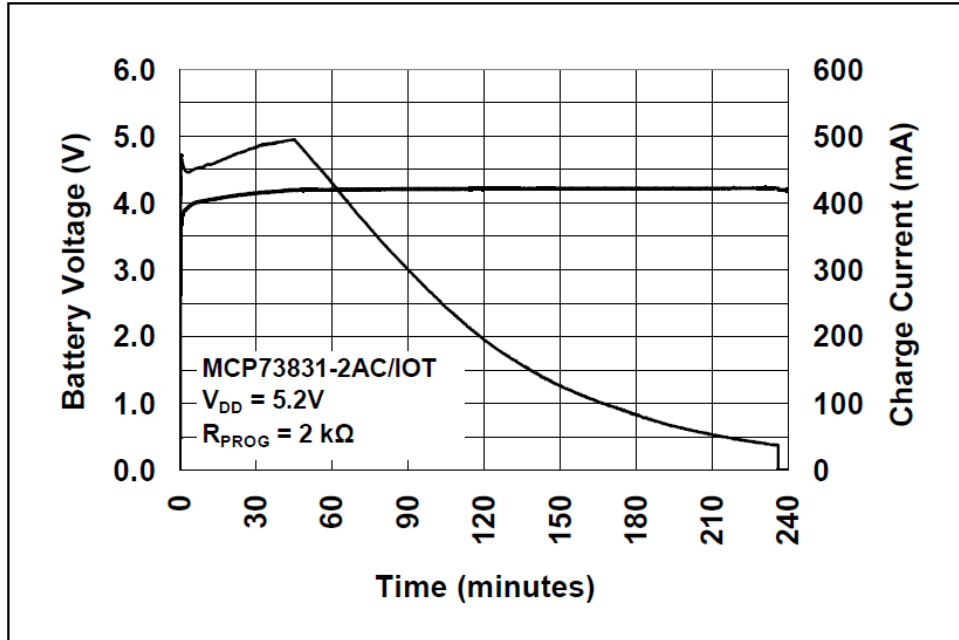


Figura 43. Gráfica del voltaje y corriente de carga frente al tiempo (batería de 1000 mAh).
Extraído de la datasheet del MCP73831/2

Toma menos de una hora para alcanzar un valor cercano al de regulación y menos de 4 para finalizar completamente y empezar el proceso de recarga automática.

Consideraciones térmicas:

Igual que se realizó en el apartado del módulo LDO se procederá a realizar los cálculos de potencia eléctrica y de la temperatura resultante a esperar en la unión. Existen 2 encapsulados, el SOT-23 con una resistencia térmica J-A de 230 °C/W y el DFN con 76 °C/W. La potencia a disipar máxima se da al final de la etapa de preacondicionamiento y se pasa al modo de cargar rápida cuando la corriente de carga y la diferencia de voltaje entrada-salida es la mayor. La condición para este cambio es que la tensión de la batería sea mayor al umbral establecido V_{PTH} que es un porcentaje de la tensión regulada V_{REG}, siendo en el menor de los casos del 64% de 4,2V, es decir, 2,69V.

$$P_{max} = I_{REG} \cdot (V_{DD} - V_{PTH}) = 0,5 \cdot (5 - 2,69) = 1,155 \text{ W}$$

Potencia máxima a disipar para no superar la temperatura de unión:

$$P_{max}^{SOT-23} = \frac{T_{J_{max}} - T_A}{\theta_{JA}} = \frac{150 - 35}{230} = 0,5 \text{ W}$$

$$P_{max}^{DFN} = \frac{T_{J_{max}} - T_A}{\theta_{JA}} = \frac{150 - 35}{76} = 1,51 \text{ W}$$

Aunque el SOT-23 no cumpliría las condiciones de potencia a disipar frente a temperatura podríamos utilizarlo dado que como se ha comentado, el MCP73831 cuenta con un sistema de regulación térmico que ajusta los ciclos de cargas para que no se sobrepasen las temperaturas máximas, pero también provocaría retardos en el tiempo de carga total. Para un rendimiento máximo y mayor seguridad la mejor opción es el encapsulado DFN-8.

Consumo:

El consumo de este módulo corresponde a cuando el módulo MCP73831 se encuentra en modo hibernación ("shutdown mode") en ausencia de batería.

$I = 200 \mu A$ (max)

4.2.4 MEMS

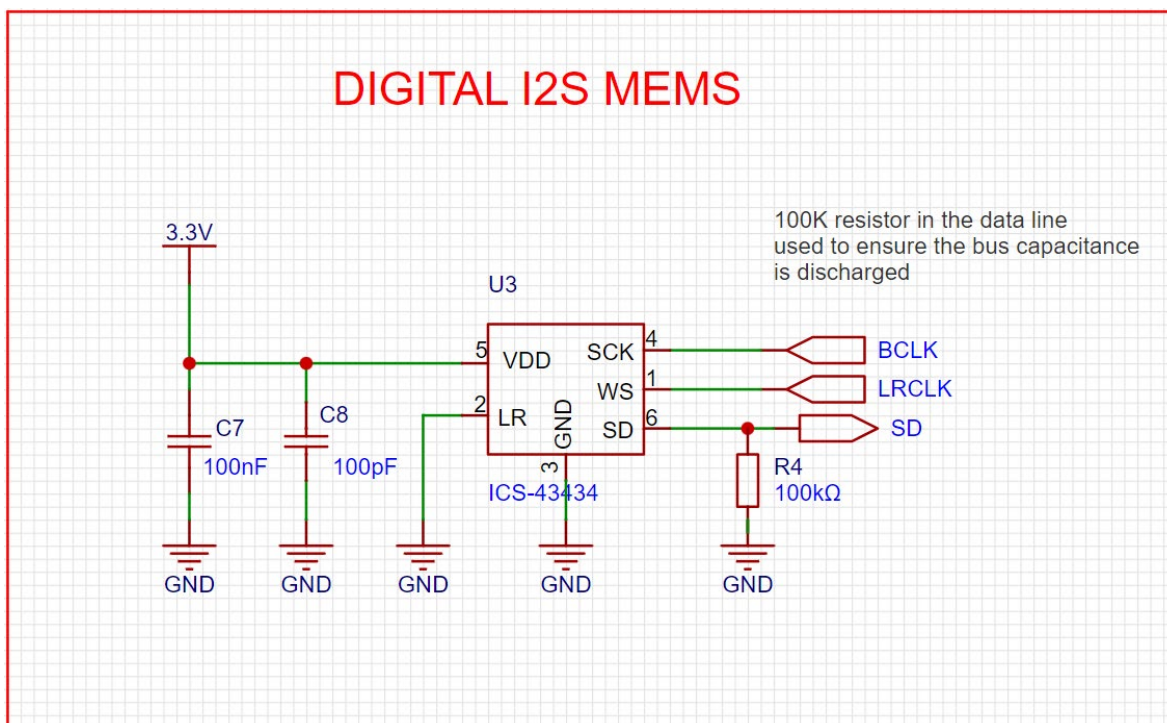


Figura 44. Módulo de captación acústico. Fragmento del esquemático de la ESAI

Aunque el micrófono INMP441 ha mostrado ser un componente de calidad y fiable se trata ya de un componente relativamente antiguo que ha sido superado en casi todas sus características por nuevos modelos desarrollados por la empresa TDK Invensense y por otros fabricantes. Se han seleccionado 2 componentes de TDK y otro de la empresa

Knowles la cual también suministra a LCSC Electronics. Los datos son extraídos directamente de los valores típicos de las hojas de datos de los fabricantes:

Modelo [1]	Sensibilidad (dB FS)	SNR(dBA)	PSR/PSRR (dBFS)[2]	AOP(dBSPL) [3]	$I_s(V_{DD} = 3.3V)$
INMP441	-26	61	-75	THD 3% 105 dB SPL	2,2 mA
ICS-43432	-26	65	-80/-90	116	1,1 mA
ICS-43434	-26	64	-98/-100	120	230 μ A
SPH0645LM4H-1	-26	65	-86/80	120	300 μ A

Tabla 3. Comparativa de MEMS digitales

1. Los valores son para el modo normal o activo excepto para el ICS-43434 que para la frecuencia de muestreo de 16 KHz estaría operando en el modo de bajo consumo.
2. Cada micrófono tiene una serie de pruebas distintas. El INMP441 carece de dato de PSRR y las condiciones para la PSR son la de señal cuadrada 100mVp-p y 217Hz. Los modelos ICS entregan el dato de PSR anterior y el de PSRR para una senoidal de 1KHz. El modelo SPH da el valor de PSRR en dB y en valor positivo por lo que asumo que utiliza la expresión clásica que se usa en LDOs y amplificadores o es una errata.
3. La hoja de datos del INMP441 no tenía el dato estándar de AOP por lo que se ha utilizado el de THD.

Todos los modelos anteriores tienen la misma forma de operar en cuanto a la interfaz I2S (formato Phillips estándar): el primer bit de dato es el más significativo, un ciclo completo de la señal de WS está constituido por 64 ciclos BCLK, la señal de datos está desfasada un ciclo de reloj respecto a WS y la profundidad de bits máxima es de 24 bits. En cuanto a la respuesta en frecuencia, los dos últimos modelos ganan por diferencia llegando hasta los 51,6 y 64 KHz respectivamente.

Como conclusión los modelos que destacan son el ICS-43434 y el SPH0645LM4H-1 como excelentes opciones relación en calidad y consumo. El único inconveniente es que estos componentes son de tipo “bottom ported”. Como vimos en la segunda sección, la disposición bottom ported permite un mayor tamaño de diafragma mayor y por lo tanto mejor calidad de recepción de sonido pero también tiene el inconveniente de que el dispositivo debe ser montado en la cara opuesta del resto del circuito, al menos para una

placa prototipo para desarrollo como en nuestro caso aunque para un producto final el componente podría montarse en la capa superior y la placa simplemente estaría al revés respecto a la parte frontal o delantera que dará la cara de la ESAI pero habría que pensar también que los leds integrados no podrían ser visualizados porque estarían también del revés al usuario. De cara a una producción en masa de un producto existe gran interés en evitar componentes en la capa inferior pues el ensamblado de ambas caras **aumenta significativamente** los costes y tiempos de producción.

Consumo:

$I = 300 \mu A$ (para una frecuencia de muestreo de 16 KHz)

4.2.5 MÓDULO ADAPTADOR USB-UART

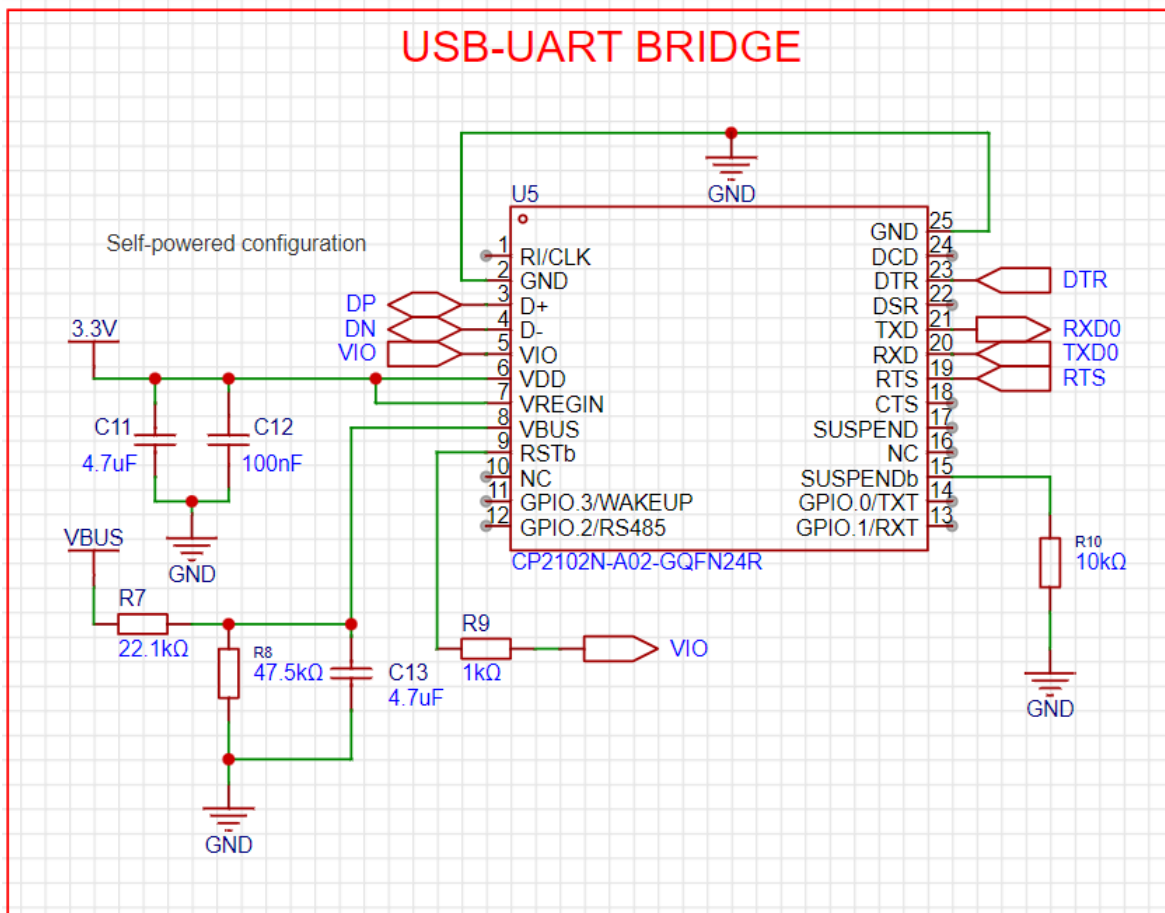


Figura 45. Módulo adaptador USB-UART. Fragmento del esquemático de la ESAI

El CP2102N [40] es un controlador altamente integrado que sirve de puente entre la

interfaz USB y la UART de un dispositivo como nuestro microcontrolador ESP32 y de ahí su denominación común de “USB bridge”. El dispositivo cuenta con las siguientes características:

- Cumple con la especificación USB 2.0 (full-speed 12 Mbps).
- Transceptor USB integrado. No será necesario añadir las típicas resistencias de terminación en las señales D+, D- que provienen del bus USB.
- Oscilador de 48 MHz integrado. No será necesario añadir oscilador externo.
- UART capaz de soportar 5-8 bits de datos, 1-2 bits de stop, todos los tipos de paridad y una tasa de baudios de 300 baud a 3 Mbaud.
- Cuenta con los controladores de puertos COM virtuales para Windows, Mac y Linux.

El dispositivo cuenta con un regulador interno para regular la tensión de bus de entrada y generar 3.3 voltios a la salida (pines VREGIN y VDD). Dado que ya tenemos nuestro regulador LDO externo queremos que el dispositivo opere en la configuración de “self-powered” y prescindir de su regulador interno por lo que llevamos la línea 3.3V de salida de nuestro LDO a los pines VREGIN y VDD, evitando así que el regulador funcione y consuma potencia.

El pin VBUS permite al integrado detectar si el bus USB está presente o no para entrar en modo suspensión. El divisor de tensión formado por R7 y R8 tiene la función de protección para que no se supere la tensión máxima en el pin VBUS, la cual según la tabla 3.10 de la hoja del componente es de VIO +2.5 y siendo VIO igual a la tensión VDD. Un caso en el que esto podría ocurrir es si llegase la señal del bus antes que la de nuestro regulador (AP2112) a VREGIN mientras el integrado no está aún encendido.

El pin SUSPENDb debe de estar a nivel bajo durante el encendido y el reinicio del sistema. Es posible que durante estos eventos dichos pines queden temporalmente en un estado indefinido por lo que se ha unido a tierra mediante una resistencia de pull-down.

Consumo:

El consumo de corriente del integrado es de 9,5 mA en modo de operación normal, esto es durante el proceso de carga de firmware o comunicación a través de la consola serial. Sin embargo, el producto final de la ESAI no utilizará este módulo durante el desempeño de su actividad por lo que se tomará el valor en estado de suspensión.

$I = 105 \mu A$

Hay que añadir también el consumo del divisor de tensión que sirve para detectar si la línea de bus USB está presente:

$$I_{Total} = 100 \mu A + \frac{5V}{21,1K + 47,5K} = 171,84 \mu A$$

4.2.6 BUS USB

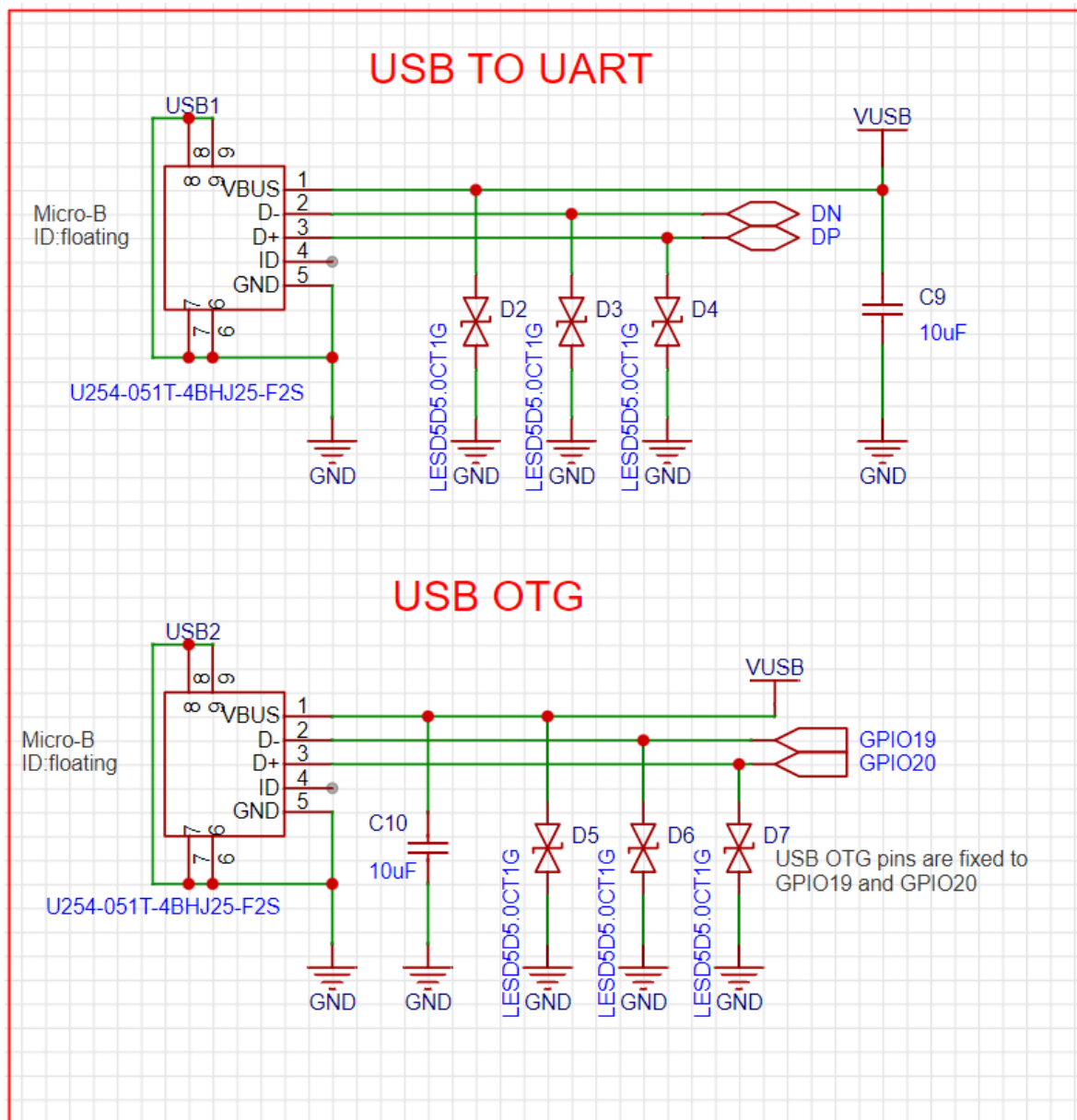


Figura 46. USB-OTG y USB-UART. Fragmento del esquemático de la ESAI

La conexión con el bus se realiza mediante dos conectores USB micro-B y por ende se les ha dejado el pin ID flotando para que puedan ser correctamente reconocidos [41].

Uno de los conectores tiene sus señales de datos directamente conectadas a los pines GPIO19 y 20 del microcontrolador, los cuales como se ha comentado son los pines establecidos para la recepción OTG y el otro conector es la entrada para el integrado

CP2102N descrito anteriormente. La cuestión que surge es, ¿por qué utilizar dos conectores USB y utilizar el adaptador USB-UART externo cuando la serie ESP32-S3 tiene incorporado un transceptor USB/JTAG? El motivo es porque existen diversas limitaciones en implementar la consola serial CDC (Communication Device Class). La mayoría de estas limitaciones provienen del hecho de que el protocolo de comunicación por consola USB CDC se implementa en software, lo que lo hace más frágil y complejo que una consola que utiliza UART:

- Si la aplicación se bloquea, la salida del “panic handler” puede no enviarse a través de USB CDC. Si la memoria utilizada por el controlador CDC se corrompe o si hay algún otro problema a nivel del sistema, CDC puede no funcionar para enviar mensajes a través de USB. Del mismo modo, si la aplicación entra en un “bootloop” antes de que el controlador USB CDC tenga la oportunidad de iniciarse, no habrá salida en la consola.

- Si la aplicación reconfigura accidentalmente los pines del periférico USB o deshabilita el periférico USB, el dispositivo USB CDC no aparecerá en el sistema al que está conectado. Lo mismo ocurrirá si la aplicación entra en modo light-sleep o en modo de suspensión profunda.

- El controlador USB CDC reserva una cierta cantidad de RAM y aumenta el tamaño del código de la aplicación lo cual en casos en los que se requiera usar el máximo provecho de la capacidad de la RAM pueda ser un inconveniente.

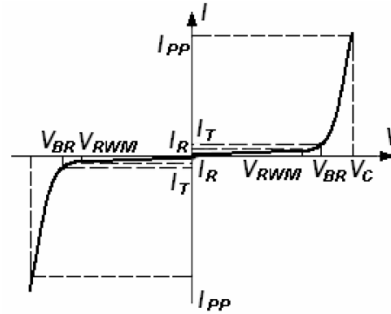
- Cuando se depura la aplicación utilizando JTAG, la operación de USB CDC puede dejar de funcionar si la CPU se detiene en un punto de interrupción (breakpoint). La operación de USB CDC depende de las interrupciones del periférico USB que se atienden periódicamente. Si la computadora anfitriona no recibe respuestas válidas del lado del dispositivo USB durante un tiempo, puede decidir desconectar el dispositivo. El tiempo real depende del sistema operativo y del controlador, y varía desde unos pocos cientos de milisegundos hasta unos pocos segundos.

Más información puede obtenerse en la página oficial de Espressif [42]

- Los componentes D2 a D7 son diodos supresores de voltajes transitorios (TVS) de doble polaridad. En caso de producirse un pico de tensión indeseado, los diodos sufrirán el efecto avalancha y derivarán la corriente a tierra para evitar dañar los posteriores circuitos integrados. La forma de operar y los parámetros a analizar de estos componentes es la siguiente:

Electrical Parameter

Symbol	Parameter
I_{PP}	Maximum Reverse Peak Pulse Current
V_C	Clamping Voltage @ I_{PP}
V_{RWM}	Working Peak Reverse Voltage
I_R	Maximum Reverse Leakage Current @ V_{RWM}
I_T	Test Current
V_{BR}	Breakdown Voltage @ I_T



Electrical Characteristics

Ratings at 25°C ambient temperature unless otherwise specified. VF = 0.9V at IF = 10mA

Device	V_{RWM} (V)	I_R (μ A) @ V_{RWM}	V_{BR} (V) @ I_T (Note 1)		I_T	V_C (V) @ $I_{PP}=5$ A*	V_C (V) @ Max I_{PP} *	I_{PP} (A)*	P_{PK} (W)*	C (pF)
	Max	Max	Min	Max	mA	Typ	Max	Max	Max	Typ
LESD5D5.0CT1G	5.0	1	5.6	7.8	1.0	11.6	18.6	9.4	174	15

Figura 47. Características del diodo LESD5D5.0CT1G [43]

La tensión V_{RWM} es aquella que asegura que el diodo esté en la región de corte con una corriente equivalente a la de fuga inversa (I_R). Si aumentamos la tensión, la corriente de fuga comenzará a aumentar gradualmente hasta llegar al valor de tensión de V_{BR} (tensión de ruptura). Generalmente se define este valor como el que produce una corriente de fuga de 1mA y a partir del cual se producirá el efecto de ruptura por avalancha en la unión del diodo. Se ha asegurado que su V_{BR} esté por encima del valor de 5V nominales de la línea de alimentación. Otro parámetro a tener en cuenta es V_C (clamping voltage o voltaje de sujeción máximo) el cual define la tensión máxima a la que estará expuesto nuestro sistema cuando se produzca una derivación teniendo en cuenta la resistancia en serie intrínseca (R_{DYN}) al diodo por la que caerá una tensión también. Por lo tanto, esto significa que puede haber más tensión antes de que se produzca la ruptura y dependerá del valor de dicha resistancia. Este valor lo podemos calcular de forma indirecta a partir de los datos de V_C y de la corriente de pulso pico (I_{PP}) que nos ofrece el fabricante. En la figura 48, se ilustra el modelo real del diodo con su resistancia interna y la expresión de la tensión de sujeción. Para un modelo más fiel al real habría que añadir la capacidad parásita en paralelo entre la línea a proteger y tierra.

La expresión de V_C es:

$$V_C = V_{BR} + I_{PP} \cdot R_{DYN}$$

despejando obtenemos:

$$R_{DYN} = \frac{V_C - V_{BR}}{I_{PP}}$$

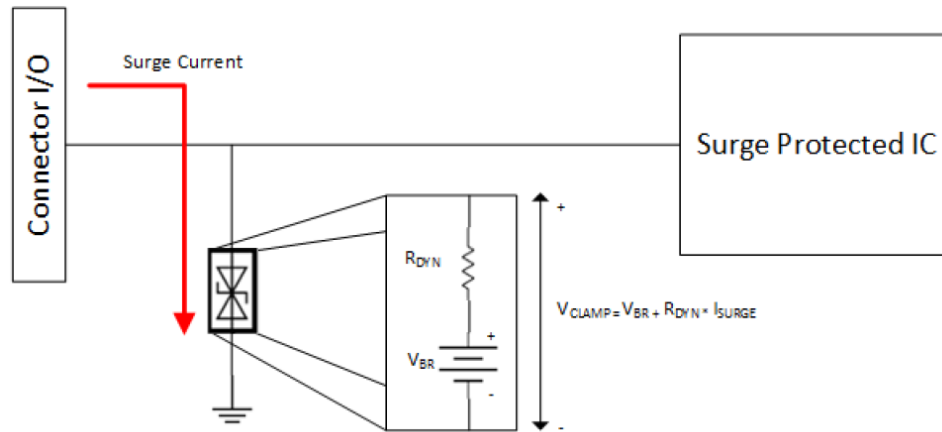


Figura 48. Modelado de un diodo TVS [44]

Para la prueba que realiza el fabricante con una $I_{PP} = 5A$, V_C es 11,6 V y considerando el valor más bajo de ruptura (5,6 V) nos arroja un valor de resistencia de 1,2 Ω , el cual para una corriente de 0,5 generaría una caída de 0,6V.

A la hora de elegir este componente para el diseño habría que realizar pruebas con prototipos y analizar los pulsos de tensión y las corrientes para determinar si ese valor de resistencia es demasiado grande. Otra cosa que no se ha tenido en cuenta es la duración de los pulsos y que podría influir en la sobrecarga térmica del diodo. En general se busca siempre que el valor de resistencia interna sea el mínimo posible para asegurarnos que el diodo opera de forma ideal y entre en ruptura cuando el valor de tensión de la línea sea el de V_{BR} .

El último parámetro a tener en cuenta es el de la capacidad parásita del diodo, el cual puede ser importante si la señal a proteger es una señal de muy alta frecuencia. Como norma general la capacitancia máxima entre líneas de señal y tierra debe ser menor a 100 picofaradios (pF) a una velocidad de datos de 125 kilobits por segundo (kb/s) y 35 pF a 1 Mbits/s [45]. Teniendo en cuenta que la velocidad de transferencia serial será de kilobits 15 pF es un valor bastante seguro.

La elección de este modelo ha sido guiada por ser una elección habitual de Espressif en sus placas de desarrollo.

Consumo:

Sumando las corrientes de fugas (1uA) en estado de operación normal para una tensión de entrada $V < V_{WRM}$ de todos los diodos TVS obtendremos:

$$I = 6 \mu A$$

4.2.7 CIRCUITO DE AUTO-ARRANQUE (AUTO-BOOTLOADER)

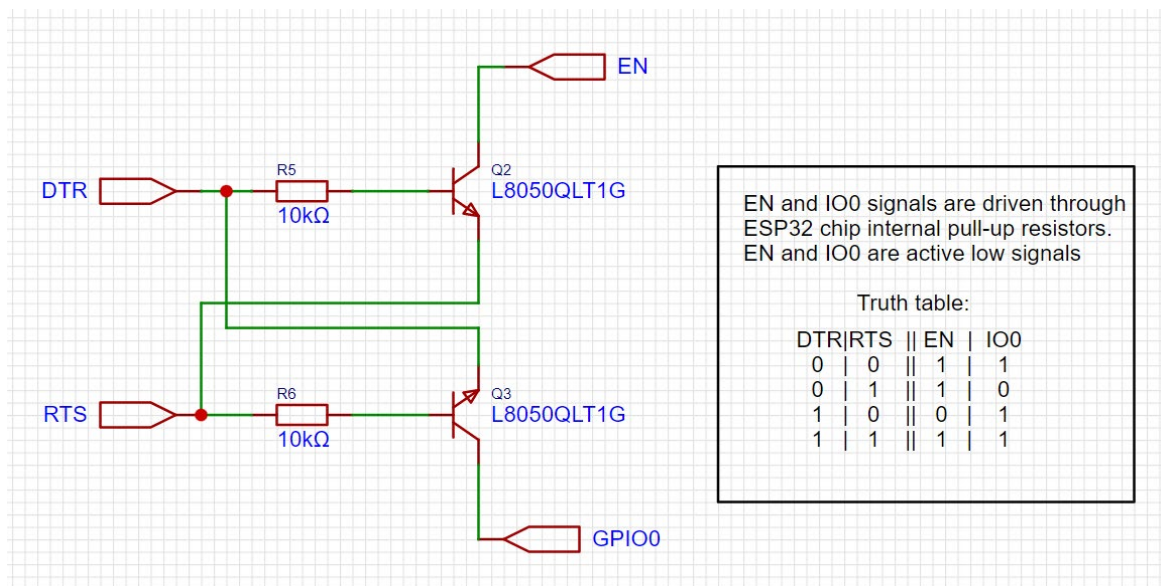


Figura 49. Circuito auto-bootloader. Fragmento del esquemático de la ESAI

Este circuito permite cargar el firmware sin la necesidad de pulsar los botones manuales de RESET y BOOT. Los dos transistores NPN bipolares junto con las resistencias de pull-up internas del ESP32 crean un circuito combinacional. Si DTR y RTS están a nivel alto o bajo a la vez los transistores estarán polarizados en la región de corte y ambos pines se mantendrán a nivel alto a través de las resistencias de pull-up internas. De esta manera podemos lograr de forma automática la secuencia correcta (activar a nivel bajo GPIO0 tras activar el pin EN) para hacer que el ESP32 entre en modo “boot” cuando establezcamos la comunicación serial con el ESP32 desde el ordenador al cargar el programa. Sin embargo, en la placa se han añadido dos pulsadores conectados a los pines EN y GPIO0 para poder resetear o hacer entrar en modo arranque de forma manual en caso de que fallara el anterior circuito.

Consumo:

Despreciable.

4.2.8 MONITOR DE BATERÍA

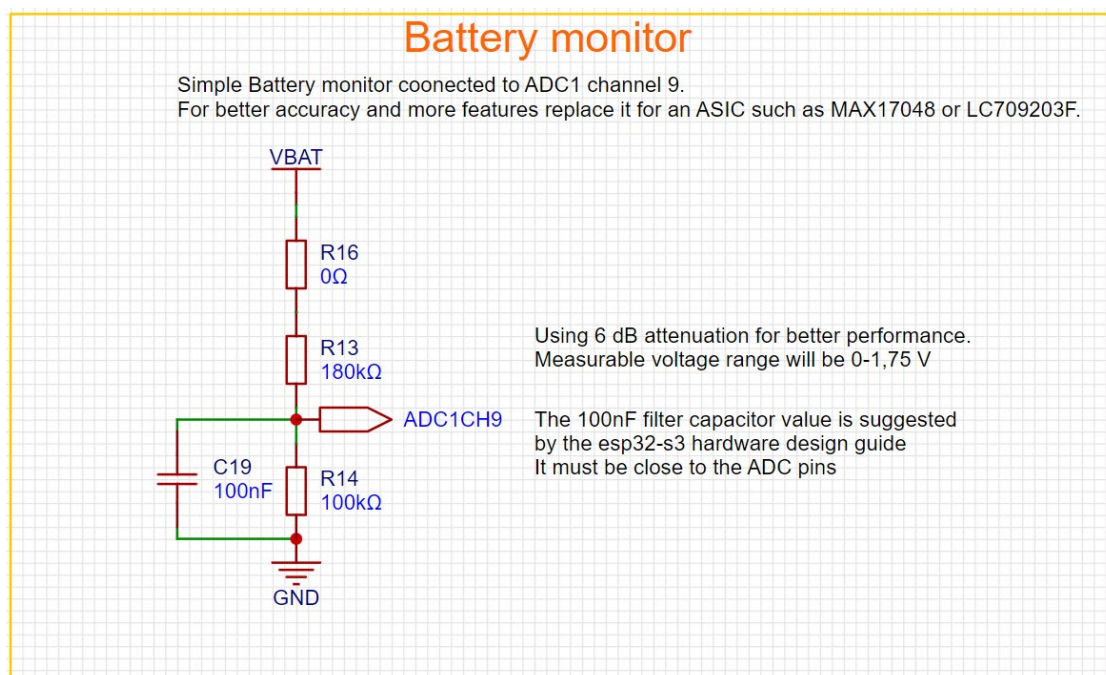


Figura 50. Monitor de batería. Fragmento del esquemático de la ESAI

El siguiente apartado consiste en un rudimentario monitor de batería basado en un divisor de tensión conectado a la entrada del canal 9 del ADC1 del microcontrolador. Los valores de resistencia se han elegido de tal forma que cuando a la entrada haya una tensión de 4,2V (máxima tensión esperada cuando la batería esté completamente cargada) la tensión en el ADC sea de 1,5V con el objetivo de que la tensión de entrada no supere los 1,6V. El motivo de esto es porque para obtener una respuesta lo más lineal posible del convertidor y mejor precisión es necesario añadir una atenuación al canal del ADC, y en concreto la que logra mejores resultados es la de 6dB como se puede apreciar en la figura 51 extraída de una publicación en “research gate” titulada “*Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things*” introducida en la 7ª conferencia de internet de las cosas y aplicaciones en Wrexham, Reino Unido [46].

Por defecto, los canales del ADC están configurados con la máxima atenuación (11 dB para la serie ESP32 y ESP32-S3). Al aumentar la atenuación el rango de tensión de entrada permitido aumenta también, pero la respuesta pierde linealidad y con ello la precisión de la medida se verá afectada.

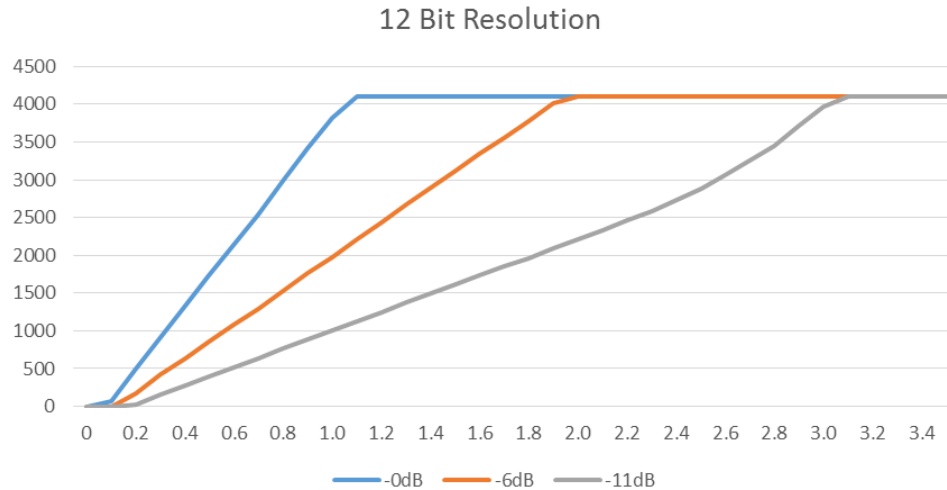


Figura 51. Comparativa de linealidad del ADC para una resolución de 12 bits

En la hoja de datos “esp32-s3 series” se especifican los rangos efectivos para cada nivel de atenuación disponible para el canal ADC:

Symbol	Parameter	Min	Max	Unit
DNL (Differential nonlinearity) ¹	ADC connected to an external 100 nF capacitor; DC signal input;	-4	4	LSB
INL (Integral nonlinearity)	Ambient temperature at 25 °C; Wi-Fi off	-8	8	LSB
Sampling rate	—	—	100	kSPS ²
Effective Range	ATTEN0	0	850	mV
	ATTEN1	0	1100	mV
	ATTEN2	0	1600	mV
	ATTEN3	0	2900	mV

Tabla 4. “ADC Characteristics” extraído del documento “ESP32-S3 series”

Para una atenuación de 6 dB (ATTEN 2) el rango efectivo variará de 0 a 1,6V por lo que el divisor anterior debe poner como máximo esa tensión de salida en caso de que la entrada sea de 4,2V. Con la disposición de R13 y R14 se tendrían 1500mV y 1321mV de salida para 4,2 y 3,7 de entrada respectivamente. Al encontrarnos dentro del rango efectivo podemos asegurar que la respuesta sea aproximadamente lineal y no sature al valor máximo.

El orden de magnitud de las resistencias ha sido elegido llegando a un compromiso entre:

-Consumo de corriente: 11,79μA para 100K/180K o 117,86 μA para 10K/18K

-Deriva de los componentes: Valores muy altos de resistencia aumentan el nivel de ruido (preferible la opción 10K/18K frente a 100K/180K).

-La resistencia interna del pin del ADC que se vería en paralelo con R14 debe de ser de un valor mucho mayor. Esta resistencia es del orden de decenas de M Ω (no hay información específica en la hoja de datos)

No se tienen en cuenta el tiempo de carga del condensador externo ni del condensador de muestreo y retención (S&H) dado que la frecuencia de medición será muy baja (cada varios segundos).

Calidad de las medidas:

El convertidor interno del ESP32 no es un convertidor de gran calidad más bien es impreciso e inestable. El ADC2 no se puede usar mientras el módulo RF esté activo y aunque en los manuales y datasheets se indique el uso del ADC1 durante el uso del módulo RF los resultados son considerablemente inestables. Para mejorar las mediciones se opta por la técnica de multimuestra y añadir un condensador de bypass (100nF de valor recomendado por la guía de diseño [47]). En la siguiente imagen se ejemplifica gráficamente la comparativa de ruido en la medición del ADC ante 3 posibles casos, sin condensador, con condensador y finalmente añadiendo condensador y utilizando multimuestreo (64 muestras promediadas) a la vez.

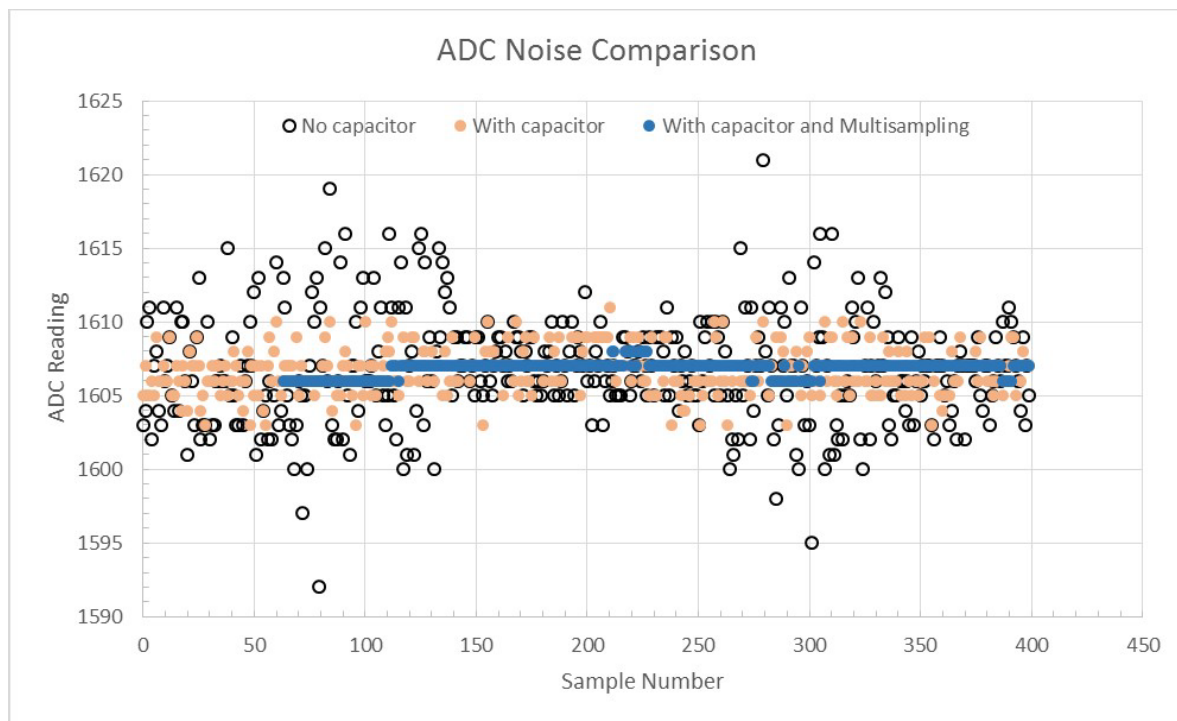


Figura 52. Comparativa de mitigación de ruido del ADC. Análisis ofrecido por Espressif [48]

Para mejorar aún más la medida puede optarse por desconectar por completo el módulo de comunicaciones cada cierta cantidad de tiempo o a voluntad del usuario a través de la web de configuración de la ESAI. El inconveniente de realizarlo periódicamente de forma autónoma es que reconectarse al WiFi y posteriormente al broker es un proceso que puede llegar a tardar varios segundos y durante ese tiempo podría ocurrir que se perdiera la detección de un evento lo cual supondría una falla en la seguridad. Este problema condicionará también la utilización de los recursos que cuenta el EPS32 para optimizar el consumo de corriente tal y como se verá con más detalle en la sección de “Consideraciones Adicionales > Ahorro energético”.

Una mejor alternativa en cuanto a calidad es utilizar circuitos integrados de bajo consumo como el LC709203F. Este CI es indicador del nivel de batería y estado relativo de baterías de ion de litio utilizando un algoritmo llamado 'HG-CVR'. Este algoritmo elimina el uso de una resistencia de sensado y proporciona información precisa incluso en condiciones inestables (temperatura, carga, envejecimiento y auto descarga). Cuenta con interfaz de comunicación I2C para ser programada y transmitir la información relativa de la carga.

Consumo:

$$I = 11,79 \mu A$$

4.2.9 PINES DE SALIDA DE LA PLACA

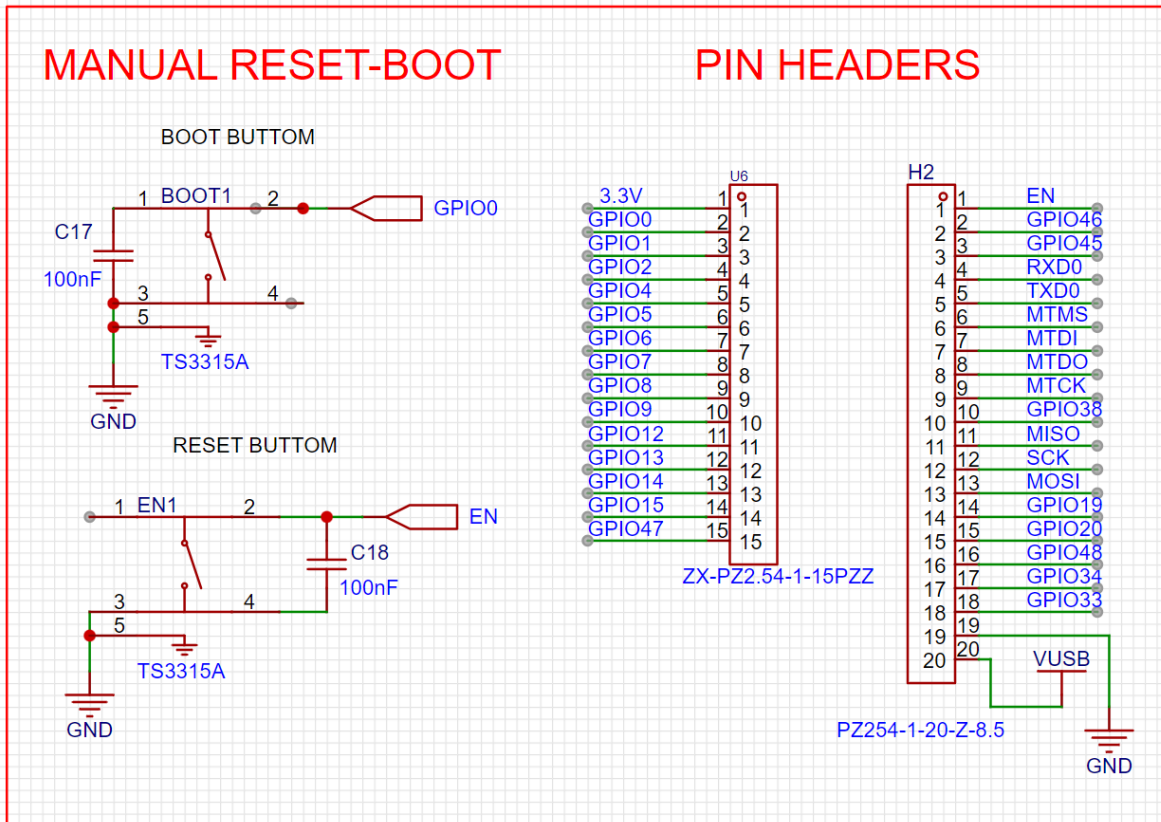


Figura 53. Botones de control y pines de salida de la placa

La placa dispone de 35 pines de salidas disponibles, de los cuales 3 de ellos están destinados a la alimentación y el resto pueden utilizarse como pines de entrada/salida de propósito general o bien para otras funciones que están indicadas mediante serigrafía en la placa PCB.

-Los pines 11-13 (MISO, SCK, MOSI) están indicados para usarlos en comunicación SPI (por ejemplo, para leer o escribir datos en una memoria). El pin de selección CS puede ser cualquier otro pin de propósito general.

-Los pines 6-9 (MTMS, MTDI, MTDO, MTCK) son los pines reservados para la interfaz JTAG. Dado que el chip ESP32-S3 tiene integrado un transceptor JTAG no harán falta componentes ni placas adicionales que sirvan de puente adaptador entre la computadora y el microcontrolador, simplemente el conector JTAG.

-Los pines 4-5 (RXD0, TXD0) se puede utilizar para realizar una conexión síncrona con otro componente o dispositivo. Estos pines son los que se utilizan para la comunicación a través del USB 1 por lo que su uso debería limitarse a cuando se utiliza el USB-OTG.

4.3 Diseño de la PCB

4.3.1 Consideraciones sobre EMI. Teoría y métodos de mejora prácticos

Tras haber explicado en profundidad en el anterior apartado cada uno de los elementos que operan en el sistema se procederá ahora a su disposición e interconexión sobre la placa de circuito impreso. Este nuevo apartado se va a abordar siguiendo las pautas y procedimientos para lograr un diseño lo más limpio y robusto posible frente a las interferencias electromagnéticas o EMI (Electromagnetic Interference), explicando brevemente las principales formas en que estas interferencias se manifiestan y las soluciones o precauciones que se han tomado a lo largo del diseño y sin necesidad del uso de fórmulas complejas o herramientas de simulación.

Anteriormente se abordó el fenómeno de la reflexión de la señal a un nivel de abstracción superior y podríamos cubrir detalladamente todas las precauciones y medidas a nivel de PCB dentro del módulo ESP32-S3-MINI-1 de Espressif donde de hecho la EMI y la EMC (compatibilidad electromagnética) son más críticas debido a que dentro se encuentra el chip de radiofrecuencia, el oscilador de 40MHz y la línea de alimentación de la antena pero sería innecesario pues ya vienen recogidas en la guía de diseño de la serie ESP32-S3 por lo que nos centraremos en las medidas para el resto del circuito apoyados en la guía High speed layout Guidelines de Texas Instruments [49].

-Trazado de pistas:

- Evitar ángulos rectos de 90° ya que puede causar la emisión de más radiación electromagnética. La capacitancia aumenta en la región de las esquinas y la impedancia característica de la pista cambia y esto genera reflexiones. En su lugar es recomendable usar cambios de dirección de como máximo 45° y en caso de señales de muy alta frecuencia es preferible realizar arcos curvos para evitar al máximo las fluctuaciones de la impedancia de la pista como en el caso de la línea de transmisión de la antena PCB al pin LNA_IN del ESP32-S3.
- Separar las señales de alta velocidad de las señales de baja velocidad y las señales digitales de las analógicas. Por ejemplo, en nuestra placa la señal de RF o las señales de reloj BCLK y WS (LRCLK) del micrófono digital deben estar separadas de las pistas de alimentación o las de puertos de propósito general.

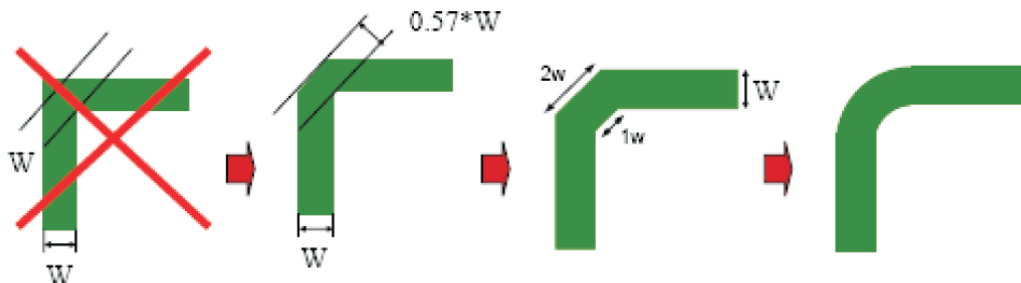


Figura 54. Trazado de pistas de peor a mejor. [49]

-Bucles de pistas y bucles de tierra (loop areas)

Una de las fuentes más común de ruido electromagnético son los bucles generados entre las pistas y sus corrientes de retorno. La interferencia se produce debido al acople por inducción electromagnética descrita por la ley de Faraday.

$$\varepsilon = - \frac{d}{dt} \int_S \vec{B} \cdot d\vec{S}$$

De la expresión anterior se desprende que la inducción se produce cuando un campo magnético fluctuante atraviesa la superficie de un conductor cerrado (bucle) y será proporcional a la variación temporal del número de líneas de campo magnético que atraviesen la superficie. La variación de este flujo magnético dependerá también del tamaño del área que es atravesado por las líneas de campo. La f.e.m (fuerza electromotriz) generada producirá una corriente que se opondrá a la variación del campo magnético variable que la produce. De manera inversa cambios de polaridad en la corriente que circulen por el bucle generarán un campo magnético variable que a su vez inducirá a conductores y otros bucles cercanos.

Por lo tanto, podemos concluir que los bucles deberán reducirse lo máximo posible especialmente cuando se traten de señales de alta frecuencia. La manera más sencilla de hacerlo es haciendo que la corriente de retorno esté lo más próximo a la fuente original.

Como diseñadores tenemos que controlar estos caminos de retornos para disminuir estos efectos indeseados. Comúnmente se suele controlar los caminos de más baja resistencia, pues es más sencillo y directo calcular la resistencia de una pista en función de su grosor, anchura y longitud. Y efectivamente éste será el camino de regreso de las señales de baja frecuencia sin embargo las señales de alta velocidad de conmutación, que además serán las que causen más estas interferencias no van a seguir el camino de menor resistencia sino el de menor impedancia y éste camino puede no ser tan evidente ni sencillo de controlar.

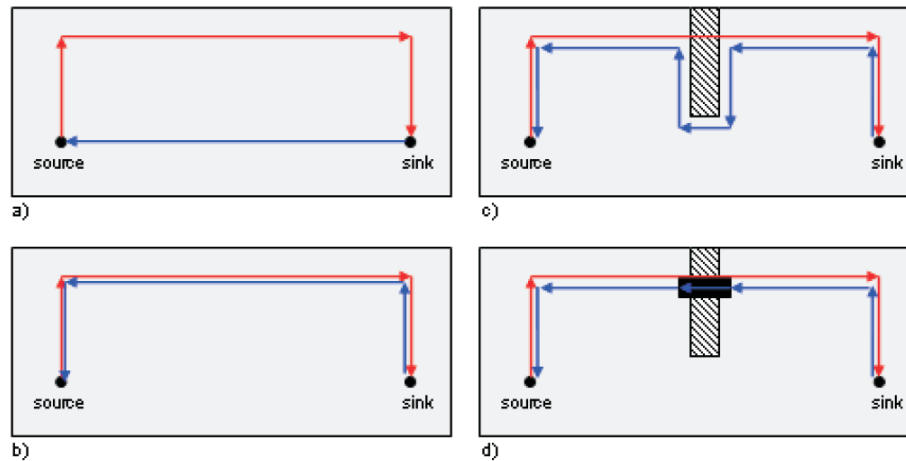


Figura 55. Ejemplos de bucles de área malos(a,c) y mejores(b,d). [49]

En la figura anterior vemos 2 ejemplos en los que tenemos una mala disposición de la corriente de retorno y en los que se crean bucles que serán fuentes potenciales de ruido. En el caso a-b se ha logrado reducir el bucle disminuyendo la distancia del camino de retorno de la señal original. En el supuesto c el camino de retorno está a una buena distancia próxima a la fuente, pero justo en medio se encuentra una ranura en la placa (zona sin cobre) y por lo tanto la corriente tiene que rodearla creando así un bucle. Si la ranura no es salvable la mejor solución en esos casos es colocar una resistencia de $0\ \Omega$ que haga de puente.

Otra característica de estos bucles es que actuarán también como antenas trasmisoras y receptoras de ondas electromagnéticas pudiendo perturbar componentes o dispositivos cercanos o por el contrario ser perturbados por una fuente radiante de radiofrecuencia (por ejemplo, el módulo ESP32).

A su vez tenemos el problema de los bucles de masa. En condiciones ideales todos los componentes del circuito tienen como referencia los 0V de tierra. A pesar de lo anterior, en la realidad puede darse el caso de que existan diferencias de potencial entre las conexiones a tierra de 2 o más componentes debido a las capacidades e inductancias parásitas, los cambios de impedancias de las pistas, por acoplamiento inductivo o radiado de un sistema perturbador etc. Al existir una diferencia de potencial y un camino entre ambos puntos se generará un bucle por el que circulará una corriente parasitaria que podrá afectar al resto del circuito tal como hemos descrito anteriormente. Estas corrientes de bucles de masa son una de las causas del ruido que se puede escuchar en un equipo de sonido cuyo diseño haya sido descuidado o mediocre.

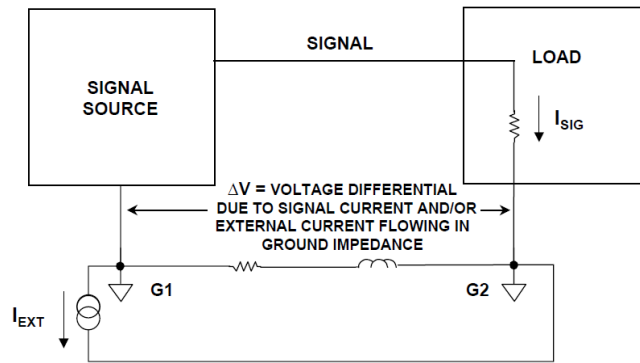


Figura 56. Corriente generada por diferencia de potencial en las tierras [50]

Una solución para el problema de los bucles es la creación de planos de masas. En nuestro caso se ha reservado una capa entera para el plano de masa (capa inferior) y en la capa superior los componentes están rodeados de otro plano. A su vez ambos planos están interconectados a través de múltiples vías. La ventaja de usar planos de masa es que al tener una conexión a tierra sólida y de muy baja impedancia, se minimiza la posibilidad de que se desarrollen diferencias de potencial significativas entre diferentes partes del circuito. Un plano de masa continuo ayuda a minimizar las áreas de bucle ya que las corrientes de señal y sus retornos pueden seguir trayectorias cercanas y paralelas en el plano de masa, disminuyendo drásticamente la posibilidad de grandes áreas de bucle y como consecuencia se reduce la inductancia en los caminos de retorno de la corriente. Cuanto más reducido sea el espaciado del plano de masa con las señales mejor, aunque en determinado punto tenemos que llegar a un compromiso por las limitaciones técnicas de la fabricación de la placa.

-Diafonía(crosstalk)

El fenómeno de diafonía es cuando dos trazas cercanas y paralelas se influyen o perturban mutuamente. Una de ellas se llama el agresor (portadora de la señal) y la otra se llama la víctima (traza afectada por el agresor). El acoplamiento de diafonía puede darse por acoplamiento inductivo, capacitivo o por las rutas de impedancia de tierra común. En general se da una combinación de los tres casos. Estos acoplamientos generan una corriente directa y otra inversa en la traza de la víctima, mientras que en un entorno homogéneo (por ejemplo, con pistas tipo stripline), las dos corrientes inducidas se cancelan entre sí. En un entorno de microstrip (tipo de pista usado en nuestro diseño), la corriente del acoplamiento inductivo tiende a ser mayor que la influencia del acoplamiento capacitivo. Por suerte el plano de tierra puede ser de gran utilidad para mitigar también los efectos de la diafonía ya que reduce la impedancia de la masa común alrededor de 40-70 dB en condiciones ideales. En la siguiente figura se ilustra el fenómeno de crosstalk entre dos trazas de tipo microstrip:

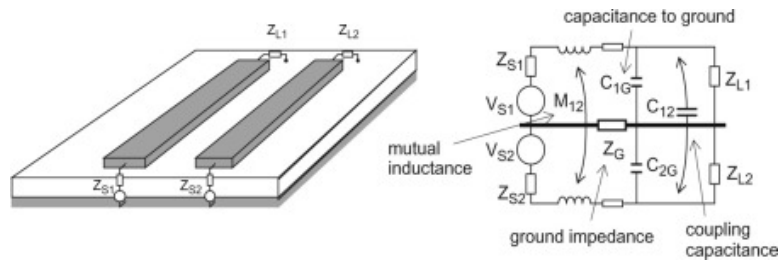


Figura 57. Circuito equivalente de diafonía [51]

El plano de tierra también puede reducir el acoplamiento de inductancia mutua al garantizar que los bucles de corriente acoplados no sean coplanares, es decir estén entre distintas capas. El acoplamiento capacitivo C_{12} no será directamente afectado por el plano de tierra, pero la reducción general de la impedancia de la traza (equivalente a decir que C_{1G} y C_{2G} han aumentado) reducirá la amplitud de la diafonía capacitiva.

Otra técnica para minimizar los efectos de la diafonía en trazas adyacentes es mantener al menos a una distancia de 2 veces el ancho de la traza. La diafonía puede ocurrir también entre señales de distintas capas adyacentes que se crucen. En ese caso lo mejor es que las señales se enruten con un ángulo de 90 grados entre sí. La siguiente figura muestra un caso práctico aplicado en la placa de la ESAI donde la señal BCLK se cruza entre capas con las señales de LRCLK y SD. Nótese que la distancia entre pistas cumple la sugerencia de espaciado de 2 veces el ancho de la pista.

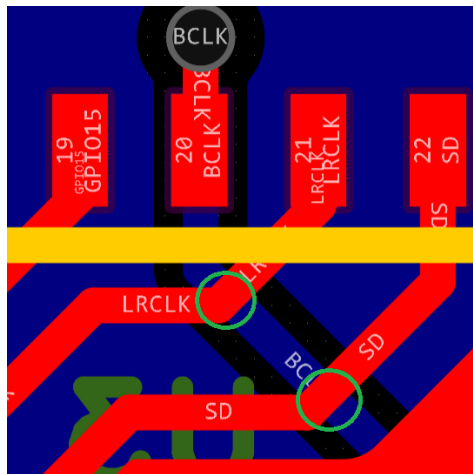


Figura 58. Buena praxis de diseño para evitar diafonía entre señales. Fragmento de PCB de la ESAI

-Vías

Situar vías u orificios pasantes en la PCB es una opción indispensable para permitir el trazado de las conexiones entre los diversos componentes. Además, las vías permiten conectar los planos de masas de la capa superior e inferior. Cuantas más conexiones haya mejor será la calidad del plano la tierra a nivel general ya que si ambos planos estuvieran conectados por una sola vía, la impedancia se vería condicionada por ese “cuello de botella”. Es buena costumbre añadir vías entre los planos repartidos a lo largo de la PCB especialmente cerca de los condensadores de desacoplo que acompañan los pines de alimentación de los distintos integrados. No obstante, hay que tener cuidado de no excederse en la cantidad de orificios pues éstos añaden una capacidad e inductancia adicional al circuito y en señales de alta frecuencia pueden llegar a ocurrir reflexiones por el cambio de impedancia. Cuando se colocan varias vías adyacentes, por ejemplo, en un bus de señales que cambia de capa, es importante la disposición y orientación para evitar que las corrientes de retorno no creen bucles.

Una consideración especial en nuestro diseño respecto a las vías es el orificio que debe tener el micrófono MEMS para que la entrada de las ondas de presión sonora en el diafragma se produzca de forma óptima. Si nos vamos al apartado de “PCB design and land pattern layout” de la hoja de datos del ICS-43434 el fabricante nos recomienda que el diámetro del orificio de la PCB debería ser mayor que el orificio del puerto de sonido del micrófono con un diámetro mínimo recomendado de 0.5 mm. Las dimensiones del integrado son las siguientes:

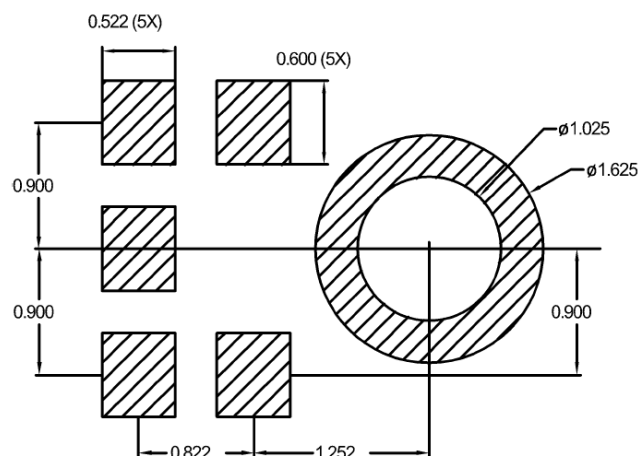


Figura 59. ICS-43434 datasheet. Distribución de los pads y dimensiones en mm

Hemos elegido para nuestro diseño un orificio con un diámetro exterior e interior de 1,3 y 1.25 mm respectivamente.

Otra consideración a tener en cuenta es la longitud de este orificio ya que todo puerto acústico tendrá su propia frecuencia de resonancia. El orificio practicado en la PCB que conecta con el puerto del micrófono se puede categorizar como una columna de aire de tipo cilindro cerrado. Esta frecuencia de resonancia dependerá de la longitud de la columna de aire (cuanto mayor sea la longitud menor será la frecuencia de resonancia) y de la velocidad del sonido en el aire para la temperatura dada. Podemos encontrar en la web de “HyperPhysics” información técnica al respecto (recomendable visitar) y una calculadora online para determinar la frecuencia de resonancia para los distintos tipos de columnas de aire [52]. Para el caso de nuestro cilindro cerrado la longitud de la columna se corresponderá con $\frac{1}{4}$ de la longitud de onda de la frecuencia de resonancia. Suponiendo un espesor de PCB estándar de 1.6 mm el armónico resonante fundamental será:

$$f_1 = \frac{V_{sonido}}{4 L_{vias}} = \frac{343,7}{4 \cdot 0,0016} = 53703,125 \text{ Hz}$$

El límite de espesor aconsejable para la PCB sería de 4,3 mm ya que supondría una frecuencia de resonancia de 20 KHz, correspondiente a la banda de audio.

-Condensadores de desacoplo:

Los condensadores de desacoplo también llamados condensadores de bypass son uno de los elementos más fundamentales y básicos en la mejora de la calidad y rendimiento de casi cualquier circuito electrónico independientemente de si se trabajan con señales de frecuencia alta o baja. Este subapartado se va a presentar de la mano de 3 expresiones archiconocidas en el mundo de la electrónica y aunque básicas son la base subyacente de las técnicas y análisis de mejora del ruido eléctrico en sistemas electrónicos:

$$V_L(t) = L \frac{di(t)}{dt} \quad (1) \quad X_L(\omega) = j\omega L \quad (2) \quad X_C(\omega) = \frac{1}{j\omega C} \quad (3)$$

Los condensadores de desacoplo se conectan entre los pines de alimentación de los circuitos integrados y tierra por un lado para actuar como pequeños reservorios de carga y por otra parte para suprimir o filtrar el ruido eléctrico que se manifiesta en forma de variaciones o picos de tensión, comúnmente denominado rizado (ripple). Este rizado de tensión respecto al valor nominal esperado en la línea de alimentación se debe principalmente a los efectos de las inductancias parásitas cuando se producen rápidas variaciones de corriente (1) aunque también pueden estar generados por los tipos de

acoplamientos de interferencia electromagnéticas tratados en los subapartados anteriores.

Estos efectos escalan con la longitud de la pista (a mayor longitud mayor inductancia parásita total) y con la frecuencia (1). Por esto, los condensadores deben de colocarse lo más cerca posible del pin y evitar que haya la mínima longitud de pista entre ellos y el pin a proteger. Esto es especialmente relevante en aquellos condensadores de bypass que tengan los menores valores de capacidad, ya que estos condensadores se utilizan para filtrar los ruidos de mayor frecuencia y como hemos mencionado los efectos inductivos escalan con la velocidad de cambio de la corriente.

La manera en la que estos condensadores suprimen el ruido de alta frecuencia es ofreciendo un camino de baja impedancia (3) hacia tierra (de ahí lo de bypass) a la corriente de alta frecuencia superpuesta sin afectar a la vez a la señal de baja frecuencia que comúnmente es la señal CC de alimentación. Idílicamente un condensador ofrecerá cada vez una impedancia más baja a medida que la frecuencia aumenta hasta llegar a cero. Sin embargo, como siempre sucede cuando te separas del mundo ideal teórico, en la vida real no ocurre eso. De hecho, a muy altas frecuencias los condensadores parecen tener un valor de impedancia mucho mayor a lo esperado. ¿A qué se debe esto? Para entenderlo tenemos que ver el modelo equivalente real de un condensador:

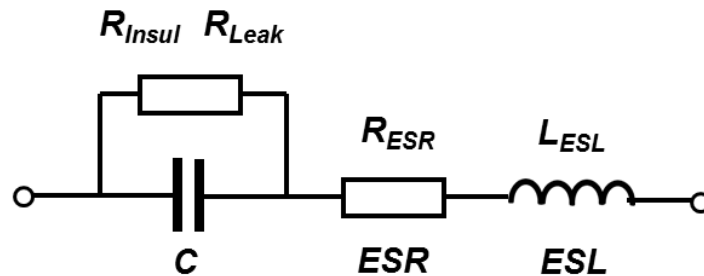


Figura 60. Modelo equivalente de un condensador [53]

Lo que tenemos en realidad es un circuito RLC formado por el condensador más la resistencia e inductancia parásita. Por lo tanto, cada condensador de bypass se comporta realmente como un circuito resonante y como consecuencia su característica capacitiva sólo tendrá validez hasta alcanzar su frecuencia de resonancia, comúnmente llamada SRF de las siglas “Self Resonant Frequency”. A partir de ahí los efectos inductivos (2) dominarán, provocando que pierda su función como deriva a tierra.

Para contrarrestar este problema podemos optar por el uso de varios condensadores de distintos valores para lograr una baja impedancia para un mayor rango de frecuencias. Por una parte, los condensadores con valores de capacidad más alta ofrecerán una impedancia más pequeña para bajas frecuencias (pero alcanzará su punto de SRF antes). Por la otra, los de menor valor tendrán una mayor impedancia para el rango anterior pero

servirán para ofrecer una baja impedancia para el rango de frecuencias más altas ya que tienen el punto SRF desplazado. La diferencia entre las SRF se debe a que en los condensadores más grandes en capacidad también tienen generalmente una resistencia e inductancia en serie mayor y viceversa y esto afecta a la frecuencia de resonancia:

$$f_r = \frac{1}{\sqrt{LC}} \Rightarrow \text{Si } L \uparrow \text{ o } C \uparrow \text{ entonces } SRF \downarrow \text{ y viceversa}$$

En la figura siguiente se muestra la impedancia de un arreglo de distintos condensadores cerámicos de multicapa, tanto la contribución de cada uno de ellos por separado como de la impedancia equivalente del paralelo de todos ellos:

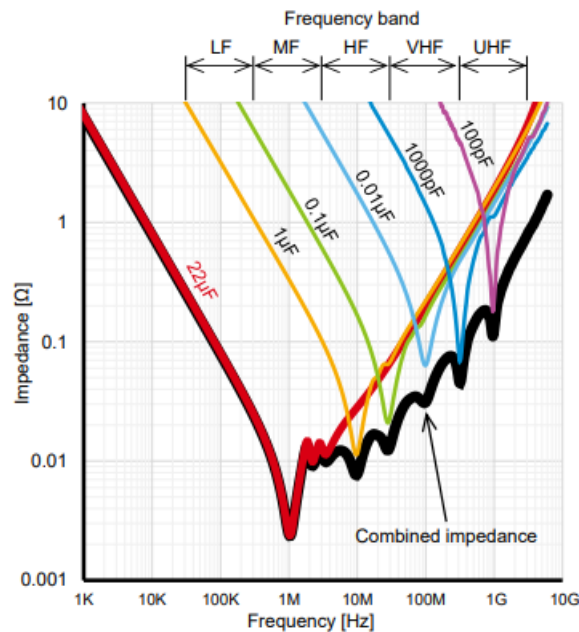


Figura 61. Impedancia característica de un arreglo de distintos MLCC [54]

Vemos como la impedancia combinada de todos los condensadores logra “coger” lo mejor de cada tipo de condensador manteniendo una impedancia baja a lo largo de varias bandas de frecuencia. Con todo, usar este método tiene un problema que no es evidente a priori y del cual a día de hoy sigue existiendo debate al respecto. Si nos fijamos entre frecuencias intermedias a las SRF de los distintos condensadores encontramos picos de impedancia de manera que la impedancia aumenta en lugar de disminuir, y la interferencia electromagnética (EMI) empeora. A este fenómeno se le denomina antirresonancia y ocurre en el punto en el que se cruzan la característica capacitiva y la característica inductiva de los distintos condensadores adyacentes en valor.

Otra técnica en la forma de implementar los condensadores de bypass es la de usar múltiples condensadores, pero de igual valor en este caso. Al estar en paralelo logramos aumentar la capacidad y a la vez reducir la ESR y la ESL del arreglo. Por ejemplo si añadimos 3 condensadores iguales en paralelo la capacidad total será el triple y la inductancia $1/3$, si ponemos 4 condensadores tendremos 4 veces la C y $1/4$ de ESL y así sucesivamente. Como resultado podemos lograr con condensadores más pequeños una impedancia total más baja y todo esto sin modificar la frecuencia de resonancia. Este método está ganando popularidad en los últimos años ya que la tecnología SMD ha avanzado tanto que es posible encapsular condensadores con tamaños pequeños y resistencias e inductancias en serie de bajo valor para una capacidad relativamente grande.

No obstante, nada impide hacer uso de ambas técnicas a la vez, es decir utilizar varios grupos de condensadores de distintos valores de capacidad. En la figura de abajo tenemos una experiencia realizada por el conocido experto y consultor técnico en diseño de PCB Zachariah Peterson en el que simula dos casos [55]. En el primer caso se simula la impedancia equivalente de 4 condensadores de distintas capacidades, a saber: C_1 , C_2 , C_3 , C_4 representados por la línea continua azul y por otro lado la impedancia equivalente de 4 grupos de condensadores con valores idénticos a los anteriores (C_1 - C_4) con 5 condensadores por cada grupo (20 en total).

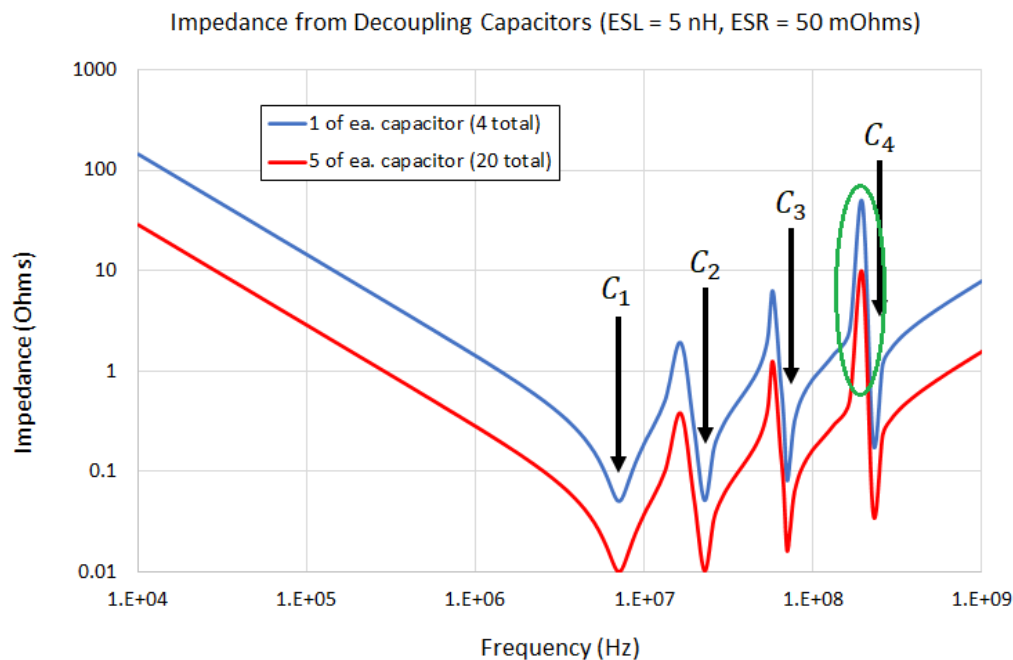


Figura 62. Resultado de usar un arreglo de condensadores distintos frente a un arreglo de condensadores distintos repetidos múltiples veces.

La ESR y ESL se han mantenido constante para todos los componentes. Tal como se esperaba la impedancia que refleja la línea roja logra ser significativamente más baja que

la azul. Sin embargo, en este experimento la impedancia alcanzada en las regiones de antirresonancia son bastante grandes (círculo verde) alcanzando valores de impedancia de hasta 10 o 70 ohms para la línea roja y azul respectivamente. La única manera de tener una curva de impedancia más estable sería añadir condensadores de valores intermedios en las zonas de antirresonancia que sean de interés.

A pesar de todo, aunque se intenta modelar de forma realista los condensadores, sigue siendo una abstracción idílica alejada de los valores reales y de los múltiples efectos parásitos de una PCB real. De contar con los recursos, la mejor solución pasa por analizar y ensayar en el laboratorio la naturaleza del ruido eléctrico que interesa filtrar y la respuesta en frecuencia de varios prototipos con las diversas topologías o técnicas mencionadas de supresión de ruido. En general es indiscutible que el empleo de múltiples condensadores de igual valor una buena práctica, aunque tiene como limitación e inconveniente el espacio y coste que implica aumentar el número de componentes.

Para el diseño del prototipo de la ESAI se ha optado por utilizar como medio de desacople dos condensadores de distinto tamaño (10uF o 4,7uF y 100nF) en todos los ICs excepto en el micrófono que se ha usado un grupo de 100nF y 100pF para prestar especial atención al ruido generado por el módulo de radiofrecuencia.

-Disposición de antena:

Se debe prestar atención al cómo disponemos el módulo en la placa base. Para minimizar la interferencia de la placa base en el rendimiento de la antena del módulo se sugiere colocar la antena PCB incorporada en el módulo **fuera de la placa base**, con el punto de **alimentación de la antena lo más cercano posible a la placa**. De no poderse colocar la antena fuera de la placa base, se debe dejar un espacio libre de al menos 15 mm ("clearance area"). La imagen siguiente de la guía de diseño de la serie ESP32-S3 contiene ejemplos de posiciones adecuadas marcadas con ✓ mientras que las posiciones sin marca no son recomendadas.

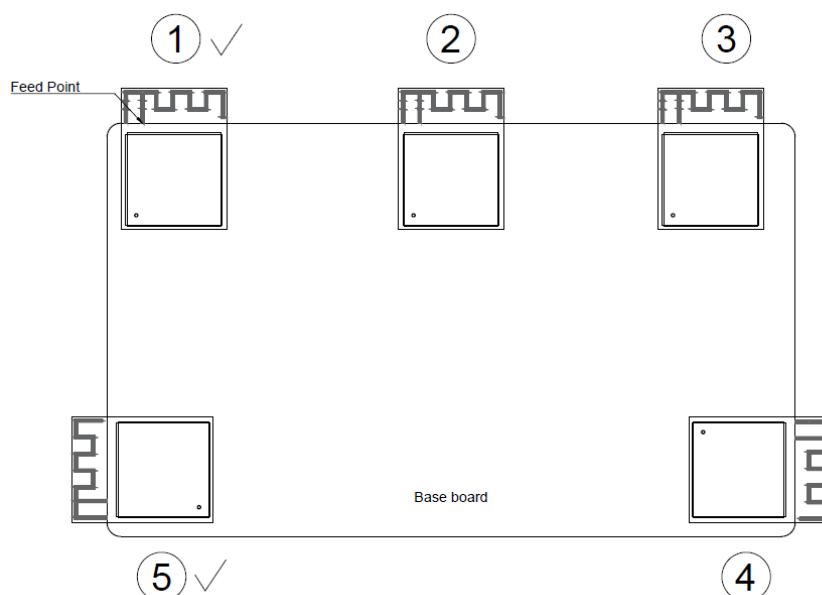


Figura 63. Guía de posicionamiento del módulo con antena PCB.
Guía de diseño del ESP32-S3

4.3.2 Vista de capas

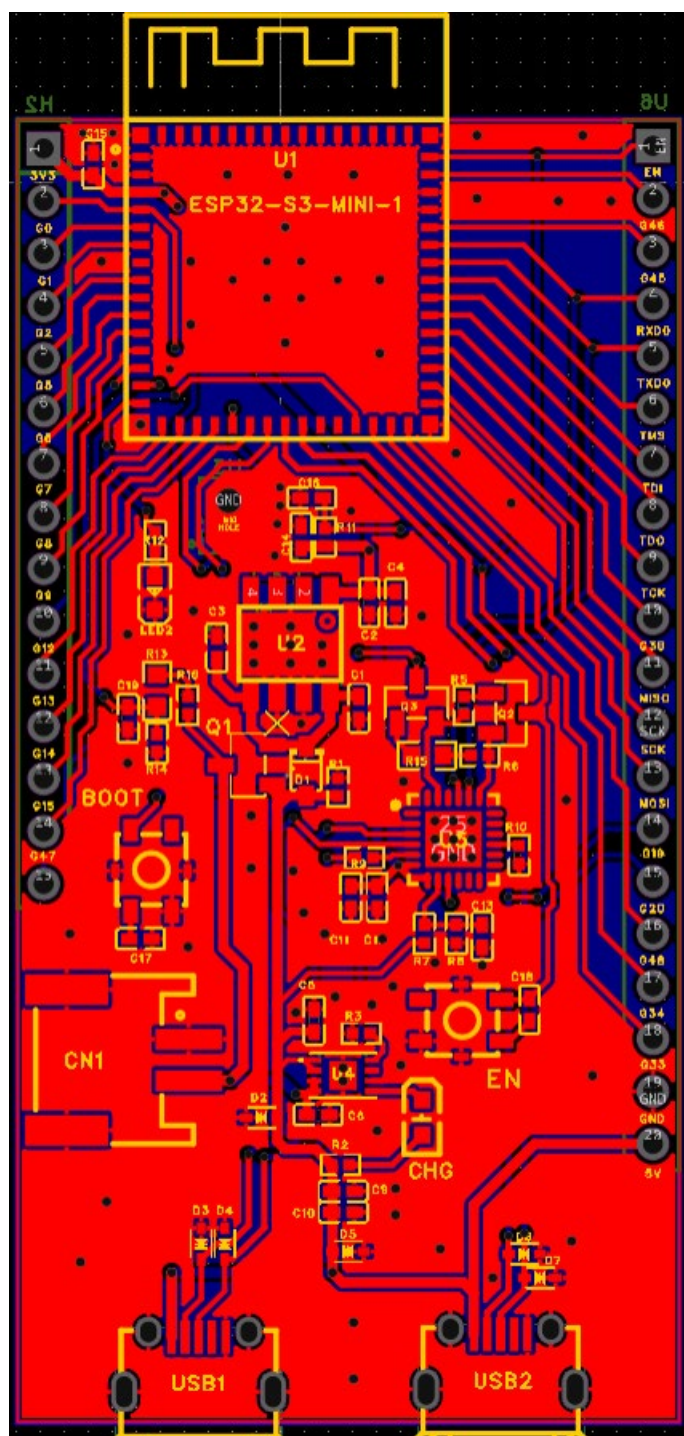


Figura 64. PCB de la ESAI. Capa superior

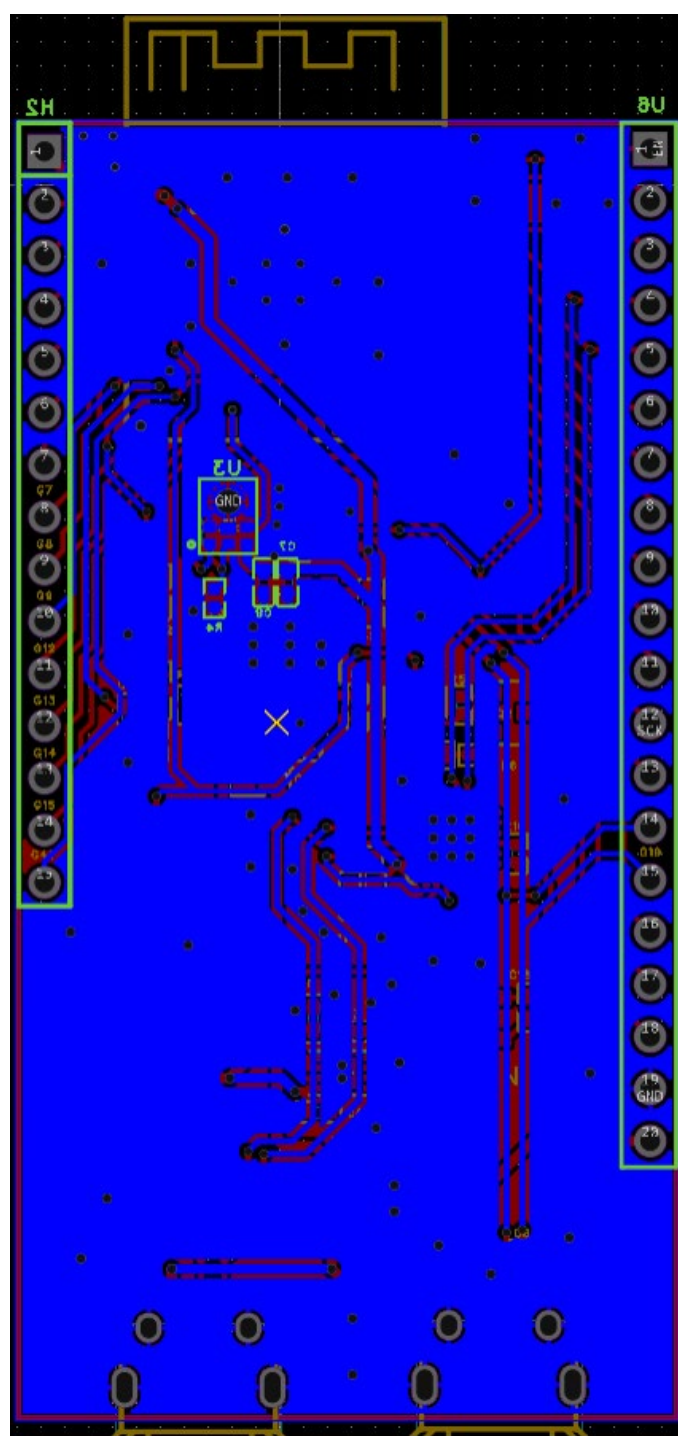


Figura 65. PCB de la ESAI. Capa inferior

4.3.3 Información técnica de la PCB:

- Material: FR-4
- Dimensiones: 63 x 32 mm
- Acabado superficial externo de cobre: 1 oz (35 μm)
- Ancho de pistas:
 - Pistas de señales: 0,254 mm (10 mil)
 - Pistas de alimentación: 0,508 mm (20 mil)
- Nº de capas: 2
- Espesor de placa: 1,6 mm

4.4 Consumo:

4.4.1 Consumo de corriente teórico aproximado

Sumando la contribución de consumo de todos los componentes calculado en el apartado de esquemático tendremos una aproximación del consumo real que tendrá el dispositivo.

$$I_{placa} = 100m + 90\mu + 200\mu + 300\mu + 171,84\mu + 6\mu + 11,79\mu = 100,78 \text{ mA}$$

Mientras que el ESP32 suma él solo 100mA el resto del circuito consume 0,78 mA aproximadamente. En una aplicación en la que el ESP32 pudiera ponerse en estados de sueño profundo o hibernación, se podrían utilizar MOSFETS como interruptores podría limitarse el consumo del MCP73831, ICS-43434, CP2102 y del monitor de batería.

Con los datos anteriores, con una batería de ion de litio de 1000 mA la ESAI podía operar durante casi 10 horas seguidas. Este valor es una mera aproximación.

De igual modo que se comentó en el apartado de consumo de la placa ESP32 DOIT DEVKIT V1, deben hacerse pruebas de potencia con las herramientas adecuadas para obtener un valor más realista y observar la forma y distribución de la señal de la corriente consumida.

4.4.2 Consideraciones adicionales de ahorro energético

El ESP32 cuenta con distintos modos para reducir el consumo de corriente a saber: modem-sleep, light-sleep y deep-sleep. El modo de modem-sleep se utiliza para reducir el consumo generado por el módulo de radiofrecuencia. A pesar de la confusión que existe con este modo, realmente consiste en apagar y encender el módulo durante intervalos de tiempo programables sin más. Los modos de sueño ligero y profundo a priori no son una opción interesante para la forma de operar de la ESAI. El salir del modo light o deep-sleep supone unos 20-40 ms. A eso habría que añadirle el tiempo que tarda el micrófono en estar operativo en caso de haber entrada en modo sueño (20 ms). 80 ms en el peor de los casos no es algo crítico. Lo que sí es un problema es el tiempo que tarda el ESP32 en conectarse al punto de acceso y posteriormente realizar la conexión con el bróker, que puede llegar a tardar varios segundos. Para nuestro caso eso es una cantidad de tiempo abismal, que podría llevar a no detectar eventos importantes durante estas transiciones.

Otra posibilidad es poner al ESP32 en otro estado paralelo de “bajo consumo” en el que se mantiene el módulo RF sólo recibiendo. Se sigue comunicando mediante I2S con el

MEMS cada cierto intervalo de tiempo (20 - 40 ms) realizando la captura de pequeñas muestras (buffer de 100-200) muestras, que a una SR de 16KHz supone unos 6,25 – 12,5 ms. Después se promediaría el valor de cada componente del buffer y si se supera un umbral, saldría del modo de “bajo consumo”. Aun así, en todo momento en todo momento el módulo RF estaría activo, pero se ahorrarían los picos de consumo al transmitir los paquetes de audio, lo que a la larga podría ser una diferencia.

5 RESULTADOS

Aunque en la sección de Firmware e Introductoria vimos el funcionamiento del sistema por pasos de forma detallado en esta sección vamos a mostrar mediante pantallazos y fotografías al sistema ESAI en funcionamiento. Por desgracia, dado que el sistema está basado en el sonido, para disfrutar plenamente el resultado del proyecto debe verse en persona, contando con al menos una ESAI en escucha, otra en chivato y algún dispositivo con conectividad y salida de audio (móvil, portátil, tablet...) con el que poder navegar y hacer uso del portal captivo de la ESAI y la interfaz web del servidor DJANGO.

1. Alimentamos el primer nodo ESAI. Nos aparecerá una nueva red disponible. Cuando nos conectemos nos aparecerá una notificación de inicio de sesión en red necesario o directamente se nos abrirá una nueva pestaña con el portal captivo que está generando la ESAI en modo SoftAP para configurar las credenciales del AP que usaremos, el modo de la estación, los eventos a detectar y la búsqueda de actualizaciones:



Figura 66. Conexión desde móvil android con el SoftAP generado por la ESAI. Creado por el autor.

Por seguridad, aparte de la clave de la red se ha establecido que el punto de acceso virtual del ESP32 sólo admita una conexión a la vez, ya que sólo está pensada para que el usuario propietario de la estación pueda tener acceso a su configuración. En caso de que esto fuese un inconveniente, se puede modificar fácilmente cambiando el argumento de la función `WiFi.softAP()`.

The screenshot shows a web browser window with the address bar displaying 'msftconnecttest.com/redirect'. The page has a light blue background and a title 'Configuración de red'. Below the title, there are two text input fields: the first is labeled 'SSID:' and the second is labeled 'Clave:'. Below these fields is a green button with the text 'Enviar'.

Figura 67. Portal captivo, configuración de red. Creado por el autor

Una vez introducido credenciales válidas:

The screenshot shows a web browser window with the address bar displaying 'msftconnecttest.com/redirect'. The page has a light gray background. In the center, there is a white rectangular box containing a green message: 'El ingreso de las credenciales de red ha sido realizado con éxito.' Below this message is a green button with the text: 'Pulse aquí para seleccionar el modo de operación de la estación'.

Figura 68. Plantilla html con mensaje de éxito de la configuración de red. Creado por el autor

Configuración de la estación acústica inteligente

Modo de la estación:

☐

Modo escucha
(activado por defecto)

☐

Modo silencio

Eventos a detectar:

☒

Detección de roturas de cristales

☐

Monitor de bebé

☐

Detección de aves

☒

Detector de presencia humana

Comprobar actualizaciones: ☐

Finalizar

Figura 69. Página web del portal para la configuración de la ESAI. Creado por el autor

← → ↻
No es seguro
msftconnecttest.com

🔍
☆
📄
📱
🔊
⋮

Configuración de la Estación Acústica Inteligente:

-Estación en modo Escucha

-Detección de roturas de cristales.

-Detección de presencia humana(voz,tarareo,silbidos,estornudos,pisadas etc)

Si no ve nada aquí es que no se ha seleccionado ningún método. Si desea configurar de nuevo los modos de empleo puede utilizar el botón en el dispositivo o hacerlo a través de la web.

Figura 70. Plantilla html final del portal. Creado por el autor

A continuación, la estación saldrá del modo portal captivo y el módulo Wifi operará como estación. Intentará conectarse a la red, si no lo logra comenzará el proceso de reconexión y reinicio de AP visto anteriormente y si lo logra verificará si se ha solicitado la búsqueda de actualizaciones de firmware. Completado lo anterior la ESAI seguirá con todas las

demás inicializaciones hasta finalmente conectarse al broker y empezar a grabar y transmitir datos:

```
<!DOCTYPE html><html><head><meta name="viewport" content="width=device-width, initial-scale=1"><meta charset="UTF-8"> <link rel="icon" href="data:,"><title> Selección de modos</title><style>html { font-family: Arial; font-size:large; display: inline-block; margin: 0px auto; text-align: center; background-color: #f2f2f2;}.container { margin: 40px auto; padding: 20px; background-color: #fff; border: 1px solid #ddd; border-radius: 5px; }h1 { color: #2ecc71; }</style></head><body><div class="container"><h1>Configuración de la Estación Acústica Inteligente: </h1><p>-Estación en modo Escucha </p><br><p>-Detección de roturas de cristales.</p><br><p>-Detección de presencia humana(voz,tarareo,silbidos,estornudos,pisadas etc)</p><br><p style='color:#e74c3c;'> Si no ve nada aquí es que no se ha seleccionado ningún método. Si desea configurar de nuevo los modos de empleo puede utilizar el botón en el dispositivo o hacerlo a través de la web.</p><br></div></body></html>
```

```
SERVIDOR >:
Iniciando conexión con red: SSID:MOVISTAR-WIFI6-9E60 . Se ha realizado con éxito la conexión con la red.No se han encontrado actualizaciones disponibles.
Iniciando conexión con el broker
Conexión con el broker realizada con éxito.
T_mqtt: 550 us
T_mqtt: 412 us
T_mqtt: 423 us
T_mqtt: 426 us
T_mqtt: 396 us
T_mqtt: 412 us
T_mqtt: 432 us
T_mqtt: 415 us
T_mqtt: 432 us
```

Figura 71. Vista de la terminal serial de la ESAI. Creado por el autor

El proceso para la estación chivata será el mismo excepto que la debemos configurar como chivata:

Figura 72. Configuración de la estación en modo chivato. Creado por el autor

Nótese que cada estación ha seleccionado distintos eventos a detectar. Esto se ha hecho para poder mostrar que el sistema funciona correctamente y que cada salida se activa

cuando se debe de activar como se verá a continuación. La estación escucha (E) detectará roturas de cristales y presencia humana y la estación chivata (C) detectará: roturas de cristales y presencia de aves. La estación a la izquierda será la estación E, con un led amarillo como actuador de salida y la de la derecha la estación C, con un led blanco como salida.

La primera prueba es reproduciendo sonidos de rotura de cristal, esperando que se activen ambos. La segunda es un ave piando y cantando, esperando que se active sólo la estación C. Podríamos hacer más pruebas con las demás combinaciones de eventos, pero sería repetitivo y además ya se mostrará con más detalle en la presentación del proyecto. En el ordenador, tenemos abierta la web del servidor Django donde se muestra la inferencia del modelo en tiempo real y se reproduce el audio captado por la estación E.

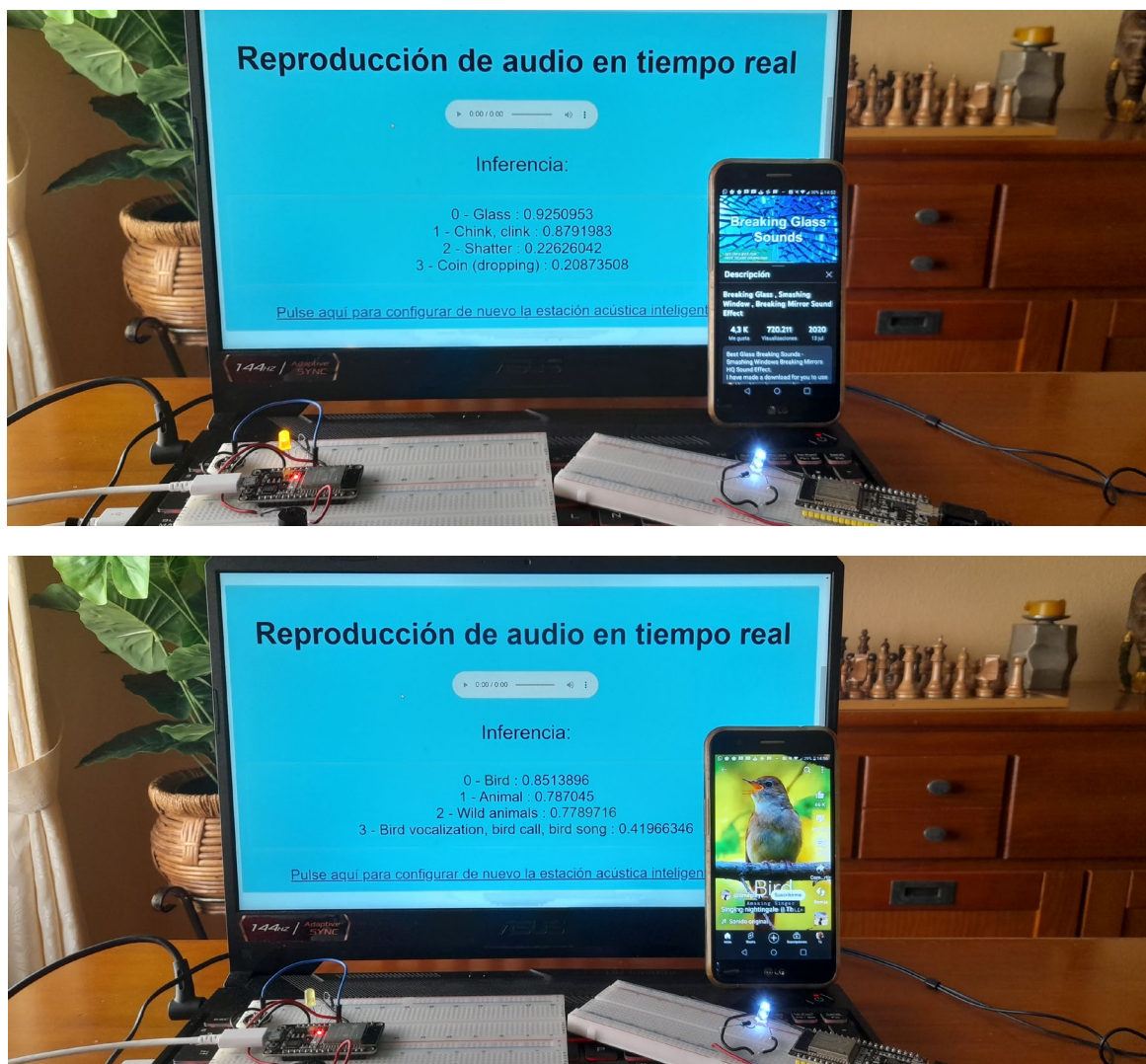


Figura 73. Pruebas realizadas para mostrar el funcionamiento total del sistema ESAI. Creado por el autor.

Si pinchamos en “Pulse aquí para configurar de nuevo la estación acústica inteligente” nos lleva a la web de configuración que es idéntica a la que se ofrecía en el portal captivo salvo por la opción de poder reiniciar el AP y elegir entre nodos ESAI:

The image shows a web configuration interface for an acoustic station (ESAI) with a light blue background. It contains several sections with colored buttons and checkboxes:

- Modo de la estación:** A section with two buttons: a pink button labeled "Modo escucha (activado por defecto)" and a blue button labeled "Modo silencio". There is a small white square checkbox to the left of the pink button.
- Eventos a detectar:** A section with four buttons: a red button "Detección de roturas de cristales", a purple button "Monitor de bebé", a green button "Detección de aves", and an orange button "Detector de presencia humana". There are small white square checkboxes between the buttons.
- Estaciones a configurar:** A section with two blue buttons: "ESAI 1" and "ESAI 2". The "ESAI 2" button has a small blue square checkbox with a white checkmark inside it. There is a small white square checkbox to the left of the "ESAI 1" button.
- Footer:** Two checkboxes at the bottom: "Comprobar actualizaciones:" (checked with a blue square) and "Cambiar AP de la ESAI:" (unchecked with a white square).

Figura 74. Plantilla html de Django para la configuración de la ESAI. Creado por el autor

La ESAI responderá inmediatamente al recibir el mensaje del broker, procesará su contenido y mandará a reiniciar el sistema. Al iniciarse intentará establecer una conexión HTTP con el servidor para solicitar la FOTA y en caso de que el servidor cuente con una versión de firmware disponible comenzará la OTA.

Al terminar el ESP32 se reiniciará y cargará el programa en la dirección de la FLASH de la OTA/NVS donde está almacenado el nuevo firmware.

El nuevo firmware es un sencillo programa que reproduce una pequeña melodía que compuse utilizando como salida de audio el pequeño zumbador negro y envía un mensaje por la terminal que da las gracias por haber asistido a la demostración.

```

T_mqtt: 424 us
T_mqtt: 424 us
Configuración cambiada.
rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jul 29 2019 12:21:46
Iniciando conexión con red: SSID:ESAI_2.4GHz . Se ha realizado con éxito la conexión con la red.Se ha encontrado una solicitud actual
ización de firmware OTA.
Comenzando actualización OTA...
Http Event On Connected
Http Event Header Sent
Http Event On Header, key=Server, value=Werkzeug/2.3.4 Python/3.11.3
Http Event On Header, key=Date, value=Tue, 09 Apr 2024 12:59:32 GMT
Http Event On Header, key=Content-Type, value=text/html; charset=utf-8
Http Event On Header, key=Content-Type, value=: application/octet-stream
Http Event On Header, key=Content-Length, value=265200
Http Event On Header, key=X-Frame-Options, value=DENY
Http Event On Header, key=X-Content-Type-Options, value=nosniff
Http Event On Header, key=Referrer-Policy, value=same-origin
Http Event On Header, key=Cross-Origin-Opener-Policy, value=same-origin
Http Event On Header, key=Connection, value=close
Http Event Disconnected
Http Event Disconnected
La actualización inalámbrica Over The Air del firmware ha sido realizada con éxito. Reiniciando unidad...
ets Jul 29 2019 12:21:46
rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2

¡¡GRACIAS POR ESTAR AQUI!!

```

Figura 75. Solicitud de OTA enviada desde el servidor, proceso de OTA y ejecución del nuevo programa. Creado por el autor.

Aunque existe una ventana de mejoras de prestaciones para futuros prototipos el proyecto logra cumplir de forma satisfactoria con todos los objetivos propuestos. La ESAI puede generar a veces falsas alarmas, aunque hay varios factores que influyen en esto:

- La calidad del diseño electrónico de las placas de desarrollo usadas y del módulo microfónico.
- Con los materiales usados, tanto la placa como el módulo se montan sobre una placa protoboard y se realizan conexiones con cables jumpers. Esto aumenta significativamente el ruido eléctrico que se introduce en el micrófono y por lo tanto dificulta la inferencia del modelo.
- La base de datos de AudioSet no contiene una cantidad proporcional de muestras de audio para todas las clases disponibles. Como consecuencia, el modelo YAMNet no ha sido entrenado de forma simétrica y tendrá mejores resultados para ciertas clases de sonidos que para otras.
- Se debe tener en cuenta también que el micrófono, aunque es omnidireccional, obtiene el sonido entrante a través del orificio perforado en la PCB del módulo. Por lo tanto, influirá la orientación relativa del módulo INMP441 respecto a la fuente sonora.
- El umbral que ajustamos influye de forma muy significativa en las falsas alarmas. Ajustar este umbral ayuda a corregir estos fallos.

Mejoras:

- Con el diseño electrónico propuesto en este trabajo se mejoraría la calidad del sonido capturado y por lo tanto se mejoraría también los resultados de la inferencia. Además, con la PSRAM integrada en el chip del prototipo, la RAM dejaría de ser un problema y se podría permitir jugar con un módulo de microcámara para la obtención de imágenes o video a la vez que se captura el sonido.
- Mejorar la experiencia al usuario con las nuevas funcionalidades del servidor Django descritas en el apartado de firmware de Django.
- Valorar mejorar el sistema de monitorización de batería del prototipo con un IC como el descrito en el apartado correspondiente del diseño electrónico.

6 Bibliografía

- [1] M. Lamrini, M. Y. Chkouri, and A. Touhafi, "Evaluating the Performance of Pre-Trained Convolutional Neural Network for Audio Classification on Embedded Systems for Anomaly Detection in Smart Cities," *Sensors (Basel)*, vol. 23, no. 13, Jul. 2023, doi: 10.3390/S23136227.
- [2] C. M. Mesa-Cantillo *et al.*, "A Sound Events Detection and Localization System based on YAMNet Model and BLE Beacons".
- [3] "Frequency-Weightings for Sound Level Measurements." Accessed: Dec. 02, 2023. [Online]. Available: <https://www.nti-audio.com/en/support/know-how/frequency-weightings-for-sound-level-measurements>
- [4] K. F. A. Darras *et al.*, "High microphone signal-to-noise ratio enhances acoustic sampling of wildlife," *PeerJ*, vol. 8, Oct. 2020, doi: 10.7717/PEERJ.9955/SUPP-13.
- [5] "Frequency Weightings - A-Weighted, C-Weighted or Z-Weighted." Accessed: Dec. 02, 2023. [Online]. Available: <https://www.noisemeters.com/help/faq/frequency-weighting/>
- [6] "What Is DB SPL (and How Does It Relate To DBA And DB HL)?" Accessed: Dec. 02, 2023. [Online]. Available: <https://audiointerfacing.com/what-is-dbspl/>
- [7] "Understanding Microphone Sensitivity | Analog Devices." Accessed: Dec. 02, 2023. [Online]. Available: <https://www.analog.com/en/analog-dialogue/articles/understanding-microphone-sensitivity.html>
- [8] "MICROPHONE SPECIFICATIONS EXPLAINED", Accessed: Dec. 02, 2023. [Online]. Available: www.invensense.com
- [9] S. Pithadia, S. Lester, and A. Verma, "LDO PSRR Measurement Simplified Application Report LDO PSRR Measurement Simplified," 2009, Accessed: Dec. 02, 2023. [Online]. Available: www.ti.com
- [10] "Comparing MEMS and Electret Condenser (ECM) Microphones | CUI Devices." Accessed: Dec. 01, 2023. [Online]. Available: <https://www.cuidevices.com/blog/comparing-mems-and-electret-condenser-microphones>
- [11] "Sensors | Free Full-Text | Design and Modeling of a MEMS Dual-Backplate Capacitive Microphone with Spring-Supported Diaphragm for Mobile Device Applications." Accessed: Dec. 03, 2023. [Online]. Available: <https://www.mdpi.com/1424-8220/18/10/3545>
- [12] "PDM vs. I²S: Comparing Digital Interfaces in MEMS Microphones | CUI Devices." Accessed: Dec. 03, 2023. [Online]. Available:

<https://www.cuidevices.com/blog/pdm-vs-i2s-comparing-digital-interfaces-in-mems-microphones>

- [13] "PDM Microphones and Sigma-Delta A/D Conversion | Electronics etc...."
Accessed: Dec. 03, 2023. [Online]. Available:
<https://tomverbeure.github.io/2020/10/04/PDM-Microphones-and-Sigma-Delta-Conversion.html>
- [14] "ARCELI Omnidireccional Micrófono Módulo I2S Interfaz Inmp441 Mems Alta Precisión Bajo Volumen Ultra Pequeño para Esp32 : Amazon.es: Informática."
Accessed: Dec. 03, 2023. [Online]. Available:
https://www.amazon.es/gp/product/B07RHMLYKD/ref=ppx_yo_dt_b_asin_title_o06_s00?ie=UTF8&psc=1
- [15] "Placa de desarrollo de ESP-32 ESP32, módulo compatible con Bluetooth, WiFi inalámbrico, doble núcleo, CP2102, filtros, 2,4 GHz, RF para Arduino - AliExpress."
Accessed: Dec. 03, 2023. [Online]. Available:
https://es.aliexpress.com/item/1005001929935550.html?spm=a2g0o.order_list.order_list_main.38.4d4b194dF7kA8Y&gatewayAdapt=glo2esp
- [16] "AZDelivery ESP32 ESP-WROOM-32 NodeMCU Modulo Wifi + Bluetooth Dev Kit C Placa de Desarrollo 2.4 GHz Dual Core con Chip CP2102 (sucesor del ESP8266) compatible con Arduino con E-Book incluido! : Amazon.es: Informática." Accessed: Dec. 03, 2023. [Online]. Available:
https://www.amazon.es/gp/product/B071P98VTG/ref=ppx_yo_dt_b_asin_title_o04_s00?ie=UTF8&th=1
- [17] "Getting Started with the ESP32 Development Board | Random Nerd Tutorials."
Accessed: Dec. 03, 2023. [Online]. Available:
<https://randomnerdtutorials.com/getting-started-with-esp32/>
- [18] "Memory Types - ESP32-S2 - — ESP-IDF Programming Guide v4.3 documentation." Accessed: Dec. 10, 2023. [Online]. Available:
<https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32s2/api-guides/memory-types.html>
- [19] "ESP32 Programmers' Memory Model. Internal memory of the MCU is probably... | by Amey Inamdar | The ESP Journal." Accessed: Dec. 13, 2023. [Online]. Available:
<https://blog.espressif.com/esp32-programmers-memory-model-259444d89387>
- [20] "ESP32 - DEBUGGING your ESP-IDF code using JTAG [VS CODE] - YouTube."
Accessed: Dec. 08, 2023. [Online]. Available:
https://www.youtube.com/watch?v=uq93H7T7cOQ&t=1019s&ab_channel=YuriR
- [21] "Networking for DevOps. In the last module, we discussed Linux... | by Bhanuprathap Reddy | Medium." Accessed: Dec. 14, 2023. [Online]. Available:
<https://avahack.medium.com/networking-for-devops-bf2a0aca5813>

- [22] "Protocolo de transferencia de hipertexto - Wikipedia, la enciclopedia libre."
Accessed: Dec. 04, 2023. [Online]. Available:
https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto
- [23] "Pila de protocolos - Wikipedia, la enciclopedia libre." Accessed: Dec. 04, 2023.
[Online]. Available: https://es.wikipedia.org/wiki/Pila_de_protocolos
- [24] "Modelo OSI - Wikipedia, la enciclopedia libre." Accessed: Dec. 04, 2023. [Online].
Available: https://es.wikipedia.org/wiki/Modelo_OSI
- [25] "Understanding the MQTT Protocol Packet Structure." Accessed: Dec. 04, 2023.
[Online]. Available: <http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/>
- [26] "Conda Documentation — conda-docs documentation." Accessed: Dec. 05, 2023.
[Online]. Available: <https://docs.conda.io/en/latest/>
- [27] "How to install and setup Django project inside Anaconda?" Accessed: Dec. 05,
2023. [Online]. Available: <https://gisgeeks.com/how-to-install-and-setup-django-project-inside-anaconda/>
- [28] "Paho Mqtt :: Anaconda.org." Accessed: Dec. 05, 2023. [Online]. Available:
<https://anaconda.org/conda-forge/paho-mqtt>
- [29] "

- [37] "Lumen Calculator (Lumen to Lux to Candela)." Accessed: Dec. 01, 2023. [Online]. Available: <https://www.omnicalculator.com/physics/lumen>
- [38] STMicroelectronics, "High efficiency switching and ultra low leakage current Schottky diode".
- [39] "Does MOSFET let current flow through source to drain as it allows it from drain to source? - Electrical Engineering Stack Exchange." Accessed: Nov. 27, 2023. [Online]. Available: <https://electronics.stackexchange.com/questions/105559/does-mosfet-let-current-flow-through-source-to-drain-as-it-allows-it-from-drain>
- [40] "USBXpress USB Bridges - USB-to-UART bridge controllers - Silicon Labs." Accessed: Nov. 08, 2023. [Online]. Available: <https://www.silabs.com/interface/usb-bridges/usbxpress>
- [41] "USB On-The-Go - Wikipedia." Accessed: Nov. 08, 2023. [Online]. Available: https://en.wikipedia.org/wiki/USB_On-The-Go
- [42] "USB OTG Console - ESP32-S2 - — ESP-IDF Programming Guide latest documentation." Accessed: Nov. 08, 2023. [Online]. Available: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/api-guides/usb-otg-console.html>
- [43] "LESD5D5.0CT1G LRC | C383211 - LCSC Electronics." Accessed: Nov. 23, 2023. [Online]. Available: https://www.lcsc.com/product-detail/Diodes-ESD_LRC-LESD5D5-0CT1G_C383211.html?gclid=CjwKCAiAjfyqBhAsEiwAUdzJGZZFn3PSxcgyr7CpmnN59XIEGwpinly9AcNi1z-epPh-vJYeRVR-xoC9EMQAvD_BwE
- [44] A. Forbes, "How to select a Surge Diode Application Report How to select a Surge Diode," 2019, Accessed: Nov. 23, 2023. [Online]. Available: www.ti.com
- [45] "Protección de circuitos con diodos de TVS de grado automotriz | DigiKey." Accessed: Nov. 23, 2023. [Online]. Available: <https://www.digikey.com/es/articles/design-in-tvs-diode-protection-enhance-can-bus-reliability>
- [46] A. Maier, A. Sharp, and Y. Vagapov, "Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things," *2017 Internet Technologies and Applications, ITA 2017 - Proceedings of the 7th International Conference*, pp. 143–148, Nov. 2017, doi: 10.1109/ITECHA.2017.8101926.
- [47] "ESP32-S3 Series Hardware Design Guidelines Introduction," 2023, Accessed: Nov. 27, 2023. [Online]. Available: www.espressif.com
- [48] "Analog to Digital Converter (ADC) - ESP32 - — ESP-IDF Programming Guide v4.4.3 documentation." Accessed: Nov. 27, 2023. [Online]. Available:

<https://docs.espressif.com/projects/esp-idf/en/v4.4.3/esp32/api-reference/peripherals/adcd.html>

- [49] A. Weiler, A. Pakosta, and A. Verma, "High-Speed Layout Guidelines Application Report High-Speed Layout Guidelines," 2006, Accessed: Nov. 30, 2023. [Online]. Available: www.ti.com
- [50] "PRINTER CIRCUIT BOARD ISSUES CHAPTER 12: PRINTED CIRCUIT BOARD (PCB) DESIGN ISSUES INTRODUCTION 12.1".
- [51] T. Williams, "Layout and grounding," *EMC for Product Designers*, pp. 298–329, 2017, doi: 10.1016/B978-0-08-101016-7.50012-2.
- [52] "Resonances of open air columns." Accessed: Dec. 01, 2023. [Online]. Available: <http://hyperphysics.phy-astr.gsu.edu/hbase/Waves/opecol.html>
- [53] "Capacitor-Series-Equivalent-Circuit-.png (649×277)." Accessed: Nov. 30, 2023. [Online]. Available: <https://upload.wikimedia.org/wikipedia/commons/9/9d/Capacitor-Series-Equivalent-Circuit-.png>
- [54] P. Device, "Application Note Performance required for bypass capacitor," 2020.
- [55] "Decoupling Capacitor Optimization and Resources for Power Integrity | Blog | Altium Designer." Accessed: Dec. 01, 2023. [Online]. Available: <https://resources.altium.com/p/decoupling-capacitor-optimization-power-integrity>
- [56] "ESP32 I2S stops working when building with arduino AND esp-idf framework - Development Platforms - PlatformIO Community." Accessed: Nov. 14, 2023. [Online]. Available: <https://community.platformio.org/t/esp32-i2s-stops-working-when-building-with-arduino-and-esp-idf-framework/18164/2>

7 ANEXO. Incidencias con el protocolo I2S.

Este anexo, que podría considerarse una extensión innecesaria de la memoria, ha sido incluido a propósito con una finalidad. Hay que entender que el 70% aproximadamente del tiempo dedicado al proyecto se basa en la aparición de errores y los esfuerzos por encontrar su causa y crear una solución al mismo. La mayoría de estos fallos están causados por errores de programación, algunos más triviales y otros menos o por falta de conocimiento aún en el campo, al fin y al cabo, este TFG surgió como idea casi como desafío y ha pavimentado todo un largo camino de aprendizaje. Sin embargo, en este caso los problemas se deben a errores e inexactitudes de la librería para controlar los drivers que el propio fabricante ofrece. Es por ello que su explicación y desarrollo es vital para obtener una idea real y práctica de cuáles son las limitaciones y particularidades de la librería.

Se pretende que este anexo pueda servir como guía y apoyo a futuros proyectos relacionados que puedan acabar encontrando estos problemas. Tener constancia de estas anomalías puede suponer una enorme cantidad de ahorro en tiempo y frustración por parte del desarrollador. Especialmente es importante dado que estas anomalías no se describen o advierten por parte del fabricante Espressif ni se describen en las propias librerías como notas o comentarios. Lo único que existe son las notas de “bugfix releases” o las notas de lanzamiento de una versión, pero cuando ya han resuelto un problema o fallo previo. Lo más parecido a una lista de fallos o “bug list”(aunque lejos de serlo) sería el apartado de “issues” del repositorio oficial de GitHub, donde los usuarios publican sus incidencias particulares y pueden o no recibir respuesta.

Incidencia:

Realizando pruebas con el micrófono INMP441 y la biblioteca de ESP-IDF “driver/i2s.h” necesaria para configurar y controlar la comunicación I2S se observó algunas cosas que llamaron la atención. Para visualizar gráficamente los datos se ha utilizado el Serial Plotter de Arduino versión 1.8.18 ya que el graficador de versiones posteriores no funcionaba correctamente.

Podemos apreciar en la figura 66 que, aunque se configuró una tasa de muestreo (de ahora en adelante SR del inglés Sample Rate) de 44,1KHz en el código, en la práctica obtenemos una frecuencia de muestreo que es justo la mitad. Para determinarlo se ha utilizado la aplicación “Frequency Generator” que permite generar diversas formas de onda para una frecuencia o rango de frecuencias deseado. La primera prueba es con una frecuencia de 10025 Hz y se aprecia que la señal obtenida es todavía una señal senoidal pero algo deformada.

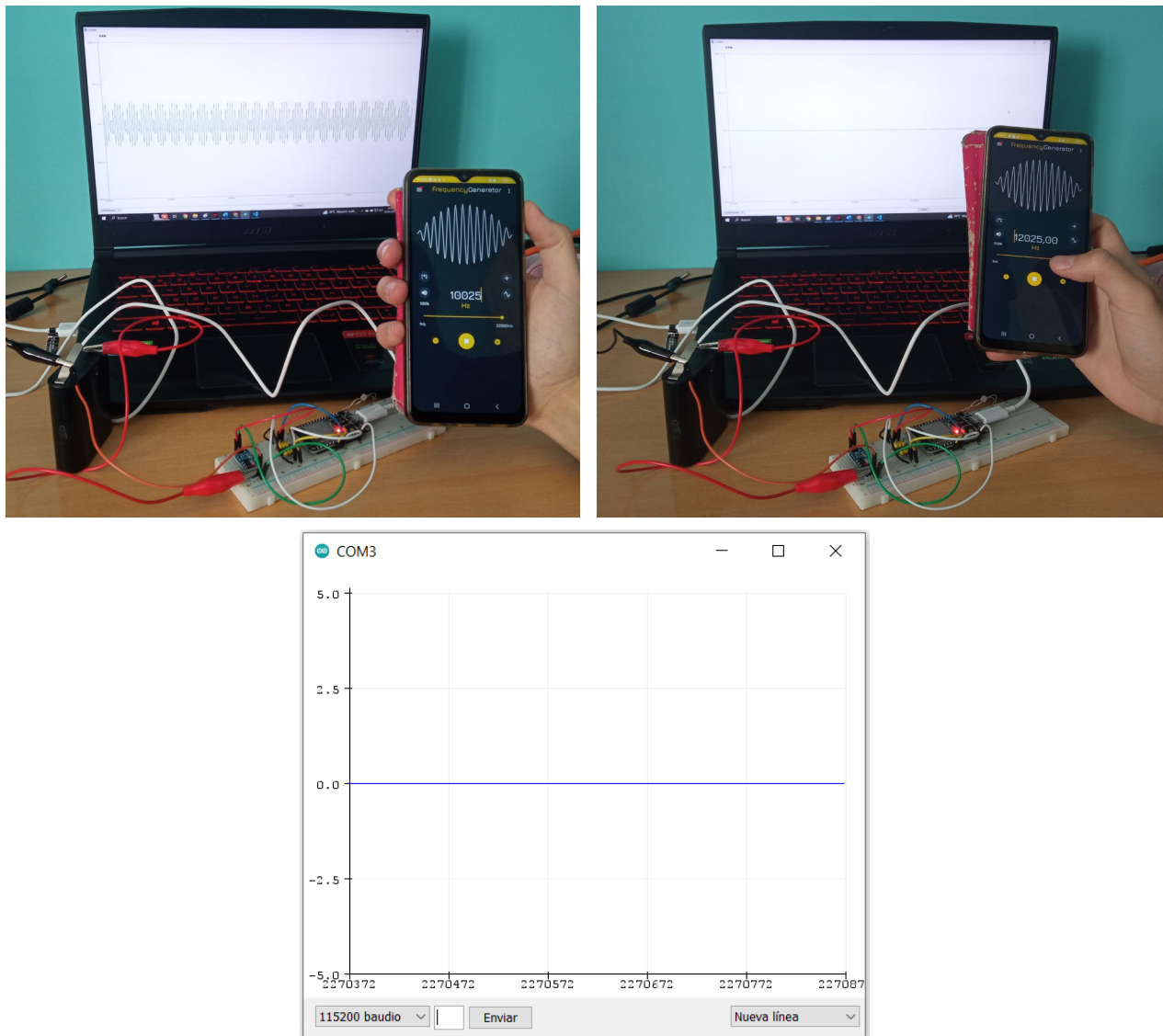


Figura 76. Pruebas realizadas con el INMP441 y la placa ESP32 DEV montados en una protoboard. Creado por el autor

La segunda prueba es a 12025 Hz y no se obtiene nada. A priori esto parecería no tener sentido a no ser que la SR fuera justo la mitad que la que hemos configurado (22050 en vez de 44100) y por lo tanto su frecuencia de Nyquist sería justo 11025 y de ahí los resultados obtenidos. La tercera imagen final corresponde al resultado que se obtiene cuando se aumenta la resolución de las muestras a 24 o 32 bits. Todas las muestras tomadas arrojan el valor 0 de forma indefinida.

Lo que resulta curioso es que la duración de los archivos es consecuente con la configuración de los parámetros de SR de la estructura I2S y el número de muestras

tomados por nuestro buffer cuando son procesadas por un reproductor o un procesador de audio como VLC Media Player o Audacity. Por ejemplo, para 32000 Hz y 32 000 muestras la duración del archivo de audio es de 1s, pero la frecuencia máxima que se capta es 8KHz (lo cual sería lo esperado para una tasa de muestreo de justo la mitad, 16KHz).

Para que esto esté ocurriendo deben estar ocurriendo varias cosas a la vez:

- La frecuencia de BCLK no es lo suficientemente alta para satisfacer la tasa de muestreo y profundidad de bits configurada.
- La mitad de los datos se están descartando ya que, si el número de muestras se mantiene, pero la SR real medida es la mitad de la configurada entonces la duración debería ser el doble, dado que la duración(d) del archivo de audio para un número de muestras determinado es inversamente proporcional a la SR:

$$d(s) = \frac{n_{muestras}}{SR \left(\frac{muestras}{s} \right)}$$

si $SR \downarrow \downarrow$ entonces $d \uparrow \uparrow$

Analizando la librería i2s.h encontramos dos funciones que podrían solucionar nuestro problema:

```
esp_err_t i2s_set_sample_rates(i2s_port_t i2s_num, uint32_t rate);  
esp_err_t i2s_set_clk(i2s_port_t i2s_num, uint32_t rate, uint32_t bits_cfg, i2s_channel_t ch);
```

Desafortunadamente no nos permite establecer una frecuencia del reloj a un valor concreto, sino que de nuevo depende de la SR, la profundidad de bits y el número de canales. Tras hacer pruebas añadiendo el uso de estas funciones se sigue observando el problema. Más aún extraño es el resultado que nos devuelve la función

```
float i2s_get_clk(i2s_port_t i2s_num);
```

la cual devuelve la tasa de muestreo configurada en vez de la frecuencia de reloj. En este punto empieza a quedar claro que la librería contiene errores e imprecisiones.

Anteriormente se analizó la interfaz de comunicación del micrófono INMP441 y uno de los detalles destacados es que por cada periodo de WS debía de haber 64 ciclos de la señal de reloj en cada periodo de la señal WS y que en consecuencia el tamaño de palabra de datos debe ser de 32 ciclos SCK. Esto significa que **independientemente de la resolución de bits que asignemos en la comunicación en el ESP32 el micrófono transmitirá 32 bits durante el primer ciclo de WS (4 bits de datos y 6 bits de alta impedancia) y en el segundo ciclo transmitirá 32 bits de alta impedancia.** Es probable que el fallo se deba a esto.

Si volvemos al archivo “i2s.h” de ESP-IDF y revisamos los parámetros de la estructura i2s_config_t vemos que una de las numerosas variables es bit_per_chan (bits por slot) que según indica los comentarios, por defecto se encuentra a 0, lo cual supone que tenga

el mismo valor que el de bits por muestra. En la siguiente prueba, vamos a proceder mantener 16 bits por muestra en la estructura, pero con la diferencia de que se establecerán 32 bits por canal (figura 67). Para la prueba vamos a utilizar una tasa de muestreo de 16 KHz por lo que deberíamos esperar poder captar frecuencias de hasta 8KHz.

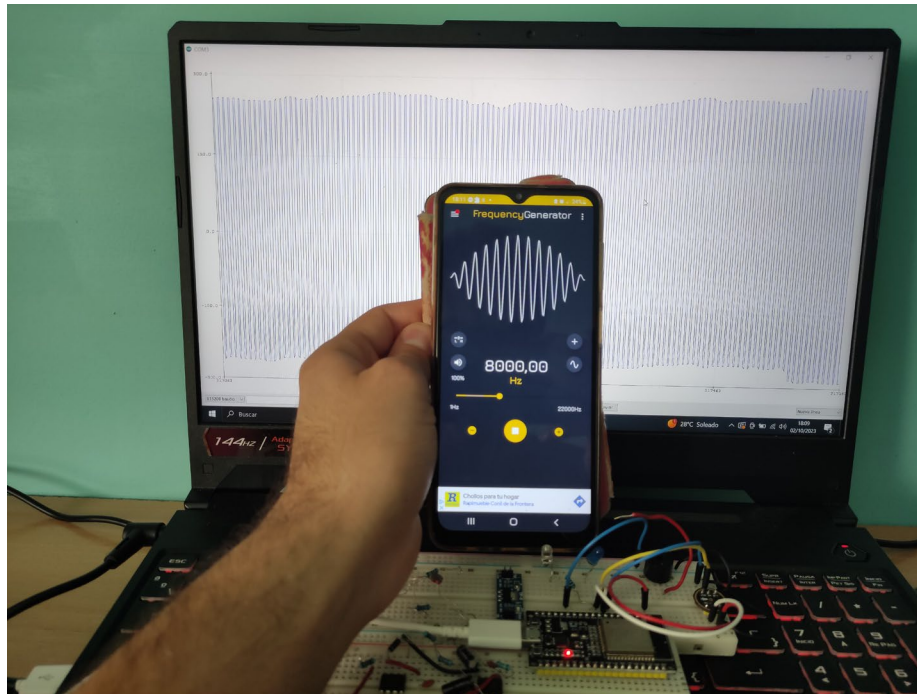


Figura 77. Resultados de la prueba con 16 bits por muestra y 32 bits de canal. Creado por el autor

Se puede considerar que el resultado de la prueba es positivo ya que la señal de 8KHz está siendo captada perfectamente, lo cual coincidiría con una SR de 16 KHz.

Sin embargo, debe de realizarse un estudio más detallado para determinar si el problema está solucionado. Para ello vamos a imprimir en el monitor serial los datos de audio de una muestra de 1 segundo. Para reducir el número de muestras necesarias, bajamos la SR a 8KHz (a efectos prácticos nos vale igual), por lo que tendremos que copiar sólo 8000 datos en vez de 16000. Creamos un programa sencillo en VS Code en el lenguaje Python que nos permita recoger los datos en un array de tipo numpy int16 y posteriormente guardar/escribir en formato binario su contenido en un archivo con extensión “.bin” (binary). En la figura siguiente se muestra el código:

```

1  import numpy as np
2  import os
3
4  data_array=[...]
5  numpy_array = np.array(data_array,dtype=np.int16)
6  file_dir="C:/Users/USUARIO/Documents/Projects/4.0 i2s_inmp441_plotter_devBoard/rawt2binario
+save"
7  filename = "8KHzTest.bin"
8  with open(file_dir + filename,"wb") as audio:
9      audio.write(numpy_array)
10 print("Escritura completada")

```

Figura 78. Código Python para guardar datos en formato binario. Creado por el autor

Los datos extraídos de la lectura serial del esp32 se copiarán entre los corchetes del vector “data_array”. Cuando se haya creado el archivo abrimos el programa de edición de sonido Audacity y realizamos los siguientes pasos: Archivo>>Importar>>Datos en Bruto>>Seleccionamos dirección y nos saldrá la siguiente pestaña:

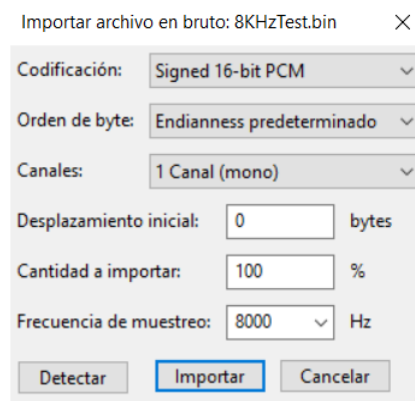


Figura 79. Pestaña de importación de archivos de Audacity. Creado por el autor.

Establecemos la tasa de muestreo, canal mono, little endian (es el Endianess por defecto) y nos saldrá nuestra pieza de audio de 1 segundo de duración:

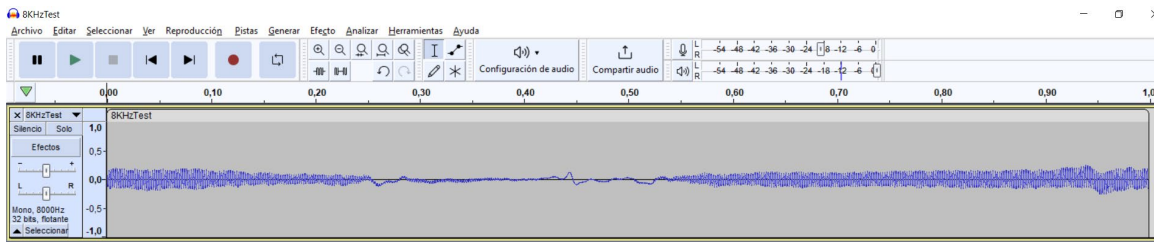


Figura 80. Visualización de los datos de audio en bruto. Creado por el autor

Al iniciar la reproducción debería escucharse un silbido pero en su lugar hay un sonido distorsionado, con un tono más grave. No puede ser otra cosa salvo que la tasa de muestreo no sea la adecuada. Pulsando click derecho sobre la muestra podemos cambiar la tasa de muestreo a 16000 Hz. El sonido ahora es normal y se corresponde al de mi silbido pero al aumentar el doble la tasa de muestreo la duración se redujo a la mitad.

Por lo tanto, modificando los bits por canal a 32 no ha resuelto el problema pues estamos en una situación similar. Lo que tenemos es 8000 muestras y una SR real de 16KHz dándonos a un clip de 500 ms cuando lo que queremos es un clip de 1 segundo con una SR de 8KHz.

Publiqué un tema relacionado con los problemas anteriores en el foro de la página oficial “www.esp32.com”. A falta de una respuesta fui redirigido al apartado de incidencias del repositorio de Espressif en GitHub y obtuve la siguiente respuesta:

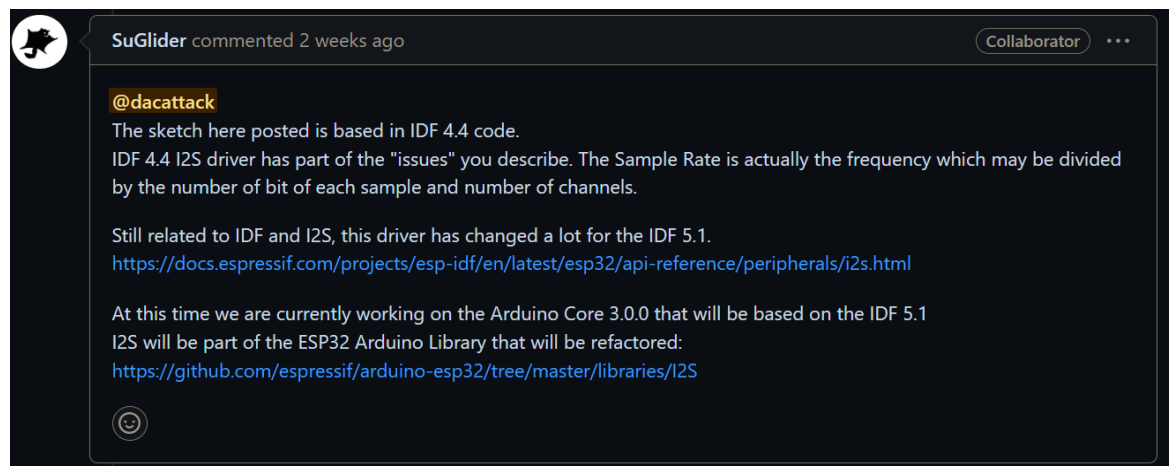


Figura 81. Respuesta ante la incidencia publicada en GitHub. Creado por el autor

En resumen, declara que el driver I2S de IDF tiene alguno de los problemas mencionados anteriormente. En el segundo enlace se encuentra la librería I2S que incorpora por defecto el framework arduino. En aras de simplificar y reducir extensión no se mostrará con detalles, pero tras realizar pruebas con dicha librería se encuentra el mismo problema.

En la guía de Espressif del protocolo I2S podemos encontrar en el apartado de “I2S File Structure” que existe una nueva versión del driver recogida en 3 cabeceras que deben ser incluidas (i2s_std.h, i2s_pdm.h, i2s_tdm.h) y sustituir en su lugar a la que ahora se llama la versión “legacy” (obsoleta), la cual es la que hemos estado utilizando hasta ahora (i2s.h). Esta versión nueva no está disponible para el marco de referencia de Arduino, es exclusiva de momento de ESP-IDF.

Con PlatformIO cambiar de entorno de trabajo es muy sencillo mediante la extensión PlatformIO de VS Code, tan solo tenemos que crear un nuevo proyecto y seleccionar el entorno de ESP-IDF. Esta nueva librería tiene algunas diferencias con la anterior versión y permite una configuración más específica y detallada como se muestra en la siguiente imagen:

```
void i2s_inmp441_init(){
    //La manera de configurar es distinta a la librería i2s.h de arduino. Primero se hace es a
    i2s_chan_config_t rx_channel_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_0, I2S_ROLE_MASTER);
    i2s_new_channel(&rx_channel_cfg, NULL, &rx_handle);
    //Ahora se crea una estructura con una configuración detallada y precisa de la comunicació
    i2s_std_config_t std_rx_cfg= {
        .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(SAMPLE_RATE),
        .slot_cfg = { //Aquí tenemos un control mucho más preciso de los parámetros del protoco
            .data_bit_width = I2S_DATA_BIT_WIDTH_16BIT, // (word length) Nº bits que se envía
            .slot_bit_width = I2S_SLOT_BIT_WIDTH_32BIT, // El slot es el espacio en la trama
            .slot_mode = I2S_SLOT_MODE_MONO, // Configurar Mono o estéreo
            .slot_mask = I2S_STD_SLOT_LEFT, // Máscara para seleccionar o filtrar
            .ws_width = 32, // Nº de BCLK ticks que la señal WS
            .ws_pol = false, // Poner a true si queremos habilita
            .bit_shift = true, // Importante, el inmp441 envía el p
            .msb_right = false
        },
        // .slot_cfg = I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_M
        .gpio_cfg = {
            .mclk = I2S_GPIO_UNUSED,
            .bclk = BCK_PIN,
            .ws = WS_PIN,
            .dout = I2S_GPIO_UNUSED,
            .din = SD_PIN,
        }
    };
}
```

Figura 82. Fragmento de código del proyecto de comunicación i2s con el framework ESP-IDF. Creado por el autor

Explicación de la estructura:

“.data_bit_width“ y “.slot_bit_width“: Igual que en la anterior librería, estas dos variables establecen la profundidad de bits de los datos a comunicar y por otro lado el ancho de bits que conforma un canal.

“.slot_mode“ y “.slot_mask“: La primera tiene relación al modo de canal(mono o estéreo) y la segunda variable es una máscara para filtrar o seleccionar el canal o los canales que se requieran.

“.ws_width“: Establece el ciclo de trabajo de la señal WS mediante el número de ciclos de reloj de bits (BCLK) durante los cuales la señal WS estará en alto(que como vimos depend del formato de comunicación).

“.ws_pol“: Controla la polaridad de la señal WS para que empiece a nivel alto o bajo.

“.bit_shift“: Desfasar o no 1 bit la señal de datos respecto a la señal WS.

“msb_right”: Poner a true para colocar los datos del canal derecho en los bits más significativos del buffer FIFO.

Mediante esta estructura podemos establecer manualmente el formato de comunicación que queramos, aunque también podemos utilizar macros que nos ahorre la tarea para configurar cualquiera de los formatos válidos (estándar, PDM, ADC/DAC etc.).

Cuando ejecutamos el programa en el EPS32 para leer y representar por el monitor serial los datos lo primero que advertimos al realizar pruebas es que si seleccionamos 16 bits de datos y 32 bits por canal obtenemos unos resultados idénticos a la prueba con la librería i2s (incluido el mismo problema cuando se realiza el procesamiento con Audacity). Cuando establecemos 32 bits de datos en lugar de ceros obtenemos un ruido constante que no varía ante la generación del sonido cerca del micrófono.

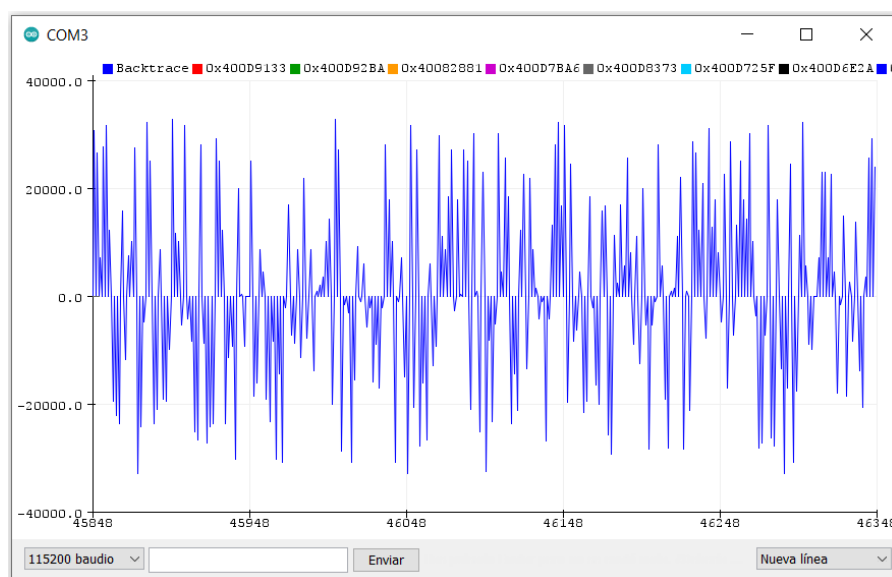


Figura 83. Prueba con la librería i2s_std.h. Creado por el autor

Esta librería tiene una importante incómoda diferencia respecto a la antigua, como se ha podido observar, en ningún momento aparecen las variables DMA. En su lugar debe realizarse manualmente la asignación de esta memoria DMA y la sincronización de las rutinas de servicio de interrupción, ISR (Interruption Service Routines). Estas diferencias pueden llevar a complicaciones en el diseño y a posibles errores adicionales y teniendo en cuenta que no se han conseguido mejores resultados que en la librería “legacy i2s.h” no hay motivo para usarla.

Resolución del problema

Atendiendo a las consultas realizada por usuarios en los foros de PlatformIO se puede apreciar que un número de ellas tienen como tema central problema de compatibilidades y errores entre el IDE de Arduino y Platformio. Por lo tanto, dado que en mi ordenador

está instalada una versión previa de Arduino(1.8.18), podemos realizar el experimento de ejecutar aquí nuestro código inicial con la librería i2s y ver qué resultados se obtienen.

Lo primero que surge al copiar el código aparecen varios errores ya que algunas de los elementos de las estructuras enumeradas se quedaron obsoletas y se reemplazaron por otras por ejemplo "I2S_COMM_FORMAT_I2S" por "I2S_COMM_FORMAT_STAND_I2S" y las variables ".mclk_multiple" y ".bit_per_channel" no existen o existen bajo otra denominación. No es problema pues "mclk" es un parámetro opcional y el ancho de canal es por defecto igual al número de bits de datos y precisamente vamos a seleccionar 32 bits. Por lo demás, el código permanece idéntico pero el resultado al ejecutar y visualizar los datos fue muy distinto, y en este caso, favorable. Parámetros SR=16KHz,32-32 ancho de datos y canal.

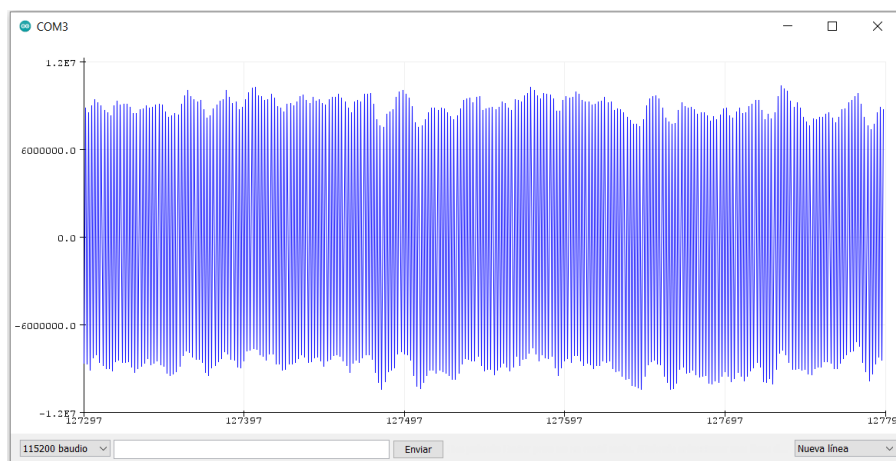


Figura 84. Prueba realizada con la versión 1.8.18 de Arduino. Creado por el autor

Con idéntica configuración I2S estamos viendo una onda de 8KHz generada cerca del micrófono siendo captada y representada como deberíamos haber esperado en un principio. Los valores, sin embargo, al ser ahora de 32 bits son mayores, aunque lo importante es que varía de forma proporcional a cuando tenía 16 bits. Al fin y al cabo, los datos de audios serán normalizados para que el valor máximo sea 1. Queda pues comprobar con Audacity si el sonido se corresponde con la SR de 16KHz o si existe el mismo problema que en pruebas anteriores. Igual que en la segunda prueba, se utilizará por simplicidad 8000 muestras a 8KHz de muestreo para dar a un clip de 1 segundo:

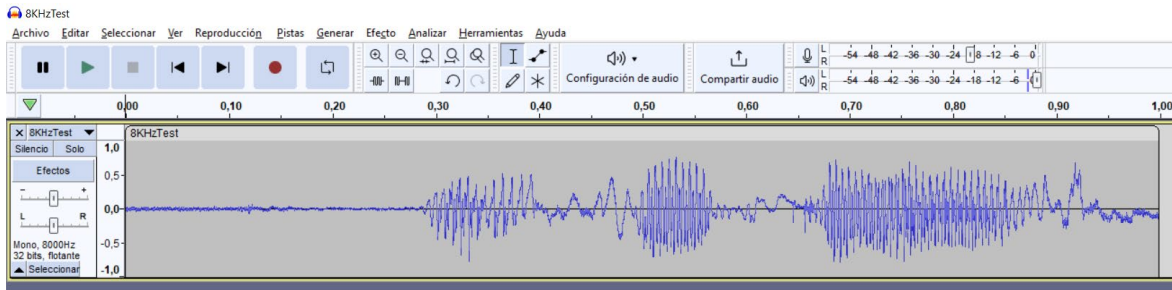


Figura 85. Audio siendo reproducido de forma exitosa en audacity. Creado por el autor

Para mi grata sorpresa el sonido se escuchaba perfectamente, por lo que hemos finalmente logrado resolver el problema. La causa es a todas luces, que la versión de la plataforma ESP-IDF que se ejecuta en mis proyectos de PlatformIO, que es la más actual 5.1.1, tiene algún tipo de fallo (aparte de los ya mencionados) con respecto a la versión de IDF que se ejecuta en el IDE de Arduino.

Continuando las investigaciones encontré una incidencia en el foro de PlatformIO [56] con un usuario que encontraba problemas con el mismo código cuando incorporaba el framework de ESP-IDF a cuando lo ejecutaba con el framework de Arduino sólo. El motivo es que al añadir ESP-IDF, como marco de referencia, se utilizaba una versión de este distinta al que cuando se ejecuta sólo Arduino. Tiene sentido si tenemos en cuenta que cada vez que hay una nueva versión de ESP-IDF, los desarrolladores de Arduino deben trabajar por actualizar sus librerías y eso puede conllevar un desfase entre las versiones de IDF usadas por el marco propio de Espressif frente al de Arduino. Continuando el hilo vemos que el propio usuario encuentra el error entre distintas versiones en cuestión, el cual es que por una razón desconocida es que, ¡los canales izquierdo y derecho están invertidos!

Es lógico deducir que la situación de este caso es exactamente la misma. Por lo tanto, volviendo al proyecto de VS Code y cambiando el elemento `I2S_CHANNEL_FMT_ONLY_LEFT` de la estructura `i2s_config_t` por `I2S_CHANNEL_FMT_ONLY_RIGHT` obtenemos el mismo resultado satisfactorio que acabamos de ver en la figura 75.

Como paso final nos queda gestionar los 32 bits que estamos recibiendo en el buffer junto con el hecho de que el micrófono INMP441 envía hasta 24 bits de resolución y que finalmente deseamos enviar paquetes de audio con una profundidad de 16 bits.

En un principio podríamos simplemente desplazar 8 bits hacia la derecha ya que el último byte no contiene información del micrófono. Pero incluso de los 24 bits útiles que transmite el micrófono, el primer byte podría también considerarse ruido poco útil. Hasta en una habitación en el silencio más absoluto que se ha podido lograr obtenemos valores erráticos:

00	00	68	00
00	00	37	C0
00	00	03	C0
00	03	72	00
FF	FF	C6	00
FF	FF	E3	C0
FF	FE	3D	C0
FF	FF	95	C0

*Figura 86. Representación hexadecimal de los 4 bytes trasmitidos por cada muestra.
Creado por el autor*

En esta imagen claramente se confirma que el primer byte no transmite información, puesto que toma sólo dos valores (00 ó C0) lo cual era de esperar. Por otro lado, los primeros dos bytes toman o valores próximos a cero o valores próximos al valor máximo. Esto corresponde a los valores más ínfimos tanto positivos como negativos de una variable de tipo entera (el valor negativo mínimo, por complemento a dos es el valor absoluto máximo para la cantidad de bits determinada o FF FF en hexadecimal para este caso. Finalmente, tal como se apuntaba, el segundo byte menos significativo toma valores que podrían considerarse casi aleatorios. Para una situación de silencio cuasi absoluto obtenemos los siguientes valores (convertidos a decimal): 104, 55, 3, 114, 198, 227, 61, 149.

Por ende, debemos eliminar este ruido fijo e innecesario, y el método más sencillo pasa por desplazar nuestro buffer de 32 bits algunos bits adicionales aparte del byte primero sin datos. En concreto lo desplazaremos $8 + 5 = 13$ bits hacia la derecha. De esta manera, tendremos 3 bits más de resolución que si tuviéramos una profundidad de 16 bits, aunque también existe mayor riesgo a que los datos de audio se saturen a algunos de los valores máximos. Por otro lado, tomando estos 3 bits menos significativos del segundo byte del bus de datos I2S podríamos decir que en cierto modo también estamos amplificando el valor de la señal en un factor de 2^3 veces respecto a una señal de 16 bits pura.

No obstante, debemos tener un pequeño detalle en cuenta relacionado con la degradación de una variable de un orden mayor a otro menor. Si nuestro buffer original (tras realizarse los desplazamientos de bits) tiene un ancho de palabra mayor a nuestro buffer de destino de 16 bits, podría darse el caso de que este albergara valores que excedieran el rango de valores posibles para una variable de tipo entera de 16 bits (65535, -65536) y esto podría originar resultados anómalos. Es por eso que tenemos que añadir una ligera estructura condicional (que podemos simplificar haciendo uso de operadores ternarios) para asegurarnos de que no se sobrepasan dichos valores. Esto es realizado por la función “audioDataCorrector” que se encuentra en el archivo “main.cpp” del proyecto”.