Universidad Zaragoza
1542

# Trabajo Fin de Máster

## Solución IoT para el control de ruido en comunidades de vecinos: Integración de MQTT y ESP32

## IoT solution for noise in residential communities: Integration of MQTT and ESP32


Autor

Moisés Zarzuela Maicas

Directores

Pablo Escribano García
Isidro Urriza Parroqué


Máster en Ingeniería Electrónica


Escuela de Ingeniería y Arquitectura

2024

# Solución IoT para el control de ruido en comunidades de vecinos: Integración de MQTT y ESP32


## RESUMEN

El presente Trabajo de Fin de Máster se ha desarrollado como parte de una colaboración con la empresa “Como tu casa”, entidad centrada en el alquiler de viviendas para corta y larga duración.

El objetivo de este proyecto es implementar un dispositivo capaz de medir los decibelios en una sala en las horas de descanso para, en caso de que se supere el nivel permitido, mande un aviso a la empresa para poder notificarlo a los inquilinos. La finalidad es evitar principalmente las molestias a los vecinos y dañar la convivencia en el edificio.

Para lograr dicho objetivo, y con las funciones que se necesitan definidas, se han elegido los componentes que se adecúan a la tarea. Se necesita conexión a internet por medio de Wi-Fi y datos móviles para el envío de información, además de un micrófono capaz de detectar los niveles de sonido. El micrófono debe ser lo suficientemente sensible como para detectar el ruido que se pueda producir en una vivienda de forma clara, pero no demasiado sensible, ya que podría saturarse. Para el tratamiento de datos se debe fijar tanto la frecuencia de muestreo como la ventana de datos para calcular los valores RMS.

Una vez se tenga el dispositivo montado y con capacidad de medir decibelios de forma más o menos precisa, se conecta por internet a un servidor MQTT, donde enviará los avisos pertinentes cuando se supere el umbral de ruido permitido. Además de enviar mensajes de aviso al servidor, el microcontrolador está diseñado para realizar acciones de forma remota, como actualizar su código o encender y apagar dos enchufes.

En el tema de la seguridad, para evitar ataques al servidor que puedan alterar el funcionamiento del mismo o de los dispositivos, se han implantado medidas de seguridad. Entre ellas está una serie de normas ACL, que incluyen una lista de acceso, permitiendo la entrada solo a usuarios registrados y una serie de normas que limitan la publicación de mensajes y suscripción a los tópicos necesarios. Otra condición para acceder al servidor es el usar una conexión segura con un certificado SSL.

Con este trabajo, se busca mejorar la convivencia en los edificios mediante el control de ruido en horario de descanso. Además, el sistema es perfectamente adaptable a futuras ampliaciones para cubrir otras necesidades.

# Índice

## Lista de acrónimos

| | |
|---|---|
| ACL | Access Control List |
| ADC | Analog to Digital Converter |
| CPU | Central Processing Unit |
| DMA | Direct Memory Access |
| DNS | Domain Name System |
| HTML | Hypertext Markup Language |
| I2C | Inter-Integrated Circuit |
| IoT | Internet of Things |
| MQTT | Message Queuing Telemetry Transport |
| OTA | Over-the-Air, Actualización remota |
| RMS | Root Mean Square |
| RTOS | Real-Time Operating System |
| SCL | Serial Clock Line |
| SDA | Serial Data Line |
| SIM | Subscriber Identity Module del Suscriptor |
| SSID | Service Set Identifier |
| SSL | Secure Sockets Layer |
| TLS | Transport Layer Security |
| UART | Universal Asynchronous Receiver-Transmitter |
| USB | Universal Serial Bus |

# 1. Introducción

## 1.1 Motivación

Este trabajo parte del interés de la empresa “Como tu casa” en un intento de evitar los ruidos a altas horas de la noche en los pisos de alquiler, que pueden producir molestia a los vecinos. La motivación principal del proyecto es el crear un ambiente de convivencia responsable, protegiendo tanto la inversión de la empresa como su reputación.

Respecto al panorama turístico español, la alta demanda de alojamiento temporal es el entorno perfecto para empresas de alquiler de viviendas, como puede ser “Como tu casa”. España es el segundo país más visitado en la comunidad internacional, con más de 70 millones de viajeros anuales de media a los que hay que sumarle el turismo nacional. Ciudades como Ibiza acogieron el año pasado a casi cuatro millones de turistas, la gran mayoría en busca de entretenimiento que a menudo incluye vida nocturna en las propias viviendas, lo cual es una fuente de ruido.

El tema central en el que se basa este trabajo es la detección de ruidos que superen el umbral permitido en las horas de descanso nocturno, puesto que son las más importantes para el descanso de los vecinos. Para ello, se dispone de un micrófono que actuará como sonómetro colocado en el salón, el cual se conectará a un servidor en la nube. Se ha elegido el salón como lugar del monitoreo puesto que es donde más afluencia de gente suele haber y, por tanto, más ruido. En el momento que el dispositivo capta un nivel de ruido superior al permitido, éste mandará un mensaje de aviso al servidor. Dicho servidor podría ofrecer más funciones para la casa, como abrir puertas o incluso controlar el aire acondicionado.

En el ámbito del Derecho administrativo, de acuerdo con el artículo 149.1.23.º de la Constitución Española [1], corresponde al Estado la definición de las bases de la protección del medio ambiente, mientras que las comunidades autónomas asumen, de forma generalizada, las competencias de desarrollo y de ejecución. Por ello, hay que acudir a lo establecido en la Ley 7/2010, de 18 de noviembre, de protección contra la contaminación acústica de Aragón [2].

La citada ley declara, en su artículo 5-a, la competencia de los municipios para aprobar ordenanzas sobre contaminación acústica de conformidad con lo establecido en el artículo 7, pudiendo estas contener aspectos que amplíen el grado de protección frente al ruido y las vibraciones establecido en dicha Ley. Por ejemplo, en Zaragoza, según el artículo 41 de la Ordenanza para la protección contra Ruidos y Vibraciones en el término municipal de Zaragoza [3], los niveles permitidos son de 40-45 dB durante el día (desde las 8:00 a las 22:00) y 27-30 dB durante la noche (desde las 22:00 a las 8:00), protegiendo así las horas de descanso de los vecinos.

En la actualidad, ya hay empresas dedicadas al control de ruido en el interior de los hogares como Minut [4] o Raixer [5]. El interés en el campo hace que crear una red propia de detectores de ruido resulte interesante.

## 1.2 Herramientas

Para la programación del microcontrolador se ha usado Arduino IDE, una plataforma que permite desarrollar y cargar el *firmware* de manera sencilla. Además, Arduino IDE permite cambiar el módulo sin tener que hacer grandes cambios a nivel de código. Como apoyo se ha usado la librería de la placa LilyGO-T-A7670E, que cuenta con un módulo de la familia ESP32, específicamente el ESP32-WROEVER-E.

La familia ESP32 es ampliamente reconocido en el ámbito de proyectos IoT *"Internet of Things"*, por sus siglas en inglés, o Internet de las cosas en español. Su bajo costo y mínimo consumo de energía lo hacen una opción muy atractiva para este tipo de proyectos en los que el dispositivo puede requerir estar alimentado a base de baterías, aumentando así su vida útil entre cambios de pilas. Además, la familia ESP32 tiene la capacidad de conectarse por Wi-Fi y Bluetooth a otros dispositivos o a la red. [6].

En un contexto en el que las tecnologías de la información y comunicación avanza a pasos agigantados, los proyectos IoT son esenciales para la gestión de comunicación entre dispositivos. Debido a la gran cantidad de datos que se procesan a cada momento, la comunicación entre dispositivos debe ser rápida y optimizar los recursos todo lo posible [7]. Es ahí donde nace MQTT, *"Message Queuing Telemetry Transport"*, por sus siglas en inglés. MQTT es un protocolo de mensajería ligera basado en el estándar Pub/Sub creado por IBM en 1999 con la finalidad de enviar mensajes con la menor latencia posible [8]. El protocolo MQTT ya se ha utilizado con anterioridad en proyectos de monitorización remota en sistemas IoT de control ambiental, para medir factores como la humedad y la temperatura [9].

La puesta en marcha del servidor MQTT y todas las configuraciones de seguridad se han apoyado en diversos cursos especializados en dichas áreas. El *broker*[1] utilizado es "EMQX open source", que facilita la gestión escalable de comunicación entre dispositivos.

## 1.3 Marco legal

Para implementar un sistema de monitoreo de ruido en las viviendas alquiladas por la empresa "Como tu casa" de forma legal, es necesario conocer y respetar las normativas que afectan a este tipo de prácticas. En este caso, el principal referente es el Reglamento General de Protección de Datos de la Unión Europea [10].

[1] Un *broker* es la entidad que recibe y distribuye los mensajes entre los dispositivos conectados mediante el control de las suscripciones y publicaciones.

Los artículos 5, 6, 13 y 17 indican que para que la detección de ruido se haga de forma lícita, la empresa debe informar a los inquilinos la finalidad de la medida tomada, dejando claro que solo se va a tomar el nivel de ruido, sin grabar conversaciones ni otros sonidos, con el fin de controlar los decibelios.

Se debe garantizar también la confidencialidad de los datos y que el acceso a los mismos esté limitado solo a personal autorizado. De ser necesario, los datos deberán borrarse tras su uso.

En lo referente al propietario, el artículo 21 de la Ley de Arrendamientos Urbanos [11] le da derecho a la implementación de medidas para proteger sus propiedades y asegurar una convivencia positiva, respetando siempre la privacidad de los inquilinos. Además, el artículo 7.2 de la Ley de Propiedad Horizontal [12], establece que << *Al propietario y al ocupante del piso o local no les está permitido desarrollar en él o en el resto del inmueble actividades prohibidas en los estatutos, que resulten dañosas para la finca o que contravengan las disposiciones generales sobre actividades molestas, insalubres, nocivas, peligrosas o ilícitas*>>. Dicho artículo se puede usar para argumentar la adopción de medidas preventivas para la limitación de ruido.

## 1.4 Objetivos

El objetivo fundamental de este trabajo es la detección de sonidos que superen el umbral acústico permitido en las horas de descanso por el bienestar de las personas que rodean al inmueble. Para ello, se ha utilizado un micrófono conectado a un amplificador, para así ser capaces de leer la intensidad del sonido y, tras obtener la equivalencia a decibelios (dB), conseguir que el microcontrolador avise de cuando se están superando los valores permitidos.

Para lograr dicho objetivo se plantean una serie de tareas:

- Estudio de múltiples micrófonos disponibles en el mercado. Existen una gran variedad con varios factores que pueden decantarnos por uno u otro. Se busca un conjunto sensible a los ruidos pero que no llegue a saturarse, ya que distorsionaría la señal captada.
- Calibración del micrófono. El dispositivo que se elija devolverá un valor en voltios que, tras pasar por el ADC, pasara a una señal en forma de bits y no en decibelios, que es lo que se pretende medir. Es por ello que se requiere la obtención de una fórmula que permita traducir la medida del micrófono a decibelios.
- Estudio de los servidores MQTT y configuración del mismo. Se parte de un servidor sin ningún tipo de seguridad, al que cualquiera puede conectarse, por lo que se deben añadir capas de seguridad para asegurar en la medida de lo posible que extraños no puedan entrar al servidor para atacarlo.
- Desarrollo del *firmware* en el microcontrolador. Una vez creado el servidor con toda su configuración se desarrolla el *firmware* que permita a la placa hacer de intermediaria entre el sensor y el servidor, y poder enviar y recibir mensajes.

- Realización de pruebas de conexión y funcionamiento. Tras conectar el micrófono a la placa y que ésta sea capaz de conectarse al servidor, se deben realizar pruebas para que no haya puntos en los que el código no se quede atascado en un punto donde no sea útil.

## 1.5 Estructura de la memoria

Este trabajo expone el proceso de desarrollo de un sistema capaz de detectar intensidades de sonido superiores a la permitida en horario de descanso de los vecinos. Para ello, se utilizará un micrófono que actuará como sonómetro, el cual estará conectado a una placa, que a su vez se comunicará a un servidor MQTT.

La memoria del trabajo está estructurada de la siguiente forma:

En el segundo capítulo se tratan los requerimientos y los componentes físicos del sistema, que se colocarán en las viviendas. Se mostrará la forma en la que se conectan y la elección de los parámetros del micrófono, incluyendo la frecuencia de muestreo, la ventana de datos y la ganancia. Por último, se calculará la conversión de los valores a decibelios.

En el tercer capítulo se ven los componentes no físicos, es decir, el servidor con sus configuraciones de seguridad, el *dashboard*, desde donde se verá la información de los dispositivos y se presentará MQTTX, una aplicación que permite conectarse al servidor, donde se podrán enviar y recibir mensajes de forma más cómoda. También se comentará el funcionamiento del *firmware* con el que funcionará el dispositivo, con la explicación de sus funciones.

En el cuarto capítulo se muestran una serie de pruebas que se han hecho a modo de test, donde se pone a prueba el funcionamiento del conjunto y viendo la respuesta que da cuando se le pasan datos correctos e incorrectos.

Por último, en el quinto capítulo se tratan las conclusiones a las que se ha llegado tras la realización del trabajo y se proponen ideas acerca del trabajo futuro para esta herramienta.

Al final del documento se encuentran los anexos. Este documento consta de tres. En el anexo A se muestra el *datasheet* del amplificador de micrófono. En el anexo B se encuentra el código con el que funciona el sonómetro y el anexo C contiene los *scripts* de Python con los que se pone a prueba la conexión a internet y al servidor con distintas entradas.

# 2. Componentes electrónicos

En este capítulo se tratará la parte física del proyecto como la elección de los aparatos electrónicos y su conexión. También se tratarán las características de los dispositivos.

Se pretende diseñar un dispositivo capaz de detectar sonidos altos, y, mediante MQTT, enviar la información a un usuario central para ver si es necesario avisar a los inquilinos. Es por ello que el microcontrolador necesitará optar a las siguientes funciones:

- Capacidad de conexión a Wi-Fi: Puesto que los datos se van a enviar y recibir por medio de un servidor MQTT ubicado en la nube, es necesario que el microcontrolador sea capaz de conectarse a la red Wi-Fi.
- Memoria no volátil: El dispositivo puede llegar a apagarse o reiniciarse por diversas razones (actualizaciones, batería agotada, desconexión por parte del inquilino). La memoria no volátil permite almacenar los datos de configuración del dispositivo para que no sea necesario el reconfigurarlo manualmente cada vez que se da este suceso.
- Disponibilidad para operar una tarjeta SIM: En caso de que la red Wi-Fi no esté disponible, el dispositivo se conectará al módem que contenga la tarjeta SIM, para acceder a Internet de forma alternativa.
- Periféricos para la conexión del micrófono: El nivel de ruido se calculará a partir de los datos obtenidos por el conjunto micrófono-amplificador, que se conectará a la placa por medio de sus pines.

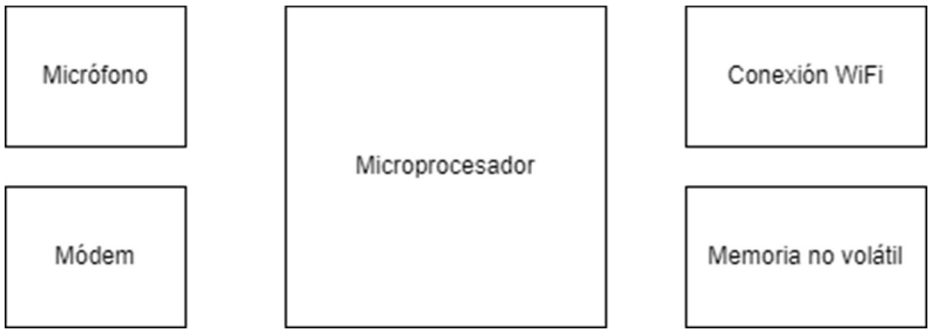



*Figura 1 Esquema de características necesarias.*

En la figura 1 se ve un diagrama mencionando las características necesarias para el funcionamiento del dispositivo.

La memoria no volátil suele estar integrada en módulos como los de las familias ESP32 o ESP8266, al igual que el módulo encargado de la conexión a redes Wi-Fi. Estas características están conectadas a la CPU *"Central Process Unit"* mediante buses de datos de alta velocidad.

Por otro lado, el Módem se conecta por UART "Universal Asynchronous Receiver-Transmitter". Este protocolo asíncrono permite la comunicación entre dos dispositivos utilizando dos cables. Un cable envía la información mientras el otro se encarga de recibirla. Para su funcionamiento, el pin TX del primer dispositivo debe conectarse al RX del segundo, y viceversa. Además, es

necesario que ambos dispositivos tengan configurados la velocidad de comunicación, por ejemplo, 115.200 baudios por segundo.

La placa será alimentada por medio de un cable USB *"Universal Serial Bus"* o bus serial universal con la electricidad de la casa, evitando así el cambio de pilas continuo.

Finalmente, el micrófono, al tener una salida analógica, debe estar conectado al módulo mediante un canal ADC *"Analog to Digital Converter"*, o conversor analógico-digital que permite convertir la señal analógica en digital para poder trabajar con ella. Además, también debe conectarse a una fuente de alimentación y a tierra.

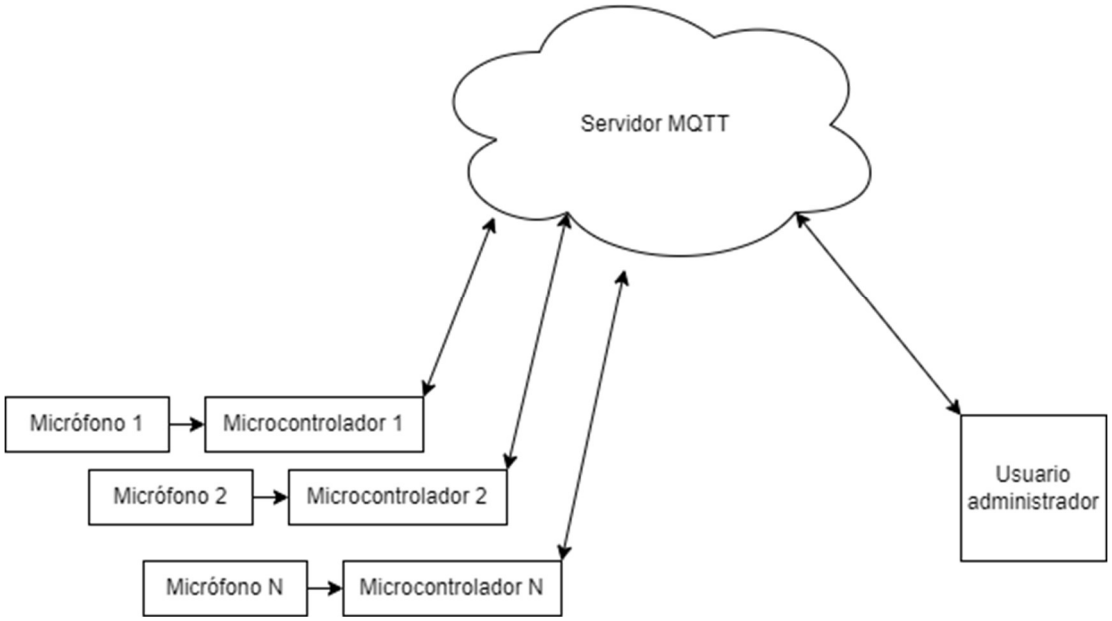



*Figura 2 Diagrama global*

La figura 2 muestra el diagrama global que se plantea. Una serie de microcontroladores que se comuniquen con el servidor sin necesidad de compartir información entre ellos y un servidor que puede comunicarse a su vez con el usuario administrador. Una vez conocidos los requisitos para la puesta en marcha del proyecto, sólo queda elegir los componentes del mismo.

## 2.1 Placa

La placa elegida para este trabajo es la LilyGo T-A7670E dadas sus características compatibles con este tipo de aplicaciones IoT, ya que integra un módulo ESP32-WROVER-E. Aunque el dispositivo estará configurado para trabajar principalmente conectado a una red Wi-Fi, la placa también incluye un módem SimCOM A7670E, que permite usar una tarjeta SIM como modo alternativo de conexión a Internet, además de poder programarse para enviar SMS de forma automática. Para la alimentación se usará el puerto USB-C que lleva incorporado. Por último, el módulo tiene integrado una memoria no volátil, por lo que tampoco será necesario preocuparse de buscar una externa. La figura 3 muestra un esquema detallando los periféricos de la placa.

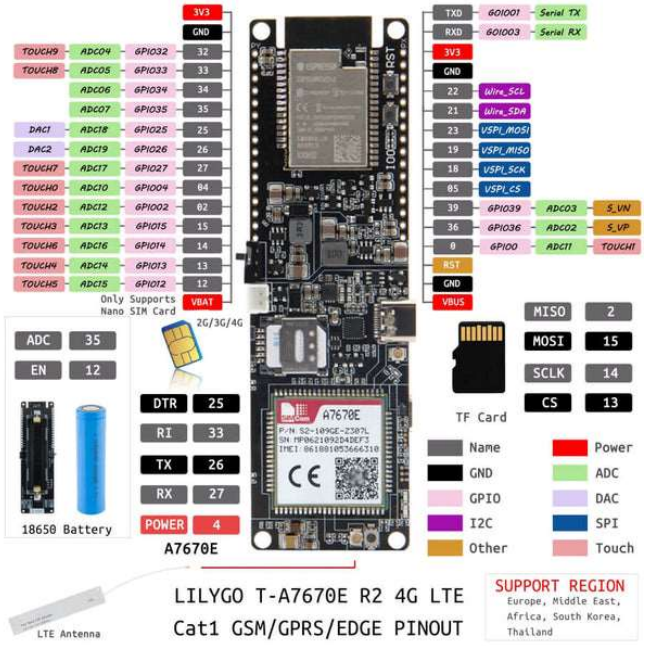



*Figura 3 Esquema de la placa LilyGo T-A7670E*

En cuanto a la conexión de dispositivos, la LilyGo T-A7670E alberga la posibilidad de conexión de varios sensores para, si se decide ampliar la funcionalidad, medir más datos como por ejemplo niveles de monóxido de carbono, lo cual aumentaría la seguridad en la vivienda.

## 2.2 MAX9814 y micrófono

Una vez seleccionada la placa con la que se va a trabajar, el siguiente paso es elegir un micrófono adecuado que cumpla con los siguientes requisitos:

- Tamaño reducido: Por estética, es preferible que el conjunto sea lo más compacto posible, para reducir el impacto visual.
- Que pueda escuchar ruidos elevados sin saturarse: Un micrófono muy sensible tomaría valores muy elevados al menor ruido presente, pudiendo llegar a saturarse y distorsionar la medida de ruido. Es por ello que se busca un micrófono con una sensibilidad suficiente para detectar valores altos pero que pueda llegar a leer correctamente diferencias más pequeñas.

Con todas las características detalladas, se ha elegido el MAX9814, un amplificador de micrófono fabricado por “Analog Devices, Inc”. El MAX9814 viene integrado en un módulo con un micrófono electret y pines que facilitan la conexión a la placa. La hoja de características del micrófono se encuentra en el ANEXO A.

### 2.2.1 Conexión de los dispositivos

Una vez seleccionados los componentes del sistema, sólo queda conectarlos. Debido a que la placa LilyGo T-A7670E trae consigo todas las funciones requeridas para el proyecto salvo la del micrófono, sólo hay que conectar este a la placa.

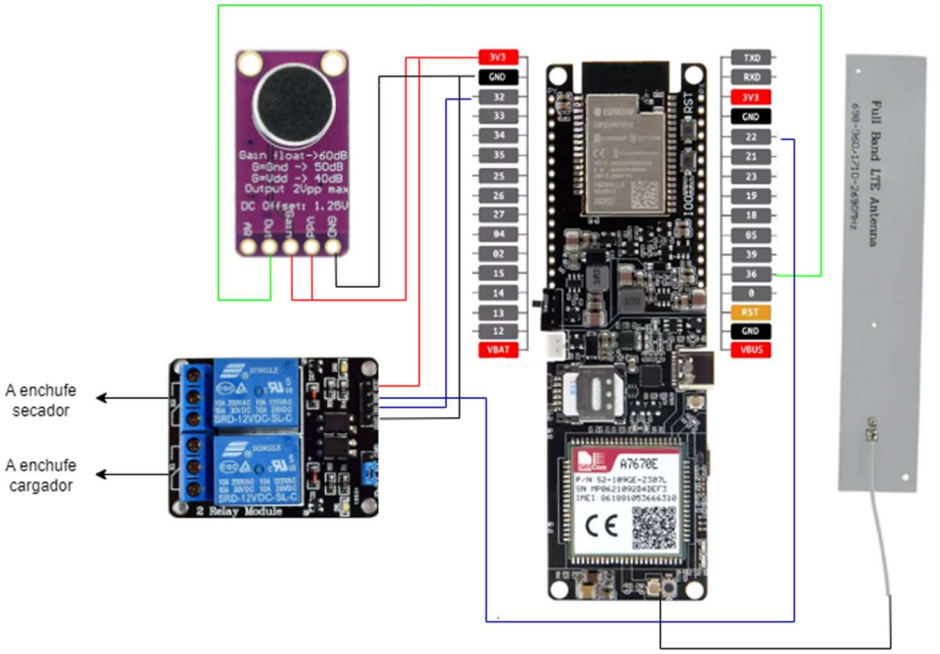



*Figura 4 Conexión de dispositivos a la placa.*

La figura 4 presenta la conexión de los componentes a la placa. Además del micrófono, se añaden dos relés conectados a los pines 22 y 32. La finalidad de estos relés es mostrar la capacidad que tiene el microcontrolador de interpretar mensajes del servidor y realizar la acción requerida. Para la conexión a internet con la tarjeta SIM se conecta una antena, capaz de captar la señal de la red. Por la parte del micrófono, el pin GND de la placa se conecta a la tierra del micrófono mientras que la alimentación va a Vdd. La salida del micrófono va conectada al pin 36, más adelante se explicará la razón. Por parte del microcontrolador, los pines ADC ofrecen valores de 12 bits, por lo que se podría esperar una salida entre 0 y 4095. Sin embargo, el *datasheet* del MAX9814 indica que la salida estará comprendida entre los 3 mV y los 2,45 V, por lo que no tomará el rango completo. Esto significa que los valores digitales que se esperan a la salida estarán en el rango de 3 hasta 3.040. El micrófono ofrece tres valores de ganancia, dependiendo de cómo esté conectada. La elección de conexión se verá más adelante.

Si el pin de la ganancia se deja sin conectar, el micrófono tomará el valor de ganancia más alto, que en este caso son 60dB. Este valor sería capaz de detectar incluso los sonidos más suaves, pero con el riesgo de saturar más fácilmente, lo cual no es ideal ya que el objetivo es diferenciar valores altos.

El pin de la ganancia conectado a tierra ofrece una ganancia de 50dB, que es el valor intermedio de los tres que ofrece. La saturación se alcanza con valores más altos en comparación con la ganancia de 60 dB.

Por último, conectando la ganancia a Vdd se obtienen 40dB de ganancia, que es el valor mínimo del micrófono. Con esta configuración la saturación es incluso más complicada, ya que se reduce la amplificación del micrófono.

Para elegir qué valor es más adecuado para el sensor, se realizan pruebas de detección. Pero antes, es importante saber con qué frecuencia se van a tomar muestras del micrófono.

### 2.2.2 Configuración del micrófono

Para poder tomar valores con criterio, se deben fijar ciertos parámetros del micrófono. Estos son la frecuencia de muestreo y la ganancia, mencionada anteriormente.

La regla de Nyquist dicta que se debe tomar muestras a una frecuencia como mínimo el doble de la frecuencia máxima que se desea detectar. El ser humano es capaz de escuchar sonidos comprendidos entre 20 Hz y 20 kHz, lo cual implica tomar muestras a una frecuencia de 40 kHz, o cada 25 µs.

La voz humana emite sonidos en una frecuencia comprendida entre los 85 Hz para las voces más graves y los 4 kHz para las más agudas. Por el lado de la música, el rango es más variado, dado que los instrumentos son capaces de producir sonidos con una mayor frecuencia. Los sonómetros comerciales abarcan un rango comprendido entre los 10 Hz y los 16 kHz [13], más que suficiente para captar la voz humana, ignorando los sonidos de frecuencia superior, pero rebajando la carga de datos para procesar. Se ha decidido seguir esta línea para centrarse en las voces humanas.

Una vez elegida la frecuencia de muestreo, se realizan pruebas con los tres valores de ganancia ofrecidos por el micrófono para ver cuál es más adecuado para la detección de sonidos fuertes. Las pruebas consisten de dos tomas de datos: una en silencio y otra con un ruido. La prueba de ruido consiste en captar una conversación con música de fondo a una intensidad suave. De este modo, se puede apreciar como el micrófono capta los datos a los que se va a enfrentar en una situación real.

En primer lugar, se deja la ganancia en 60 dB, el modo de mayor amplificación. A continuación, en la figura 5, se muestra una gráfica de los valores que toma en una habitación en silencio. El ruido ronda los 25 dB. Debido a la falta de sonómetro, la medida de decibelios se ha realizado con una aplicación que emula un sonómetro con muy buenas reseñas debido a su bajo margen de error.

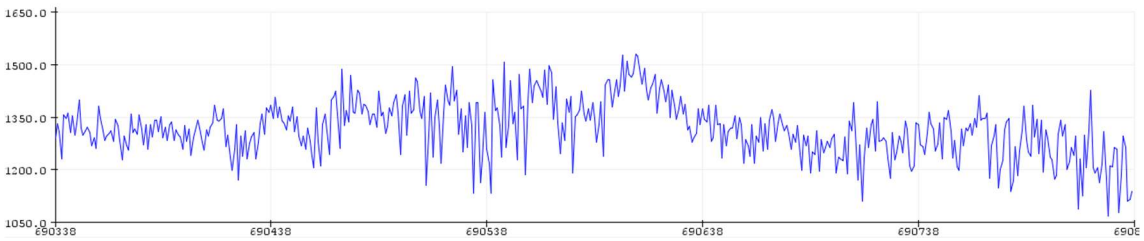



*Figura 5 Valores tomados del micrófono con ganancia 60dB en una habitación silenciosa.*

Se puede apreciar que el micrófono toma valores comprendidos en su mayoría entre los valores de 1200 y 1500, un rango bastante amplio para estar en silencio. A continuación, está la gráfica de una conversación con música de fondo. La intensidad del sonido ronda los 55 decibelios.

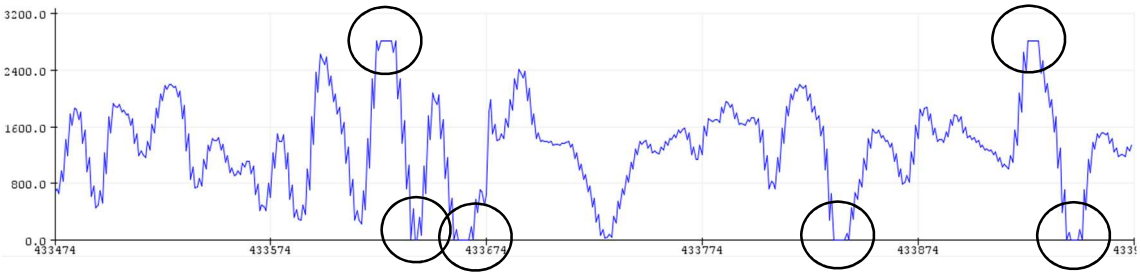



*Figura 6 Valores tomados del micrófono con ganancia 60dB en una habitación con ruido.*

En la figura 6 se puede apreciar la saturación del micrófono tanto por la parte inferior como en la superior, donde se ven mesetas. Dichos puntos están marcados con un círculo. Esto no es un resultado correcto, ya que al llegar a su límite está alterando la medición. No es un comportamiento adecuado para un micrófono cuya función es el ser capaz de detectar ruidos fuertes, por lo que esta configuración queda descartada.

A continuación, en la figura 7, se muestran los resultados obtenidos con la ganancia puesta en 50 dB. En este caso, se esperan unos niveles más compactos en el caso silencioso y picos más bajos cuando haya ruido.


*Figura 7 Valores tomados del micrófono con ganancia 50dB en una habitación silenciosa.*

En este caso los valores están comprendidos en el rango de 1300 y 1380, una diferencia de 80, que es menos de la mitad que cuando la ganancia era de 60 dB. En la figura 8 se ven los valores que toma cuando hay ruido en la habitación.

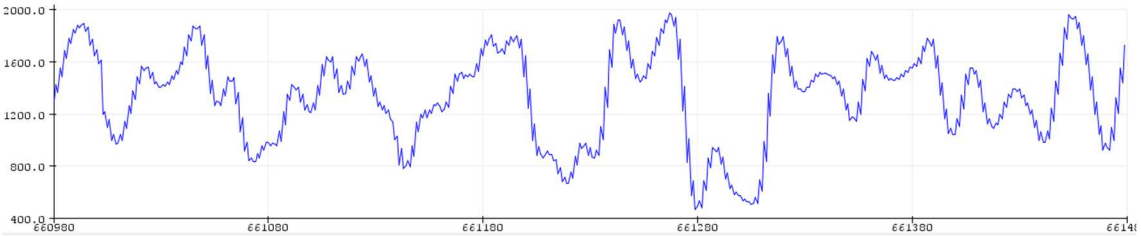



*Figura 8 Valores tomados del micrófono con ganancia 50dB en una habitación con ruido.*

En este caso las ondas toman valores más reducidos, sin llegar a saturar. Este comportamiento es aceptable ya que no distorsiona el ruido detectado, por lo que no se va a descartar a esperas del resultado de la última prueba. Por último, en la figura 9, se prueba la ganancia de 40dB.

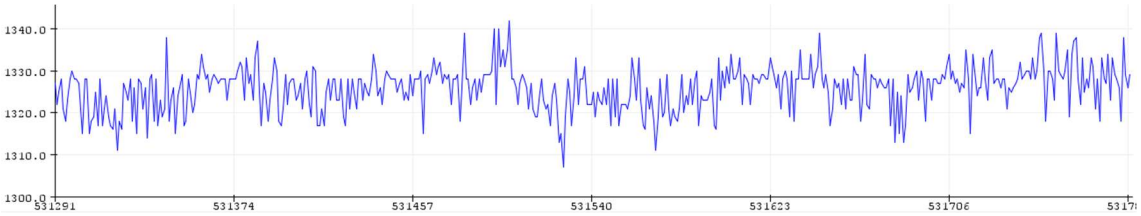



*Figura 9 Valores tomados del micrófono con ganancia 40dB en una habitación silenciosa.*

En la prueba de la habitación en silencio los valores se concentran en la zona comprendida entre los 1310 y los 1340, una diferencia de 30, que es menos de la mitad que en el caso anterior. Esto muestra que la señal es bastante estable en este punto.

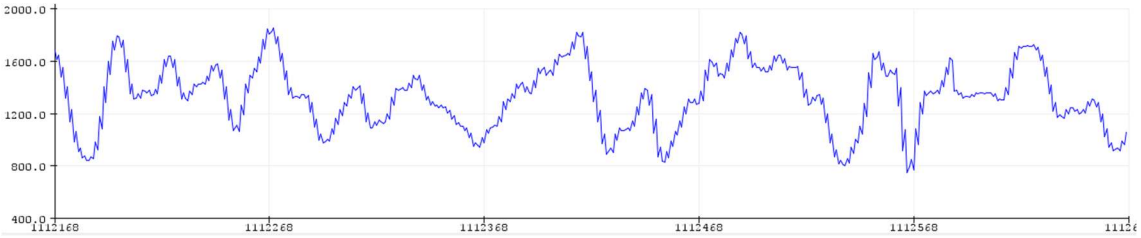



*Figura 10 Valores tomados del micrófono con ganancia 40dB en una habitación con ruido.*

Para finalizar, como se ve en la figura 10, se usa la ganancia de 40dB en el micrófono para medir los valores con una conversación y música de fondo. La amplitud de las ondas es menor que en el caso anterior. Esto nos deja con dos posibles ganancias que en teoría funcionan bien.

Las medidas tomadas con anterioridad muestran una onda producida por el ruido, pero un valor suelto no equivale a un nivel de decibelios. Es por ello que se utiliza el RMS *"Root Mean Square"* o valor eficaz.

Un detalle que no debe pasarse por alto para el cálculo del valor RMS es que sólo hay que tener en cuenta la variación de la onda. En las imágenes anteriores, se puede ver que los valores no oscilan en torno al 0, sino que tienen un *offset*. En todos los casos la onda oscila en torno a un valor medio de 1.324. Dicho *offset* debe ser sustraído al valor que ofrece el ADC.

La fórmula del RMS agrupa múltiples muestras para sacar un valor promedio utilizando los cuadrados, que en este caso sí que puede tomarse en cuenta para el cálculo de decibelios. Para reducir el impacto de sonidos puntuales como la caída de un objeto, la ventana de datos es de 16.000 muestras o 1 segundo.

Para la elección de la ganancia se va a repetir la prueba anterior en el caso de silencio y hablando, pero mirando los valores RMS. Esto se puede hacer ya que se ha comprobado que el micrófono no se satura con esos niveles de ruido, por lo que el cálculo del valor RMS no se verá distorsionado. Primero se muestran en la figura 11 los resultados con la ganancia de 40 dB.

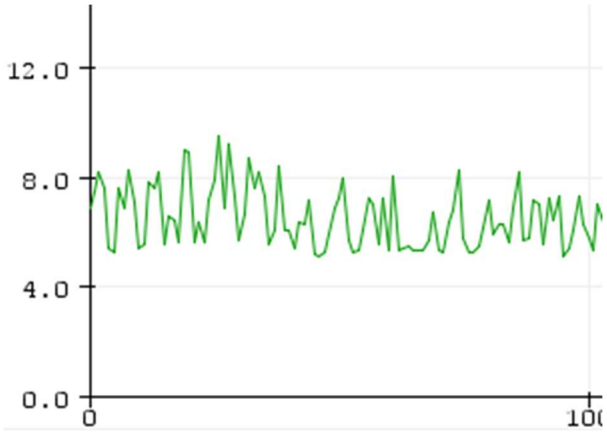



*Figura 11 Valores RMS en una habitación silenciosa y ganancia a 40 dB.*

Con la habitación en silencio, el micrófono obtiene valores rondando el 7, con poca variación. Esto es un buen resultado, porque para un tono uniforme en la habitación, la salida es estable.

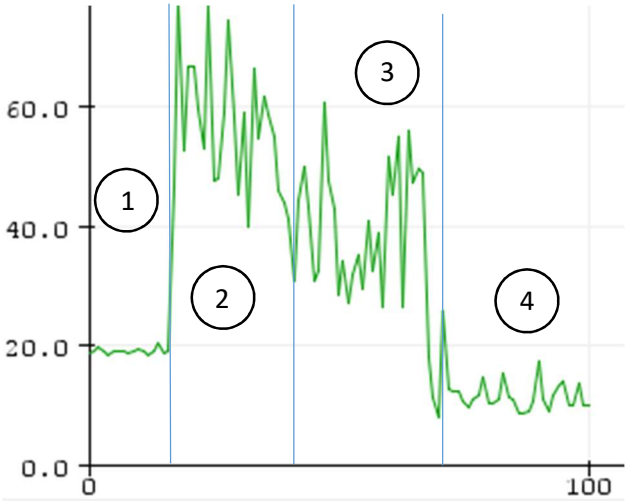



*Figura 12 Valores RMS en una habitación con ruido y ganancia a 40 dB.*

La figura 12 muestra los valores obtenidos con diferentes fuentes de sonido. En primer lugar, están los valores cuando se escucha música con una intensidad de 30 dB. La zona 2 añade la voz humana. Debido a que el tono no es uniforme a lo largo del tiempo, se detectan picos de diferente valor. La medida tomada por el sonómetro del móvil es de 43 dB, con variaciones amplias debido a la inconsistencia del tono. El punto 3 representa a la misma persona hablando sin la música, a unos 38 dB, bajando los valores detectados. Por último, el punto 4 representa a la persona hablando en voz baja sin música. En ese momento, el sonómetro marcaba 28 dB. Este resultado parece adecuarse al objetivo que persigue, ya que diferencia correctamente distintas intensidades de sonido. A continuación, se ve la prueba con la ganancia en 50 dB.

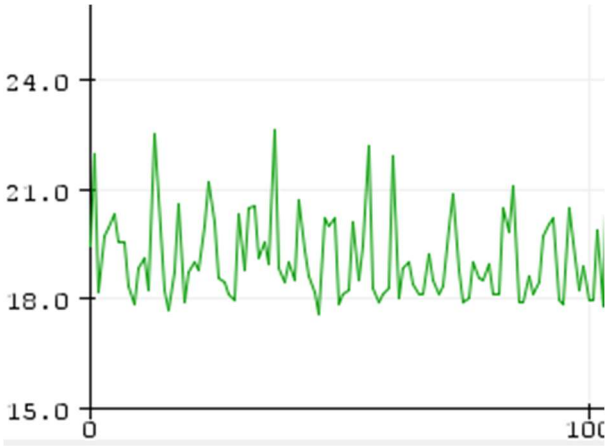



*Figura 13 Valores RMS en una habitación silenciosa y ganancia a 50 dB.*

En la figura 13 se puede ver que el sensor, con una sensibilidad mayor, capta un valor notablemente superior, en torno a los 20, con una variación más amplia. Esto puede llegar a ser un problema si la variación aumenta en niveles más altos

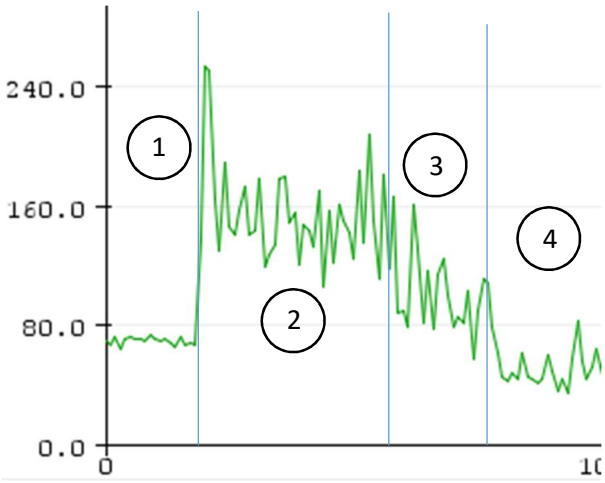



*Figura 14 Valores RMS en una habitación con ruido y ganancia a 50 dB.*

En la figura 14 aparecen los valores RMS calculados con una ganancia de 50 dB en los mismos casos de la prueba anterior. Hay una diferencia mayor en los valores, lo que permite diferenciar más fácilmente los niveles de ruido. Con estos resultados, se propone el uso de una ganancia de 50 dB ya que, aunque en sonidos de intensidad similar la variación sea mayor, también es mayor la diferencia entre niveles de ruido, cosa que facilita el diferenciarlos.

### 2.2.3 Cálculo de decibelios dB

Por último, para el cálculo de los decibelios, se propone hacer una regresión para estimar la equivalencia. Para ello, se emitirá un sonido a un volumen estable para ver la respuesta del micrófono y, con ayuda de una hoja de cálculo en Excel, obtener la fórmula que se ajuste a dichas muestras.

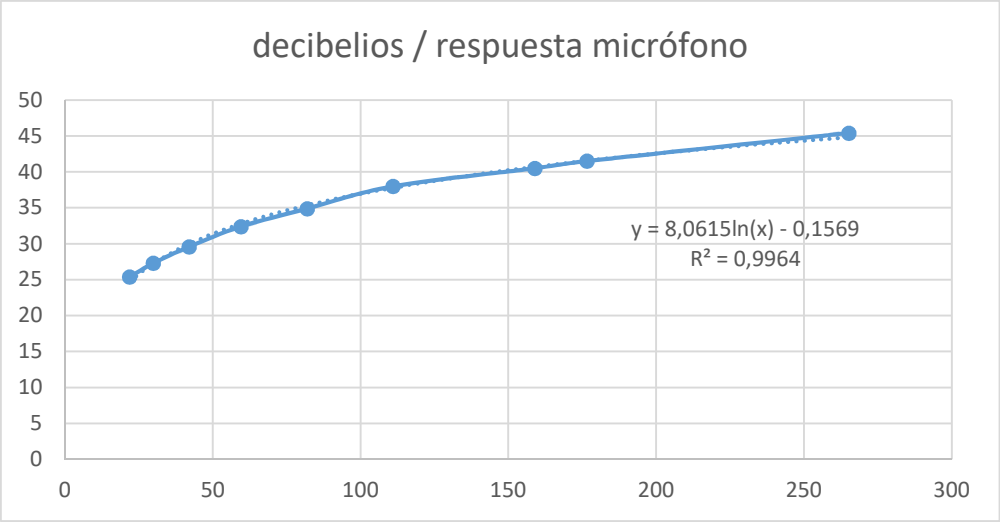



*Gráfico 1 Correlación entre los valores medidos del micrófono y los decibelios*

En el gráfico 1 aparecen valores medidos por el micrófono y su correspondencia en decibelios medidos por la aplicación antes mencionada. La equivalencia resultante viene dada en la ecuación 1:

$$dB = 8{,}0615ln(x) - 0{,}1569$$

*Ecuación 1 Equivalencia valor medido del micrófono a decibelios*

Calcular un logaritmo es un proceso costoso para un microcontrolador, por lo que se va a linealizar la fórmula. No se pretende tener una medida exacta de los decibelios, sino tener la posibilidad de estimar el nivel de ruido. La linealización se va a hacer en tres tramos.

Tramo 1: En el primer tramo se tomarán los valores de decibelios más bajos. En esta zona una variación pequeña en el valor del micrófono supone una diferencia mayor en la escala de decibelios comparado con las medidas más grandes. La fórmula viene dada por la ecuación 2.

$$dB\ (tramo\ 1) = 0{,}2072x + 20{,}948 \begin{cases} x > 22 \\ x \leq 42 \end{cases}$$

*Ecuación 2 Equivalencia del tramo 1*

Tramo 2: El segundo tramo comprende los valores intermedios. La pendiente será menor. Este tramo se rige por la ecuación 3.

$$dB\ (tramo\ 2) = 0{,}1196x + 24{,}915 \begin{cases} x > 42 \\ x \leq 111 \end{cases}$$

*Ecuación 3 Equivalencia del tramo 2*

Tramo 3: Este tramo comprende los valores más elevados, donde existe una diferencia mayor entre los niveles del micrófono comparado con la diferencia de decibelios. Sigue la ecuación 4.

$$dB\ (tramo\ 3) = 0{,}0477x + 32{,}866\ cuando\ x > 111$$

*Ecuación 4 Equivalencia del tramo 3*

Una vez linealizada por tramos se puede comparar con la función original. En el gráfico 2 se puede ver que ambas funciones son muy similares.

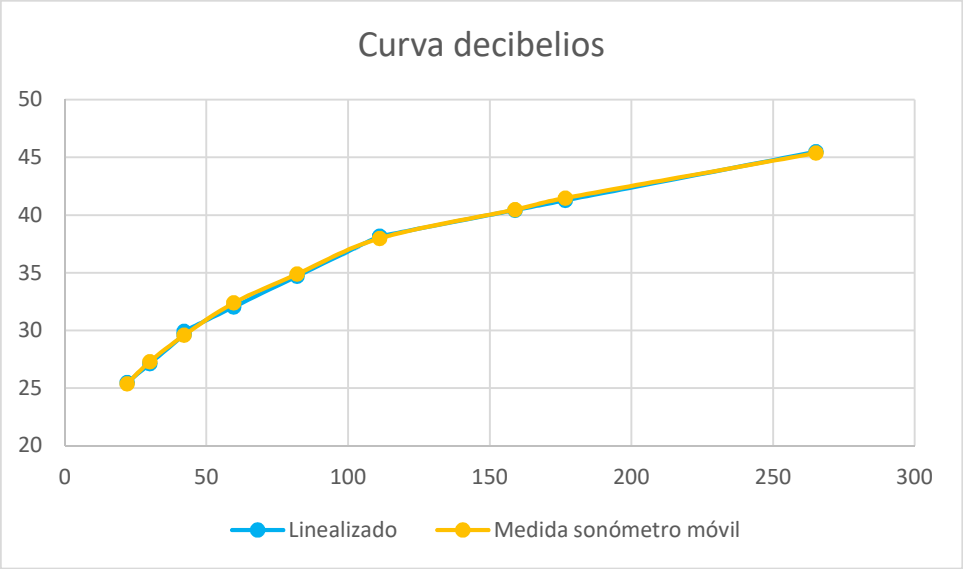



*Gráfico 2 Comparación de la función original con la función linealizada*

Tras obtener la equivalencia entre valores, queda definir un umbral el cual no deberá superarse. Según la ordenanza municipal de Zaragoza, el valor máximo en decibelios que debe registrarse es de 30 dB, los cuales deben ser medidos desde el piso afectado, no desde el foco del ruido. La prueba de sonido consiste en, con un sonómetro en cada vivienda, subir el volumen de ruido en el piso progresivamente hasta que el vecino registre un valor de 30 dB. En ese momento se revisa el valor en la vivienda donde se va a poner el micrófono y el valor que esté dando en ese momento será el umbral de sonido que no se deberá superar. Debido a las diferencias entre edificios, no es posible fijar un único umbral de sonido para todas las viviendas, sino que es un parámetro diferente para cada una.

# 3. Diseño del *firmware* y configuración del *broker*

En este capítulo se presenta por una parte las configuraciones del servidor, donde se habla sobre sus reglas de acceso y barreras de seguridad, el *firmware* con el que funciona el dispositivo y se hace un breve resumen acerca del *dashboard* con el que se pueden realizar diferentes acciones como ver los clientes conectados, los tópicos activos y los mensajes.

## 3.1 Firmware

En el siguiente apartado se explica el funcionamiento del dispositivo, desglosando las funciones que se ejecutarán. El *firmware* se divide en dos partes: *setup* y *loop*. La parte de *setup* se ejecuta una vez cada vez que se inicie el dispositivo mientras que el *loop* se ejecuta continuamente.

El diagrama de flujo de la imagen 15 trata la parte de *setup* del firmware, donde se comprueba el estado de la memoria no volátil, se configura el puerto que se utiliza para la toma de datos, se configuran las conexiones a la red Wi-Fi, al servidor MQTT y se configura la función de actualizaciones remotas OTA *"Over-The-Air"*. Por último, se crean las tareas RTOS *"Real Time Operating System"*. Este tipo de tareas se ejecutan paralelamente al *loop*. Las funciones de configuración de datos se tratan en el próximo capítulo.

El protocolo OTA permite que el dispositivo reciba actualizaciones de *firmware* de forma remota, lo cual evita el problema de ir casa por casa. Para realizar la actualización, el nuevo firmware deberá estar en un repositorio de GitHub para que el microcontrolador pueda descargarlo usando LittleFS, para después cargarlo. LittleFS "Little File System" es un sistema de archivos liviano diseñado para microcontroladores con almacenamiento en memoria flash. Esta función requiere conexión a internet y que el repositorio sea público.

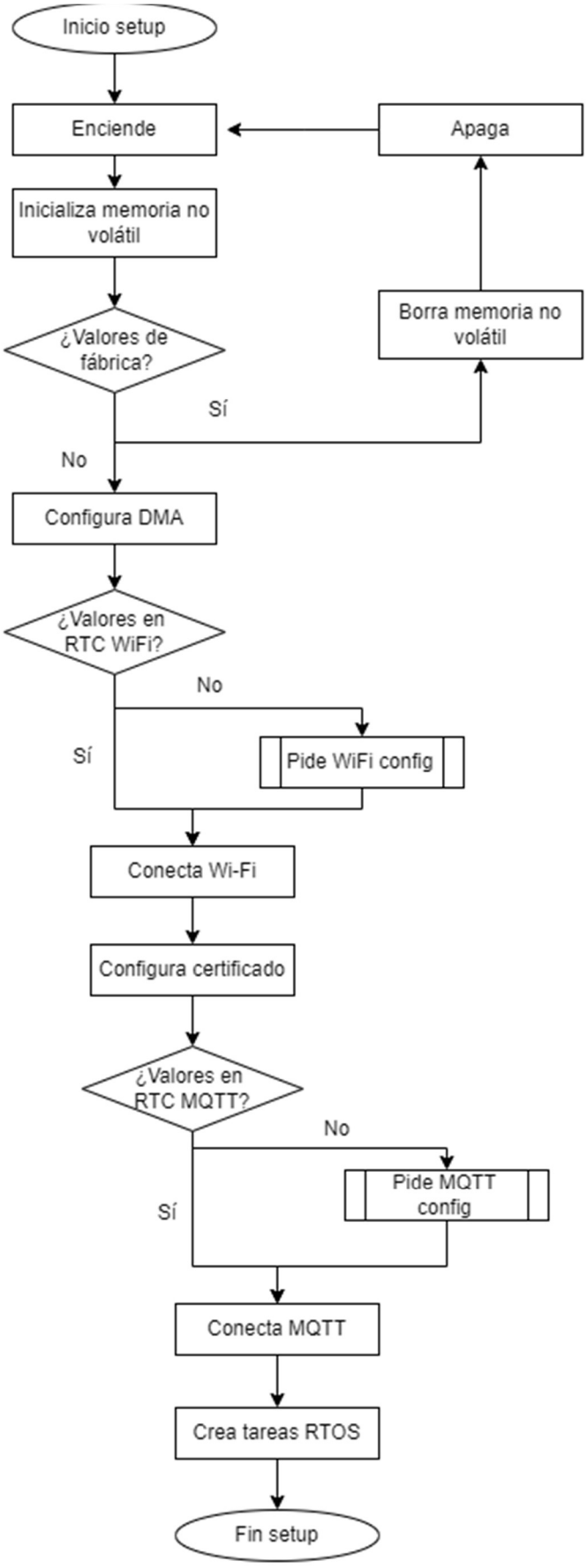



*Figura 15 Diagrama de flujo parte setup del código.*

En la figura 16 se muestra el diagrama de bloques que hace referencia al *loop*.

Para recepción de mensajes el microcontrolador entra en la función de cliente MQTT, donde lee los mensajes que se le han podido enviar para ejecutar las funciones ordenadas, en caso de que las haya. Se le pueden ordenar dos acciones: encender/apagar dos relés y actualizar el sistema.

Cada dispositivo se conecta al servidor con un nombre de usuario y *client ID*. En el punto siguiente, donde se presentan el servidor y las normas ACL, se muestra que un cliente podía conectarse a los tópicos que llevaran su *client ID*, para diferenciarlo de otros dispositivos cliente. Si un dispositivo recibe un mensaje "Enciende secador" en el tópico Acción/ "*Client ID*", el microcontrolador activará el pin conectado al relé asignado al enchufe del secador, encendiéndolo. Funciona de igual manera con el mensaje "Enciende cargador". Del mismo modo, si el microcontrolador recibe el mensaje "Apaga secador", el microcontrolador cumplirá la orden, desactivando el pin y apagando el enchufe conectado al mismo. El otro mensaje que desemboca en una acción es el mensaje "Actualiza" en los tópicos Actualización y Actualización/ *"Client ID"*. Una vez el dispositivo recibe este mensaje cambia el estado de *pendiente* a "True".

Por último, el microcontrolador comprueba la hora. Esta medida se hace para apagar la función de lectura del micrófono, la cual se explica a continuación. Al querer evitar los ruidos por la noche, se limita el trabajo del micrófono al horario de 22:00 a 8:00. Es una opción alternativa al *Deep-sleep*, ya que se necesita la conexión a internet en caso de que haya alguna actualización. También será necesaria en caso de añadir funciones que deban funcionar durante el día, como detectores de monóxido de carbono, control de cerraduras o termostato.


*Figura 16 Diagrama de flujo parte loop del código.*

Paralelamente al bucle definido antes, se ejecutan continuamente dos tareas: comprobación de conexiones a Wi-Fi y al servidor MQTT y el monitoreo del ruido. Para éstas funciones se ha usado la librería FreeRTOS. La tarea principal de FreeRTOS es la gestión de tareas para que se ejecuten de forma eficiente.

Cada minuto, el microcontrolador comprobará que está conectado a la red Wi-Fi. En caso de no estarlo, intentará la conexión. Si esto falla, encenderá el módem donde se encuentra una tarjeta SIM y la configurará para conectarse a la red y al servidor. Si el dispositivo está conectado a la red Wi-Fi y el estado *pendiente* está en *True*, el microcontrolador realizará la actualización de *firmware*. La actualización del estado *pendiente* está explicada en la parte de *setup*. Esta acción sólo se realiza mediante Wi-Fi ya que tras la actualización el dispositivo se reiniciará para utilizar el *firmware* actualizado, requiriendo conectarse la primera vez por Wi-Fi. Adicionalmente, si la conexión es por medio de Wi-Fi, se comprobará la conexión al servidor MQTT, intentando la conexión en caso de estar desconectado. Si el estado de conexión inicial era el de tarjeta SIM, y el dispositivo es capaz de conectarse a la red Wi-Fi, se apagará el módem.

La otra tarea es la recogida de datos por parte del micrófono. En esta función se toman valores durante un segundo para después obtener el valor RMS y transformarlo a un valor en decibelios. Si dicho valor supera el umbral permitido, el microcontrolador enviará un mensaje al servidor, notificando de la infracción. Esta función tiene al principio una condición, que es estar en el periodo de muestreo. Si se encuentra en horario de día, la función no tomará ningún dato.

Tomar muestras a 16 kHz puede resultar en un consumo de recursos excesivo para la CPU. La solución que se plantea es el uso del DMA, *“Direct Memory Access”*, por sus siglas en inglés o acceso directo a la memoria en español. El DMA es un método que permite la transferencia de datos entre los periféricos y la memoria u otros periféricos sin requerir la intervención de la CPU. El uso de DMA puede mejorar el rendimiento de la transferencia de datos mientras que libera la CPU para la realización de otras tareas. La placa LilyGo T-A7670E tiene dos pines capaces de realizar la conexión mencionada, el 36 y el 39.

Los diagramas de flujo de estas tareas están en la figura 17.

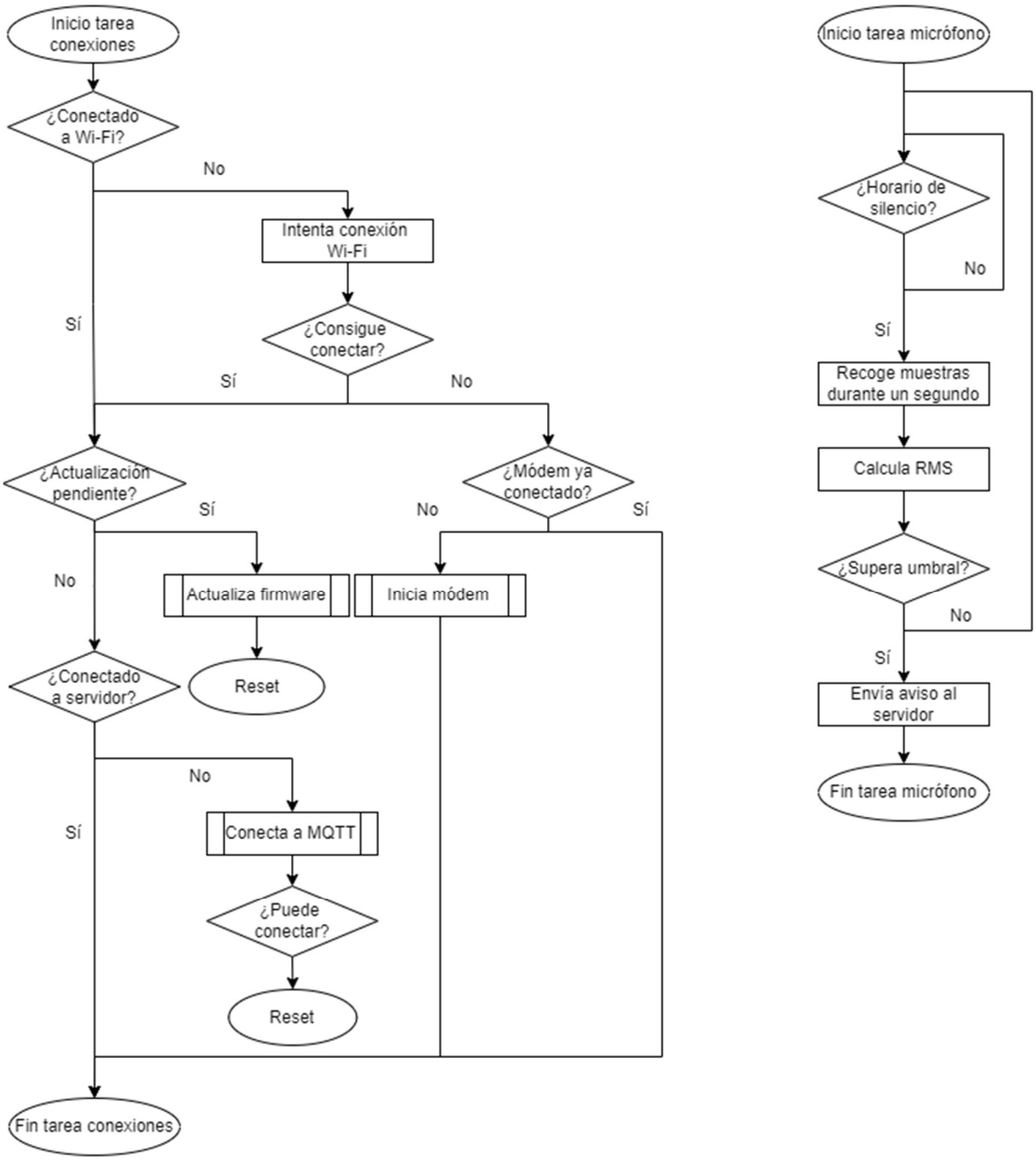



*Figura 17 Diagramas de flujo de las funciones RTOS.*

## 3.2 Configuración del servidor y seguridad

A continuación, se muestra un resumen acerca del *broker* utilizado para controlar el servidor, así como las configuraciones que se le han aplicado para el uso que se le va a dar.

El *broker* elegido ha sido EMQX, de la empresa “EMQ”. EMQX permite la conexión de 100 millones de dispositivos de forma simultánea con una latencia inferior a 10 milisegundos, lo cual cumple con creces el objetivo que se pretende abordar. Además, permite personalizar el acceso al servidor por medio de

normas de autenticación y conexión por puertos TLS *"Transport Layer Security"*/SSL *"Socket Secure Layer"*, lo cual añade una capa de seguridad.

El servidor está alojado en la nube, para permitir el acceso al *broker* desde cualquier dispositivo con acceso al mismo. Para ello, se han contratado los servicios de "Google Cloud". "Google Cloud" es una plataforma de servicios de computación en la nube creada por Google. Es utilizada por empresas de todo el mundo para almacenar datos y procesar información.

En este momento contamos con un servidor extremadamente vulnerable, ya que cualquiera podría acceder al mismo en cualquier momento, sin necesidad de identificarse, tan sólo conociendo la IP de la máquina virtual creada en "Google Cloud". Una máquina virtual es un entorno *software* que emula un sistema informático dentro de otro sistema físico. Para evitar una situación de ataque se plantean diferentes capas de seguridad, las cuales se muestran a continuación.

### 3.2.1 Evitar conexiones anónimas y de usuarios no permitidos

El primer punto que se aborda es el permitir la conexión sólo al servidor a dispositivos autorizados. Para ello utilizaremos uno de los administradores de datos con la interfaz más amistosa para el usuario promedio, "phpMyAdmin". En él, es posible configurar las normas de acceso al servidor mediante ACL *"Access Control List"* por sus siglas en inglés o lista de control de acceso. Dichas normas recogen la lista de usuarios aceptados y sus contraseñas y los permisos que tienen dichos usuarios (posibilidad de publicar y/o suscribirse a diferentes tópicos).

| id | username | password | salt | is_superuser | created |
|---|---|---|---|---|---|
| 1 | emqx | efa1f375d76194fa51a3556a97e641e61685f914d446979da5... | NULL | 1 | NULL |
| 2 | cliente | efa1f375d76194fa51a3556a97e641e61685f914d446979da5... | NULL | 0 | NULL |

*Figura 18 Lista de usuarios permitidos en el servidor con su contraseña.*

En la figura 18 se ven los usuarios que pueden entrar al servidor y su contraseña. La contraseña está encriptada por SHA256. Ambas contraseñas pueden cambiarse en cualquier momento. En dicha imagen, se ve que el usuario "emqx" tiene permisos de ser súper usuario, mientras que el usuario "cliente" es un usuario normal, sin permisos especiales. Se entiende que "emqx" es el usuario de la empresa que recolecta toda la información dada por los demás usuarios.

| id | allow<br>0: deny, 1: allow | ipaddr<br>IpAddress | username<br>Username | clientid<br>ClientId | access<br>1: subscribe, 2: publish, 3: pubsub | topic<br>Topic Filter |
|---|---|---|---|---|---|---|
| 1 | 0 | NULL | $all | NULL | 3 | +/# |
| 2 | 1 | NULL | $all | NULL | 2 | Ruido/%c |
| 3 | 1 | NULL | $all | NULL | 3 | Accion/%c |
| 4 | 1 | NULL | $all | NULL | 1 | Actualizacion |
| 5 | 1 | NULL | $all | NULL | 1 | Actualizacion/%c |
| 6 | 1 | NULL | $all | NULL | 2 | Conexion |

*Figura 19 Lista de normas ACL del servidor.*

En la figura 19 se aprecian una serie de normas sobre las acciones que pueden realizar los usuarios del servidor. La primera de todas prohíbe a todos los usuarios la suscripción y publicación de mensajes en cualquier tópico. Es una medida general ya que el resto de normas dará permisos concretos de donde pueden tomar acción. La siguiente norma permite a los usuarios publicar en el tópico Ruido/ "*Client ID*". De esta forma, damos acceso a todos los dispositivos a enviar la información de sus micrófonos cuando éstos detecten un nivel de ruido superior al permitido.

La tercera norma autoriza a los dispositivos la suscripción al tópico Acción/ "*Client ID*", la cual permite a los dispositivos recibir mensajes en dicho tópico para actuar en consecuencia al mensaje recibido.

Las normas cuarta y quinta permiten a los dispositivos suscribirse a los tópicos Actualización y Actualización/ "*Client ID*". Esta medida permite actualizar el *firmware* de dispositivos concretos si se especifica cuál o todos a la vez si el mensaje es enviado al tópico Actualización.

Por último, está la norma que permite la publicación de mensajes al tópico Conexión, donde los dispositivos enviarán mensajes informando de si están conectados o desconectados.

### 3.2.2 Permitir sólo conexiones seguras

MQTT es un protocolo ligero que no cuenta con medidas de seguridad por defecto. Si dejáramos el servidor así, los mensajes se transmitirían en texto plano, lo cual implicaría la posibilidad de sufrir ataques de interceptación de mensajes. Estos ataques, también llamados *Man in the middle*, permiten la manipulación de mensajes enviados.

Es por ello que se exigirán conexiones con SSL/TLS. Al usar este elemento de seguridad, se cifra la conexión entre el servidor y el cliente, lo cual complica que los mensajes que puedan llegar a ser interceptados, leídos o modificados por extraños.

En el momento de conexión del cliente al servidor, el *broker* muestra un certificado que verifica su identidad, para así evitar que el cliente se conecte a un servidor equivocado. Una opción adicional sería el utilizar certificados mutuos, donde el cliente también tendría que mostrar su certificado al *broker*, para asegurar que es un cliente legítimo.

## 3.3 *Dashboard* y cliente MQTTX

Con el servidor en funcionamiento y un dispositivo conectado solo queda ver lo que el administrador del servidor puede controlar. Todo ello se hace desde el *dashboard*, que permite controlar lo que está ocurriendo en todo momento. Desde el *dashboard* se puede ver la lista de dispositivos conectados con sus suscripciones, los tópicos existentes, además de poder conectarse con un usuario para mandar los mensajes de control.

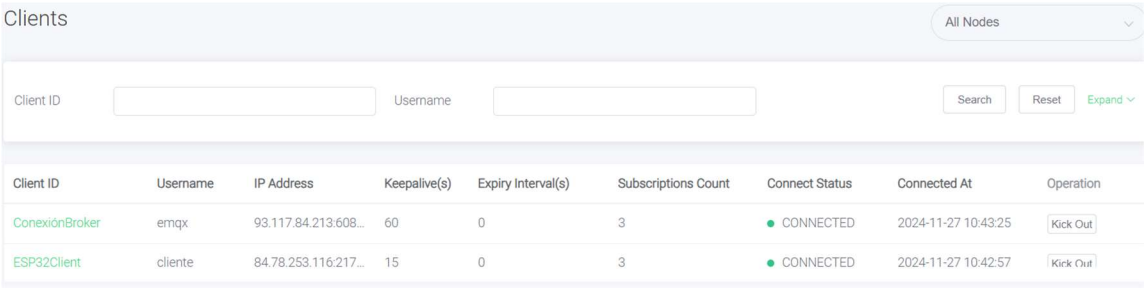



*Figura 20 Apartado vista de clientes.*

La figura 20 muestra que hay dos clientes conectados al servidor. Junto al *Client ID*, se ven otros datos como el nombre de usuario, la dirección IP desde la que se conecta, el número de suscripciones, que debería ser siempre 3 para los detectores de ruido, más información como la hora de conexión o su estado de conexión. El segundo cliente es el propio del bróker. El bróker tiene un apartado de conexión al servidor para enviar y recibir mensajes.

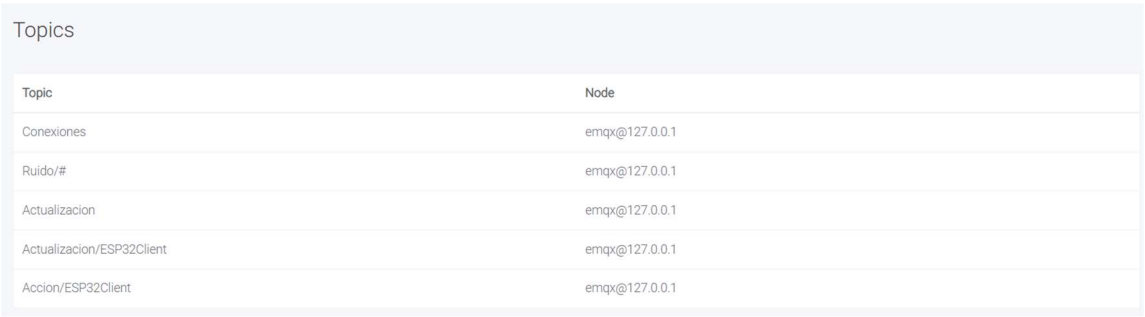



*Figura 21 Apartado vista de tópicos existentes*

En la figura 21 se muestran los tópicos a los que se han suscrito los diferentes clientes. El # del tópico ruido/# es un comodín multinivel. Con esta medida, el administrador no tiene que conectarse a un tópico diferente de forma manual cada vez que se conecte un dispositivo, sino que lo está por defecto. Esta sección se puede relacionar con la figura 22, que muestra las suscripciones de cada dispositivo. Aquí se puede comprobar que los detectores de ruido no se hayan conectado a tópicos que no deberían.

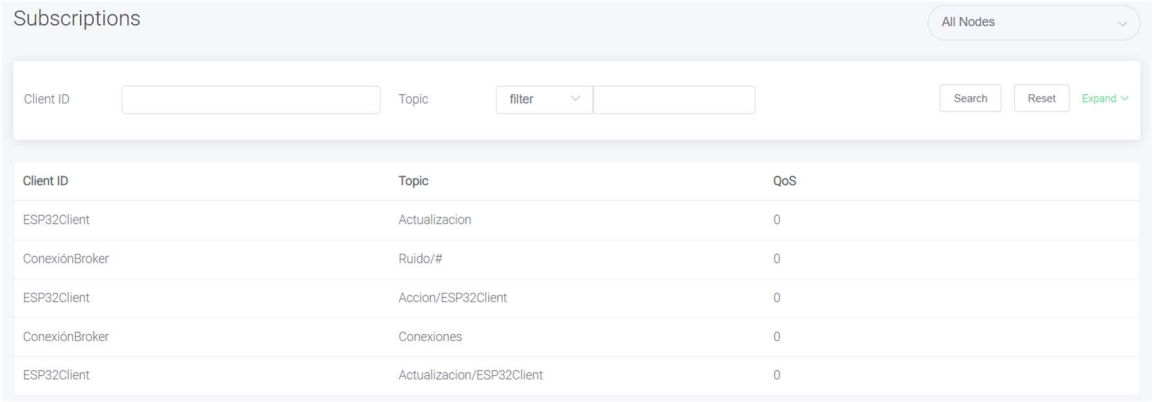



*Figura 22 Apartado vista de suscripciones*

Si bien el *dashboard* permite la suscripción y envío de mensajes a los tópicos, existe una herramienta más visual para el usuario: MQTTX.

Esta aplicación permite conectarse al servidor de la misma forma que los microcontroladores o el *dashboard*, pero con la ventaja de poder filtrar los mensajes por tópicos. En la figura 23 se ve como el cliente “emqx”, que es súper usuario por las reglas ACL mencionadas anteriormente, está conectado con SSL y suscrito a los tópicos de interés.

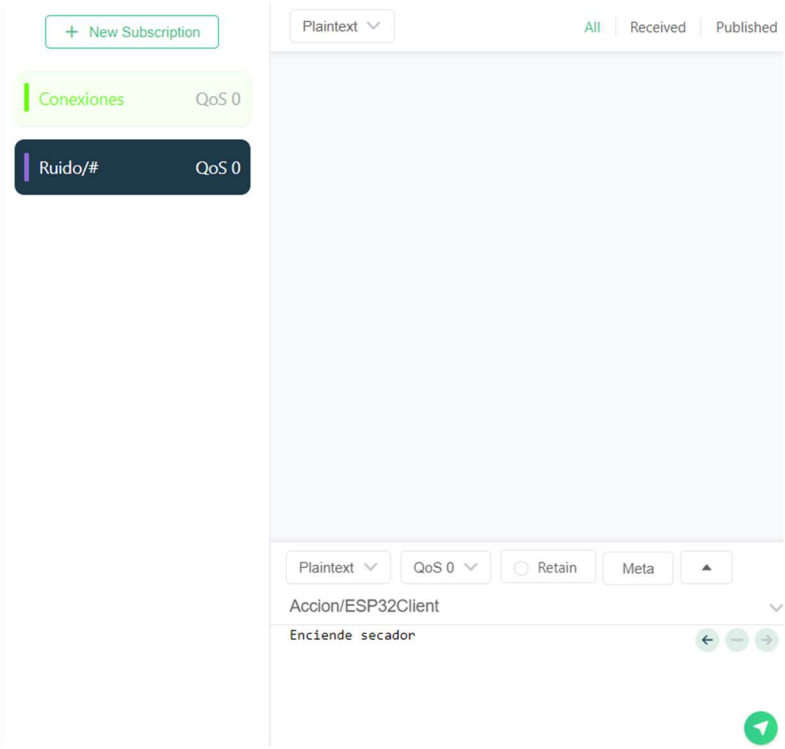



*Figura 23 Aplicación cliente MQTTX*

# 4. Pruebas en el *setup*

En el siguiente apartado se pondrá a prueba el *firmware* encargado del control de ruido y envío de información cuando se detecten sonidos altos. Se pretende llevar este desarrollo a un entorno de producción, por lo que es necesario la validación del correcto funcionamiento del mismo.

En el *setup* existen funciones que requieren acción humana para introducir los datos para la conexión a la red Wi-Fi y al servidor MQTT, lo que puede ser un foco de errores.

Para explorar todas las vías posibles se han creado varios *scripts* de Python que responden de forma automática a los mensajes recibidos por puerto serie desde el microcontrolador. Las zonas de mayor conflicto son las conexiones a internet, ya sea de forma manual o escaneando las redes y la conexión al servidor MQTT, puesto que estas configuraciones son necesarias para poder enviar los datos. Una vez se hace la conexión, no es necesario volver a introducir los valores puesto que se guardan en la memoria no volátil.

## 4.1 Conexión Wi-Fi por escáner

La primera opción que se presenta para la conexión Wi-Fi es por medio del escaneado de redes. En este caso el usuario tendrá una lista de redes detectadas para elegir a cuál conectarse. Tras ello, se pide una contraseña y la confirmación de los datos.

En este apartado, se contemplan cuatro casos, nombrados en la tabla 1:

| | SSID | Contraseña | Confirmación | Salida esperada |
|---|---|---|---|---|
| Caso 1 | SSID | Contraseña incorrecta | Y | No conecta, reinicia el ESP32 |
| Caso 2 | XXX | XXX | N | “Por favor, ingrese nuevamente la configuración.” |
| Caso 3 | SSID | Contraseña correcta | Y | “¡Conectado a Wi-Fi!” |

*Tabla 1 Casos de intento de conexión a Wi-Fi y las salidas esperadas.*

En la tabla anterior no se contemplan los errores que pueden darse si en el apartado “Confirmación” se manda un mensaje diferente de “Y” o “N”, en cuyo caso, el usuario simplemente recibe un aviso de “Entrada inválida. Por favor ingrese 'Y' para Sí o 'N' para No.”. El usuario queda atrapado en un bucle hasta que responde “Y” o “N”. Tampoco se permitirá al usuario confirmar los datos si la contraseña está en blanco. La respuesta del microcontrolador en ese caso será “Contraseña no introducida. Por favor, intente de nuevo.”.

Otra medida de control que se ha incluido en este *script* es qué pasaría si el usuario, cuando se le pregunta que opción de configuración Wi-Fi, responde algo diferente a “E” (escaneo) o “M” (manual). La respuesta esperada del microcontrolador es volver a preguntar hasta que se responda con un valor válido.

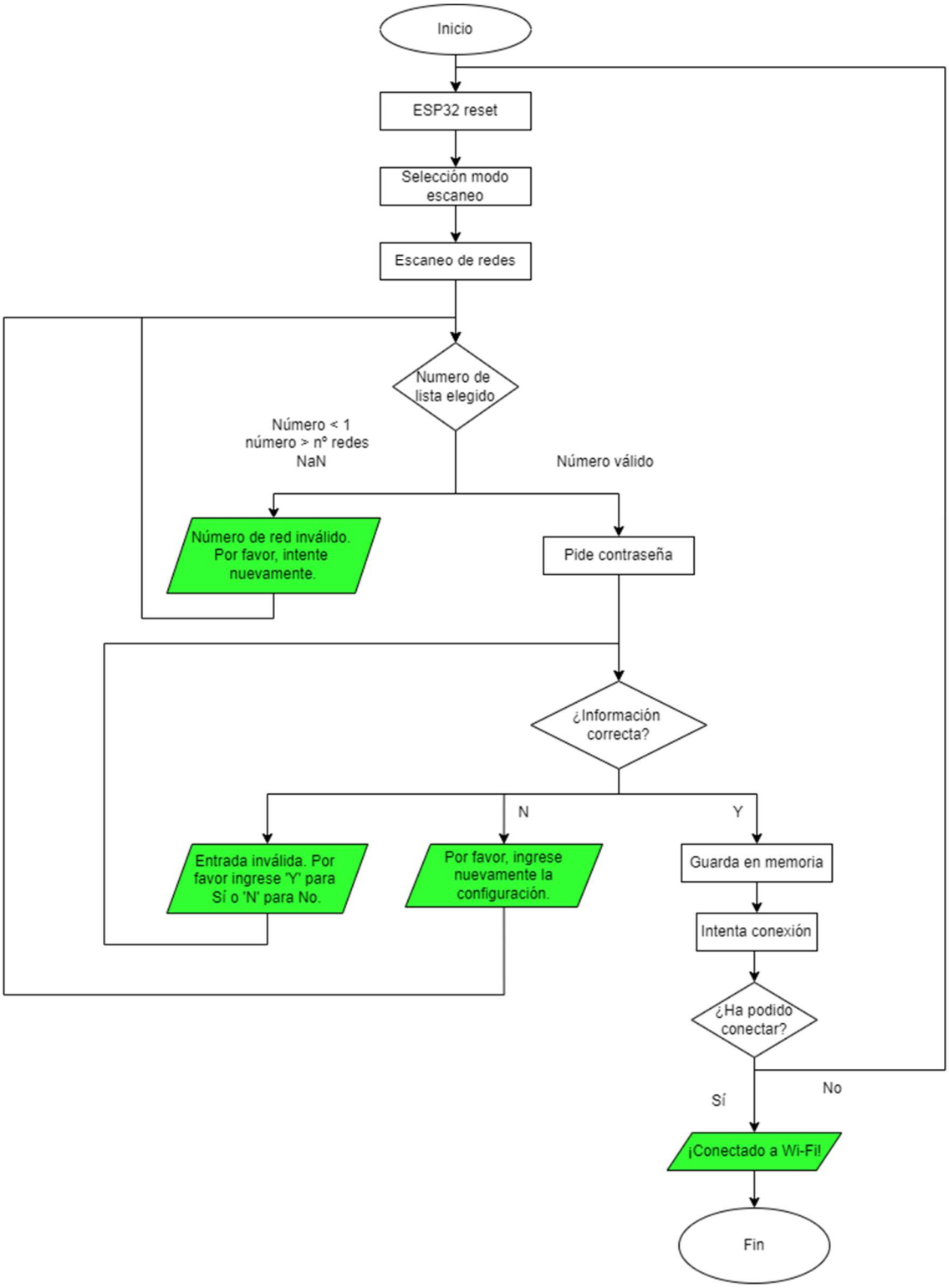



*Figura 24 Diagrama de flujo de la función conectar a Wi-Fi mediante escaneo de redes.*

El diagrama de bloques de la figura 24 se muestra el flujo del código con las diferentes opciones según las respuestas del usuario. En este caso se piden tres entradas del usuario: el número correspondiente a la red elegida, la contraseña y la confirmación. A continuación, en la figura 25, se pueden ver los resultados de la prueba.

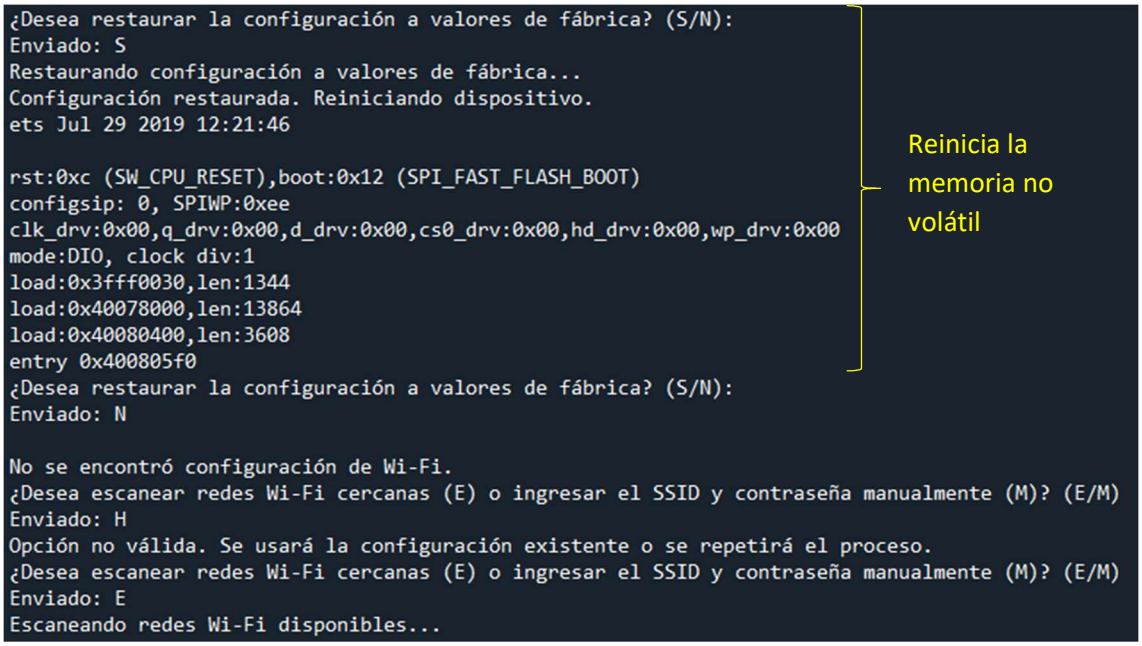



*Figura 25 Captura en Python donde se borra la memoria no volátil y se elige la conexión mediante escaneo.*

En primer lugar, se limpia la memoria. Si hubiera datos en ella no se podría configurar la red, al estar ya configurada. Tras reiniciar el dispositivo, el microcontrolador da la opción de elegir el modo de conexión a la red. Se va a comprobar el método de escaneo, por lo que se introduce “E”. Se puede ver que la primera vez que da a elegir el método de conexión se elige una opción no válida. El microcontrolador la detecta y manda un aviso para después volver a enviar la pregunta. Por último, al entrar en modo escaneo, el dispositivo empieza la búsqueda de redes disponibles.

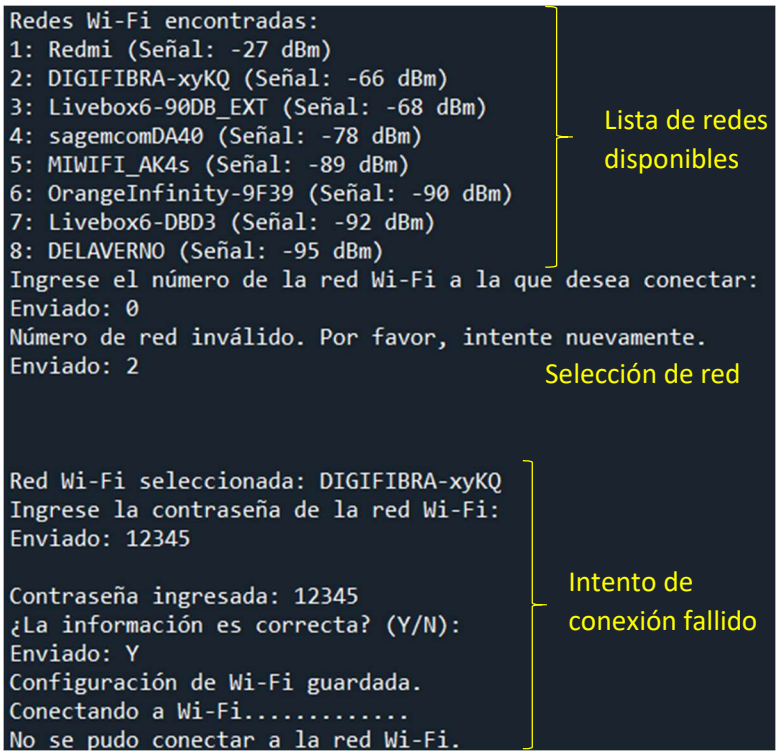



*Figura 26 Captura en Python donde se muestra el listado de redes disponibles y se elige un SSID incorrecto.*

En la figura 26 se presenta el primer intento de conexión. Tras buscar las redes disponibles para el dispositivo, éste manda un listado de redes a las que puede conectarse, para que el usuario elija la propia. Esto se hace eligiendo el número asociado a cada red. Este caso va a probar una conexión fallida, por lo que no se elige ni la red ni la contraseña correcta. El resultado de la conexión, como cabía esperar, es fallido. También se ha enviado un número de red inválido, ya que “0” no es un valor existente en la lista. La respuesta ante este tipo de entradas es volver a pedir la información.

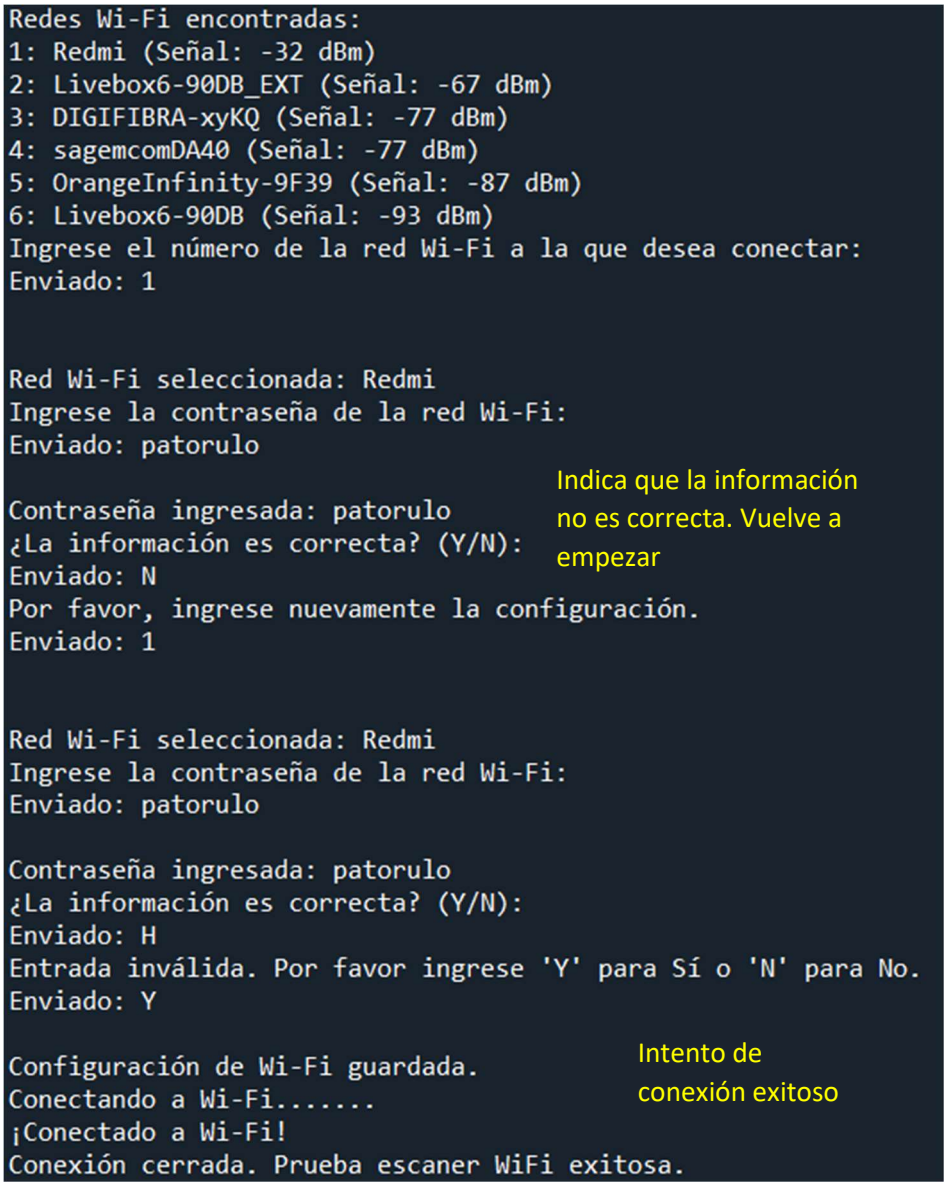



*Figura 27 Captura en Python donde se indica que la información no es correcta, y después se intenta la conexión con valores correctos.*

Por último, en la figura 27, se intenta la conexión con los valores adecuados, lo que permite el acceso a internet. También se aprecia que cuando la confirmación es negativa, el microcontrolador simplemente vuelve a pedir la información para la conexión sin tener que escanear de nuevo las redes disponibles.

## 4.2 Conexión Wi-Fi manual

La opción alternativa de conexión a la red Wi-Fi es de forma manual. En este caso, el dispositivo pide al usuario el SSID de la red a la que quiere conectarse, su contraseña y una confirmación de que los valores introducidos son correctos.

Las combinaciones de datos de entradas con las salidas son exactamente iguales a las del apartado anterior.

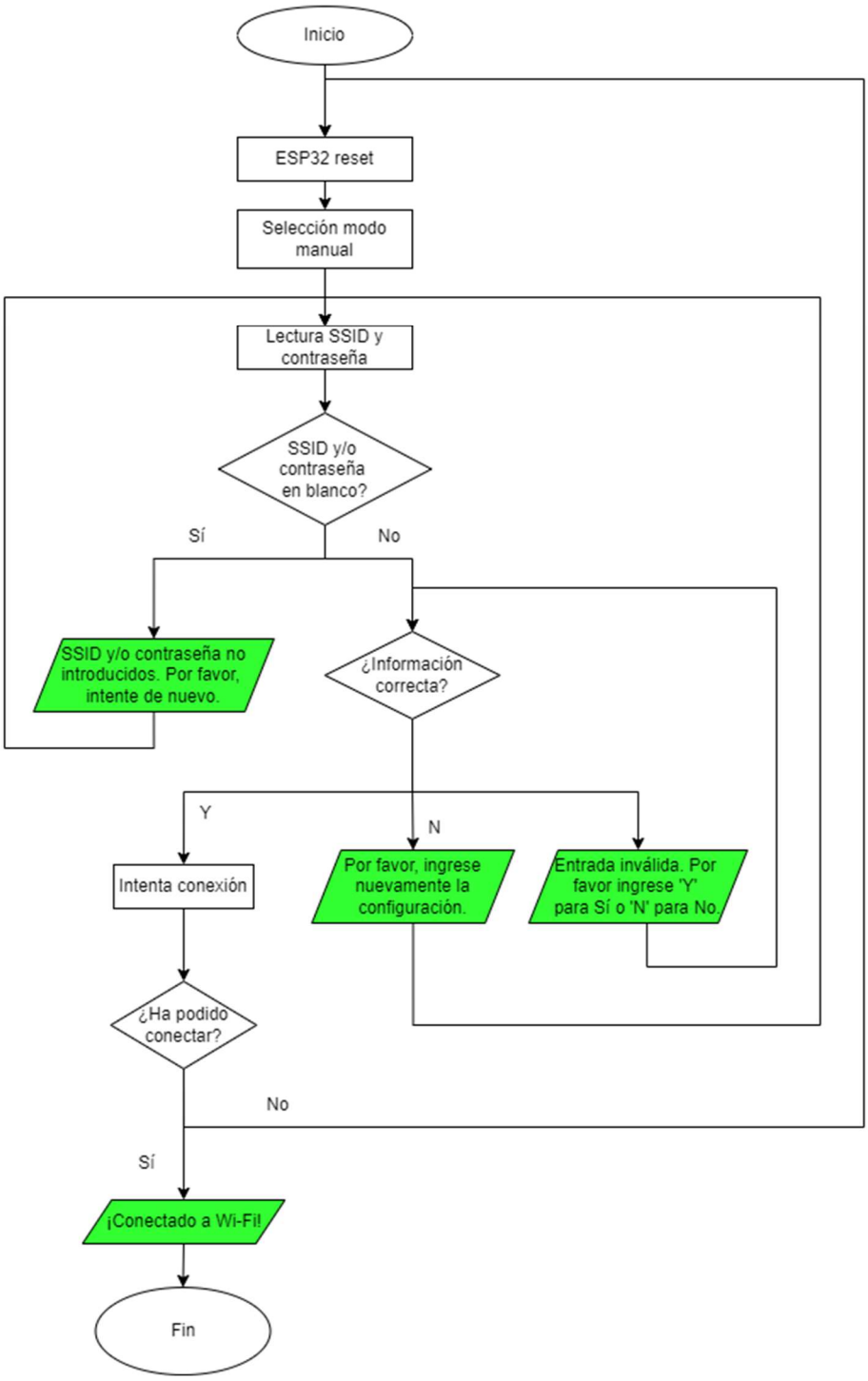



*Figura 28 Diagrama de flujo de conexión Wi-Fi de forma manual*

El diagrama de bloques de la figura 28 muestra el curso que sigue el *script* de Python para comprobar todos los casos de la tabla. El *script* comienza con un reseteo automático de la memoria no volátil. Se da por hecho que la selección de conexión será manual. En ese momento se piden el SSID de la red y su contraseña. Si alguna de las entradas se queda en blanco se volverán a pedir ambos hasta que los dos sean cumplimentados. Después, se pide la confirmación al usuario de que los valores introducidos son correctos. Esta vez, el bucle solo aceptará las respuestas "Y" (sí) o "N" (no), repitiendo la pregunta en caso de introducir un valor diferente. Si la respuesta es "N" se volverán a pedir los datos requeridos con anterioridad. Cuando la respuesta es "Y" el microcontrolador intentará la conexión Wi-Fi en la red seleccionada con la contraseña indicada. En caso de no poder conectarse el dispositivo se reiniciará mientras que en caso de conectarse mostrará por puerto serie el mensaje "Conectado a la red Wi-Fi", concluyendo el proceso de conexión.

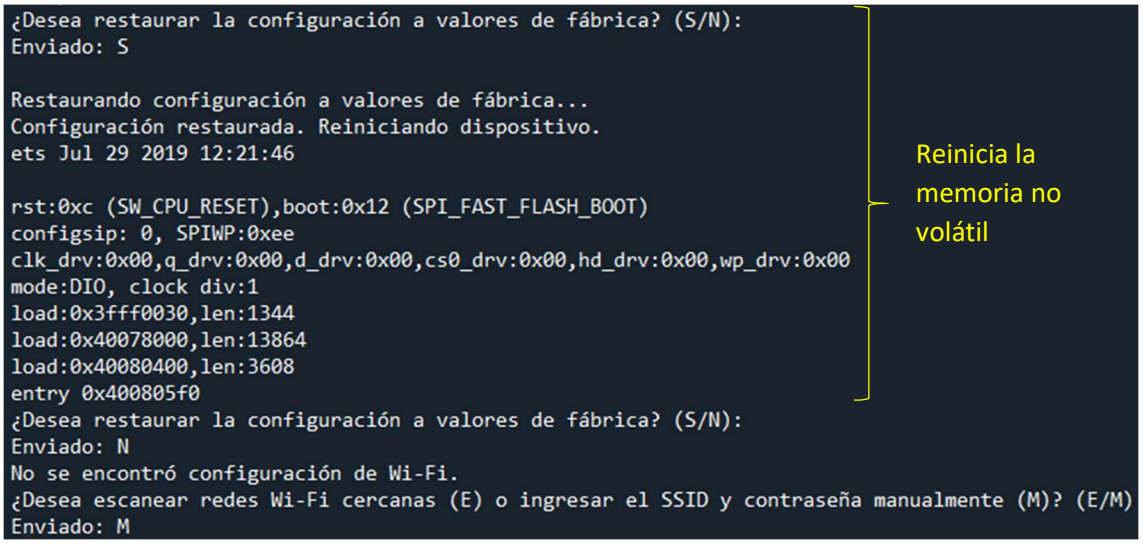



*Figura 29 Captura en Python donde se borra la memoria no volátil y se elige la conexión de modo manual.*

En la figura 29 se pueden como se borra la memoria no volátil y se selecciona el modo manual.

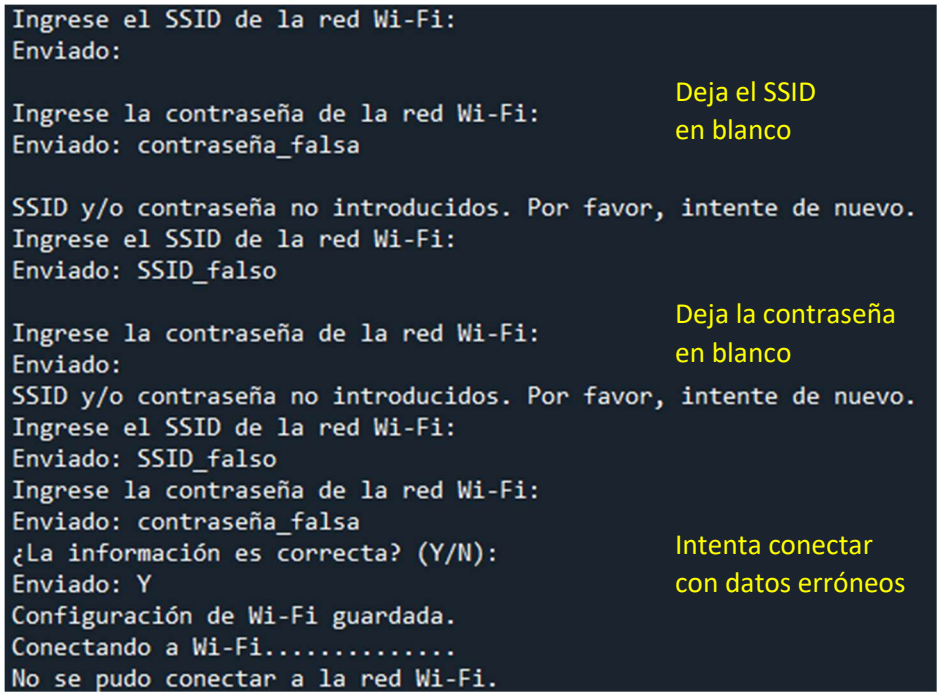



*Figura 30 Captura en Python donde se intenta la conexión con datos en blanco o erróneos.*

La figura 30 muestra tres intentos de conexión: el primero deja el SSID en blanco y el segundo la contraseña. En ambos la respuesta del microcontrolador es la misma, pidiendo al usuario que vuelva a intentarlo ya que ha dejado apartados en blanco. Por último, se intenta conectar a la red con un SSID y una contraseña erróneos, fallando, como era de esperar, la conexión. Esta respuesta es la misma que si de introdujera el SSID adecuado con una contraseña equivocada y viceversa. Tras el fallo en el intento de conexión, el microcontrolador se reinicia para buscar otra configuración.

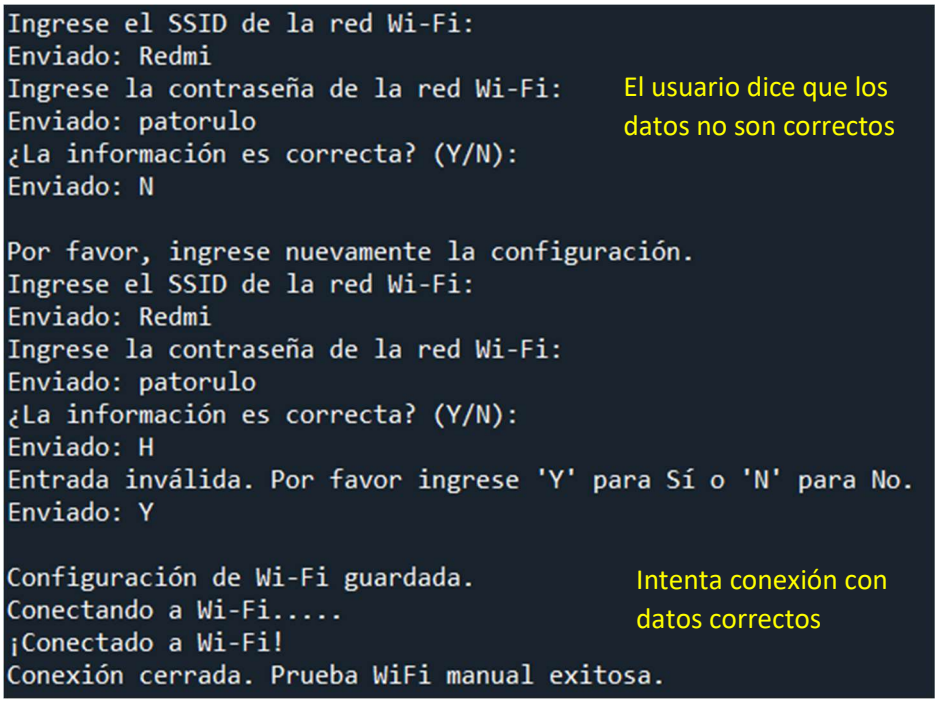



*Figura 31 Captura en Python donde se intenta la conexión con los datos correctos.*

Para acabar con esta prueba se presentan los casos en los que la confirmación es negativa, lo cual lleva al usuario a volver a introducir los datos y el caso en el que el SSID y la contraseña son correctos, lo que permite la conexión a la red Wi-Fi. Se puede apreciar también el intento de confirmar los datos con una entrada "H", la cual es rechazada por el microcontrolador, que pide de nuevo la confirmación. Las respuestas del microcontrolador se ven en la figura 31.

## 4.3 Conexión a servidor MQTT

Una vez conectados a internet sólo queda por configurar la conexión al servidor MQTT. Para ello, el microcontrolador pedirá al usuario que introduzca el servidor al que quiere conectarse, el usuario con el que quiere entrar y la contraseña. Estos dos últimos valores son iguales para todos los dispositivos, ya que entrarán con una cuenta limitada a enviar y recibir mensajes en tópicos definidos en las reglas ACL. A continuación, en la tabla 2, se muestran los valores de entrada y sus respectivas salidas.

| | Servidor | Registro usuario | Salida esperada |
|---|---|---|---|
| Caso 1 | Servidor equivocado | XXX | "Fallo al conectar, rc=-2" |
| Caso 2 | Servidor correcto | Usuario/contraseña equivocados | "Fallo al conectar, rc= 5" |
| Caso 3 | Servidor correcto | Usuario y contraseña correctos | "Conectado." |

*Tabla 2 Casos de intento de conexión al servidor MQTT y las salidas esperadas.*

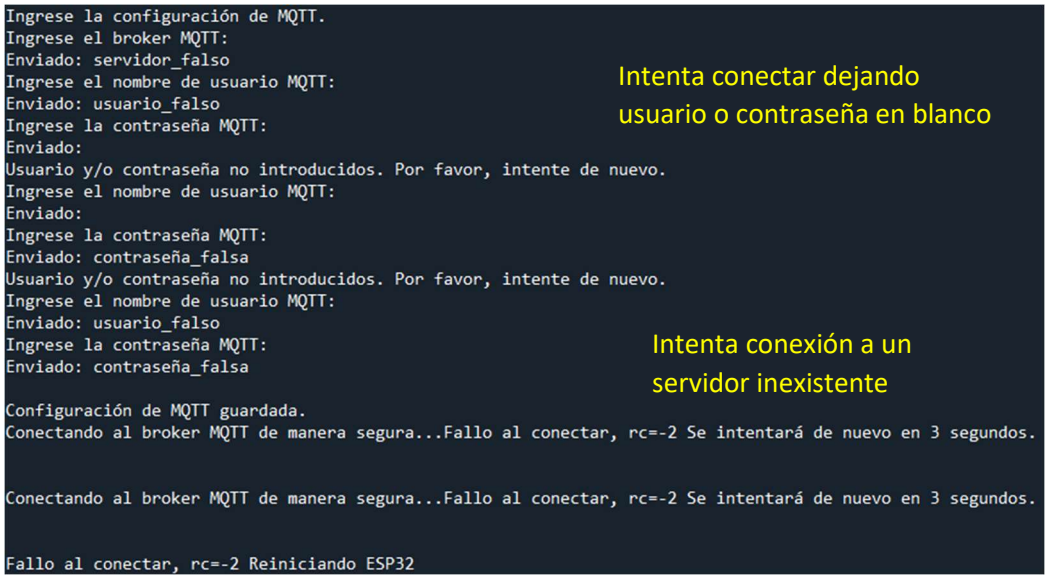



*Figura 32 Captura en Python donde se intenta conectar sin usuario o contraseña y después a un servidor inexistente*

La figura 32 muestra como en la configuración se requiere un usuario y una contraseña. No se permite que ningún campo quede vacío. Tras introducir valores equivocados en el servidor y usuario, el microcontrolador intenta conectarse a él, recibiendo un error del tipo -2. Este error indica que el cliente no ha conseguido traducir el nombre del dominio a una dirección IP válida mediante DNS *"Domain Name System"*. Tras tres intentos de conexión, el microcontrolador se reinicia para volver a configurar la conexión al servidor.

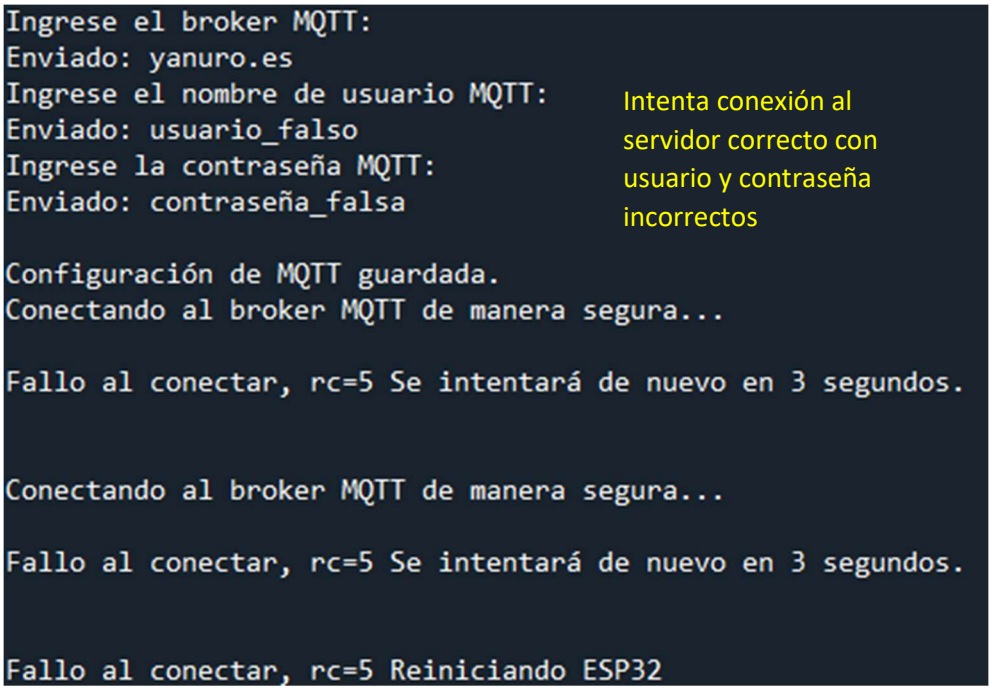



*Figura 33 Captura en Python donde se intenta ingresar al servidor con credenciales falsas*

El siguiente intento de conexión es utilizando el nombre del dominio correcto, pero con credenciales no reconocidas por las reglas ACL mencionadas en el capítulo anterior, lo que resulta en un error tipo 5. Este error indica que el cliente que intenta conectarse no tiene permiso ya que no está autorizado, en este caso, por las reglas ACL. Esto se ve en la figura 33.

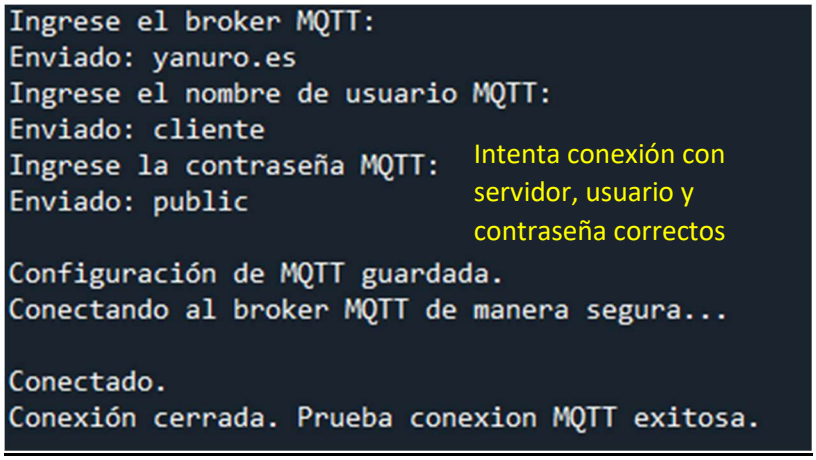



*Figura 34 Captura en Python donde se intenta la conexión al servidor con credenciales correctas*

Por último, en la figura 34, la conexión exitosa al servidor se da cuando el servidor es válido y las credenciales están registradas en las reglas ACL. Hay ocasiones en las que la conexión puede demorarse, pero al final se conecta. Es por eso que en todos los intentos se dan 3 oportunidades antes de resetear el microcontrolador. Si se hubiera añadido el código de error en este caso, este respondería con rc=0, lo que indica que la conexión se ha realizado con éxito.

# 5. Conclusiones y líneas futuras

## 5.1 Conclusiones

A lo largo de este proyecto se ha descrito paso a paso el diseño y desarrollo de un dispositivo capaz de medir el nivel de ruido existente en una sala. Inicialmente se han presentado los componentes del dispositivo en base a los objetivos que se planteaban, justificando su elección. Tras ello, se ha explicado la configuración del micrófono y de cómo se procesan los datos para traducir los miles de valores por segundo recogidos por el micrófono a valores en decibelios, los cuales sí se pueden medir y comparar.

Por el otro lado, se ha presentado también el servidor MQTT utilizado para recoger las alertas de sonido y enviar mensajes de acción al dispositivo, así como las configuraciones de seguridad por medio de conexiones seguras y una lista de accesos.

En conclusión, el presente trabajo culmina con un dispositivo totalmente funcional e independiente, que no requiere visitas al propio inmueble para su mantenimiento o actualización, salvo para añadir componentes. La función de activar o desactivar enchufes con un mensaje desde el servidor abre las puertas a futuras ampliaciones, consolidando al dispositivo como una solución versátil y escalable.

## 5.2 Líneas futuras

Una vez tratados los puntos del trabajo, solo queda definir las líneas futuras que puede tomar el proyecto para sumar funcionalidad o hacer la visualización de datos más clara.

La primera ruta que puede tomar el trabajo es la creación de una interfaz más amistosa con el usuario a la hora de recibir y enviar mensajes MQTT desde el sistema central. Un ejemplo podría ser una página HTML que permita ver un resumen de todos los dispositivos conectados y los avisos que ha tenido que enviar.

Por otro lado, y siguiendo el tema de sensores, se puede añadir un detector de monóxido de carbono conectado a una alarma. Esto no requeriría añadir un microcontrolador, ya que la placa que se está usando cuenta con pines disponibles.

Poniendo el punto de mira en la domotización del hogar, es posible utilizar el microcontrolador como base de operaciones de la casa, y podría usarse para el control de luces, aire acondicionado o incluso de abrir y cerrar las puertas. Esta última aplicación elimina la necesidad de entrega de llaves, tanto al principio como al final del periodo de alquiler, agilizando el proceso para ambas partes.

En cuanto a la energía consumida, actualmente es de 0,3W, un valor elevado si se planteara la alimentación por baterías. Se propone el estudio de uso de técnicas de reducción de energía como el *light sleep*, que permitiría reducir el

consumo, o la desactivación temporal de la radio Wi-Fi, que permite mantener la conexión, hasta que sea necesaria el envío o recepción de mensajes.

# 6. Bibliografía

[1] Página web de la Agencia Estatal, Boletín Oficial del Estado [en línea] [consulta: 13 de noviembre de 2024]. Disponible en: https://www.boe.es/eli/es/c/1978/12/27/(1)/con

[2] Página web de la Agencia Estatal, Boletín Oficial del Estado [en línea] [consulta: 13 de noviembre de 2024]. Disponible en: https://www.boe.es/eli/es-ar/l/2010/11/18/7/con

[3] Página web del Ayuntamiento de Zaragoza [en línea] [consulta: 13 de noviembre de 2024]. Disponible en: https://www.zaragoza.es/sede/servicio/normativa/eli/es-ar-01502973/odnz/2001/12/05/(1)

[4] Página web de Minut [en línea] [consulta: 13 de noviembre de 2024]. Disponible en: https://store.minut.com/

[5] Página web de Raixer [en línea] [consulta: 13 de noviembre de 2024]. Disponible en: https://www.raixer.com/product

[6] GUBBI, Jayavardhana, et al. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, 2013, vol. 29, no 7, p. 1645-1660.

[7] Maier, A., Sharp, A., & Vagapov, Y. (2017, September). Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things. In *2017 Internet Technologies and Applications (ITA)* (pp. 143-148). IEEE.

[8] Soni, D., & Makwana, A. (2017, April). A survey on mqtt: a protocol of internet of things (iot). In *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)* (Vol. 20, pp. 173-177).

[9] Atmoko, R. A., Riantini, R., & Hasin, M. K. (2017, May). IoT real time data acquisition using MQTT protocol. In *Journal of Physics: Conference Series* (Vol. 853, No. 1, p. 012003). IOP Publishing.

[10] Página web de la Agencia Estatal, Boletín Oficial del Estado [en línea] [consulta: 13 de noviembre de 2024]. Disponible en: https://www.boe.es/doue/2016/119/L00001-00088.pdf

[11] Página web de la Agencia Estatal, Boletín Oficial del Estado [en línea] [consulta: 13 de noviembre de 2024]. Disponible en: https://www.boe.es/eli/es/l/1994/11/24/29/con

[12] Página web de la Agencia Estatal, Boletín Oficial del Estado [en línea] [consulta: 13 de noviembre de 2024]. Disponible en: https://www.boe.es/eli/es/l/1960/07/21/49/con

[13] Página web de Svantek [en línea] [consulta: 27 de noviembre de 2024]. Disponible en: https://svantek.com/es/

# ANEXO A: *Datasheet* MAX9814

EVALUATION KIT AVAILABLE

# MAX9814

# Microphone Amplifier with AGC and Low-Noise Microphone Bias

## General Description

The MAX9814 is a low-cost, high-quality microphone amplifier with automatic gain control (AGC) and low-noise microphone bias. The device features a low-noise preamplifier, variable gain amplifier (VGA), output amplifier, microphone-bias-voltage generator and AGC control circuitry.

The low-noise preamplifier has a fixed 12dB gain, while the VGA gain automatically adjusts from 20dB to 0dB, depending on the output voltage and the AGC threshold. The output amplifier offers selectable gains of 8dB, 18dB, and 28dB. With no compression, the cascade of the amplifiers results in an overall gain of 40dB, 50dB, or 60dB. A trilevel digital input programs the output amplifier gain. An external resistive divider controls the AGC threshold and a single capacitor programs the attack/release times. A trilevel digital input programs the ratio of attack-to-release time. The hold time of the AGC is fixed at 30ms. The low-noise microphone-bias-voltage generator can bias most electret microphones.

The MAX9814 is available in the space-saving, 14-pin TDFN package. This device is specified over the -40°C to +85°C extended temperature range.

## Applications

Digital Still Cameras

Digital Video Cameras

PDAs

Bluetooth Headsets

Entertainment Systems (e.g., Karaoke)

Two-Way Communicators

High-Quality Portable Recorders

IP Phones/Telephone Conferencing

## Features

- **Automatic Gain Control (AGC)**
- **Three Gain Settings (40dB, 50dB, 60dB)**
- **Programmable Attack Time**
- **Programmable Attack and Release Ratio**
- **2.7V to 5.5V Supply Voltage Range**
- **Low Input-Referred Noise Density of 30nV/√Hz**
- **Low THD: 0.04% (typ)**
- **Low-Power Shutdown Mode**
- **Internal Low-Noise Microphone Bias, 2V**
- **Available in the Space-Saving, 14-Pin TDFN (3mm x 3mm) Package**
- **-40°C to +85°C Extended Temperature Range**

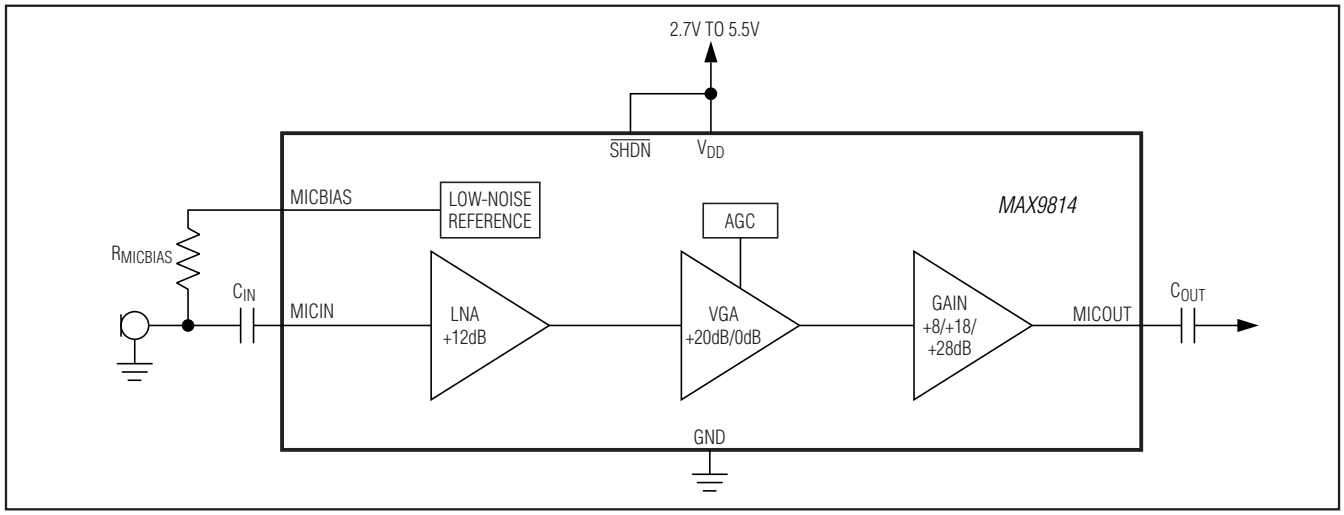
## Ordering Information

| PART | TEMP RANGE | PIN-PACKAGE |
|---|---|---|
| MAX9814ETD+T | -40°C to +85°C | 14 TDFN-EP* |

*+Denotes a lead(Pb)-free/RoHS-compliant package.*
*T = Tape and reel.*
**EP = Exposed pad.*

***Pin Configurations appear at end of data sheet.***

## Simplified Block Diagram




**For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim's website at www.maximintegrated.com.**

19-0764; Rev 2; 6/09

## ABSOLUTE MAXIMUM RATINGS

$V_{DD}$ to GND .....-0.3V to +6V
All Other Pins to GND.....-0.3V to ($V_{DD}$ + 0.3V)
Output Short-Circuit Duration.....Continuous
Continuous Current (MICOUT, MICBIAS).....±100mA
All Other Pins .....±20mA

Continuous Power Dissipation ($T_A$ = +70°C)
14-Pin TDFN-EP
(derate 16.7mW/°C above +70°C).....1481.5mW
Operating Temperature Range .....-40°C to +85°C
Junction Temperature.....+150°C
Lead Temperature (soldering, 10s) .....+300°C
Bump Temperature (soldering) Reflow.....+235°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

## ELECTRICAL CHARACTERISTICS

($V_{DD}$ = 3.3V, $\overline{SHDN}$ = $V_{DD}$, $C_{CT}$ = 470nF, $C_{CG}$ = 2µF, GAIN = $V_{DD}$, $T_A$ = $T_{MIN}$ to $T_{MAX}$, unless otherwise specified. Typical values are at $T_A$ = +25°C.) (Note 1)

| PARAMETER | SYMBOL | CONDITIONS | MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|---|
| **GENERAL** | | | | | | |
| Operating Voltage | $V_{DD}$ | Guaranteed by PSRR test | 2.7 | | 5.5 | V |
| Supply Current | $I_{DD}$ | | | 3.1 | 6 | mA |
| Shutdown Supply Current | $I_{\overline{SHDN}}$ | | | 0.01 | 1 | µA |
| Input-Referred Noise Density | $e_n$ | BW = 20kHz, all gain settings | | 30 | | $nV/\sqrt{Hz}$ |
| Output Noise | | BW = 20kHz | | 430 | | $µV_{RMS}$ |
| Signal-to-Noise Ratio | SNR | BW = 22Hz to 22kHz ($500mV_{RMS}$ output signal) | | 61 | | dB |
| | | A-weighted | | 64 | | |
| Dynamic Range | DR | (Note 2) | | 60 | | dB |
| Total Harmonic Distortion Plus Noise | THD+N | $f_{IN}$ = 1kHz, BW = 20Hz to 20kHz, $R_L$ = 10kΩ, $V_{TH}$ = 1V (threshold = $2V_{P-P}$), $V_{IN}$ = $0.5mV_{RMS}$, $V_{CT}$ = 0V | | 0.04 | | % |
| | | $f_{IN}$ = 1kHz, BW = 20Hz to 20kHz, $R_L$ = 10kΩ, $V_{TH}$ = 0.1V (threshold = $200mV_{P-P}$), $V_{IN}$ = $30mV_{RMS}$, $V_{CT}$ = 2V | | 0.2 | | |
| Amplifier Input BIAS | $V_{IN}$ | | 1.14 | 1.23 | 1.32 | V |
| Maximum Input Voltage | $V_{IN\_MAX}$ | 1% THD | | 100 | | $mV_{P-P}$ |
| Input Impedance | $Z_{IN}$ | | | 100 | | kΩ |
| Maximum Gain | A | GAIN = $V_{DD}$ | 39.5 | 40 | 40.5 | dB |
| | | GAIN = GND | 49.5 | 50 | 50.6 | |
| | | GAIN = unconnected | 59.5 | 60 | 60.5 | |
| Minimum Gain | | GAIN = $V_{DD}$ | 18.7 | 20 | 20.5 | dB |
| | | GAIN = GND | 29.0 | 30 | 30.8 | |
| | | GAIN = unconnected | 38.7 | 40 | 40.5 | |
| Maximum Output Level | $V_{OUT\_RMS}$ | 1% THD+N, $V_{TH}$ = MICBIAS | | 0.707 | | $V_{RMS}$ |
| Regulated Output Level | | AGC enabled, $V_{TH}$ = 0.7V | 1.26 | 1.40 | 1.54 | $V_{P-P}$ |
| AGC Attack Time | $t_{ATTACK}$ | $C_{CT}$ = 470nF (Note 3) | | 1.1 | | ms |
| Attack/Release Ratio | A/R | A/R = GND | | 1:500 | | ms/ms |
| | | A/R = $V_{DD}$ | | 1:2000 | | |
| | | A/R = unconnected | | 1:4000 | | |

## ELECTRICAL CHARACTERISTICS (continued)

($V_{DD}$ = 3.3V, $\overline{SHDN}$ = $V_{DD}$, $C_{CT}$ = 470nF, $C_{CG}$ = 2µF, GAIN = $V_{DD}$, $T_A$ = $T_{MIN}$ to $T_{MAX}$, unless otherwise specified. Typical values are at $T_A$ = +25°C.) (Note 1)

| PARAMETER | SYMBOL | CONDITIONS | MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|---|
| MICOUT High Output Voltage | $V_{OH}$ | $I_{OUT}$ sourcing 1mA | | 2.45 | | V |
| MICOUT Low Output Voltage | $V_{OL}$ | $I_{OUT}$ sinking 1mA | | 3 | | mV |
| MICOUT Bias | | MICOUT unconnected | 1.14 | 1.23 | 1.32 | V |
| Output Impedance | $Z_{OUT}$ | | | 50 | | Ω |
| Minimum Resistive Load | $R_{LOAD\_MIN}$ | | | 5 | | kΩ |
| Maximum Capacitive Drive | $C_{LOAD\_MAX}$ | | | 200 | | pF |
| Maximum Output Current | $I_{OUT\_MAX}$ | 1% THD, $R_L$ = 500Ω | | 1 | 2 | mA |
| Output Short-Circuit Current | $I_{SC}$ | | 3 | 8 | | mA |
| Power-Supply Rejection Ratio | PSRR | AGC mode; $V_{DD}$ = 2.7V to 5.5V (Note 4) | 35 | 50 | | dB |
| | | f = 217Hz, $V_{RIPPLE}$ = 100m$V_{P-P}$ (Note 5) | | 55 | | |
| | | f = 1kHz, $V_{RIPPLE}$ = 100m$V_{P-P}$ (Note 5) | | 52.5 | | |
| | | f = 10kHz, $V_{RIPPLE}$ = 100m$V_{P-P}$ (Note 5) | | 43 | | |
| **MICROPHONE BIAS** | | | | | | |
| Microphone Bias Voltage | $V_{MICBIAS}$ | $I_{MICBIAS}$ = 0.5mA | 1.84 | 2.0 | 2.18 | V |
| Output Resistance | $R_{MICBIAS}$ | $I_{MICBIAS}$ = 1mA | | 1 | | Ω |
| Output Noise Voltage | $V_{MICBIAS\_NOISE}$ | $I_{MICBIAS}$ = 0.5mA, BW = 22Hz to 22kHz | | 5.5 | | $µV_{RMS}$ |
| Power-Supply Rejection Ratio | PSRR | DC, $V_{DD}$ = 2.7V to 5.5V | 70 | 80 | | dB |
| | | $I_{MICBIAS}$ = 0.5mA, $V_{RIPPLE}$ = 100m$V_{P-P}$, $f_{IN}$ = 1kHz | | 71 | | |
| **TRILEVEL INPUTS (A/R, GAIN)** | | | | | | |
| Tri-Level Input Leakage Current | | A/R or GAIN = $V_{DD}$ | $0.5V_{DD}$ / 180kΩ | $0.5V_{DD}$ / 100kΩ | $0.5V_{DD}$ / 50kΩ | mA |
| | | A/R or GAIN = GND | $0.5V_{DD}$ / 180kΩ | $0.5V_{DD}$ / 100kΩ | $0.5V_{DD}$ / 50kΩ | |
| Input High Voltage | $V_{IH}$ | | $V_{DD}$ x 0.7 | | | V |
| Input Low Voltage | $V_{IL}$ | | | | $V_{DD}$ x 0.3 | V |
| Shutdown Enable Time | $t_{ON}$ | | | 60 | | ms |
| Shutdown Disable Time | $t_{OFF}$ | | | 40 | | ms |
| **DIGITAL INPUT ($\overline{SHDN}$)** | | | | | | |
| $\overline{SHDN}$ Input Leakage Current | | | -1 | | +1 | µA |
| Input High Voltage | $V_{IH}$ | | 1.3 | | | V |
| Input Low Voltage | $V_{IL}$ | | | | 0.5 | V |
| **AGC THRESHOLD INPUT (TH)** | | | | | | |
| TH Input Leakage Current | | | -1 | | +1 | µA |

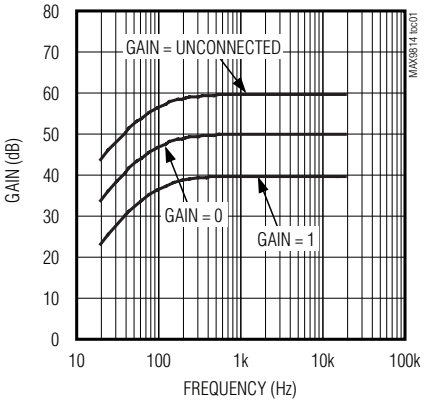
**Note 1:** Devices are production tested at $T_A$ = +25°C. Limits over temperature are guaranteed by design.
**Note 2:** Dynamic range is calculated using the EIAJ method. The input is applied at -60dBFS (0.707$µV_{RMS}$), $f_{IN}$ = 1kHz.
**Note 3:** Attack time measured as time from AGC trigger to gain reaching 90% of its final value.
**Note 4:** CG is connected to an external DC voltage source, and adjusted until $V_{MICOUT}$ = 1.23V.
**Note 5:** CG connected to GND with 2.2µF.

# Microphone Amplifier with AGC and Low-Noise Microphone Bias

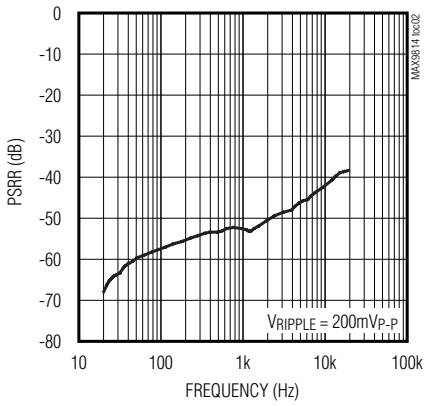
## Typical Operating Characteristics

($V_{DD}$ = 5V, $C_{CT}$ = 470nF, $C_{CG}$ = 2.2µF, $V_{TH}$ = $V_{MICBIAS}$ x 0.4, GAIN = $V_{DD}$ (40dB), AGC disabled, no load, $R_L$ = 10kΩ, $C_{OUT}$ = 1µF, $T_A$ = +25°C, unless otherwise noted.)

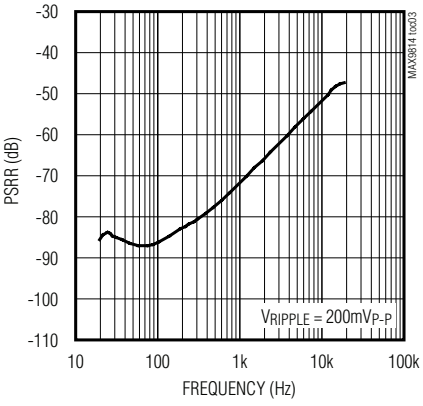
**GAIN vs. FREQUENCY**



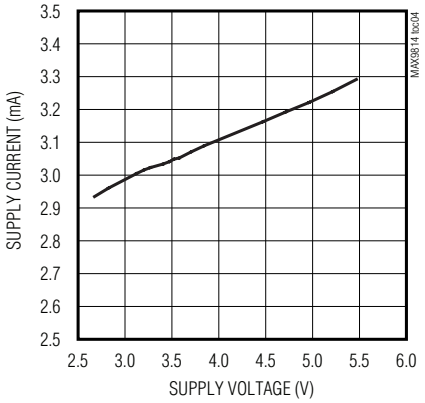

**POWER-SUPPLY REJECTION RATIO vs. FREQUENCY**



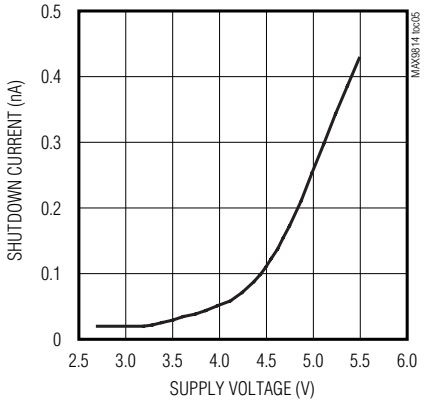

**MICBIAS POWER-SUPPLY REJECTION RATIO vs. FREQUENCY**



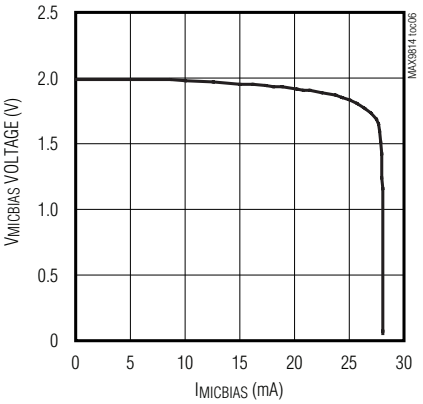

**SUPPLY CURRENT vs. SUPPLY VOLTAGE**



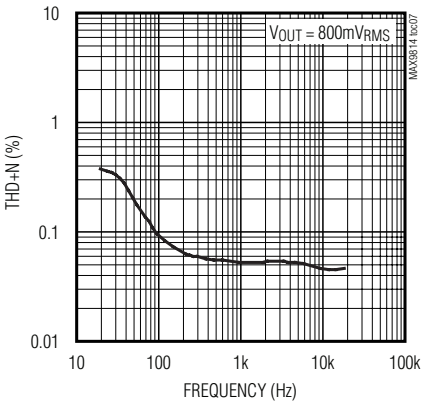

**SHUTDOWN CURRENT vs. SUPPLY VOLTAGE**



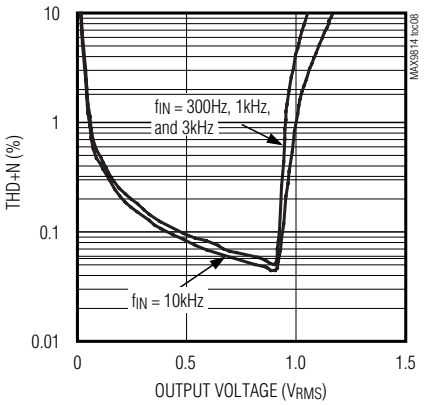

**MICROPHONE BIAS VOLTAGE vs. MICROPHONE BIAS SOURCE CURRENT**



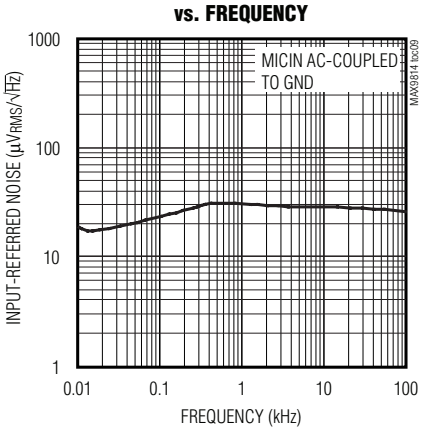

**TOTAL HARMONIC DISTORTION PLUS NOISE vs. FREQUENCY**




**TOTAL HARMONIC DISTORTION PLUS NOISE vs. OUTPUT VOLTAGE**



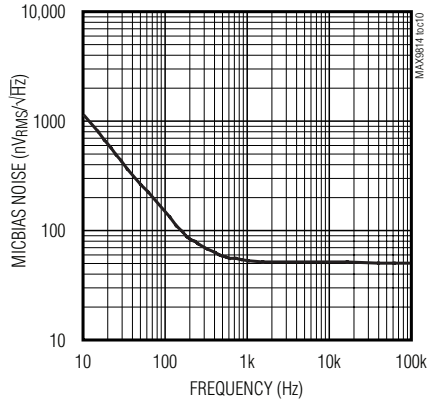

**INPUT-REFERRED NOISE vs. FREQUENCY**



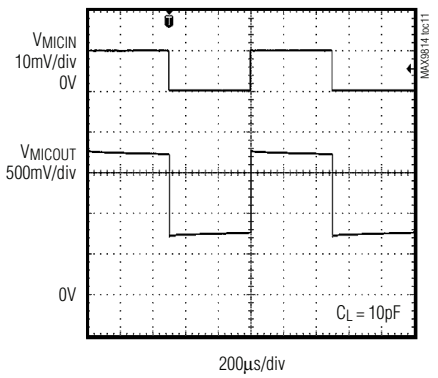
## Typical Operating Characteristics (continued)

($V_{DD}$ = 5V, $C_{CT}$ = 470nF, $C_{CG}$ = 2.2µF, $V_{TH}$ = $V_{MICBIAS}$ x 0.4, GAIN = $V_{DD}$ (40dB), AGC disabled, no load, $R_L$ = 10kΩ, $C_{OUT}$ = 1µF, $T_A$ = +25°C, unless otherwise noted.)

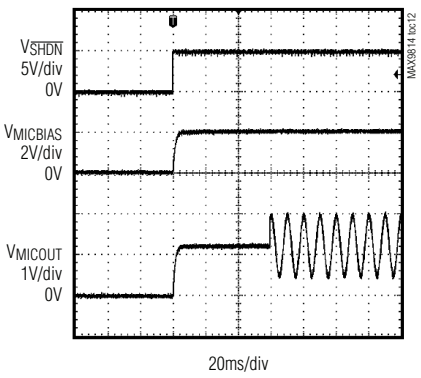
**MICBIAS NOISE vs. FREQUENCY**



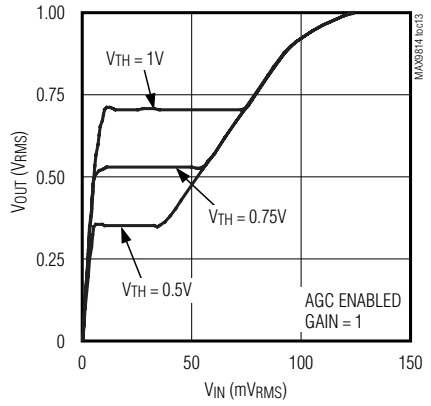

**SMALL-SIGNAL PULSE RESPONSE**



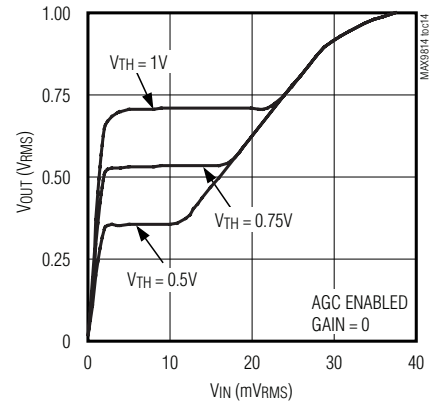

**TURN-ON RESPONSE**



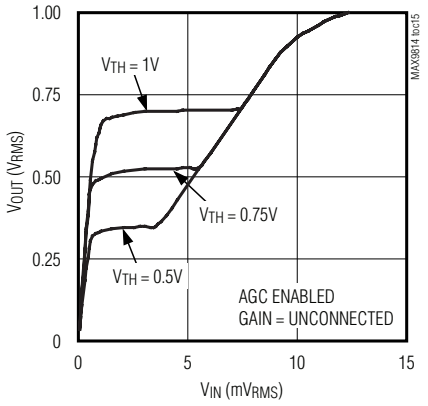

**VOUT vs. VIN**



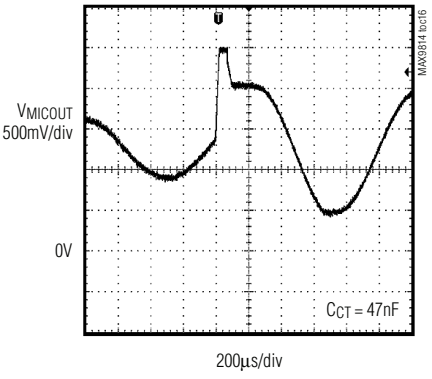

**VOUT vs. VIN**



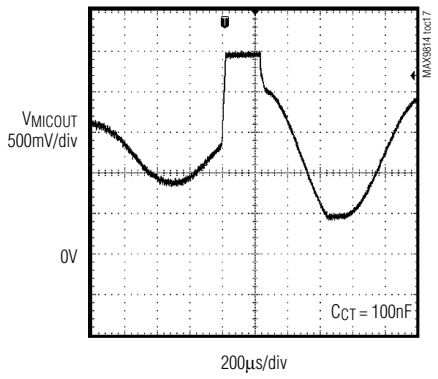

**VOUT vs. VIN**



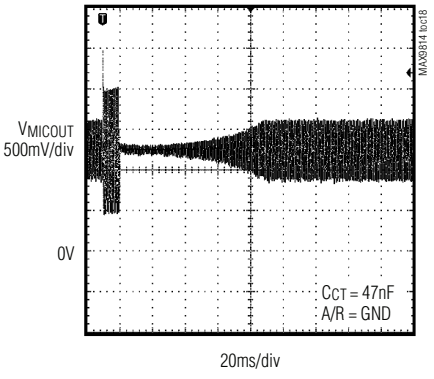

**ATTACK TIME**




**ATTACK TIME**




**HOLD AND RELEASE TIME**

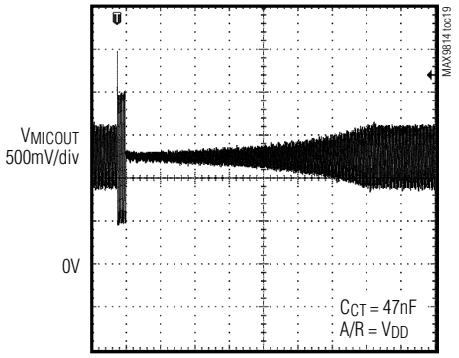


# Microphone Amplifier with AGC and Low-Noise Microphone Bias

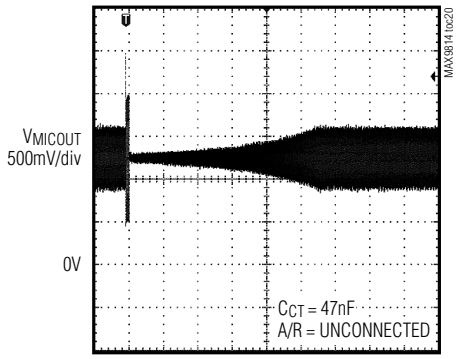
## Typical Operating Characteristics (continued)

($V_{DD}$ = 5V, $C_{CT}$ = 470nF, $C_{CG}$ = 2.2µF, $V_{TH}$ = $V_{MICBIAS}$ x 0.4, GAIN = $V_{DD}$ (40dB), AGC disabled, no load, $R_L$ = 10kΩ, $C_{OUT}$ = 1µF, $T_A$ = +25°C, unless otherwise noted.)

**HOLD AND RELEASE TIME**




**HOLD AND RELEASE TIME**




## Pin Description

| PIN | NAME | FUNCTION |
|---|---|---|
| TDFN | | |
| 1 | CT | Timing Capacitor Connection. Connect a capacitor to CT to control the Attack and Release times of the AGC. |
| 2 | $\overline{SHDN}$ | Active-Low Shutdown Control |
| 3 | CG | Amplifier DC Offset Adjust. Connect a 2.2µF capacitor to GND to ensure zero offset at the output. |
| 4, 11 | N.C. | No Connection. Connect to GND. |
| 5 | $V_{DD}$ | Power Supply. Bypass to GND with a 1µF capacitor. |
| 6 | MICOUT | Amplifier Output |
| 7 | GND | Ground |
| 8 | MICIN | Microphone Noninverting Input |
| 9 | A/R | Tri-Level Attack and Release Ratio Select. Controls the ratio of attack time to release time for the AGC circuit.<br>A/R = GND: Attack/Release Ratio is 1:500<br>A/R = $V_{DD}$: Attack/Release Ratio is 1:2000<br>A/R = Unconnected: Attack/Release Ratio is 1:4000 |
| 10 | GAIN | Tri-Level Amplifier Gain Control.<br>GAIN = $V_{DD}$, gain set to 40dB.<br>GAIN = GND, gain set to 50dB.<br>GAIN = Unconnected, uncompressed gain set to 60dB. |
| 12 | BIAS | Amplifier Bias. Bypass to GND with a 0.47µF capacitor. |
| 13 | MICBIAS | Microphone Bias Output |
| 14 | TH | AGC Threshold Control. TH voltage sets gain control threshold. Connect TH to MICBIAS to disable the AGC. |
| — | EP | Exposed Pad. Connect the TDFN EP to GND. |

## Detailed Description

The MAX9814 is a low-cost, high-quality microphone amplifier with automatic gain control (AGC) and a low-noise microphone bias. The MAX9814 consists of several distinct circuits: a low-noise preamplifier, a variable gain amplifier (VGA), an output amplifier, a microphone-bias-voltage generator, and AGC control circuitry.

An internal microphone bias voltage generator provides a 2V bias that is suitable for most electret condenser microphones. The MAX9814 amplifies the input in three distinct stages. In the first stage, the input is buffered and amplified through the low-noise preamplifier with a gain of 12dB. The second stage consists of the VGA controlled by the AGC. The VGA/AGC combination is capable of varying the gain from 20dB to 0dB. The output amplifier is the final stage in which a fixed gain of 8dB, 18dB, 20dB is programmed through a single tri-level logic input. With no compression from the AGC, the MAX9814 is capable of providing 40dB, 50dB, or 60dB gain.

### Automatic Gain Control (AGC)

A device without AGC experiences clipping at the output when too much gain is applied to the input. AGC prevents clipping at the output when too much gain is applied to the input, eliminating output clipping. Figure 1 shows a comparison of an over-gained microphone input with and without AGC.

The MAX9814's AGC controls the gain by first detecting that the output voltage has exceeded a preset limit. The microphone amplifier gain is then reduced with a selectable time constant to correct for the excessive output-voltage amplitude. This process is known as the attack time. When the output signal subsequently lowers in amplitude, the gain is held at the reduced state for a short period before slowly increasing to the normal value. This process is known as the hold and release time. The speed at which the amplifiers adjust to changing input signals is set by the external timing capacitor $C_{CT}$ and the voltage applied to A/R. The AGC threshold can be set by adjusting $V_{TH}$. Gain reduction is a function of input signal amplitude with a maximum AGC attenuation of 20dB. Figure 2 shows the effect of an input burst exceeding the preset limit, output attack, hold and release times.

If the attack and release times are configured to respond too fast, audible artifacts often described as "pumping" or "breathing" can occur as the gain is rapidly adjusted to follow the dynamics of the signal. For best results, adjust the time constant of the AGC to accommodate the source material. For applications in which music CDs are the main audio source, a 160µs attack time with an 80ms release time is recommended. Music applications typically require a shorter release time than voice or movie content.

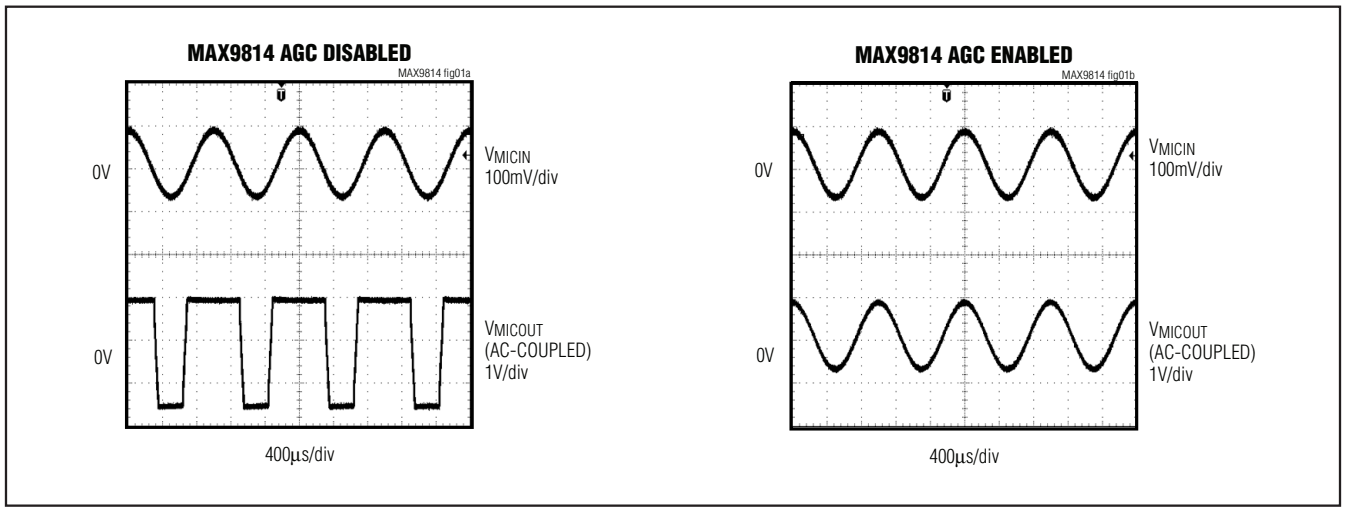



*Figure 1. Microphone Input with and Without AGC*

# Microphone Amplifier with AGC and Low-Noise Microphone Bias

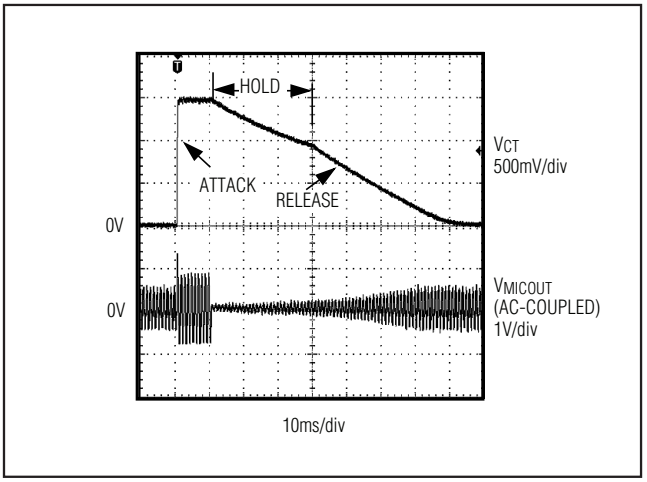



*Figure 2. Input Burst Exceeding AGC Limit*

### Attack Time

The attack time is the time it takes for the AGC to reduce the gain after the input signal has exceeded the threshold level. The gain attenuation during the attack time is exponential, and defined as one-time constant. The time constant of the attack is given by 2400 x $C_{CT}$ seconds (where $C_{CT}$ is the external timing capacitor):

- Use a short attack time for the AGC to react quickly to transient signals, such as snare drum beats (music) or gun shots (DVD).
- Use a longer attack time to allow the AGC to ignore short-duration peaks and only reduce the gain when a noticeable increase in loudness occurs. Short-duration peaks are not reduced, but louder passages are. This allows the louder passages to be reduced in volume, thereby maximizing output dynamic range.

### Hold Time

Hold time is the delay after the signal falls below the threshold level before the release phase is initiated. Hold time is internally set to 30ms and nonadjustable. The hold time is cancelled by any signal exceeding the set threshold level, and the attack time is reinitiated.

### Release Time

The release time is how long it takes for the gain to return to its normal level after the output signal has fallen below the threshold level and 30ms hold time has expired. Release time is defined as release from a 20dB gain compression to 10% of the nominal gain setting after the input signal has fallen below the TH threshold and the 30ms hold time has expired. Release time is adjustable and has a minimum of 25ms. The release time is set by picking an attack time using $C_{CT}$ and setting the attack-to-release time ratio by configuring A/R as shown in Table 1:

- Use a small ratio to maximize the speed of the AGC.
- Use a large ratio to maximize the sound quality and prevent repeated excursions above the threshold from being independently adjusted by the AGC.

### AGC Output Threshold

The output threshold that activates AGC is adjustable through the use of an external resistive divider. Once the divider is set, AGC reduces the gain to match the output voltage to the voltage set at the TH input.

### Microphone Bias

The MAX9814 features an internal low-noise microphone bias voltage capable of driving most electret condenser microphones. The microphone bias is regulated at 2V to provide that the input signal to the low-noise preamplifier does not clip to ground.

## Applications Information

### Programming Attack and Release Times

Attack and release times are set by selecting the capacitance value between CT and GND, and by setting the logic state of A/R (Table 1). A/R is a tri-level logic input that sets the attack-to-release time ratio.

**Table 1. Attack-and-Release Ratios**

| A/R | ATTACK/RELEASE RATIO |
|---|---|
| GND | 1:500 |
| $V_{DD}$ | 1:2000 |
| Unconnected | 1:4000 |

The attack and release times can be selected by utilizing the corresponding capacitances listed in Table 2.

**Table 2. Attack-and-Release Time**

| $C_{CT}$ | $t_{ATTACK}$ (ms) | $t_{RELEASE}$ (ms) | | |
|---|---|---|---|---|
| | | A/R = GND | A/R = $V_{DD}$ | A/R = UNCONNECTED |
| 22nF | 0.05 | 25 | 100 | 200 |
| 47nF | 0.11 | 55 | 220 | 440 |
| 68nF | 0.16 | 80 | 320 | 640 |
| 100nF | 0.24 | 120 | 480 | 960 |
| 220nF | 0.53 | 265 | 1060 | 2120 |
| 470nF | 1.1 | 550 | 2200 | 4400 |
| 680nF | 1.63 | 815 | 3260 | 6520 |
| 1µF | 2.4 | 1200 | 4800 | 9600 |

### Setting the AGC Threshold

To set the output-voltage threshold at which the microphone output is clamped, an external resistor-divider must be connected from MICBIAS to ground with the output of the resistor-divider applied to TH. The voltage $V_{TH}$ determines the peak output-voltage threshold at which the output becomes clamped. The maximum signal swing at the output is then limited to two times $V_{TH}$ and remains at that level until the amplitude of the input signal is reduced. To disable AGC, connect TH to MICBIAS.

### Microphone Bias Resistor

MICBIAS is capable of sourcing 20mA. Select a value for $R_{MICBIAS}$ that provides the desired bias current for the electret microphone. A value of 2.2kΩ is usually sufficient for a microphone of typical sensitivity. Consult the microphone data sheet for the recommended bias resistor.

### Bias Capacitor

The BIAS output of the MAX9814 is internally buffered and provides a low-noise bias. Bypass BIAS with a 470nF capacitor to ground.

### Input Capacitor

The input AC-coupling capacitor ($C_{IN}$) and the input resistance ($R_{IN}$) to the microphone amplifier form a highpass filter that removes any DC bias from an input signal (see the *Typical Application Circuit/Functional Diagram*). $C_{IN}$ prevents any DC components from the input-signal source from appearing at the amplifier outputs. The -3dB point of the highpass filter, assuming zero source impedance due to the input signal source, is given by:

$$f_{-3dB\_IN} = \frac{1}{2\pi \times R_{IN} \times C_{IN}}$$

Choose $C_{IN}$ such that $f_{-3dB\_IN}$ is well below the lowest frequency of interest. Setting $f_{-3dB\_IN}$ too high affects the amplifier's low-frequency response. Use capacitors with low-voltage coefficient dielectrics. Aluminum electrolytic, tantalum, or film dielectric capacitors are good choices for AC-coupling capacitors. Capacitors with high-voltage coefficients, such as ceramics (non-C0G dielectrics), can result in increased distortion at low frequencies.

### Output Capacitor

The output of the MAX9814 is biased at 1.23V. To eliminate the DC offset, an AC-coupling capacitor ($C_{OUT}$) must be used. Depending on the input resistance ($R_L$) of the following stage, $C_{OUT}$ and $R_L$ effectively form a highpass filter. The -3dB point of the highpass filter, assuming zero output impedance, is given by:

$$f_{-3dB\_OUT} = \frac{1}{2\pi \times R_L \times C_{OUT}}$$

### Shutdown

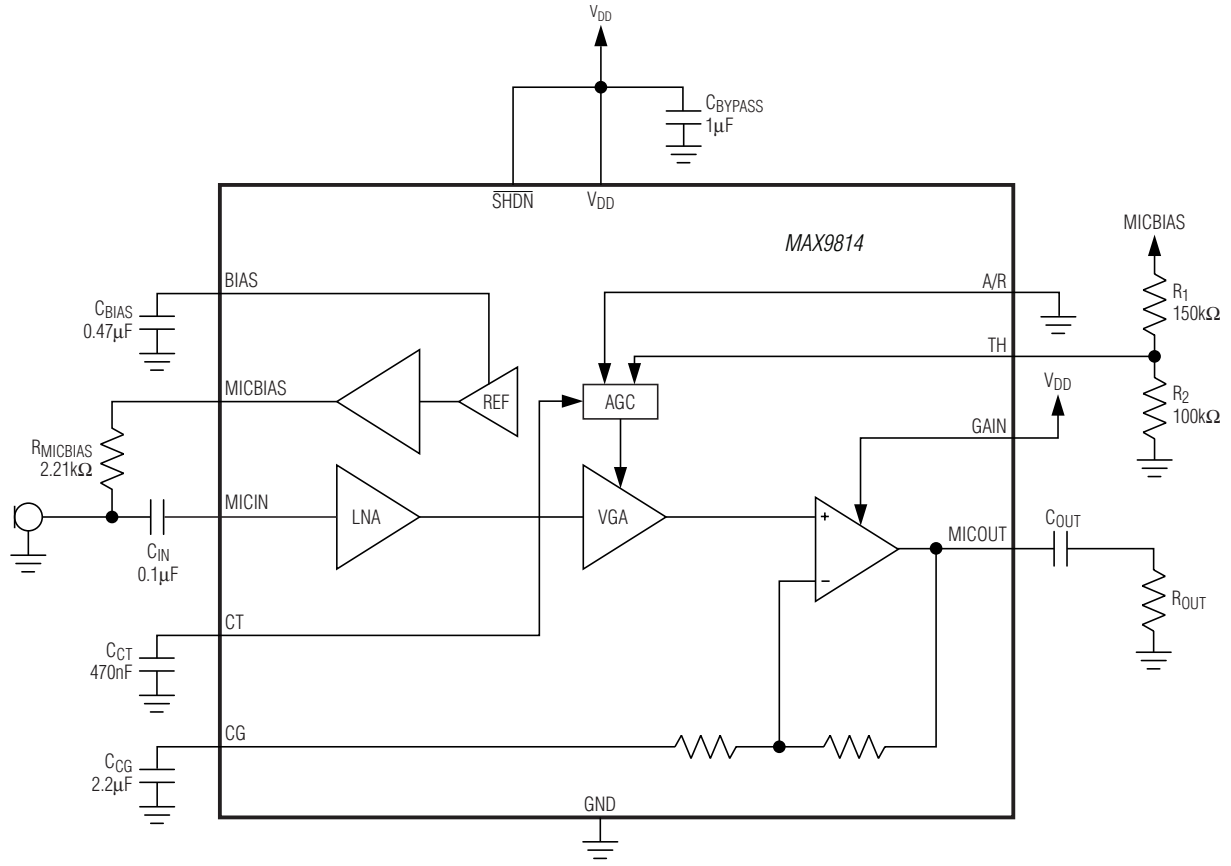
The MAX9814 features a low-power shutdown mode. When $\overline{SHDN}$ goes low, the supply current drops to 0.01µA, the output enters a high-impedance state, and the bias current to the microphone is switched off. Driving $\overline{SHDN}$ high enables the amplifier. Do not leave $\overline{SHDN}$ unconnected.

### Power-Supply Bypassing and PCB Layout

Bypass the power supply with a 0.1µF capacitor to ground. Reduce stray capacitance by minimizing trace lengths and place external components as close to the device as possible. Surface-mount components are recommended. In systems where analog and digital grounds are available, connect the MAX9814 to analog ground.

# Microphone Amplifier with AGC and Low-Noise Microphone Bias

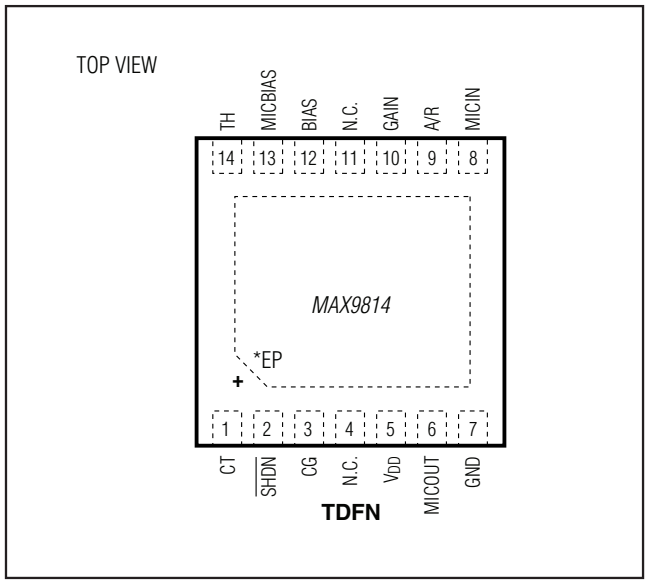
## Typical Application Circuit/Functional Diagram




*THE DEVICE HAS BEEN CONFIGURED WITH AN ATTACK TIME OF 1.1ms, 40dB GAIN, AND AN ATTACK-AND-RELEASE RATIO OF 1:500.

# Microphone Amplifier with AGC and Low-Noise Microphone Bias

## Pin Configuration




## Chip Information

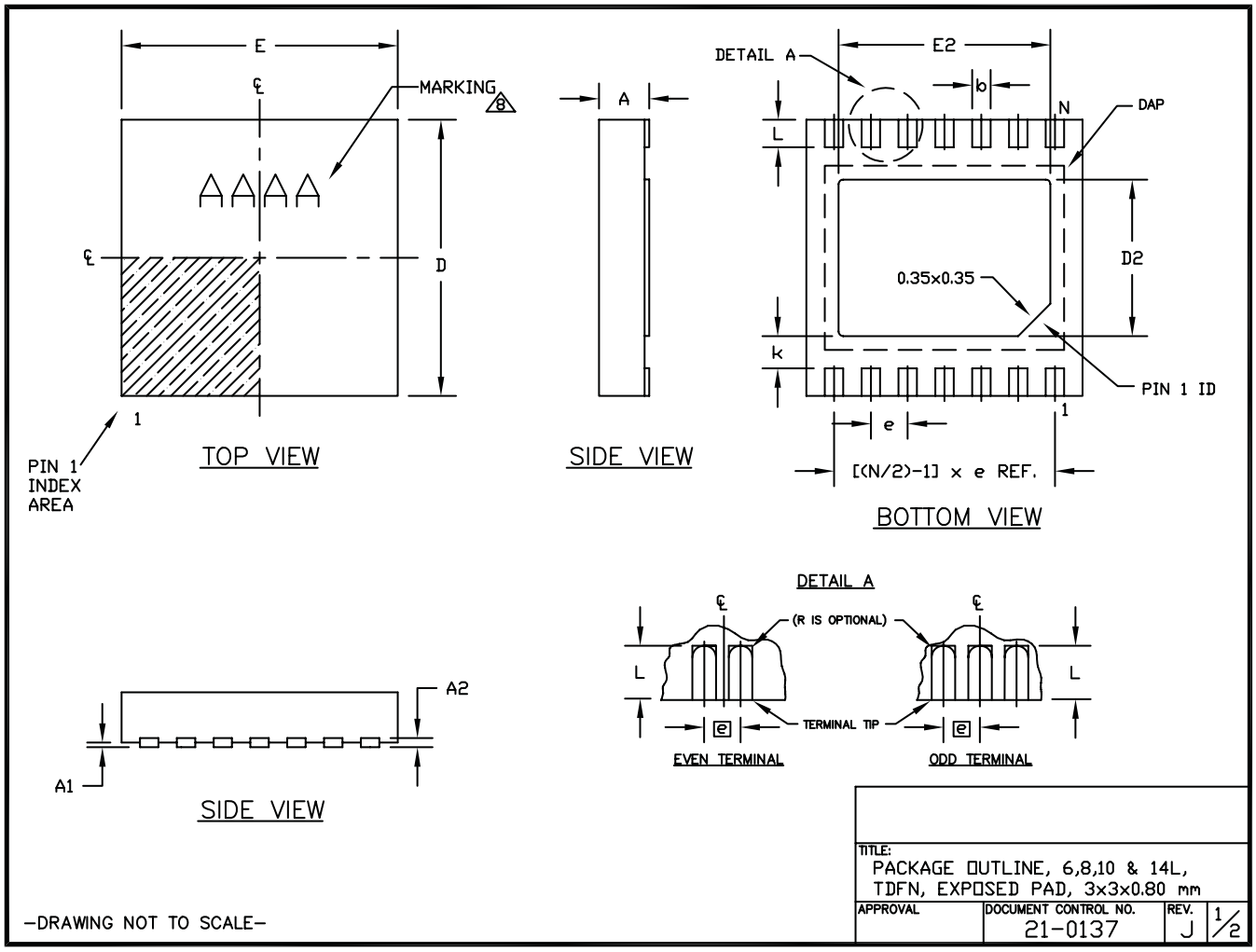
PROCESS: BiCMOS

# Microphone Amplifier with AGC and Low-Noise Microphone Bias

## Package Information

For the latest package outline information and land patterns, go to www.maxim-ic.com/packages.

| PACKAGE TYPE | PACKAGE CODE | DOCUMENT NO. |
|---|---|---|
| 14 TDFN-EP | T1433-2 | 21-0137 |



## Package Information (continued)

For the latest package outline information and land patterns, go to **www.maxim-ic.com/packages**.

| COMMON DIMENSIONS | | |
|---|---|---|
| SYMBOL | MIN. | MAX. |
| A | 0.70 | 0.80 |
| D | 2.90 | 3.10 |
| E | 2.90 | 3.10 |
| A1 | 0.00 | 0.05 |
| L | 0.20 | 0.40 |
| k | 0.25 MIN. | |
| A2 | 0.20 REF. | |

| PACKAGE VARIATIONS | | | | | | | |
|---|---|---|---|---|---|---|---|
| PKG. CODE | N | D2 | E2 | e | JEDEC SPEC | b | [(N/2)-1] x e |
| T633-2 | 6 | 1.50±0.10 | 2.30±0.10 | 0.95 BSC | MO229 / WEEA | 0.40±0.05 | 1.90 REF |
| T833-2 | 8 | 1.50±0.10 | 2.30±0.10 | 0.65 BSC | MO229 / WEEC | 0.30±0.05 | 1.95 REF |
| T833-3 | 8 | 1.50±0.10 | 2.30±0.10 | 0.65 BSC | MO229 / WEEC | 0.30±0.05 | 1.95 REF |
| T1033-1 | 10 | 1.50±0.10 | 2.30±0.10 | 0.50 BSC | MO229 / WEED-3 | 0.25±0.05 | 2.00 REF |
| T1033MK-1 | 10 | 1.50±0.10 | 2.30±0.10 | 0.50 BSC | MO229 / WEED-3 | 0.25±0.05 | 2.00 REF |
| T1033-2 | 10 | 1.50±0.10 | 2.30±0.10 | 0.50 BSC | MO229 / WEED-3 | 0.25±0.05 | 2.00 REF |
| T1433-1 | 14 | 1.70±0.10 | 2.30±0.10 | 0.40 BSC | - - - - | 0.20±0.05 | 2.40 REF |
| T1433-2 | 14 | 1.70±0.10 | 2.30±0.10 | 0.40 BSC | - - - - | 0.20±0.05 | 2.40 REF |
| T1433-3F | 14 | 1.70±0.10 | 2.30±0.10 | 0.40 BSC | - - - - | 0.20±0.05 | 2.40 REF |

NOTES:
1. ALL DIMENSIONS ARE IN mm. ANGLES IN DEGREES.
2. COPLANARITY SHALL NOT EXCEED 0.08 mm.
3. WARPAGE SHALL NOT EXCEED 0.10 mm.
4. PACKAGE LENGTH/PACKAGE WIDTH ARE CONSIDERED AS SPECIAL CHARACTERISTIC(S).
5. DRAWING CONFORMS TO JEDEC MO229, EXCEPT DIMENSIONS "D2" AND "E2", AND T1433-1 & T1433-2.
6. "N" IS THE TOTAL NUMBER OF LEADS.
7. NUMBER OF LEADS SHOWN ARE FOR REFERENCE ONLY.
8. MARKING IS FOR PACKAGE ORIENTATION REFERENCE ONLY.
9. ALL DIMENSIONS APPLY TO BOTH LEADED (-) AND PbFREE (+) PKG. CODES.

-DRAWING NOT TO SCALE-

| TITLE: PACKAGE OUTLINE, 6,8,10 & 14L, TDFN, EXPOSED PAD, 3x3x0.80 mm | | | |
|---|---|---|---|
| APPROVAL | DOCUMENT CONTROL NO. 21-0137 | REV. J | 2/2 |

# Microphone Amplifier with AGC and Low-Noise Microphone Bias

## Revision History

| REVISION NUMBER | REVISION DATE | DESCRIPTION | PAGES CHANGED |
|---|---|---|---|
| 0 | 3/07 | Initial release | — |
| 1 | 2/09 | Updated *Ordering Information*, *Absolute Maximum Ratings*, *Pin Description*, and *Pin Configuration* sections to include EP for TDFN package | 1, 2, 6, 11 |
| 2 | 6/09 | Removed UCSP package | 1, 2, 6, 11, 12 |

# ANEXO B: Firmware sonómetro

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <Preferences.h>
#include <FS.h>
#include <LittleFS.h>
#include <HTTPClient.h>
#include <Update.h>
#include "driver/i2s.h"
#include <math.h>
#include "utilities.h"
#include <TinyGsmClient.h>
#include <StreamDebugger.h>
#include "esp_task_wdt.h";


//Variables y definiciones tarjeta SIM
#define TINY_GSM_RX_BUFFER          1024 // Buffer a 1kb
#define DUMP_AT_COMMANDS //Muestra comandos AT por puerto serie
const char *SIMCARD_PIN_CODE = "4764";
StreamDebugger debugger(SerialAT, Serial);
TinyGsm modem(debugger);

//Variables y definiciones DMA y micrófono
#define SAMPLE_RATE 16000         // Frecuencia de muestreo en Hz
#define BUFFER_SIZE 1024          // Tamaño del buffer de lectura
#define OFFSET 1324               // Offset a restar a cada muestra

//Variables RMS
static volatile unsigned long ultimoRMS = 0; // Tiempo de la última calculación RMS
static double sumaCuadrados = 0; // Suma de los cuadrados de las muestras
static int contadorMuestras = 0; // Contador de muestras
double ValorRMS = 0;
bool alertaEnviada = false;
float dB = 0;

// WiFi (inicialmente vacías)
char ssid[100];
char password[100];

// Definir los pines de control del relé
const int relePin1 = 22;  // Pin para el primer relé
const int relePin2 = 32;  // Pin para el segundo relé

// MQTT Variables (inicialmente vacías)
char mqtt_broker[100];
char topic[100];
char mqtt_username[100];
char mqtt_password[100];
int mqtt_port = 8883;
String topicfiesta = "Ruido/";
String topicaccion = "Accion/";
String topicActualizacion = "Actualizacion/";
int intentos = 0;
```

```
String clientId = "ESP32Client";
bool pendiente = false;
bool conWifi = false;
bool usandoSIM = false;  // Variable para indicar si se está usando la SIM

const uint8_t mqtt_client_id = 0;
uint32_t check_connect_millis = 0;
unsigned long tiempoAnterior = 0;
const long intervalo = 60000;

const char* server = "https://github.com";  // Dominio de GitHub
const char* path = "/moiruloyeyuro/TFM2/releases/latest/download/helloworld.ino.bin";  // Ruta
completa al archivo


// Definir el cliente TLS (Seguro)
WiFiClientSecure espClient;  // Declara el cliente WiFi seguro
PubSubClient client(espClient);  // Usa el cliente seguro en PubSubClient


// NTP Client
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", 3600, 60000); // "pool.ntp.org" es un servidor NTP
común

Preferences preferences; // Objeto para almacenar preferencias


// Declaración de tareas FreeRTOS
TaskHandle_t taskWiFiHandle;
TaskHandle_t taskMonitorRMSHandle;

// Certificado del servidor MQTT
const char* mqtt_server_cert =
    "-----BEGIN CERTIFICATE-----\n"
    "MIIFBjCCAu6gAwIBAgIRAIp9PhPWLzDvI4a9KQdrNPgwDQYJKoZIhvcNAQELBQAw\n"
    "TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh\n"

"cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMjQwMzEzMDAwMDAw\
n"

"WhcNMjcwMzEyMjM1OTU5WjAzMQswCQYDVQQGEwJVUzEWMBQGA1UEChMNTGV0J3M
g\n"
    "RW5jcnlwdDEMMAoGA1UEAxMDUjExMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB\n"
    "CgKCAQEAuoe8XBsAOcvKCs3UZxD5ATylTqVhyybKUvsVAbe5KPUoHu0nsyQYOWcJ\n"
    "DAjs4DqwO3cOvfPlOVRBDE6uQdaZdN5R2+97/1i9qLcT9t4x1fJyyXJqC4N0lZxG\n"
    "AGQUmfOx2SLZzaiSqhwmej/+71gFewiVgdtxD4774zEJuwm+UE1fj5F2PVqdnoPy\n"
    "6cRms+EGZkNIGIBloDcYmpuEMpexsr3E+BUAnSeI++JjF5ZsmydnS8TbKF5pwnnw\n"
    "SVzgJFDhxLyhBax7QG0AtMJBP6dYuC/FXJuluwme8f7rsIU5/agK70XEeOtlKsLP\n"
    "Xzze41xNG/cLJyuqC0J3U095ah2H2QIDAQABo4H4MIH1MA4GA1UdDwEB/wQEAwIB\n"

"hjAdBgNVHSUEFjAUBggrBgEFBQcDAgYIKwYBBQUHAwEwEgYDVR0TAQH/BAgwBgEB\n"
    "/wIBADAdBgNVHQ4EFgQUxc9GpOr0w8B6bJXELbBeki8m47kwHwYDVR0jBBgwFoAU\n"

"ebRZ5nu25eQBc4AIiMgaWPbpm24wMgYIKwYBBQUHAQEEJjAkMCIGCCsGAQUFBzAC\n"

"hhZodHRwOi8veDEuaS5sZW5jci5vcmcvMBMGA1UdIAQMMAowCAYGZ4EMAQIBMCcG\n"
    "A1UdHwQgMB4wHKAaoBiGFmh0dHA6Ly94MS5jLmxlbmNyLm9yZy8wDQYJKoZIhvcN\n"
    "AQELBQADggIBAE7iiV0KAxyQOND1H/lxXPjDj7I3iHpvsCUf7b632IYGjukJhM1y\n"
    "v4Hz/MrPU0jtvfZpQtSlET41yBOykh0FX+ou1Nj4ScOt9ZmWnO8m2OG0JAtIIE38\n"
```

```
    "01S0qcYhyOE2G/93ZCkXufBL713qzXnQv5C/viOykNpKqUgxdKlEC+Hi9i2DcaR1\n"
    "e9KUwQUZRhy5j/PEdEglKg3l9dtD4tuTm7kZtB8v32oOjzHTYw+7KdzdZiw/sBtn\n"
    "UfhBPORNuay4pJxmY/WrhSMdzFO2q3Gu3MUBcdo27goYKjL9CTF8j/Zz55yctUoV\n"
    "aneCWs/ajUX+HypkBTA+c8LGDLnWO2NKq0YD/pnARkAnYGPfUDoHR9gVSp/qRx+Z\n"
    "WghiDLZsMwhN1zjtSC0uBWiugF3vTNzYIEFfaPG7Ws3jDrAMMYebQ95JQ+HIBD/R\n"
    "PBuHRTBpqKlyDnkSHDHYPiNX3adPoPAcgdF3H2/W0rmoswMWgTlLn1Wu0mrks7/q\n"
    "pdWfS6PJ1jty80r2VKsM/Dj3YIDfbjXKdaFU5C+8bhfJGqU3taKauuz0wHVGT3eo\n"
    "6FlWkWYtbt4pgdamlwVeZEW+LM7qZEJEsMNPrfC03APKmZsJgpWCDWOKZvkZcvjV\n"
    "uYkQ4omYCTX5ohy+knMjdOmdH9c7SpqEWBDC86fiNex+O0XOMEZSa8DA\n"
    "-----END CERTIFICATE-----\n";

// Estructura para almacenar la información de las redes Wi-Fi
struct RedWiFi {
    int numero;
    String ssid;
};

// Array de estructuras para almacenar las redes Wi-Fi encontradas
RedWiFi redesWiFi[200];

// Funciones
void pedirWifiConfiguracion();
void conectarWiFi();
void pedirMQTTConfiguracion();
void callback(char* topic, byte* payload, unsigned int length);
void publicarAviso();
void escanearRedesWiFi();
void descargaArchivo();
void actualizaFirmware(const char* fileName);
void preparaSIM();

void setup() {



    // Configurar los pines como salida
    pinMode(relePin1, OUTPUT);
    pinMode(relePin2, OUTPUT);

    // Inicializar ambos relés en estado apagado
    digitalWrite(relePin1, HIGH);
    digitalWrite(relePin2, HIGH);

    Serial.begin(115200);

    topicfiesta.concat(clientId);
    topicaccion.concat(clientId);
    topicActualizacion.concat(clientId);
    const char *topicoFiesta = topicfiesta.c_str();
    const char *topicoAccion = topicaccion.c_str();
    const char *topicoactualizacion = topicActualizacion.c_str();


    esp_task_wdt_init(10,true);
    // Inicializar las preferencias (RTC)
    preferences.begin("wifi-config", false);

    // Preguntar si se desea restaurar a valores de fábrica
    Serial.println("¿Desea restaurar la configuración a valores de fábrica? (S/N):");
    char respuesta = ' ';
```

```
  unsigned long start_time = millis();
  while (millis() - start_time < 5000) {
    if (Serial.available()) {
      String input = Serial.readStringUntil('\n');
      if (input.length() > 0) {
        respuesta = toupper(input.charAt(0));
        break;
      }
    }
  }

  if (respuesta == 'S') {
    Serial.println("Restaurando configuración a valores de fábrica...");
    preferences.clear(); // Borrar todas las configuraciones almacenadas
    Serial.println("Configuración restaurada. Reiniciando dispositivo.");
    ESP.restart(); // Reiniciar el dispositivo para aplicar los cambios
  }

  // Configurar DMA
  i2s_config_t i2s_config = {
  .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX |
I2S_MODE_ADC_BUILT_IN),
  .sample_rate = SAMPLE_RATE,          // Frecuencia de muestreo en Hz
  .bits_per_sample = I2S_BITS_PER_SAMPLE_16BIT, // Resolución de bits por muestra
  .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,  // Solo un canal (mono)
  .communication_format = I2S_COMM_FORMAT_I2S_MSB,
  .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,     // Asignación de interrupciones
  .dma_buf_count = 4,                           // Número de buffers DMA
  .dma_buf_len = BUFFER_SIZE / 2                // Longitud de cada buffer
  };

  // Configuración del I2S
  i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
  i2s_set_adc_mode(ADC_UNIT_1, ADC1_CHANNEL_0); // Canal ADC1_0 (GPIO36)

  // Habilitar el ADC en modo I2S
  i2s_adc_enable(I2S_NUM_0);

  // Intentar montar LittleFS
  if (!LittleFS.begin()) {
    Serial.println("LittleFS no pudo ser montado, intentando formatear...");
    if (LittleFS.format()) {
      Serial.println("Sistema de archivos LittleFS formateado exitosamente.");
      if (!LittleFS.begin()) {
        Serial.println("Error al montar el sistema de archivos LittleFS después de formatear.");
        return;
      }
    } else {
      Serial.println("Error al formatear el sistema de archivos LittleFS.");
      return;
    }
  }

  // Leer configuraciones de la memoria no volátil
  bool wifi_config_existe = preferences.getBool("wifi_config", false);

  if (wifi_config_existe) {
    Serial.println("Configuración de Wi-Fi existente encontrada.");
    // Si existe configuración, cargarla
    preferences.getString("ssid", ssid, sizeof(ssid));
```

```
        preferences.getString("password", password, sizeof(password));
    } else {
        Serial.println("No se encontró configuración de Wi-Fi.");
        // Si no existe configuración, pedirla al usuario
        pedirWifiConfiguracion();
    }

    //Primera conexión a WiFi
    Serial.print("Conectando a Wi-Fi...");
    WiFi.begin(ssid, password);
    unsigned long startAttemptTime = millis();
    bool connected = false;

    while (millis() - startAttemptTime < 5000) { // 5 segundos de intento
        if (WiFi.status() == WL_CONNECTED) {
            connected = true;
            break;
        }
        delay(500);
        Serial.print(".");
    }

    if (connected) {
        Serial.println("\n¡Conectado a Wi-Fi!");
        Serial.print("Dirección IP: ");
        Serial.println(WiFi.localIP());

        // Iniciar el cliente NTP
        timeClient.begin();
        timeClient.update();
    } else {
        Serial.println("\nNo se pudo conectar a la red Wi-Fi.");
        ESP.restart(); // Reiniciar si no se puede conectar
    }

    // Configurar el certificado del servidor MQTT
    espClient.setCACert(mqtt_server_cert);

    //Configuración MQTT
    preferences.begin("mqtt-config", false);
    bool mqtt_config_existe = preferences.getBool("mqtt_config", false);

    if (mqtt_config_existe) {
        preferences.getString("mqtt_broker", mqtt_broker, sizeof(mqtt_broker));
        mqtt_port = preferences.getInt("mqtt_port", 8883); // Usar el puerto MQTT sobre TLS
        preferences.getString("mqtt_username", mqtt_username, sizeof(mqtt_username));
        preferences.getString("mqtt_password", mqtt_password, sizeof(mqtt_password));
    } else {
        pedirMQTTConfiguracion();
    }

    //Primera conexión al servidor
    client.setServer(mqtt_broker, mqtt_port);
    client.setCallback(callback);

    while (!client.connected()) {
        Serial.print("Conectando al broker MQTT de manera segura...");
        if (client.connect(clientId.c_str(), mqtt_username, mqtt_password, "Conexiones", 1, true,
"Desconectado")) {
            Serial.println("Conectado.");
```

```
            client.publish("Conexiones", "ESP32 conectado", false); // Publicar mensaje de conexión
            //Suscripción a tópicos
            client.subscribe("Actualizacion");
            client.subscribe(topicaccion.c_str());
            client.subscribe(topicActualizacion.c_str());
          } else {
            if (intentos < 2){
              Serial.print("Fallo al conectar, rc=");
              Serial.print(client.state());
              Serial.println(" Se intentará de nuevo en 3 segundos.");
              delay(3000);
              intentos = intentos + 1;
              if (intentos == 2){
                Serial.print("Fallo al conectar, rc=");
                Serial.print(client.state());
                Serial.println("Reiniciando ESP32");
                ESP.restart();
              }
            }
          }
    }

    timeClient.begin();

    // Crear tareas de FreeRTOS
    xTaskCreatePinnedToCore(conectarWiFi, "conectarWiFi", 8192, NULL, 1, &taskWiFiHandle,
0);
    xTaskCreatePinnedToCore(publicarAviso, "publicarAviso", 8192, NULL, 1,
&taskMonitorRMSHandle, 0);
}

void loop() {
    client.loop(); // Para leer mensajes entrantes

    unsigned long tiempoActual = millis(); // Captura el tiempo actual
    // Verifica si ha pasado un minuto
    if (tiempoActual - tiempoAnterior >= intervalo) {
        tiempoAnterior = tiempoActual; // Guarda el tiempo actual para la próxima verificación
        esHorarioRestringido();
    }
}


//Tareas FreeRTOS
void publicarAviso(void *parameter) {

  while(true){
    if (esHorarioRestringido()){
      Serial.println("Horario restringido: Microfono desactivado");
    } else {
    uint16_t buffer[BUFFER_SIZE];  // Buffer para almacenar los datos de muestreo
    size_t bytesRead;

    // Lee datos del ADC usando DMA y almacena en el buffer
    i2s_read(I2S_NUM_0, (void *)buffer, sizeof(buffer), &bytesRead, portMAX_DELAY);

    // Calcula el número de muestras leídas
    int samplesRead = bytesRead / sizeof(uint16_t);

    // Acumula los cuadrados de las muestras (restando el offset y elevando al cuadrado)
```

```
    for (int i = 0; i < samplesRead; i++) {
      int16_t adjustedValue = (int16_t)buffer[i] - OFFSET;
      sumaCuadrados += pow(adjustedValue, 2);
      contadorMuestras++;
    }

    // Verifica si ha pasado un segundo
    if (millis() - ultimoRMS >= 1000) {
      // Calcular el valor RMS
      ValorRMS = sqrt(sumaCuadrados / contadorMuestras);
      alertaEnviada = false; //Para enviar la alerta una vez por valor si es necesario

      // Mostrar el valor RMS
      Serial.print("Valor RMS: ");
      Serial.println(ValorRMS);

      // Reinicia las variables
      sumaCuadrados = 0.0;
      contadorMuestras = 0;
      ultimoRMS = millis(); // Actualiza el tiempo del último cálculo
    }
    if (ValorRMS <= 42){
      dB = 0.2072*ValorRMS + 20.948;
    } else if (ValorRMS >= 111){
      dB = 0.0477*ValorRMS + 32.866;
    } else {
      dB = 0.1196*ValorRMS + 24.915;
    }

    // Publica alerta cuando hay ruido
    if (dB > 40){
      if (!alertaEnviada){
        String ValordBString = String(dB);
        client.publish(topicfiesta.c_str(), ValordBString.c_str(), false);
        Serial.println("Alerta");
        alertaEnviada = true;
      }
    }
  }
  vTaskDelay(1000/portTICK_PERIOD_MS);
  }


}


void conectarWiFi(void *parameter) {
    while (true) {
        if (!WiFi.isConnected()) {  // Si no está conectado a WiFi
            Serial.println("Conectando a WiFi...");
            WiFi.begin(ssid, password);
            int intentos = 0;
            while (WiFi.status() != WL_CONNECTED && intentos < 5) {
                delay(1000);
                Serial.print(".");
                intentos++;
            }
            if (WiFi.isConnected()) {  // Si logra conectarse a WiFi
                Serial.println("Conectado a WiFi.");
                conWifi = true;
```

```
            if (usandoSIM) {  // Si estaba usando la SIM, apágala
                apagarSIM();
                usandoSIM = false;
            }
        } else {  // Si no logra conectarse, usa la SIM
            if (!usandoSIM) {  // Evitar reiniciar la SIM si ya está en uso
                Serial.println("No se pudo conectar a WiFi. Usando SIM...");
                conWifi = false;
                preparaSIM();
                usandoSIM = true;
            }
        }
    } else {  // Si ya está conectado a WiFi
        Serial.println("Sigo conectado a WiFi");
        conWifi = true;
    }
    if (conWifi){
      if (!client.connected()){
        while (!client.connected()) {
            Serial.print("Conectando al broker MQTT de manera segura...");
            if (client.connect(clientId.c_str(), mqtt_username, mqtt_password, "Conexiones", 1,
true, "Desconectado")) {
                Serial.println("Conectado.");
                client.publish("Conexiones", "ESP32 conectado", false); // Publicar mensaje de
conexión
                // Puedes suscribirte a topics si es necesario
                client.subscribe("Actualizacion");
                client.subscribe(topicaccion.c_str());
                client.subscribe(topicActualizacion.c_str());
                Serial.println("Suscrito a todo desde WiFi");
            } else {
              if (intentos < 2){
                Serial.print("Fallo al conectar, rc=");
                Serial.print(client.state());
                Serial.println(" Se intentará de nuevo en 3 segundos.");
                delay(3000);
                intentos = intentos + 1;
                if (intentos == 2){
                  Serial.print("Fallo al conectar, rc=");
                  Serial.print(client.state());
                  Serial.println(" Reiniciando ESP32");
                  ESP.restart();
                }
              }
            }
        }
   } else {
        Serial.println("Sigo en MQTT");
        conWifi = false;
   }
    }
    // Descarga archivo si hay una tarea pendiente
    if (pendiente) {
        descargaArchivo();
    }

    vTaskDelay(60000 / portTICK_PERIOD_MS);  // Retardo de 60 segundos antes de
reintentar
   }
}
```

```
//Resto de tareas
bool esHorarioRestringido() {
  timeClient.update();
  int currentHour = timeClient.getHours();

  // Verifica si está entre 8 a.m. y 10 p.m.
  return (currentHour >= 8 && currentHour < 22);
}

void pedirWifiConfiguracion() {
   Serial.println("¿Desea escanear redes Wi-Fi cercanas (E) o ingresar el SSID y contraseña
manualmente (M)? (E/M)");

   char opcion = ' ';
   unsigned long start_time = millis();
   while (millis() - start_time < 20000) { // Esperar 20 segundos
      if (Serial.available()) {
         String input = Serial.readStringUntil('\n');
         if (input.length() > 0) {
            opcion = toupper(input.charAt(0));
            break;
         }
      }
   }

   if (opcion == 'E') {
      escanearRedesWiFi();
      Serial.println("Ingrese el número de la red Wi-Fi a la que desea conectar:");
      int redSeleccionada;
      String ssid_input, password_input;
      char confirmacion;

      while (true) {
        // Limpiar el buffer del puerto serie antes de pedir la selección de red
        while (!Serial.available()); // Esperar entrada del usuario
        String input = Serial.readStringUntil('\n'); // Leer la entrada hasta que se presione Enter
        input.trim(); // Eliminar espacios en blanco al inicio o al final

        // Verificar si la entrada es un número válido
        redSeleccionada = input.toInt();
        Serial.flush(); // Limpiar buffer para asegurar lectura correcta

        // Validar selección de red
        if (redSeleccionada < 1 || redSeleccionada > WiFi.scanNetworks()) {
           Serial.println("Número de red inválido. Por favor, intente nuevamente.");
           continue;
        } else {
           ssid_input = redesWiFi[redSeleccionada - 1].ssid;
           Serial.print("Red Wi-Fi seleccionada: ");
           Serial.println(ssid_input);

           Serial.println("Ingrese la contraseña de la red Wi-Fi:");
           while (!Serial.available()); // Esperar entrada del usuario
           password_input = Serial.readStringUntil('\n');
           password_input.trim(); // Eliminar espacios en blanco y saltos de línea

           Serial.print("Contraseña ingresada: ");
```

```
            Serial.println(password_input);

            // Confirmar con el usuario
            Serial.println("¿La información es correcta? (Y/N):");

            while (true) {
               // Limpiar el buffer del puerto serie antes de la confirmación
               while (!Serial.available()); // Esperar entrada del usuario
                 String confirm_input = Serial.readStringUntil('\n');
                 confirm_input.trim();  // Eliminar espacios y saltos de línea

               // Verificar si la confirmación es válida
               if (confirm_input.length() > 0) {
                  confirmacion = toupper(confirm_input.charAt(0));
               } else {
                  Serial.println("Entrada inválida. Por favor ingrese 'Y' para Sí o 'N' para No.");
                  Serial.flush(); // Limpiar el buffer si la entrada es inválida
               }

               // Actuar según la confirmación
               if (confirmacion == 'Y') {
                  ssid_input.toCharArray(ssid, sizeof(ssid));
                  password_input.toCharArray(password, sizeof(password));

                  // Guardar configuración en la memoria RTC
                  preferences.putBool("wifi_config", true);
                  preferences.putString("ssid", ssid);
                  preferences.putString("password", password);

                  Serial.println("Configuración de Wi-Fi guardada.");
                  break;
               } else if (confirmacion == 'N') {
                  Serial.println("Por favor, ingrese nuevamente la configuración.");
                  // Volver al inicio del proceso
                  break;
               } else {
                  Serial.println("Entrada inválida. Por favor ingrese 'Y' para Sí o 'N' para No.");
                  Serial.flush(); // Limpiar el buffer si la entrada es inválida
               }
            }

            // Si la confirmación es correcta ('Y'), salir del bucle
            if (confirmacion == 'Y') {
               break;
            }
         }
      }

   } else if (opcion == 'M') {
      char confirmacion;

while (true) {


      while (true) {  // Bucle que sigue pidiendo usuario y contraseña si no son válidos
            Serial.println("Ingrese el SSID de la red Wi-Fi:");
            while (!Serial.available()); // Esperar entrada del usuario
               String ssid_input = Serial.readStringUntil('\n');
               ssid_input.trim();
               ssid_input.toCharArray(ssid, sizeof(ssid));
```

```
            //Serial.println(ssid_input);

          Serial.println("Ingrese la contraseña de la red Wi-Fi:");
            while (!Serial.available()); // Esperar entrada del usuario
            String password_input = Serial.readStringUntil('\n');
            password_input.trim();
            password_input.toCharArray(password, sizeof(password));
            //Serial.println(password_input);


         // Verificar si ambos campos son válidos
         if (ssid_input.length() > 0 && password_input.length() > 0) {
            break;  // Salir del bucle si ambos valores son correctos
         } else {
            // Si alguno de los dos está vacío, mostrar el mensaje y repetir el ciclo
            Serial.println("SSID y/o contraseña no introducidos. Por favor, intente de nuevo.");
         }
       }

      // Confirmar con el usuario
      Serial.println("¿La información es correcta? (Y/N):");
      while (true) {
         while (!Serial.available()); // Esperar entrada del usuario
         String confirm_input = Serial.readStringUntil('\n');
         if (confirm_input.length() > 0) {
            confirmacion = toupper(confirm_input.charAt(0));
         } else {
            confirmacion = 'N';  // Valor predeterminado si no hay entrada
         }

         // Validar confirmación
         if (confirmacion == 'Y') {
            // Guardar configuración en la memoria RTC
            preferences.putBool("wifi_config", true);
            preferences.putString("ssid", ssid);
            preferences.putString("password", password);

            Serial.println("Configuración de Wi-Fi guardada.");
            break;  // Salir del bucle de confirmación y del proceso
         } else if (confirmacion == 'N') {
            Serial.println("Por favor, ingrese nuevamente la configuración.");
            break;  // Salir del bucle de confirmación, pero reiniciar el proceso
         } else {
            Serial.println("Entrada inválida. Por favor ingrese 'Y' para Sí o 'N' para No.");
            Serial.flush(); // Limpiar buffer
         }
      }

      // Si la confirmación fue 'Y', salimos del bucle principal
      if (confirmacion == 'Y') {
         break;  // Salir del bucle principal, ya que la configuración es correcta
      }
}
   } else {
      Serial.println("Opción no válida. Se usará la configuración existente o se repetirá el
proceso.");
      pedirWifiConfiguracion();
   }
}
```

```
void escanearRedesWiFi() {
   Serial.println("Escaneando redes Wi-Fi disponibles...");
   int numRedes = WiFi.scanNetworks();

   if (numRedes == -1) {
      Serial.println("Error al escanear redes Wi-Fi.");
      ESP.restart();
   } else if (numRedes == 0) {
      Serial.println("No se encontraron redes Wi-Fi.");
      ESP.restart();
   } else {
      Serial.println("Redes Wi-Fi encontradas:");
      for (int i = 0; i < numRedes && i < 200; i++) { // Limitar a 200 redes
         redesWiFi[i].numero = i + 1; // Número de red
         redesWiFi[i].ssid = WiFi.SSID(i); // SSID de la red
         Serial.printf("%d: %s (Señal: %d dBm)\n", i + 1, WiFi.SSID(i).c_str(), WiFi.RSSI(i));
      }
   }
}

void pedirMQTTConfiguracion() {
   Serial.println("Ingrese la configuración de MQTT.");
   while (true){
       // Solicitar el broker
      Serial.println("Ingrese el broker MQTT: ");
      while (!Serial.available());
      String broker_input = Serial.readStringUntil('\n');
      broker_input.trim();
      broker_input.toCharArray(mqtt_broker, sizeof(mqtt_broker));
      if (broker_input.length() > 0) {
         break;  // Salir del bucle si ambos valores son correctos
      } else {
         // Si alguno de los dos está vacío, mostrar el mensaje y repetir el ciclo
         Serial.println("Nombre del servidor no definido. Por favor, intente de nuevo.");
      }
   }


   // Solicitar usuario y contraseña, con validación
   while (true) {  // Bucle que sigue pidiendo usuario y contraseña si no son válidos
      Serial.println("Ingrese el nombre de usuario MQTT:");
      while (!Serial.available());
      String user_input = Serial.readStringUntil('\n');
      user_input.trim();
      user_input.toCharArray(mqtt_username, sizeof(mqtt_username));

      Serial.println("Ingrese la contraseña MQTT:");
      while (!Serial.available());
      String pass_input = Serial.readStringUntil('\n');
      pass_input.trim();
      pass_input.toCharArray(mqtt_password, sizeof(mqtt_password));

      // Verificar si ambos campos son válidos
      if (user_input.length() > 0 && pass_input.length() > 0) {
         break;  // Salir del bucle si ambos valores son correctos
      } else {
         // Si alguno de los dos está vacío, mostrar el mensaje y repetir el ciclo
         Serial.println("Usuario y/o contraseña no introducidos. Por favor, intente de nuevo.");
      }
   }
```

```
    // Guardar configuración en la memoria RTC
    preferences.putBool("mqtt_config", true);
    preferences.putString("mqtt_broker", mqtt_broker);
    preferences.putInt("mqtt_port", mqtt_port);
    preferences.putString("mqtt_username", mqtt_username);
    preferences.putString("mqtt_password", mqtt_password);

    Serial.println("Configuración de MQTT guardada.");
}



void callback(char* topicReceived, byte* payload, unsigned int length) {
    String mensajeRecibido;
    for (int i = 0; i < length; i++) {
        mensajeRecibido += (char)payload[i]; // Convertir cada byte a carácter y agregarlo a
mensajeRecibido
    }
    Serial.println(mensajeRecibido);
    if (String(topicReceived) == "Actualizacion" or String(topicReceived) == topicActualizacion){
      if (mensajeRecibido == "Actualiza"){
        pendiente = true;
      }
    }
    if (String(topicReceived) == topicaccion){
    if (mensajeRecibido == "Enciende secador") {
      digitalWrite(relePin1, LOW);
      Serial.println("Secador encendido");
    } else if (mensajeRecibido == "Apaga secador") {
      digitalWrite(relePin1, HIGH);
      Serial.println("Secador apagado");
    } else if (mensajeRecibido == "Enciende cargador") {
      digitalWrite(relePin2, LOW);
      Serial.println("Cargador encendido");
    } else if (mensajeRecibido == "Apaga cargador") {
      digitalWrite(relePin2, HIGH);
      Serial.println("Cargador apagado");
    } else if (mensajeRecibido == "Apaga todo") {
      digitalWrite(relePin1, HIGH);
      digitalWrite(relePin2, HIGH);
      Serial.println("Todo apagado");
    } else if (mensajeRecibido == "Enciende todo") {
      digitalWrite(relePin1, LOW);
      digitalWrite(relePin2, LOW);
      Serial.println("Todo encendido");
}
}
}



void descargaArchivo() {
  HTTPClient http;

  // Crear la URL completa
  String url = String(server) + path;

  Serial.println("Descargando archivo...");
```

```
  http.begin(url);  // Iniciar la conexión
  http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);  // Habilitar redirección

  int httpCode = http.GET();  // Realizar la solicitud GET

  if (httpCode == HTTP_CODE_OK) {
    Serial.println("Archivo descargado con éxito!");

    // Verificar el tamaño del archivo recibido
    int contentLength = http.getSize();
    Serial.printf("Tamaño del archivo descargado: %d bytes\n", contentLength);

    // Abrir archivo en LittleFS para escribir los datos
    File file = LittleFS.open("/helloworld.ino.bin", "w");
    if (!file) {
      Serial.println("Error al abrir el archivo en LittleFS");
      http.end(); // Finalizar la conexión HTTP
      return;
    }

    // Leer y escribir el archivo en fragmentos de 1024 bytes
    WiFiClient* stream = http.getStreamPtr();
    uint8_t buffer[1024];
    int totalWritten = 0;

    while (http.connected() && totalWritten < contentLength) {
      size_t size = stream->available();
      if (size > 0) {
        int c = stream->readBytes(buffer, ((size > sizeof(buffer)) ? sizeof(buffer) : size));
        file.write(buffer, c);
        totalWritten += c;
      }
    }

    file.close();
    Serial.printf("Archivo guardado en LittleFS. Bytes escritos: %d\n", totalWritten);

    // Verificar el tamaño del archivo guardado
    File verifyFile = LittleFS.open("/helloworld.ino.bin", "r");
    if (verifyFile) {
      size_t fileSize = verifyFile.size();
      Serial.printf("Tamaño del archivo guardado en LittleFS: %d bytes\n", fileSize);
      if (fileSize != contentLength) {
        Serial.println("¡Advertencia! El tamaño del archivo guardado no coincide con el archivo
descargado.");
      }
      verifyFile.close();
    } else {
      Serial.println("Error al abrir el archivo para verificación.");
    }

    // Llamamos a la función para flashear el firmware
    actualizaFirmware("/helloworld.ino.bin");

  } else {
    Serial.print("Error al descargar el archivo. Código de respuesta: ");
    Serial.println(httpCode);
  }

  http.end();  // Finalizar la conexión HTTP
```

```
}

void actualizaFirmware(const char* fileName) {
  // Verificar si el archivo existe antes de intentar flashear
  if (!LittleFS.exists(fileName)) {
    Serial.println("Error: El archivo no existe en LittleFS.");
    return;
  }

  // Abrir el archivo binario para flashear
  File file = LittleFS.open(fileName, "r");
  if (!file) {
    Serial.println("Error al abrir el archivo para flashear.");
    return;
  }

  size_t fileSize = file.size();
  Serial.printf("Flasheando %s, tamaño: %d bytes\n", fileName, fileSize);

  // Verificar si el archivo tiene un tamaño mayor que 0
  if (fileSize == 0) {
    Serial.println("El archivo no tiene datos, abortando.");
    file.close();
    return;
  }

  // Iniciar la actualización del firmware
  if (!Update.begin(fileSize)) {
    Serial.println("Error al iniciar la actualización");
    file.close();
    return;
  }

  // Actualizar el firmware escribiendo el archivo binario en el ESP32
  size_t written = Update.writeStream(file);

  if (written == fileSize) {
    Serial.println("Firmware actualizado correctamente. Reiniciando...");
    if (Update.end()) {
      Serial.println("Actualización completa. Reiniciando...");
      ESP.restart();  // Reiniciar para cargar el nuevo firmware
    } else {
      Serial.println("Error al finalizar la actualización");
    }
  } else {
    Serial.println("Error al escribir el archivo binario");
  }

  file.close();
}


void preparaSIM(){
  while(true){
    Serial.begin(115200); // Set console baud rate
    Serial.println("Preparando SIM");
    SerialAT.begin(115200, SERIAL_8N1, MODEM_RX_PIN, MODEM_TX_PIN);

#ifdef BOARD_POWERON_PIN
    pinMode(BOARD_POWERON_PIN, OUTPUT);
```

```
    digitalWrite(BOARD_POWERON_PIN, HIGH);
#endif

    // Set modem reset pin ,reset modem
    pinMode(MODEM_RESET_PIN, OUTPUT);
    digitalWrite(MODEM_RESET_PIN, !MODEM_RESET_LEVEL);
    delay(100);
    digitalWrite(MODEM_RESET_PIN, MODEM_RESET_LEVEL);
    delay(2600);
    digitalWrite(MODEM_RESET_PIN, !MODEM_RESET_LEVEL);

    pinMode(BOARD_PWRKEY_PIN, OUTPUT);
    digitalWrite(BOARD_PWRKEY_PIN, LOW);
    delay(100);
    digitalWrite(BOARD_PWRKEY_PIN, HIGH);
    delay(100);
    digitalWrite(BOARD_PWRKEY_PIN, LOW);

    // Check if the modem is online
    Serial.println("Iniciando módem");

    int retry = 0;
    while (!modem.testAT(1000)) {
        Serial.println(".");
        if (retry++ > 10) {
            digitalWrite(BOARD_PWRKEY_PIN, LOW);
            delay(100);
            digitalWrite(BOARD_PWRKEY_PIN, HIGH);
            delay(1000);
            digitalWrite(BOARD_PWRKEY_PIN, LOW);
            retry = 0;
        }
    }
    Serial.println();

    // Check if SIM card is online
    SimStatus sim = SIM_ERROR;
    while (sim != SIM_READY) {
        sim = modem.getSimStatus();
        switch (sim) {
        case SIM_READY:
            Serial.println("SIM card online");
            break;
        case SIM_LOCKED:
            Serial.println("The SIM card is locked. Please unlock the SIM card first.");

            modem.simUnlock(SIMCARD_PIN_CODE);
            break;
        default:
            break;
        }
        delay(1000);
    }


#ifdef NETWORK_APN
    Serial.printf("Set network apn : %s\n", NETWORK_APN);
    modem.sendAT(GF("+CGDCONT=1,\"IP\",\""), NETWORK_APN, "\"");
    if (modem.waitResponse() != 1) {
        Serial.println("Error en el APN");
```

```
        continue;
    }
#endif

    // Check network registration status and network signal status
    int16_t sq ;
    Serial.print("Conectando a red");
    RegStatus status = REG_NO_RESULT;
    while (status == REG_NO_RESULT || status == REG_SEARCHING || status ==
REG_UNREGISTERED) {
        status = modem.getRegistrationStatus();
        switch (status) {
        case REG_UNREGISTERED:
        case REG_SEARCHING:
            sq = modem.getSignalQuality();
            Serial.printf("[%lu] Signal Quality:%d\n", millis() / 1000, sq);
            delay(1000);
            break;
        case REG_DENIED:
            Serial.println("Network registration was rejected, please check if the APN is correct");
            continue ;
        case REG_OK_HOME:
            Serial.println("Online registration successful");
            break;
        case REG_OK_ROAMING:
            Serial.println("Network registration successful, currently in roaming mode");
            break;
        default:
            Serial.printf("Registration Status:%d\n", status);
            delay(1000);
            break;
        }
    }
    Serial.println();


    Serial.printf("Registration Status:%d\n", status);
    delay(1000);

    String ueInfo;
    if (modem.getSystemInformation(ueInfo)) {
        Serial.print("Inquiring UE system information:");
        Serial.println(ueInfo);
    }

    if (!modem.enableNetwork()) {
        Serial.println("Enable network failed!");
        continue;
    }

    delay(1000);

    String ipAddress = modem.getLocalIP();
    Serial.print("Network IP:"); Serial.println(ipAddress);

    // Initialize MQTT, use SSL, skip authentication server
    modem.mqtt_begin(true);


    if (!mqtt_connect()) {
```

```
      continue ;
    }
    Serial.println("Modem funcionando OK");
    break;
  }
}

bool mqtt_connect()
{

    const char *broker = mqtt_broker;
    uint16_t broker_port = static_cast<uint16_t>(mqtt_port);
    const char *client_id = clientId.c_str();
    const char *username = mqtt_username;
    const char *password = mqtt_password;
    const char *topicoFiesta = topicfiesta.c_str();
    const char *topicoAccion = topicaccion.c_str();
    const char *topicoactualizacion = topicActualizacion.c_str();
    Serial.print("Conectando a ");
    Serial.print(broker);


    // Set up Hivemq  ROOT certificate
    modem.mqtt_set_certificate(mqtt_server_cert);

    bool ret = modem.mqtt_connect(mqtt_client_id, broker, broker_port, client_id, username,
password);
    if (!ret) {
      Serial.println("Error"); return false;

    }
    Serial.println("Éxito");

    if (modem.mqtt_connected()) {
      Serial.println("MQTT conectado");
    } else {
      return false;
    }
    // Set MQTT processing callback
    modem.mqtt_set_callback(callbackSIM);
    // Subscribe to topic
    modem.mqtt_subscribe(mqtt_client_id, topicoFiesta);
    modem.mqtt_subscribe(mqtt_client_id, topicoAccion);
    modem.mqtt_subscribe(mqtt_client_id, topicoactualizacion);
    modem.mqtt_subscribe(mqtt_client_id, "Actualizacion");
    Serial.println("Suscrito a todo desde SIM");

    return true;
}

void callbackSIM(const char *topic, const uint8_t *payload, uint32_t len)
{
    String mensajeRecibido;
    const char *topicoFiesta = topicfiesta.c_str();
    const char *topicoAccion = topicaccion.c_str();
    const char *topicoactualizacion = topicActualizacion.c_str();
    for (int i = 0; i < len; i++) {
      mensajeRecibido += (char)payload[i]; // Convertir cada byte a carácter y agregarlo a
mensajeRecibido
    }
```

```
    if (String(topic) == "Actualizacion" or String(topic) == topicoactualizacion){
      if (mensajeRecibido == "Actualiza"){
        pendiente = true;
      }
    }
   if (String(topic) == topicoAccion) {
    if (mensajeRecibido == "Enciende secador") {
        digitalWrite(relePin1, LOW);
        Serial.println("Secador encendido");
    } else if (mensajeRecibido == "Enciende cargador") {
        digitalWrite(relePin2, LOW);
        Serial.println("Cargador encendido");
    } else if (mensajeRecibido == "Apaga secador") {
        digitalWrite(relePin1, HIGH);
        Serial.println("Secador apagado");
    } else if (mensajeRecibido == "Apaga cargador") {
        digitalWrite(relePin2, HIGH);
        Serial.println("Cargador apagado");
    } else if (mensajeRecibido == "Apaga todo"){
        digitalWrite(relePin1, HIGH);
        digitalWrite(relePin2, HIGH);
        Serial.println("Todo apagado");
    } else if (mensajeRecibido == "Enciende todo"){
        digitalWrite(relePin1, LOW);
        digitalWrite(relePin2, LOW);
        Serial.println("Todo encendido");
    }
   }
}

void apagarSIM() {
    Serial.println("Apagando modem SIM...");
    modem.sendAT("+CMQTTSTOP");
    Serial.println("Cerrando conexion MQTT");
    modem.sendAT("+CPOF");  // Enviar el comando AT para apagar el modem
    if (modem.waitResponse() != 1) {
        Serial.println("Error apagando");
    } else {
        Serial.println("Modem apagado correctamente.");
    }
}
```

# ANEXO C: Scripts de Python

## Conexión Wi-Fi escaneo

```
import serial
import time
errorescaneo = 0
estado = 0
reset = 0

def conectar_arduino(puerto, baudrate=115200, timeout=1):
    """Conectar con el Arduino usando pySerial."""
    try:
        arduino = serial.Serial(puerto, baudrate, timeout=timeout)
        time.sleep(2)  # Esperar a que Arduino se reinicie al conectar
        print("Conexión establecida con Arduino en el puerto", puerto)
        return arduino
    except serial.SerialException as e:
        print("Error al conectar con Arduino:", e)
        return None

def enviar_datos(arduino, mensaje):
    """Enviar un mensaje a Arduino."""
    if arduino:
        arduino.write(mensaje.encode())
        print(f"Enviado: {mensaje}")

def leer_respuesta(arduino):
    """Leer la respuesta enviada por Arduino."""
    if arduino:
        try:
            respuesta = arduino.readline().decode('utf-8').strip()
            print(f"{respuesta}")
            return respuesta
        except Exception as e:
            print("Error al leer la respuesta:", e)
            return None

if __name__ == "__main__":
    # Define el puerto serie de tu Arduino (ej. COM3 en Windows, /dev/ttyUSB0 en Linux)
    puerto_serie = 'COM4'  # Cambia esto al puerto correcto

    # Conectar al Arduino
    arduino = conectar_arduino(puerto_serie)

    if arduino:
        while True:
            # Leer la respuesta de Arduino
            recibido = leer_respuesta(arduino)
            if recibido == '¿Desea restaurar la configuración a valores de fábrica? (S/N):':
                if reset == 1:
                    enviar_datos(arduino, 'N')
                    reset = 0
                else:
                    enviar_datos(arduino, 'S')
                    reset = 1
```

```
        if recibido == '¿Desea escanear redes Wi-Fi cercanas (E) o ingresar el SSID y
contraseña manualmente (M)? (E/M)':
            if errorescaneo == 0:
                enviar_datos(arduino, 'H')
                errorescaneo = errorescaneo + 1
            else:
                enviar_datos(arduino, 'E')
        # Si el mensaje es "norte" o "sur", enviar "sur" como respuesta
        if recibido == 'Número de red inválido. Por favor, intente nuevamente.':
                enviar_datos(arduino, '2')

        if recibido == 'Ingrese el número de la red Wi-Fi a la que desea conectar:':
            if estado == 0:
                enviar_datos(arduino, '0')
            elif estado > 0:
                enviar_datos(arduino, '1')


        if recibido == 'Ingrese la contraseña de la red Wi-Fi:':
            if estado < 2:
                enviar_datos(arduino, '12345')
            else:
                enviar_datos(arduino, 'patorulo')
        if recibido == '¿La información es correcta? (Y/N):':
             if estado == 3:
                 enviar_datos(arduino, 'H')
             elif estado < 2:
                 enviar_datos(arduino, 'Y')
                 estado = estado + 1
             else:
                 enviar_datos(arduino, 'N')
                 estado = estado + 1
        if recibido == "Entrada inválida. Por favor ingrese 'Y' para Sí o 'N' para No.":
            enviar_datos(arduino, 'Y')
        if recibido == 'Por favor, ingrese nuevamente la configuración.':
            enviar_datos(arduino, '1')
    # Cerrar la conexión
        if recibido == "¡Conectado a Wi-Fi!":
            arduino.close()
            print("Conexión cerrada. Prueba escaner WiFi exitosa.")
            break
```

## Conexión Wi-Fi manual

```
import serial
import time
estado = 0
reset = 0

def conectar_arduino(puerto, baudrate=115200, timeout=1):
    """Conectar con el Arduino usando pySerial."""
    try:
        arduino = serial.Serial(puerto, baudrate, timeout=timeout)
        time.sleep(2)  # Esperar a que Arduino se reinicie al conectar
        print("Conexión establecida con Arduino en el puerto", puerto)
        return arduino
    except serial.SerialException as e:
        print("Error al conectar con Arduino:", e)
        return None

def enviar_datos(arduino, mensaje):
    """Enviar un mensaje a Arduino."""
    if arduino:
        arduino.write(mensaje.encode())
        print(f"Enviado: {mensaje}")

def leer_respuesta(arduino):
    """Leer la respuesta enviada por Arduino."""
    if arduino:
        try:
            respuesta = arduino.readline().decode('utf-8').strip()
            print(f"{respuesta}")
            return respuesta
        except Exception as e:
            print("Error al leer la respuesta:", e)
            return None

if __name__ == "__main__":
    # Define el puerto serie de tu Arduino (ej. COM3 en Windows, /dev/ttyUSB0 en Linux)
    puerto_serie = 'COM4'  # Cambia esto al puerto correcto

    # Conectar al Arduino
    arduino = conectar_arduino(puerto_serie)

    if arduino:
        while True:
            # Leer la respuesta de Arduino
            recibido = leer_respuesta(arduino)
            if recibido == '¿Desea restaurar la configuración a valores de fábrica? (S/N):':
                if reset == 1:
                    enviar_datos(arduino, 'N')
                    reset = 0
                else:
                    enviar_datos(arduino, 'S')
                    reset = 1

            if recibido == '¿Desea escanear redes Wi-Fi cercanas (E) o ingresar el SSID y
contraseña manualmente (M)? (E/M)':
                enviar_datos(arduino, 'M')


            if recibido == 'Ingrese el SSID de la red Wi-Fi:':
```

```
            if estado == 0:
                enviar_datos(arduino, ' ')
            elif estado == 1 or estado == 2:
                enviar_datos(arduino, 'SSID_falso')
            elif estado > 2:
                enviar_datos(arduino, 'Redmi')


        if recibido == 'Ingrese la contraseña de la red Wi-Fi:':
            if estado == 0 or estado == 2:
                enviar_datos(arduino, 'contraseña_falsa')
                if estado == 0:
                    estado = estado + 1
            elif estado == 1:
                enviar_datos(arduino, ' ')
                estado = estado + 1
            else:
                enviar_datos(arduino, 'patorulo')
        if recibido == '¿La información es correcta? (Y/N):':
             if estado == 2 or estado == 5:
                 enviar_datos(arduino, 'Y')
                 estado = estado +1
             elif estado == 3:
                 enviar_datos(arduino, 'N')
                 estado = estado + 1
             elif  estado == 4:
                 enviar_datos(arduino, 'H')
                 estado = estado + 1
        if recibido == "Entrada inválida. Por favor ingrese 'Y' para Sí o 'N' para No.":
            enviar_datos(arduino, 'Y')

    # Cerrar la conexión
        if recibido == "¡Conectado a Wi-Fi!":
            arduino.close()
            print("Conexión cerrada. Prueba WiFi manual exitosa.")
            break
```

## Conexión a servidor MQTT

```
import serial
import time
intento = 0
estado = 0
reset = 0

def conectar_arduino(puerto, baudrate=115200, timeout=1):
    """Conectar con el Arduino usando pySerial."""
    try:
        arduino = serial.Serial(puerto, baudrate, timeout=timeout)
        time.sleep(2)  # Esperar a que Arduino se reinicie al conectar
        print("Conexión establecida con Arduino en el puerto", puerto)
        return arduino
    except serial.SerialException as e:
        print("Error al conectar con Arduino:", e)
        return None

def enviar_datos(arduino, mensaje):
    """Enviar un mensaje a Arduino."""
    if arduino:
        arduino.write(mensaje.encode())
        print(f"Enviado: {mensaje}")

def leer_respuesta(arduino):
    """Leer la respuesta enviada por Arduino."""
    if arduino:
        try:
            respuesta = arduino.readline().decode('utf-8').strip()
            print(f"{respuesta}")
            return respuesta
        except Exception as e:
            print("Error al leer la respuesta:", e)
            return None

def resetear_arduino(arduino):
    """Reiniciar el Arduino usando el control DTR."""
    if arduino:
        arduino.setDTR(False)
        time.sleep(0.1)
        arduino.setDTR(True)
        print("Arduino reiniciado.")

if __name__ == "__main__":
    # Define el puerto serie de tu Arduino (ej. COM3 en Windows, /dev/ttyUSB0 en Linux)
    puerto_serie = 'COM4'  # Cambia esto al puerto correcto

    # Conectar al Arduino
    arduino = conectar_arduino(puerto_serie)

    if arduino:
        while True:
            # Leer la respuesta de Arduino
            recibido = leer_respuesta(arduino)
            if recibido == '¿Desea restaurar la configuración a valores de fábrica? (S/N):':
                if reset == 1:
                    enviar_datos(arduino, 'N')
                    reset = 0
                else:
```

```
                enviar_datos(arduino, 'S')
                reset = 1

        if recibido == '¿Desea escanear redes Wi-Fi cercanas (E) o ingresar el SSID y
contraseña manualmente (M)? (E/M)':
            enviar_datos(arduino, 'M')
        if recibido == 'Ingrese el SSID de la red Wi-Fi:':
            enviar_datos(arduino, 'Redmi')
        if recibido == 'Ingrese la contraseña de la red Wi-Fi:':
            enviar_datos(arduino, 'patorulo')
        if recibido == '¿La información es correcta? (Y/N):':
            enviar_datos(arduino, 'Y')


        if recibido == 'Ingrese el broker MQTT:':
            if estado < 3:
                enviar_datos(arduino, 'servidor_falso')
            else:
                enviar_datos(arduino, 'yanuro.es')

        if recibido == 'Ingrese el nombre de usuario MQTT:':
            if estado == 0 or estado == 2 or estado == 3:
                enviar_datos(arduino, 'usuario_falso')
            elif estado == 1:
                enviar_datos(arduino, ' ')
            else:
                enviar_datos(arduino, 'cliente')

        if recibido == 'Ingrese la contraseña MQTT:':
            if estado == 1 or estado == 2 or estado == 3:
                enviar_datos(arduino, 'contraseña_falsa')
            elif estado == 0:
                enviar_datos(arduino, ' ')
            else:
                enviar_datos(arduino, 'public')
        if recibido == 'Usuario y/o contraseña no introducidos. Por favor, intente de nuevo.':
            estado = estado + 1

            # Detectar mensaje de error del ESP32
        if recibido == "Conectando al broker MQTT de manera segura...Fallo al conectar, rc=-2
Se intentará de nuevo en 3 segundos." or recibido == "Fallo al conectar, rc=5 Se intentará de
nuevo en 3 segundos.":
            intento = intento + 1
            if intento == 3:
                estado = estado + 1
                intento = 0

    # Cerrar la conexión
        if recibido == "Conectado.":
            arduino.close()
            print("Conexión cerrada. Prueba conexion MQTT exitosa.")
            break
```