UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Desarrollo de un sintetizador a partir de un controlador M5Stack Core 2.

Trabajo Fin de Grado

Grado en Ingeniería de Sistemas de Telecomunicación, Sonido e Imagen

AUTOR/A: García Carratalá, Pascual

Tutor/a: Marín-Roig Ramón, José

Cotutor/a: Pérez Pascual, Mª Asunción

CURSO ACADÉMICO: 2023/2024

RESUMEN:

En el campo de la síntesis de sonido y la música electrónica, la combinación de *hardware* y *software* facilita la creación de dispositivos versátiles. Este trabajo se enfoca en el diseño de un sintetizador usando el controlador M5Stack, basado en el microcontrolador ESP32. Este controlador proporciona una base sólida para un sintetizador compacto y portátil con capacidades avanzadas.

El objetivo es aprovechar las características del M5Stack para crear un sintetizador polifónico con una interfaz de usuario intuitiva. Se cubrirán aspectos de *hardware* y *software*, incluyendo la configuración del controlador, la implementación de algoritmos de síntesis de sonido y la creación de una interfaz eficiente. Además, se presentarán los principios de la síntesis de sonido, las capacidades del *hardware* M5Stack y las técnicas de programación necesarias para desarrollar un sintetizador funcional. Este proyecto no solo ofrecerá un sintetizador innovador, sino también una base para futuras investigaciones en música electrónica y sistemas embebidos.

PALABRAS CLAVE: Sintetizador; Síntesis de sonido; ESP32; Procesamiento de señales

ABSTRACT:

In the field of sound synthesis and electronic music, the combination of hardware and software facilitates the creation of versatile devices. This work focuses on designing a synthesizer using the M5Stack controller, based on the ESP32 microcontroller operating system. This controller provides a solid foundation for a compact and portable synthesizer with advanced capabilities.

The goal is to leverage the features of the M5Stack to create a polyphonic synthesizer with an intuitive user interface. Both *hardware* and software aspects will be covered, including the controller configuration, implementation of sound synthesis algorithms, and creation of an efficient interface. Additionally, the principles of sound synthesis, the capabilities of the M5Stack hardware, and the programming techniques necessary to develop a functional synthesizer will be presented. This project will not only offer an innovative synthesizer but also provide a foundation for future research in electronic music and embedded systems.

KEY WORDS: Synthesizer, Sound synthesis, ESP32, Signal processing.

ÍNDICE

LISTA DE SIGLAS

[ADSR]: *Attack, Decay, Sustain, Release.* (Ataque, Decaimiento, Sostenimiento, Liberación)

[VCO]: *Voltage-Controlled Oscillator*. (Oscilador Controlado por Voltaje)

[DCO]: *Digital Controlled Oscillator*. (Oscilador Controlado Digitalmente)

[EQ]: *Equalizer*. (Ecualizador)

[LPF]: *Low-pass filter*. (Filtro Paso Bajo)

[HPF]: *High-pass filter*. (Filtro Paso Alto)

[BPF]: *Band-Pass Filter*. (Filtro Paso Banda)

[LFOs]: *Low-Frequency Oscillators.* (Osciladores de Baja Frecuencia)

[PMU]: *Power Management Unit*. (Unidad de Gestión de Energía)

[GPIO]: *General Purpose Input/Output*. (Entrada/Salida de Propósito General)

[IDE]: *Integrated Development Enviroment*. (Entorno de Desarrollo Integrado)

LISTA DE FIGURAS

1. Introducción

En este capítulo se introduce el proyecto de desarrollo de un sintetizador utilizando el controlador M5Stack Core 2. Se justifica la elección del tema, se detallan las motivaciones personales y académicas que impulsaron la realización del proyecto, así como los objetivos y la metodología empleada para llevarlo a cabo. Además, se describen los alcances y las limitaciones del proyecto y se detalla la estructura de la memoria.

1.1. Justificación

Desarrollar un sintetizador con el M5Stack Core 2 se eligió por varias razones personales y académicas. Como estudiante de ingeniería en sistemas de telecomunicaciones, sonido e imagen, este proyecto integra conocimientos en electrónica, programación, procesamiento de señales y acústica, permitiéndome aplicar y consolidar habilidades adquiridas durante la formación universitaria.

Campos involucrados:

El proyecto abarca varias disciplinas. En electrónica y *hardware*, requiere una comprensión de circuitos electrónicos y el uso de componentes como microcontroladores, amplificadores y altavoces. En programación y algoritmos, implica el uso de C/C++ para generar y manipular sonidos. En procesamiento de señales, aplica técnicas de análisis y manipulación de señales. En acústica y teoría del sonido, exige entender cómo se generan, modulan y perciben las ondas sonoras.

Habilidades prácticas:

El desarrollo de este sintetizador no solo consolida conocimientos teóricos, sino que también proporciona habilidades prácticas esenciales en ingeniería de telecomunicaciones y sonido. Estas habilidades incluyen el diseño y prototipado de circuitos, la depuración y solución de problemas en *hardware* y *software*, y la creatividad e innovación en el diseño y personalización de sonidos y efectos.

1.2. Motivaciones

Desde hace varios años, me ha apasionado la síntesis de audio y la creación de música electrónica. Desarrollar este sintetizador es la culminación de un objetivo tanto personal como académico. Además, la industria de la tecnología musical y el audio profesional está en constante expansión. La experiencia práctica en el diseño y programación de sintetizadores me abrirá puertas a futuras oportunidades laborales y de investigación en este campo.

1.3. Objetivos y metodología

El **objetivo principal** del proyecto es desarrollar un sintetizador de audio utilizando el M5Stack Core 2, integrando múltiples osciladores, efectos de audio como *reverb* y *delay*, filtros y una interfaz de usuario intuitiva.

Entre los objetivos secundarios se incluyen

- Diseñar y programar osciladores para generar diversas formas de onda.
- Implementar efectos de audio, desarrollar filtros de audio y configurar una envolvente ADSR (*Attack, Decay, Sustain, Release*) para ajustar las características sonoras de manera precisa.
- Desarrollar una interfaz de usuario en la pantalla táctil del M5Stack Core 2 para controlar y ajustar parámetros en tiempo real.
- Integrar componentes físicos como botones y *encoders* para el control físico del sintetizador.

La metodología del proyecto está estructurada en varias fases. Inicialmente, se realizará una revisión de la teoría relacionada con sintetizadores de audio, osciladores, efectos de audio y filtros, lo cual proporcionará una base teórica para la planificación del diseño y desarrollo del sintetizador. Posteriormente, se definirán las funcionalidades del sintetizador, especificando los componentes necesarios y su integración con el M5Stack Core 2. Se procederá con el diseño detallado del *hardware*, esquematizando el circuito en una protoboard y seleccionando los componentes electrónicos adecuados. En paralelo, se desarrollará el *software* necesario para el funcionamiento del sintetizador, que incluye la programación de los osciladores para la generación de señales de audio, la implementación de efectos, y la creación de filtros y envolvente para modular las señales de salida. La interfaz de usuario será diseñada para ofrecer una experiencia intuitiva y funcional, permitiendo ajustes en tiempo real de los parámetros del sintetizador. La fase de integración consistirá en unificar todos los módulos de *software* y *hardware*, asegurando su correcto funcionamiento conjunto y realizando pruebas para verificar la funcionalidad y rendimiento del sintetizador, ajustando y optimizando el sistema según sea necesario.

1.4. Estructura de la memoria

La memoria del proyecto está organizada en los siguientes capítulos, cada uno de los cuales aborda aspectos específicos del desarrollo del sintetizador:

El primer capítulo, titulado **Introducción**, proporciona una visión general del proyecto, explicando la justificación, objetivos, metodología, alcances y limitaciones.

El segundo capítulo, **Marco teórico**, aborda los fundamentos teóricos necesarios para el desarrollo del sintetizador.

El tercer capítulo, **Diseño del sistema**, describe detalladamente el diseño del sistema del sintetizador, incluyendo un diagrama de bloques y la integración de componentes de *hardware* y *software*.

En el cuarto capítulo, **Implementación del *hardware***, detalla las especificaciones técnicas del *hardware* utilizado, y el proceso de conexión de los componentes adicionales.

El quinto capítulo, **Desarrollo del *software***, explica el desarrollo del *software* necesario, las herramientas y entornos de desarrollo utilizados, y la implementación de los módulos de *software*.

El sexto capítulo, **Pruebas y resultados**, describe la metodología de pruebas, las pruebas realizadas, y presenta los resultados obtenidos.

El séptimo capítulo, **Conclusiones**, resume los logros y desafíos del proyecto, reflexiona sobre las limitaciones, y propone posibles mejoras y trabajos futuros.

El octavo capítulo, **Bibliografía**, presenta una lista completa de todas las fuentes y documentos consultados.

El noveno capítulo, **Anexos,** contiene información adicional relevante para el proyecto, como diagramas detallados y código fuente completo.

2. Marco teórico

En este capítulo se abordan los fundamentos teóricos necesarios para comprender el proyecto. Se explican conceptos clave relacionados con los componentes y técnicas utilizadas en el diseño del sintetizador.

2.1. Principios de síntesis de sonido

La síntesis de audio es una técnica fundamental en la creación de sonidos y música electrónica, que permite la generación y manipulación de señales sonoras mediante dispositivos electrónicos y *software*. A continuación, se exploran los principios básicos de la síntesis de audio y su historia y evolución a través del tiempo.

2.1.1. Envolvente ADSR

Las envolventes controlan la evolución temporal de parámetros del sonido, típicamente la amplitud, y están definidas por fases. La envolvente ADSR (ver en la Figura 1) es la más común y comprende[1]:

- ***Attack*** (ataque): tiempo para alcanzar el nivel máximo desde cero.
- ***Decay*** (decaimiento): tiempo para descender del nivel máximo al nivel de sostenimiento.
- ***Sustain*** (sostenimiento): nivel mantenido mientras la nota se mantiene.
- ***Release*** (liberación): tiempo para descender desde el nivel de sostenimiento hasta cero una vez que se libera la nota.

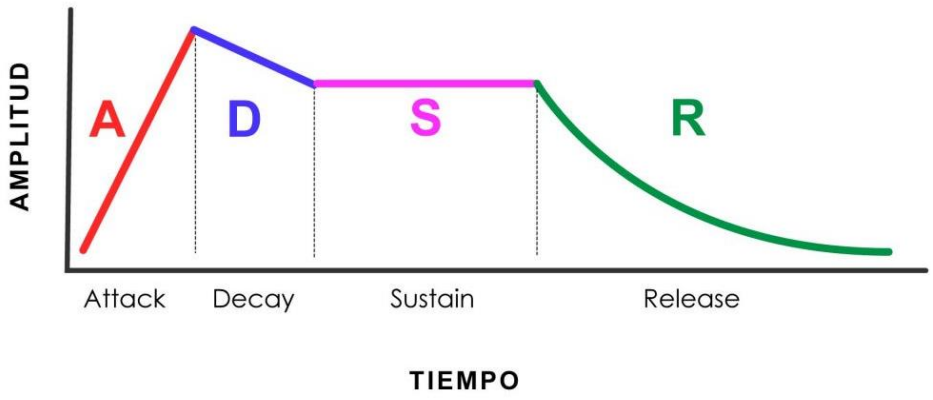

Figura 1: esquema de la envolvente de una señal. Extraído de https://djexpressions.net/que-es-el-ADSR-y-como-podemos-aprovecharlo/

[1] ADSR explained (sound design basics). (2022, November 15). Stickz.

2.1.2. Síntesis

La **síntesis substractiva** es una técnica popular que parte de una señal rica en armónicos (generada por osciladores) y utiliza filtros para restar frecuencias no deseadas, esculpiendo así el sonido final. Se usa mucho en sintetizadores analógicos y virtuales.[2]

La **síntesis aditiva** construye sonidos complejos sumando señales de onda simples (senoides) de diferentes frecuencias y amplitudes. Este método ofrece un control preciso sobre el espectro armónico del sonido, aunque puede ser computacionalmente intensivo.[3]

La **síntesis granular** descompone el sonido en pequeños fragmentos llamados "granos". Estos granos se pueden manipular y recombinar para crear texturas sonoras nuevas y complejas. Es especialmente efectiva para producir efectos espaciales y timbres cambiantes.[4]

La **síntesis de modelado físico** utiliza modelos matemáticos para simular las propiedades físicas de instrumentos acústicos. Este método puede generar sonidos muy realistas y dinámicos, imitando el comportamiento de cuerdas, membranas, y otros materiales resonantes.[5]

Los principios de la síntesis de audio abarcan una variedad de técnicas que permiten la creación y manipulación de sonidos desde cero. Cada método tiene sus propias características y aplicaciones, ofreciendo un amplio campo de posibilidades creativas para músicos y diseñadores de sonido. La comprensión de estos principios es fundamental para desarrollar y manipular sintetizadores de manera efectiva.

2.1.3. Tipos de osciladores

Los osciladores son el núcleo de cualquier sintetizador, responsables de generar las señales de sonido básicas que luego serán moldeadas y manipuladas por otros componentes como filtros, envolventes y efectos. Un oscilador produce una onda repetitiva a una determinada frecuencia, creando las tonalidades y timbres fundamentales utilizados en la síntesis de sonido.

Existen varios tipos de osciladores utilizados en los sintetizadores, cada uno con características únicas, como se puede observar en la Figura 2. Los **osciladores de onda sinusoidal** generan una onda pura y suave que contiene solo la frecuencia fundamental,

---

[2] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

[3] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

[4] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

[5] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

sin armónicos adicionales. Este tipo de oscilador es ideal para crear sonidos claros y simples, como los tonos puros de un generador de frecuencias.[6]

Los **osciladores de onda triangular** y **diente de sierra** añaden armónicos al sonido, creando ondas más ricas y complejas. La onda triangular tiene un sonido más suave y es ideal para replicar instrumentos acústicos suaves como flautas. La onda diente de sierra, por otro lado, contiene una amplia gama de armónicos, lo que la hace adecuada para crear sonidos brillantes y agresivos, como los utilizados en el bajo y los leads de la música electrónica.[7]

Los **osciladores de onda cuadrada** generan una onda que alterna abruptamente entre dos niveles, produciendo un sonido rico en armónicos impares. Este tipo de oscilador es conocido por su uso en los sintetizadores de los años 80 y es ideal para crear sonidos de bajo y efectos percusivos.[8]

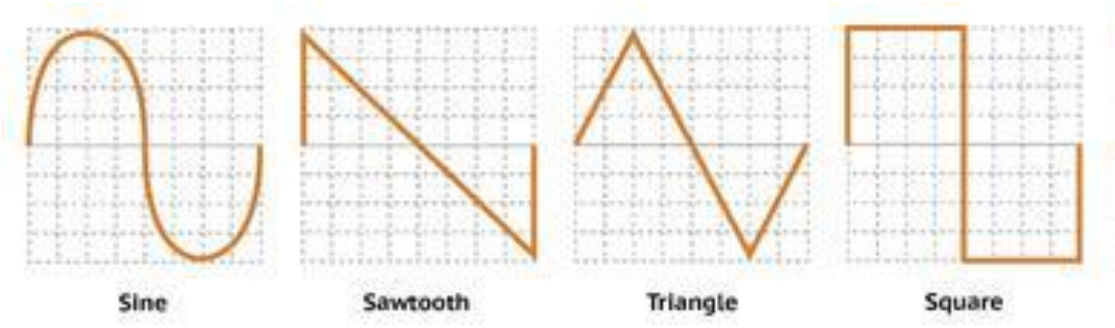

Figura 2: representación de los tipos de señales: sinusoidal, diente de sierra, triangular y cuadrada. Extraído de https://holawave.store/blogs/noticias/que-es-un-oscilador

En los sintetizadores modernos, los osciladores pueden generar múltiples formas de onda y permitir su mezcla para crear timbres más complejos. Los **osciladores controlados por voltaje** (VCO) y los **osciladores digitales** (DCO) son dos tipos comunes. Los VCOs, utilizados en sintetizadores analógicos, varían su frecuencia en respuesta a una señal de control de voltaje, mientras que los DCOs, más estables y precisos, utilizan circuitos digitales para mantener la afinación.[9]

Los osciladores también pueden sincronizarse entre sí para crear efectos de sincronización (SYNC), donde un oscilador reinicia la fase del otro, resultando en sonidos drásticos y crean timbres más agresivos y ricos en armónicos.

---

[6] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

[7] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

[8] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

[9] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

2.1.4. Efectos de audio

Los efectos de audio son técnicas y procesos utilizados para modificar o mejorar las características de una señal de sonido, ya sea en aplicaciones musicales, cinematográficas o de producción de contenido multimedia. Estos efectos juegan un papel crucial en la creación de sonidos ricos y variados, proporcionando profundidad, textura y dinamismo a las grabaciones. Entre los efectos más comunes se encuentran la *reverb*, el *delay*, el *chorus* y la distorsión.

La ***reverb*** simula el efecto del sonido reflejándose en superficies de un espacio, creando una sensación de profundidad y ambiente (ver en la Figura 3). Este efecto es fundamental para dar vida a grabaciones secas o para situar el audio en un contexto espacial específico[10]. El ***delay***, por otro lado, añade repeticiones del sonido original en intervalos específicos, creando ecos que pueden variar desde un simple retardo hasta complejas cadenas de repeticiones rítmicas.[11] (ver en la Figura 4)

El ***chorus*** es un efecto que duplica la señal de audio y le aplica ligeras variaciones en el tiempo y el tono, dando la impresión de que varias fuentes están reproduciendo el mismo sonido simultáneamente (ver en la Figura 5). Este efecto se suele usar en voces e instrumentos para enriquecer y ensanchar el sonido[12]. La **distorsión**, ampliamente utilizada en la música rock y metal, altera la forma de onda del sonido, añadiendo armónicos y aumentando el contenido de frecuencias altas, lo que resulta en un sonido más agresivo y poderoso.

Otros efectos importantes incluyen el ***flanger*** y el ***phaser***, que modulan la fase de la señal de audio para crear barridos de frecuencias, añadiendo movimiento y complejidad al sonido.[13]

El **ecualizador** (EQ) permite ajustar las diferentes bandas de frecuencia de una señal de audio, mejorando el balance tonal y destacando o atenuando ciertas frecuencias. Finalmente, la **compresión** controla la dinámica del audio, reduciendo la brecha entre los sonidos más fuertes y los más suaves, lo que resulta en una mezcla más coherente y profesional.[14]

[10] Reveillac, J.-M.(2018). Musical sound effects: analog and digital sound processing. ISTE.
[11] Reveillac, J.-M.(2018). Musical sound effects: analog and digital sound processing. ISTE.
[12] Reveillac, J.-M.(2018). Musical sound effects: analog and digital sound processing. ISTE.
[13] Reveillac, J.-M.(2018). Musical sound effects: analog and digital sound processing. ISTE.
[14] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

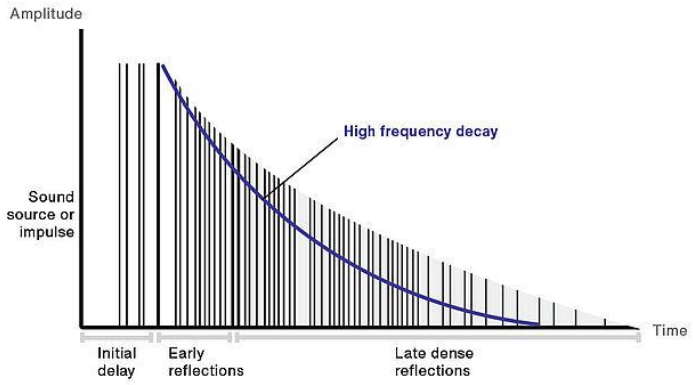

Figura 3: respuesta en el tiempo de una señal con reverb. Extraído de https://www.lewitt-audio.com/blog/what-is-reverb Extraído de https://www.lewitt-audio.com/blog/what-is-reverb

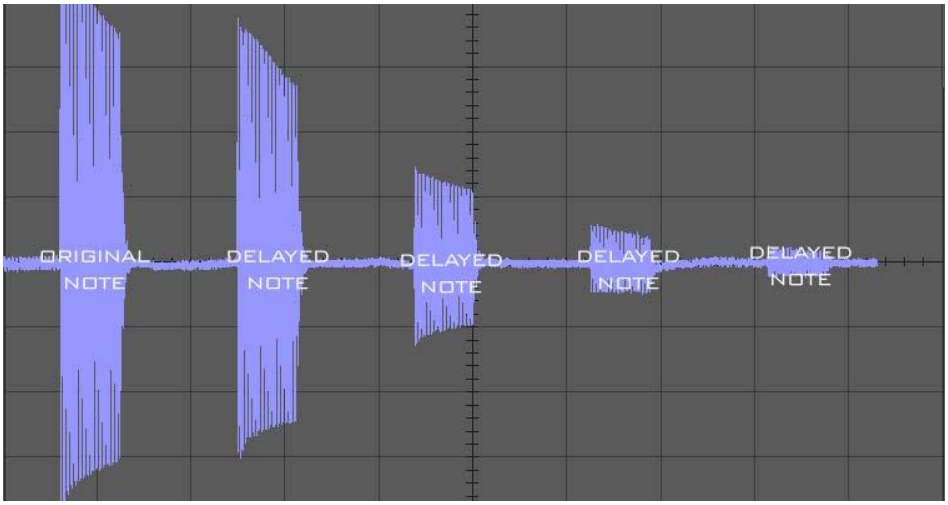

Figura 4: representación de la aplicación de delay sobre una señal. Extraído de https://www.txirula.com/blog/que-es-delay.html Extraído de https://www.txirula.com/blog/que-es-delay.html

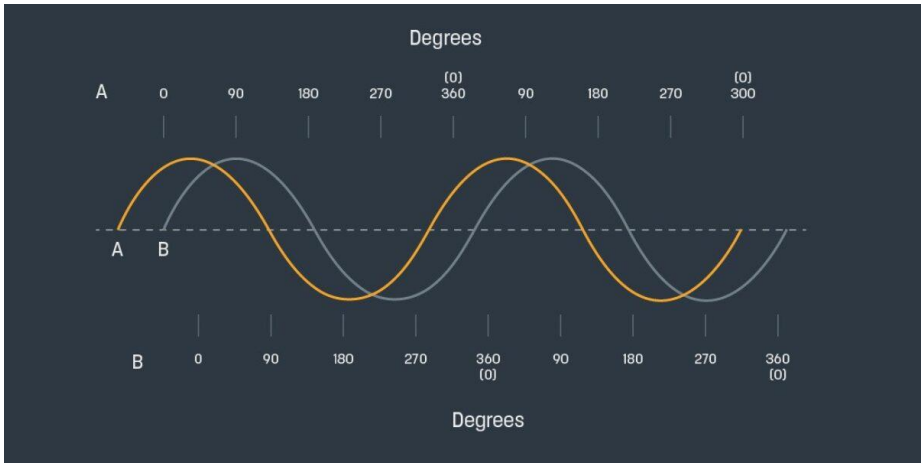

Figura 5: representación del efecto de chorus. Extraído de https://blog.native-instruments.com/chorus-effect/ Extraído de https://blog.native-instruments.com/chorus-effect/

2.1.5. Filtros de bandas

Los filtros de audio son componentes esenciales en los sintetizadores, utilizados para esculpir y transformar las señales de sonido generadas por los osciladores. Permiten a los músicos y diseñadores de sonido crear timbres únicos y complejos, así como eliminar frecuencias no deseadas, otorgando carácter y personalidad a los sonidos, como en la mostrada en la Figura 7.

Existen varios tipos de filtros comúnmente utilizados en sintetizadores, cada uno con aplicaciones específicas. Los **filtros de paso bajo (*low-pass filters*, LPF)** permiten que las frecuencias por debajo de un cierto punto de corte pasen sin atenuación mientras reducen las frecuencias superiores a ese punto. Este tipo de filtro es fundamental para suavizar sonidos, crear bajos profundos y eliminar armónicos agudos no deseados. Es especialmente útil en la síntesis sustractiva, donde un sonido complejo y rico en armónicos se esculpe para obtener el timbre deseado.[15]

Por otro lado, los **filtros de paso alto (*high-pass filters*, HPF)** permiten el paso de las frecuencias altas mientras atenúan las bajas. Son utilizados para eliminar el exceso de graves y dar claridad a los sonidos. Esto es útil para sonidos de percusión o efectos que requieren una definición más clara en el rango de frecuencias altas.[16]

Los **filtros de paso banda (*band-pass filters*, BPF)** permiten el paso de un rango específico de frecuencias mientras atenúan las frecuencias fuera de este rango. En los sintetizadores, los filtros de paso banda son útiles para aislar y resaltar ciertas frecuencias, proporcionando un control preciso sobre el timbre y la textura del sonido. Este tipo de filtro es frecuentemente empleado en la creación de efectos de barrido y para enfocar sonidos en una banda de frecuencia específica.[17]

Además, los **filtros de banda eliminada (*notch filters*)**, que atenúan un rango estrecho de frecuencias, son efectivos para eliminar frecuencias indeseadas específicas sin afectar el resto del espectro sonoro. En la síntesis de sonido, estos filtros pueden ser utilizados para eliminar resonancias no deseadas o para crear efectos de *phasing*.[18]

Los **filtros resonantes** añaden una resonancia o realce en la frecuencia de corte, creando un pico pronunciado que puede dar un carácter distintivo al sonido. Al aumentar la

[15] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

[16] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

[17] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

[18] Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

resonancia, los filtros pueden producir un efecto de barrido más pronunciado, muy apreciado en la música electrónica para crear sonidos dinámicos y enérgicos.

En los sintetizadores, los filtros suelen ser modulables, permitiendo cambios dinámicos en tiempo real mediante envolventes, **LFOs** (*Low-Frequency Oscillators*) y otros moduladores. Esto agrega movimiento y vida al sonido, permitiendo la creación de texturas sonoras desarrolladas y complejas.

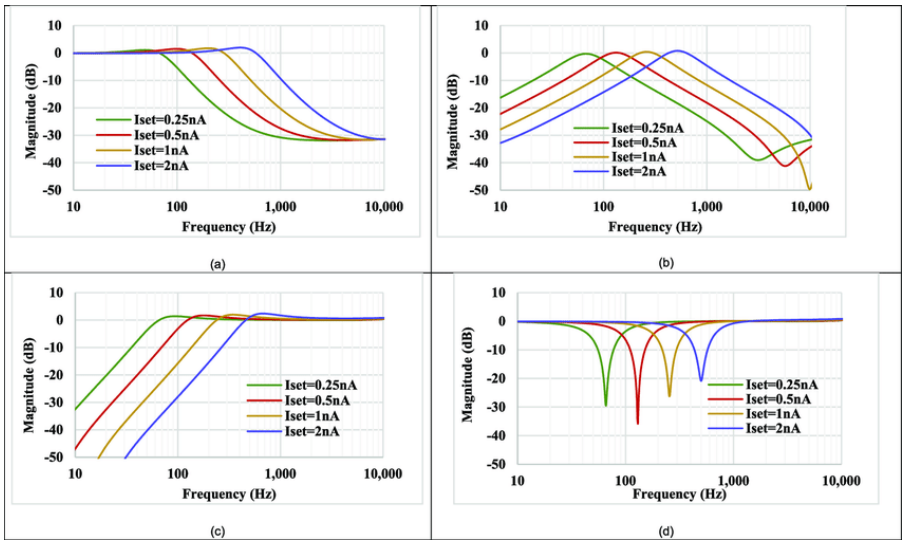

Figura 6: representación diferentes tipos de filtros (LPF, BPF, HPF, Notch). Extraído de https://www.researchgate.net/figure/Magnitude-characteristics-of-the-LPF-a-BPF-b-HPF-c-and-BSF-d-with-different_fig5_360036832 Extraído de https://www.researchgate.net/figure/Magnitude-characteristics-of-the-LPF-a-BPF-b-HPF-c-and-BSF-d-with-different_fig5_360036832

3. Diseño del sistema

Este capítulo se centra en la descripción detallada del diseño del sistema del sintetizador y sus requisitos técnicos. Se presenta un diagrama de bloques que ilustra la estructura general del sintetizador y se describen en detalle los componentes tanto de *hardware* como de *software*. Se explica cómo se integran estos componentes para formar un sistema funcional.

3.1. Diagrama de bloques del sintetizador

A continuación, se presenta un diagrama de bloques del sintetizador en la Figura 7:

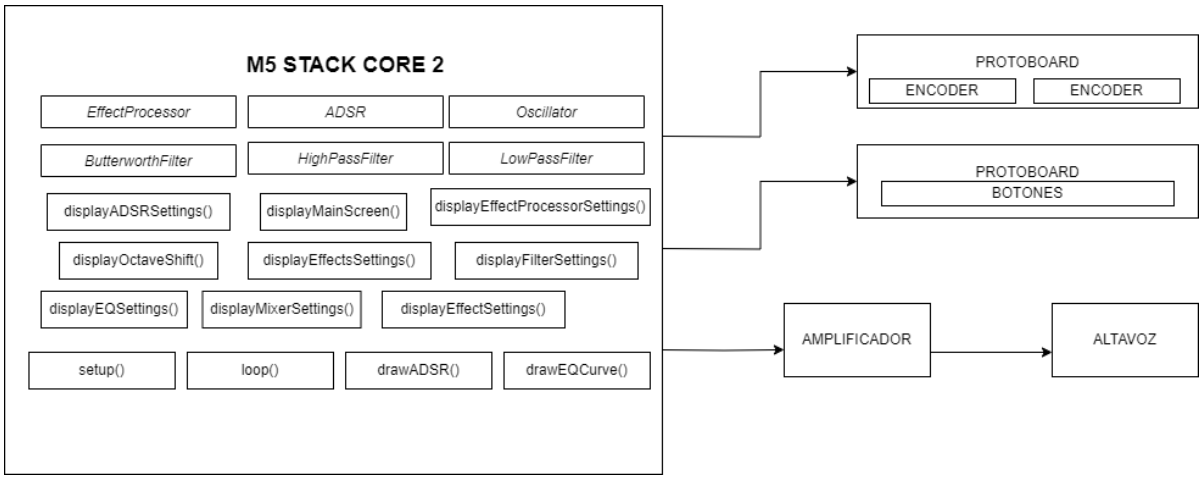

Figura 7: Representación del diagrama de bloques del sintetizador..

3.2. Descripción general del sintetizador

El corazón del sintetizador lo forma el microcontrolador M5Stack Core 2dentro, con el que se implementarán los módulos básicos para generar y procesar señales. Externamente, se incluyen dos placas de prototipado electrónico en las que se incluirán los *encoders* utilizados para el control de la frecuencia de corte de los filtros o de los parámetros de los efectos, por ejemplo. En la segunda placa se introducen varios pulsadores (botones) para permitir la generación de notas musicales simulando la disposición de un piano, y generando las frecuencias correspondientes a estas. Por último, se ha añadido un amplificador (**TPA2005d1**) conectado a un altavoz que permitirá reproducir los sonidos generados por el sintetizador.

A continuación, se mostrarán las especificaciones del sistema

1. **Osciladores:** cuatro osciladores que generan señales de diferentes formas de onda. Cada oscilador tiene controles para la frecuencia, la amplitud y la forma de onda.

Formas de Onda	Frecuencia	Amplitud
Senoidal	65,4Hz-15.804Hz	Fija
Triangular	65,4Hz-15.804Hz	Fija
Cuadrada	65,4Hz-15.804Hz	Fija
Diente de sierra	65,4Hz-15.804Hz	Fija

Tabla 1: rango de parámetros de los osciladores

2. **Efectos de audio:** incluye *reverb*, *delay* y *chorus*, cada uno con parámetros ajustables para personalizar el efecto aplicado a la señal de audio.

Efecto	Parámetros	Rango
Reverb	DecayFactor y DelayLength	0.0-1.0 y 10-1000ms
Delay	Feedback y Length	0.0-0.9 y 10-2000 ms
Chorus	Depth, Rate, Phase (Fijo)	0.0-1.0 y 0.1-5.0 Hz

Tabla 2: rango de parámetros de los efectos

3. **Envolvente ADSR:** controla la dinámica de las señales de audio, permitiendo ajustar los tiempos de ataque, decaimiento, nivel de sostenido y liberación.

Parámetros	Rango
Ataque	0.0-4.0 s
Decaimiento	Fijo
Sostenimiento	0.0-1.0
Liberación	Fijo

Tabla 3: rango de parámetros de las envolvente ADSR

4. **Filtros:** incluirá una parte de ecualización donde se podrá aplicar un filtro paso bajo y un filtro paso alto y configurar su frecuencia de corte.

Filtro	Parámetros	Rango
F. Paso Alto	Frec. de corte	20-400 Hz
	Resonancia	Fija
F. Paso Bajo	Frec. de corte	2,5- 20 kHz
	Resonancia	Fija

Tabla 4: rango de parámetros de los filtros

5. **Interfaz de usuario:** implementada en la pantalla táctil del M5Stack Core2, permite visualizar y modificar los parámetros de los osciladores y efectos en tiempo real.

6. **Botones pulsadores y *encoders*:** proporcionan una forma adicional de interactuar con el sintetizador, permitiendo cambiar parámetros y formas de onda de manera física.

3.3. Interfaz del usuario (UI) en M5Stack Core 2

La interfaz de usuario del controlador M5Stack Core 2 está diseñada para proporcionar un acceso intuitivo y eficiente a las diversas funcionalidades del dispositivo. La implementación se centra en el uso de la pantalla táctil capacitiva, los botones virtuales y los *encoders* rotatorios, permitiendo a los usuarios interactuar con el dispositivo de manera sencilla y efectiva.

Pantalla principal

La pantalla principal actúa como el centro de control desde donde se puede acceder a las configuraciones de los osciladores, efectos y el mezclador (ver en la Figura 8). Utiliza botones virtuales y componentes gráficos para la navegación. Los botones virtuales están organizados en secciones claramente definidas para los osciladores, efectos y el mezclador, facilitando la navegación y el acceso rápido a las configuraciones deseadas.

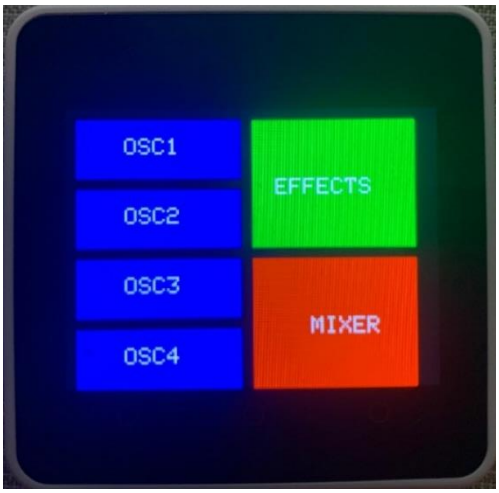

Figura 8: Imagen real de la Pantalla principal.

Configuración de osciladores

Cada oscilador tiene su propia pantalla de configuración, accesible desde la pantalla principal. Los usuarios pueden ajustar la frecuencia, la amplitud, el tipo de onda y el cambio de octava utilizando los *encoders* y la pantalla táctil, como se ve en la Figura 9. La pantalla de configuración de osciladores muestra la frecuencia actual, la amplitud y el tipo de onda. También incluye un botón para encender o apagar el oscilador y una visualización del desplazamiento de octava.

Figura 9: Imagen real de la Pantalla de configuración del oscilador 1.

Configuración de efectos

La configuración de efectos permite ajustar parámetros específicos y activar/desactivar los efectos de *reverb*, *delay* y *chorus*, mostrado en la Figura 10. Los *encoders* se utilizan para ajustar los parámetros mientras que la pantalla táctil permite la navegación y selección. La pantalla de configuración de efectos muestra los parámetros ajustables del efecto seleccionado y permite encender o apagar el efecto mediante un botón virtual.

- ***Reverb***: los parámetros incluyen el *pre-delay* y el tiempo de reverberación.
- ***Delay***: los parámetros incluyen la retroalimentación y el tiempo de *delay*.
- ***Chorus***: los parámetros incluyen el retardo y el pitch.

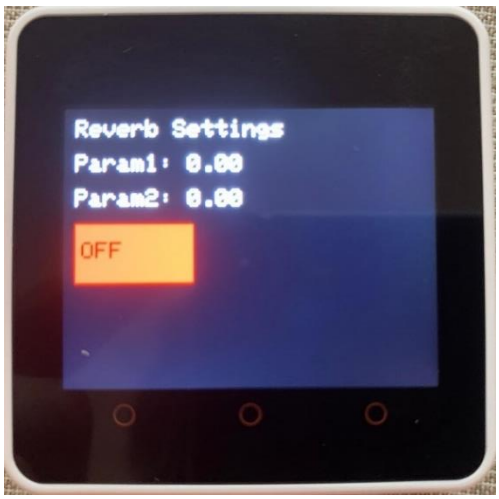

Figura 10: Imagen real de la Pantalla de configuración del reverb.

Pantalla del mezclador

El mezclador permite acceder a la configuración de los filtros paso bajo y paso alto, así como los parámetros de la envolvente ADSR. También se encuentran los botones de encendido/apagado de dichas componentes, como se observa en la Figura 11).

Figura 11: Imagen real de la Pantalla del mezclador.

Configuración de la envolvente ADSR

La configuración de la envolvente ADSR incluye parámetros para el ataque, decaimiento, sostenido y liberación mostrada en la Figura 12. La pantalla de configuración de ADSR muestra estos parámetros en forma numérica y de curva y permite ajustarlos utilizando los *encoders*.

- ***Attack***: controla el tiempo que tarda la señal en alcanzar su nivel máximo.
- ***Decay***: controla el tiempo que tarda la señal en disminuir a su nivel sostenido.
- ***Sustain***: controla el nivel sostenido durante la duración de la nota.
- ***Release***: controla el tiempo que tarda la señal en disminuir a cero después de que la nota se ha liberado.

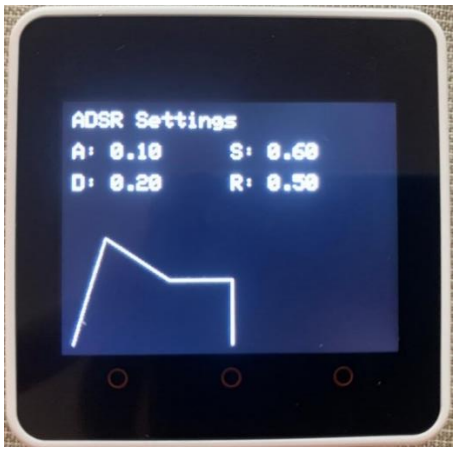

Figura 12: Imagen real de la Pantalla de configuración del ADSR.

Configuración de la ecualización (EQ)

La configuración de la EQ permite ajustar los filtros paso bajo y paso alto. La pantalla de configuración de EQ muestra la curva de respuesta en frecuencia de los filtros, proporcionando una representación visual clara de cómo se está afectando la señal de

audio, como se ve en la Figura 13. Los *encoders* se utilizan para ajustar las frecuencias de corte de los filtros, y la pantalla táctil permite activar o desactivar los filtros.

Figura 13: Imagen real de la Pantalla de configuración de los filtros.

4. Implementación del *hardware*

En este capítulo se describen las especificaciones técnicas del *hardware* utilizado y se detalla el proceso de conexión de los componentes adicionales necesarios para el funcionamiento del sintetizador. Se incluyen diagramas y explicaciones sobre el montaje y la configuración del *hardware*.

4.1. Especificaciones técnicas del M5Stack Core 2

El M5Stack Core 2 es una avanzada plataforma de desarrollo basada en el potente microcontrolador ESP32, diseñada para facilitar la creación de proyectos interactivos y aplicaciones inteligentes. Su diseño modular y expansible, junto con muchas funcionalidades integradas, lo convierte en una herramienta ideal para principiantes y desarrolladores experimentados. En las siguientes imágenes podemos observar la parte delantera, en la Figura 14, y su parte trasera, Figura 15.

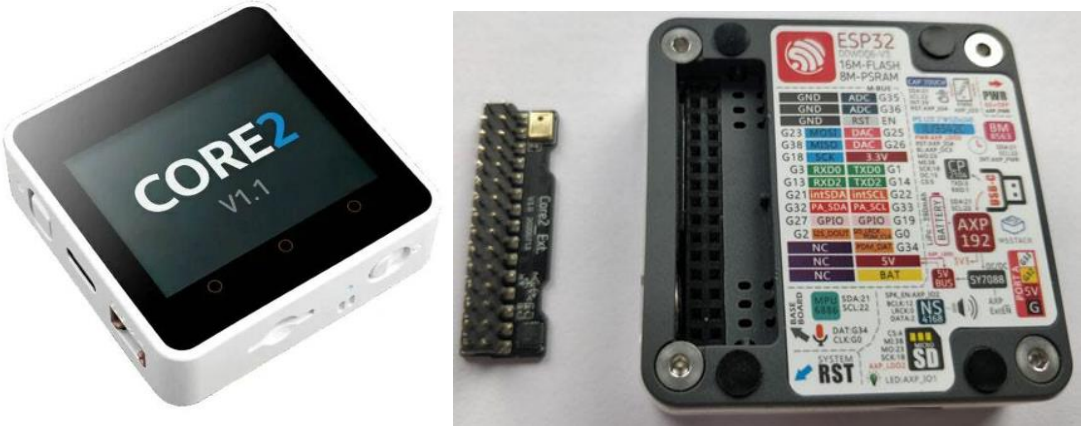

Figura 14: fotografía de la parte frontal del M5Stack Core 2. Extraído de https://core-electronics.com.au/m5stack-core2-esp32-iot-development-kit-v11.html

Figura 15: fotografía de la parte trasera del M5Stack Core 2. Extraído de https://circuitdigest.com/review/m5stack-core2-an-esp32-based-iot-development-kit

Destaca por su microcontrolador ESP32, que cuenta con una CPU dual-core Tensilica lx6 que opera a una frecuencia de hasta 240 MHz, proporcionando un rendimiento elevado para tareas complejas. Además, incluye 520 KB de SRAM y 16 MB de memoria flash, ofreciendo suficiente espacio para aplicaciones avanzadas.

Adicionalmente, este dispositivo incorpora una pantalla táctil capacitiva de 2 pulgadas con una resolución de 320 x 240 píxeles, facilitando la creación de interfaces de usuario intuitivas y dinámicas. Los sensores integrados incluyen un acelerómetro y giroscopio (MPU6886) para la detección precisa de movimientos y orientación, un micrófono (SPM1423) para captura de audio y reconocimiento de voz, y un altavoz (1W-092) para salida de audio clara y nítida. También cuenta con un reloj en tiempo real (BM8563), un motor de vibración para alertas táctiles, un LED indicador azul, un amplificador de

potencia I2S (NS4168) para mejorar la calidad del audio, y un PMU (AXP192) para la gestión eficiente de la energía.

El M5Stack Core 2 ofrece diversas interfaces de expansión, incluyendo un conector Grove, 38 pines GPIO, interfaces I2C y SPI, un puerto USB tipo C para alimentación y comunicación, y una ranura para tarjeta microSD para expansión de almacenamiento. Su batería de iones de litio de 390 mAh da autonomía razonable para aplicaciones móviles y portátiles, y puede alimentarse con un voltaje de entrada de 5V @ 500mA.

4.2. Conexión de componentes

Para completar las funcionalidades del sintetizador se incluye en el circuito los siguientes componentes para que su funcionamiento sea correcto:

- **Cables tipo Dupont macho – macho (ver en la Figura 16):** estos son cables de conexión flexibles con conectores macho en ambos extremos. Se utilizan comúnmente para realizar conexiones rápidas y temporales entre componentes electrónicos en una protoboard.

Figura 16: imagen de los cables Dupont macho-macho. Extraído de https://tienda.bricogeek.com/cables/1578-cables-dupont-macho-macho-40-cm-40-unidades.html

- **Pack pulsadores de colores (12) (ver en la Figura 17):** conjunto de pulsadores (botones) de diferentes colores. Estos pulsadores son ideales para usarse como botones de entrada en proyectos electrónicos.

Figura 17imagen del pack de pulsadores. Extraído de https://tienda.bricogeek.com/home/508-pack-pulsadores-de-colores-15-unidades.html

- **Potenciómetro *encoder* de rotación (2) (ver en la Figura 18):** conjunto de pulsadores (botones) de diferentes colores. Estos pulsadores son ideales para usarse como botones de entrada en proyectos electrónicos.

Figura 18: imagen del potenciómetro encoder de rotación. Extraído de https://tienda.bricogeek.com/componentes/197-potenciometro-encoder-de-rotacion.html

- **Placa de prototipo 16x5cm (2) (ver en la Figura 19):** con 830 puntos de conexión, utilizada para el montaje y prueba de circuitos electrónicos sin necesidad de soldadura.

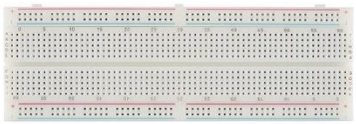

Figura 19: imagen de la placa de prototipo. Extraído de https://tienda.bricogeek.com/herramientas-de-prototipado/239-placa-de-prototipo-16x5cm.html

- **Altavoz (Figura 20):** un altavoz de 3w con caja de protección. Diseñado para proporcionar salida de audio en proyectos electrónicos.

Figura 20: imagen del altavoz. Extraído de https://tienda.bricogeek.com/varios/938-altavoz-con-caja-3w.html

- **Amplificador de audio mono – TPA2005d1 (ver en la Figura 21):** aumenta la señal de audio para poder ser escuchada en altavoces.

Figura 21:imagen del amplificador de audio TPA2005d1. Extraído de https://tienda.bricogeek.com/componentes/675-amplificador-de-audio-mono-tpa2005d1.html

A continuación, se presenta tanto el esquema electrónico de las conexiones del circuito, en la Figura 23, como una imagen del montaje del sistema, en la Figura 24):

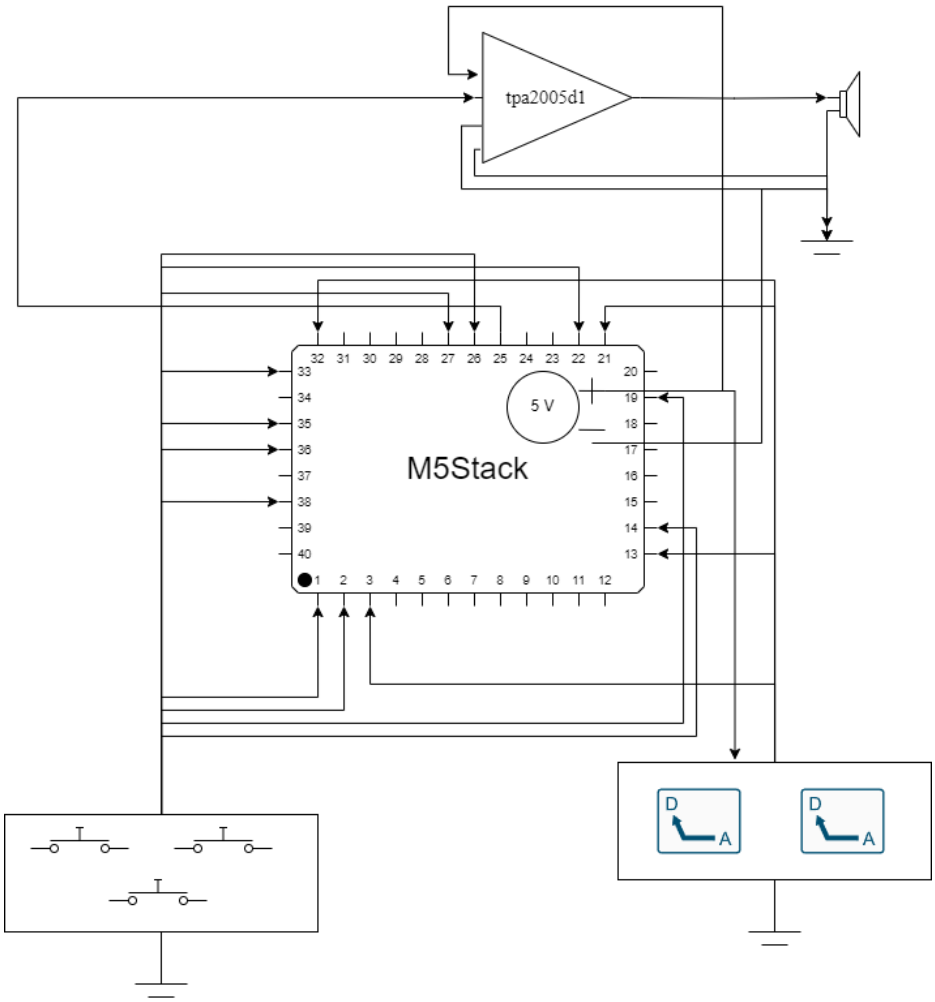

Figura 22: esquema electrónico de las conexiones del circuito.

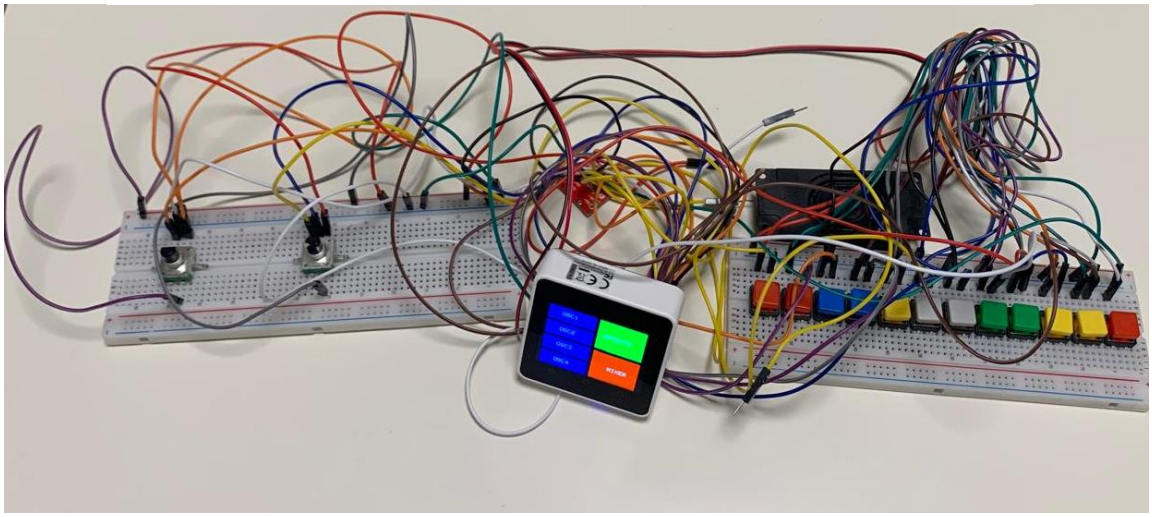

Figura 23: imagen del montaje completo del sistema.

5. Desarrollo del *software*

Este capítulo detalla el desarrollo del *software* necesario para el funcionamiento del sintetizador. Se describe la implementación de los diferentes módulos de *software* que controlan los osciladores, efectos de audio, envolventes y filtros.

5.1. Entorno de desarrollo

El **Arduino *Integrated Development Environment*** (IDE) es una plataforma de desarrollo diseñada para programar y cargar código en las placas de microcontroladores Arduino. Desde su lanzamiento, ha democratizado el acceso a la programación y la electrónica, permitiendo a estudiantes, aficionados y profesionales crear una amplia variedad de proyectos interactivos de manera accesible y eficiente.

Interfaz y funcionalidades (ver en la Figura 24)

Editor de código: área de texto para escribir y editar código.

Consola de mensajes: muestra información sobre el estado de la compilación y la carga del código, así como mensajes de error y advertencias.

Barra de herramientas: incluye botones para verificar (compilar) el código, cargarlo en la placa, abrir y guardar archivos, y acceder a la configuración.

Bibliotecas y ejemplos

Cuenta con una extensa colección de bibliotecas que simplifican tareas complejas como controlar sensores, motores, pantallas y módulos de comunicación. Viene preinstalado con muchas bibliotecas estándar y permite a los usuarios instalar bibliotecas adicionales a través del gestor de bibliotecas. Además, incluye numerosos ejemplos de código organizados por categorías y relacionados con diversas bibliotecas. Por ejemplo, para este proyecto se han utilizado las siguientes bibliotecas *M5Core2.h, vector* y *math.h*

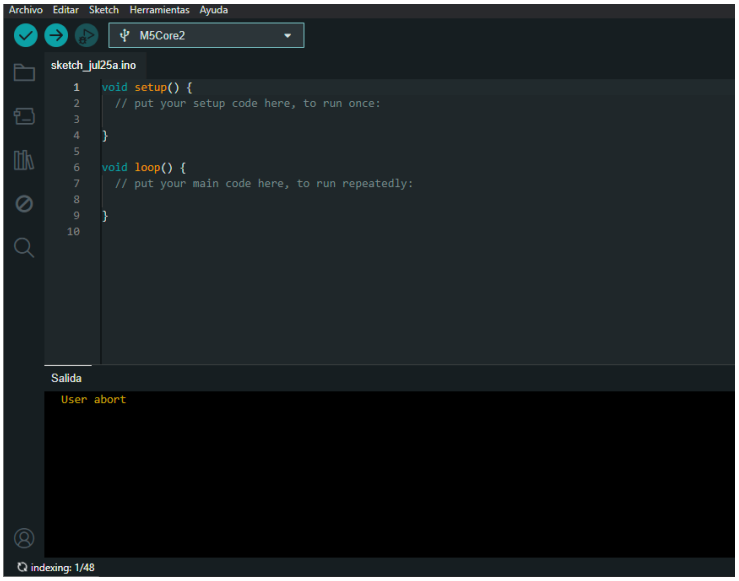

Figura 24: captura de pantalla de la interfaz del editor de código Arduino IDE.

5.2. Implementación de los osciladores

En la implementación de los osciladores para el controlador, se han utilizado varias técnicas y elementos para asegurar una operación eficiente y una experiencia de usuario intuitiva. A continuación, se detalla la estructura y funcionalidad del código utilizado para gestionar los osciladores, así como la forma en que estos interactúan con otros componentes del controlador.

1. Definición de la clase *Oscillator*

La clase *Oscillator* , expuesta en Figura 25, se define para manejar los parámetros y la generación de las formas de onda. Esta clase incluye los siguientes atributos y métodos:

Atributos:

- *baseFrequency*: La frecuencia base del oscilador.
- *amplitude*: La amplitud de la onda generada.
- *waveType*: El tipo de onda (SINE, SQUARE, TRIANGLE, SAWTOOTH). Este parámetro se controla a través del *encoder* conectado al GPIO XX.
- *phase*: La fase actual de la onda.
- *isOn*: Indica si el oscilador está activado.
- *octaveShift*: Permite ajustar la frecuencia en octavas. Este parámetro se controla a través del *encoder* conectado al GPIO XX.

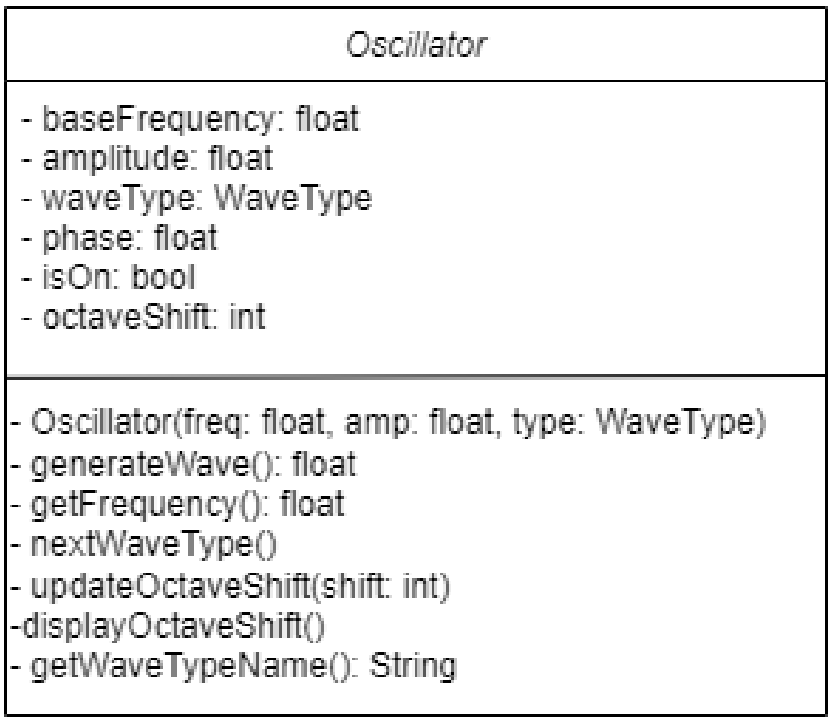

Figura 25: representación de la clase Oscillator.

Métodos:

- *Oscillator(float freq, float amp, WaveType type, bool on = true)*: Constructor que inicializa el oscilador con frecuencia, amplitud, tipo de onda y estado.
- *generateWave()*: Genera la onda correspondiente según el tipo y la frecuencia base.
- *getFrequency()*: Calcula y devuelve la frecuencia ajustada según el desplazamiento de octava.
- *nextWaveType()*: Cambia al siguiente tipo de onda.
- *updateOctaveShift(int shift)*: Actualiza el desplazamiento de octava.
- *getWaveTypeName()*: Devuelve el nombre del tipo de onda actual como una cadena de caracteres.

2. Generación y Aplicación de las Ondas

El método *generateWave()* (Figura 26) es el responsable de crear la onda de acuerdo con el tipo de onda seleccionado y la frecuencia base. La fase se incrementa en cada llamada para simular el paso del tiempo, y se ajusta la forma de onda según el tipo seleccionado.

```
float generateWave() {
    if (!isOn) return 0.0;
    float sample = 0.0;
    float increment = 2 * PI * getFrequency() / SAMPLE_RATE;
    phase += increment;
    if (phase > 2 * PI) phase -= 2 * PI;

    switch (waveType) {
        case SINE:
            sample = sin(phase);
            break;
        case SQUARE:
            sample = (sin(phase) >= 0) ? 1.0 : -1.0;
            break;
        case TRIANGLE:
            sample = asin(sin(phase)) * (2 / PI);
            break;
        case SAWTOOTH:
            sample = (2.0 / PI) * (phase - PI);
            break;
    }

    return sample * amplitude;
}
```

Figura 26: captura de pantalla del código de generateWave().

3. Integración en el Loop Principal

En el *loop* principal, se llama a *generateWave()* para cada oscilador activo y se suma su salida para crear la señal final. Esta señal luego se pasa a través de los filtros y efectos antes de ser enviada al DAC para su reproducción.

4. Interacción con los *Encoders* y la Pantalla Táctil

Los *encoders* y la pantalla táctil permiten a los usuarios ajustar en tiempo real los parámetros de los osciladores. Cada *encoder* está configurado para modificar un parámetro específico, como el tipo de onda o el desplazamiento de octava mediante la funcion *HandleOscillatorSettings()* (Figura 27). La pantalla táctil proporciona una interfaz intuitiva para seleccionar y configurar los osciladores.

Ejemplo de Ajuste con *Encoders*:

- *Encoder* 1: Ajusta el desplazamiento de octava.
- *Encoder* 2: Cambia el tipo de onda.

Cada vez que se detecta un cambio en el estado del *encoder*, se actualizan los parámetros del oscilador correspondiente y se refleja en la pantalla.

```
void handleOscillatorSettings(String oscLabel, Oscillator &osc) {
    displayOscillatorSettings(oscLabel, osc);
    while (true) {
        M5.update();

        // Leer el encoder1 para ajustar la OCTAVA
        int currentEncoderState1 = digitalRead(ENCODER1_CLK);
        if (currentEncoderState1 != lastEncoderState1) {
            int dtState = digitalRead(ENCODER1_DT);
            if (dtState == LOW && currentEncoderState1 == HIGH) {
                osc.updateOctaveShift(-1);
            } else if (dtState == HIGH && currentEncoderState1 == HIGH) {
                osc.updateOctaveShift(1);
            }
            displayOscillatorSettings(oscLabel, osc);
        }
        lastEncoderState1 = currentEncoderState1;

        // Leer el encoder2 para cambiar el tipo de onda
        int currentEncoderState2 = digitalRead(ENCODER2_CLK);
        if (currentEncoderState2 != lastEncoderState2) {
            int dtState = digitalRead(ENCODER2_DT);
            if (dtState == LOW && currentEncoderState2 == HIGH) {
                osc.nextWaveType();
            } else if (dtState == HIGH && currentEncoderState2 == HIGH) {
                osc.nextWaveType();
            }
            displayOscillatorSettings(oscLabel, osc);
        }
        lastEncoderState2 = currentEncoderState2;

        if (M5.BtnA.wasPressed()) {
            displayMainScreen();
            break;
        }
        if (M5.Touch.ispressed()) {
            TouchPoint_t innerPoint = M5.Touch.getPressPoint();
            int ix = innerPoint.x;
            int iy = innerPoint.y;
            if (ix > 10 && ix < 110 && iy > 130 && iy < 180) {
                osc.isOn = !osc.isOn;
                displayOscillatorSettings(oscLabel, osc);
            }
        }
    }
}
```

Figura 27: captura de pantalla del código de handleOscillatorSettings.

5. Visualización y Control

La visualización de los parámetros de los osciladores en la pantalla táctil del M5Stack Core2 es crucial para una interacción efectiva. Los usuarios pueden ver y ajustar en tiempo real la frecuencia, el tipo de onda y el desplazamiento de octava, proporcionando un control detallado sobre el sonido generado, mediante la función *displayOscillatorSettings()* (Figura 28).

```
void displayOscillatorSettings(String oscLabel, Oscillator &osc) {
    M5.Lcd.fillScreen(BLACK);
    M5.Lcd.setCursor(10, 10);
    M5.Lcd.print(oscLabel + " Settings");

    // Controles para frecuencia, amplitud y tipo de onda
    M5.Lcd.setCursor(10, 40);
    M5.Lcd.print("Freq: " + String(osc.getFrequency()));

    M5.Lcd.setCursor(10, 70);
    M5.Lcd.print("Amp: " + String(osc.amplitude));

    M5.Lcd.setCursor(10, 100);
    M5.Lcd.print("Wave: " + osc.getWaveTypeName());

    // Mostrar el desplazamiento de octava
    displayOctaveShift(osc);

    // Botón de encendido/apagado
    M5.Lcd.fillRect(10, 130, 100, 50, osc.isOn ? GREEN : RED);
    M5.Lcd.setCursor(15, 145);
    M5.Lcd.setTextColor(BLACK);
    M5.Lcd.print(osc.isOn ? "ON" : "OFF");
    M5.Lcd.setTextColor(WHITE);
}
```

Figura 28: captura de pantalla del código de displayOscillatorSettings.

5.3. Implementación de los efectos (*reverb*, *delay*, etc.)

La implementación de los efectos de *reverb*, *delay* y *chorus* en el controlador está diseñada para proporcionar un enriquecimiento y variabilidad del sonido generado por los osciladores. A continuación, se detalla la estructura y funcionalidad del código utilizado para gestionar estos efectos y su integración con otros componentes del controlador.

1. Definición de la clase *EffectProcessor*

La clase *EffectProcessor* expuesta en Figura 29, se define para manejar los parámetros y la aplicación de los efectos de *reverb*, *delay* y *chorus* a la señal de audio. Esta clase incluye los siguientes atributos y métodos:

- **Atributos:**
    - *isReverbOn*: estado del *reverb*, indica si el efecto está activado.

- *reverbDecayFactor*: factor de decaimiento del *reverb*. Este parámetro se controla a través del *encoder* conectado al GPIO XX.
- *reverbDelayLength*: longitud del retardo para el *reverb*. Este parámetro se controla a través del *encoder* conectado al GPIO XX.
- *reverbBuffer*: buffer circular para almacenar las muestras del *reverb*.
- *isDelayOn*: estado del *delay*, indica si el efecto está activado.
- *delayFeedback*: retroalimentación del *delay*. Este parámetro se controla a través del *encoder* conectado al GPIO XX.
- *delayLength*: longitud del retardo para el *delay*. Este parámetro se controla a través del *encoder* conectado al GPIO XX.
- *delayBuffer*: buffer circular para almacenar las muestras del *delay*.
- *isChorusOn*: estado del *chorus*, indica si el efecto está activado.
- *chorusDepth*: profundidad del *chorus*. Este parámetro se controla a través del *encoder* conectado al GPIO XX.
- *chorusRate*: frecuencia de modulación del *chorus*. Este parámetro se controla a través del *encoder* conectado al GPIO XX.
- *chorusPhase*: fase de la modulación del *chorus*.

- **Métodos:**
  - *EffectProcessor()*: Constructor
  - *applyReverbEffect(float &sample)*: aplica el efecto de *reverb* a la señal de entrada.
  - *applyDelayEffect(float &sample)*: aplica el efecto de *delay* a la señal de entrada.
  - *applyChorusEffect(float &sample)*: aplica el efecto de *chorus* a la señal de entrada.

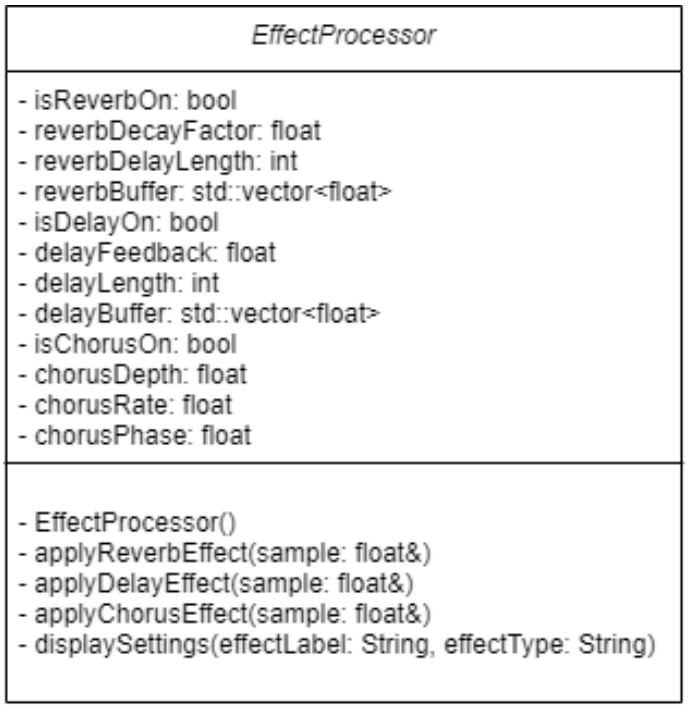

Figura 29: representación de la clase EffectoProcessor.

2. Aplicación de los efectos

Cada efecto tiene su propia función dentro de la clase *EffectProcessor* para modificar la señal de audio según los parámetros configurados.

- ***Reverb***

El efecto de *reverb* añade una serie de reflexiones que decaen con el tiempo, mediante la función *applyReverbEffect()* (Figura 30) simulando la reverberación de un espacio cerrado. Se utiliza un buffer circular para almacenar las muestras retrasadas.

```
void applyReverbEffect(float &sample) {
    if (!isReverbOn) return;

    int index = reverbBuffer.size() - reverbDelayLength;
    float delayedSample = reverbBuffer[index];
    reverbBuffer.push_back(sample + reverbDecayFactor * delayedSample);
    reverbBuffer.erase(reverbBuffer.begin());
    sample = sample * (1.0 - reverbDecayFactor) + delayedSample * reverbDecayFactor;
}
```

Figura 30: captura de pantalla del código de applyReverbEffect.

- ***Delay***

El efecto de *delay* crea repeticiones de la señal de audio con un cierto retardo, utilizando un buffer para almacenar las muestras y reproducirlas después de un período de tiempo especificado, mediante la función a*pplyDelayEffect()* (Figura 31)

```
void applyDelayEffect(float &sample) {
    if (!isDelayOn) return;

    int index = delayBuffer.size() - delayLength;
    if (index < 0) index += delayBuffer.size(); // Ajustar índice si es negativo

    float delayedSample = delayBuffer[index];
    delayBuffer.push_back(sample + delayFeedback * delayedSample);

    if (delayBuffer.size() > delayLength * 2) { // Evitar que el buffer crezca indefinidamente
        delayBuffer.erase(delayBuffer.begin());
    }

    sample = sample * (1.0 - delayFeedback) + delayedSample * delayFeedback;
}
```

Figura 31: captura de pantalla del código de applyDelayEffect.

- ***Chorus***

El efecto de *chorus* duplica la señal de audio con pequeñas variaciones en el tiempo y la afinación, utilizando un oscilador de baja frecuencia (LFO) para generar una fluctuación periódica en la señal. Se aplica mediante la función de *applyChorusEffect()* (Figura 32).

```
void applyChorusEffect(float &sample) {
    if (!isChorusOn) return;

    int chorusDelay = chorusDepth * (0.5 + 0.5 * sin(2 * PI * chorusRate * chorusPhase / SAMPLE_RATE));
    int index = delayBuffer.size() - chorusDelay;
    sample += delayBuffer[index];
    chorusPhase += 1;
    if (chorusPhase >= SAMPLE_RATE) chorusPhase -= SAMPLE_RATE;
}
};
```

Figura 32: captura de pantalla del código de applyDelayEffect.

3.Integración en el loop principal

Los efectos se aplican a la señal generada y filtrada antes de ser enviada al DAC.

4. Interacción con los *encoders* y visualización.

Los *encoders* permiten al usuario ajustar los parámetros de cada efecto en tiempo real de forma similar al modo en que se hace en el oscilador. La pantalla de configuración de efectos se actualiza mediante la función *displayEffectProcessorSettings()* (Figura 33), que muestra los valores actuales y permite su modificación.

```
void displayEffectProcessorSettings(const String &effectLabel, const String &effectType) {
    M5.Lcd.fillScreen(BLACK);
    M5.Lcd.setTextColor(WHITE);
    M5.Lcd.setTextSize(2);

    M5.Lcd.setCursor(10, 10);
    M5.Lcd.print(effectLabel + " Settings");

    if (effectType == "Reverb") {
        M5.Lcd.setCursor(10, 40);
        M5.Lcd.print("Decay: " + String(effectProcessor.reverbDecayFactor, 2));

        M5.Lcd.setCursor(10, 70);
        M5.Lcd.print("Delay: " + String(effectProcessor.reverbDelayLength));

        M5.Lcd.fillRect(10, 100, 100, 50, effectProcessor.isReverbOn ? GREEN : RED);
        M5.Lcd.setCursor(15, 115);
        M5.Lcd.setTextColor(BLACK);
        M5.Lcd.print(effectProcessor.isReverbOn ? "ON" : "OFF");
        M5.Lcd.setTextColor(WHITE);
    } else if (effectType == "Delay") {
        M5.Lcd.setCursor(10, 40);
        M5.Lcd.print("Feedback: " + String(effectProcessor.delayFeedback, 2));

        M5.Lcd.setCursor(10, 70);
        M5.Lcd.print("Delay: " + String(effectProcessor.delayLength));

        M5.Lcd.fillRect(10, 100, 100, 50, effectProcessor.isDelayOn ? GREEN : RED);
        M5.Lcd.setCursor(15, 115);
        M5.Lcd.setTextColor(BLACK);
        M5.Lcd.print(effectProcessor.isDelayOn ? "ON" : "OFF");
        M5.Lcd.setTextColor(WHITE);
    } else if (effectType == "Chorus") {
        M5.Lcd.setCursor(10, 40);
        M5.Lcd.print("Depth: " + String(effectProcessor.chorusDepth, 2));

        M5.Lcd.setCursor(10, 70);
        M5.Lcd.print("Rate: " + String(effectProcessor.chorusRate, 2));

        M5.Lcd.fillRect(10, 100, 100, 50, effectProcessor.isChorusOn ? GREEN : RED);
        M5.Lcd.setCursor(15, 115);
        M5.Lcd.setTextColor(BLACK);
        M5.Lcd.print(effectProcessor.isChorusOn ? "ON" : "OFF");
        M5.Lcd.setTextColor(WHITE);
    }
}
```

Figura 33: captura de pantalla del código de displayEffectProcessorSettings

5.4. Implementación de la envolvente ADSR

La implementación de la envolvente ADSR (*Attack*, *Decay*, *Sustain*, *Release*) en el controlador está diseñada para proporcionar un control preciso y flexible sobre la dinámica del sonido generado por los osciladores. A continuación, se detalla la estructura y funcionalidad del código utilizado para gestionar la envolvente ADSR y su integración con otros componentes del controlador.

1. Definición de la clase ADSR

La clase *ADSR,* expuesta en Figura 34, se define para manejar los parámetros y la aplicación de la envolvente *ADSR* a la señal de audio. Esta clase incluye los siguientes atributos y métodos:

- **Atributos**:
    - *Attack:* Tiempo de ataque, que determina cuánto tarda la señal en alcanzar su valor máximo después de ser activada. Este parámetro se controla a través del *encoder* conectado al GPIO XX.
    - *Decay:* Tiempo de decaimiento, que determina cuánto tarda la señal en reducirse desde el valor máximo hasta el nivel de sostenimiento.
    - *Sustain:* Nivel de sostenimiento, que es el nivel de amplitud mantenido mientras se sostiene la nota. Este parámetro se controla a través del *encoder* conectado al GPIO XX.
    - *Release:* Tiempo de liberación, que determina cuánto tarda la señal en reducirse desde el nivel de sostenimiento hasta cero después de que se libera la nota.
    - *noteOn:* Estado de la nota, indica si está activa.
    - *isOn:* Estado del ADSR, indica si la envolvente está activada.
    - *noteOnTime y noteOffTime:* Tiempos de activación y desactivación de la nota.
- **Métodos:**
    - *ADSR():* Constructor.
    - *noteOnTrigger(float currentTime):* Activa la envolvente y registra el tiempo de activación.
    - *noteOffTrigger(float currentTime):* Desactiva la envolvente y registra el tiempo de desactivación.
    - *applyEnvelope(float input, float currentTime):* Aplica la envolvente ADSR a la señal de entrada en función del tiempo transcurrido.

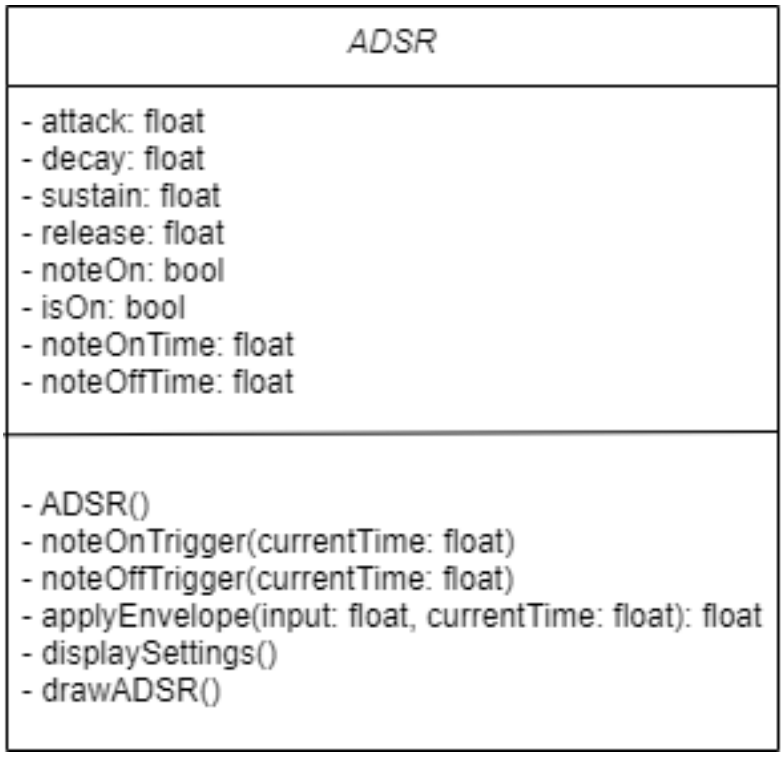

Figura 34: representación de la clase ADSR.

2. Aplicación de la envolvente ADSR

El método *applyEnvelope()* (Figura 35) toma una muestra de entrada y el tiempo actual, y aplica la envolvente ADSR a esta muestra. La señal resultante se modula en función de los parámetros de ataque, decaimiento, sostenimiento y liberación. Esto se logra mediante una serie de cálculos que determinan el valor de la señal en cada etapa de la envolvente.

```
float applyEnvelope(float input, float currentTime) {
    if (!isOn) return input;

    float time = currentTime - noteOnTime;
    if (noteOn) {
        if (time < attack) {
            return input * (time / attack);
        } else if (time < attack + decay) {
            return input * ((1.0 - (time - attack) / decay) * (1.0 - sustain) + sustain);
        } else {
            return input * sustain;
        }
    } else {
        float releaseTime = currentTime - noteOffTime;
        if (releaseTime < release) {
            return input * sustain * (1.0 - releaseTime / release);
        } else {
            return 0.0;
        }
    }
}
};
```

Figura 35: captura de pantalla del código de applyEnvelope.

3.Integración en el loop principal

4. Interacción con los *encoders*

Los *encoders* permiten modificar en tiempo real los parámetros de la envolvente ADSR, ofreciendo una interfaz intuitiva para el usuario. Cada *encoder* está configurado para ajustar uno de los parámetros de la envolvente:

- ***Encoder* 1**: ajusta el tiempo de ataque (*Attack*).
- ***Encoder* 2**: ajusta el tiempo de decaimiento (*Decay*).

Cada vez que se detecta un cambio en el estado del *encoder*, se actualizan los parámetros de la envolvente correspondiente y se redibuja la gráfica de la envolvente en la pantalla.

```
void displayADSRSettings() {
    M5.Lcd.fillScreen(BLACK);
    M5.Lcd.setTextColor(WHITE);
    M5.Lcd.setTextSize(2);

    M5.Lcd.setCursor(10, 10);
    M5.Lcd.print("ADSR Settings");

    // Controles para Attack y Decay en la columna derecha
    M5.Lcd.setCursor(10, 40);
    M5.Lcd.print("A: " + String(adsr.attack, 2));

    M5.Lcd.setCursor(10, 70);
    M5.Lcd.print("D: " + String(adsr.decay, 2));

    // Controles para Sustain y Release en la columna izquierda
    M5.Lcd.setCursor(160, 40);
    M5.Lcd.print("S: " + String(adsr.sustain, 2));

    M5.Lcd.setCursor(160, 70);
    M5.Lcd.print("R: " + String(adsr.release, 2));

    drawADSR();
}
```

Figura 36: captura de pantalla del código de displayADSRSettings.

5. Visualización y control

La visualización en pantalla de los parámetros de la envolvente (Figura 36) y su forma de onda es crucial para una interacción efectiva. El método *drawADSR* (Figura 37) se encarga de dibujar la forma de onda de la envolvente ADSR en la pantalla, proporcionando una representación visual clara de cómo afectará la envolvente a la señal de audio.

```
void drawADSR() {
    int baseY = 230;
    int baseX = 10;
    int width = 300;
    int height = 100;

    M5.Lcd.fillRect(baseX, baseY - height, width, height, BLACK);

    int attackX = baseX + width * adsr.attack;
    int decayX = attackX + width * adsr.decay;
    int sustainY = baseY - height * adsr.sustain;
    int releaseX = baseX + width * adsr.release;

    M5.Lcd.drawLine(baseX, baseY, attackX, baseY - height, WHITE);
    M5.Lcd.drawLine(attackX, baseY - height, decayX, sustainY, WHITE);
    M5.Lcd.drawLine(decayX, sustainY, releaseX, sustainY, WHITE);
    M5.Lcd.drawLine(releaseX, sustainY, releaseX, baseY, WHITE);
}
```

Figura 37: captura de pantalla del código de drawADSR.

5.5. Implementación de los filtros de bandas

La implementación de filtros en el controlador está diseñada para ofrecer un control preciso y flexible sobre la modulación de frecuencias del sonido generado. Los filtros utilizados son de tipo paso alto (HPF) y paso bajo (LPF), ambos de 4º orden utilizando la estructura de Butterworth para garantizar una transición suave y una respuesta en frecuencia efectiva. A continuación, se detalla la estructura y funcionalidad del código utilizado para gestionar estos filtros y su integración con otros componentes del controlador.

1. Definición de las clases de filtros

La clase *ButterworthFilter* expuesta en Figura 38 es una clase base para los filtros de paso bajo y paso alto de 4º orden.

- **Atributos**:
    - *Cutoff:* Frecuencia de corte del filtro.
    - *resonance:* Resonancia del filtro.
    - *isOn:* Estado del filtro, indica si el filtro está activado.

- *a, b:* Coeficientes del filtro.
- *x, y:* Vectores para almacenar los valores de entrada y salida del filtro.

- **Métodos:**
  - *calculateCoefficients()*: Calcula los coeficientes del filtro.
  - *applyFilter(float &sample)*: Aplica el filtro a la muestra de audio (Figura 39).

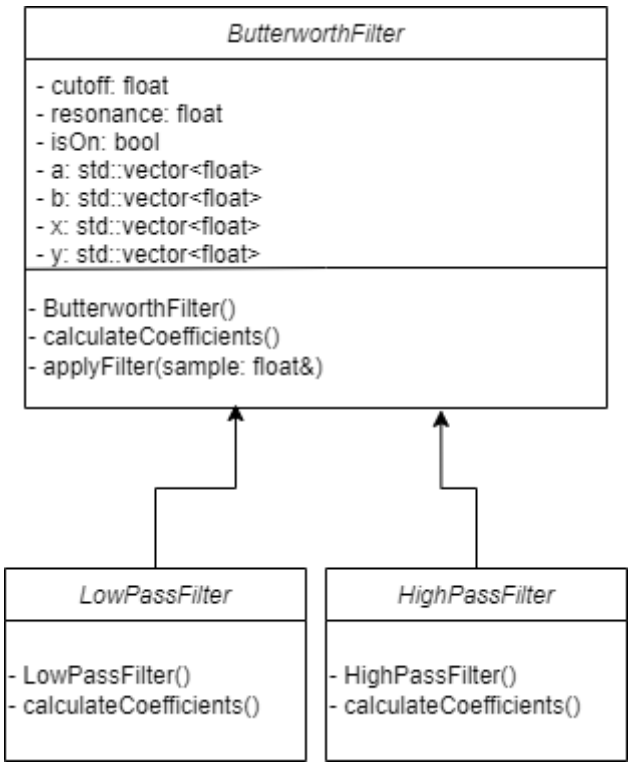

Figura 38: representación de las clases ButterworthFilter, LowPassFilter y HighPassFilter.

```
    void applyFilter(float &sample) {
        if (isOn) {
            for (int i = 4; i > 0; --i) {
                x[i] = x[i - 1];
                y[i] = y[i - 1];
            }
            x[0] = sample;
            y[0] = b[0] * x[0] + b[1] * x[1] + b[2] * x[2] - a[1] * y[1] - a[2] * y[2];
            sample = y[0];
        }
    }
};
```

Figura 39: captura de pantalla del código de applyFilter.

2. Aplicación de filtros de Butterworth

Los filtros de paso bajo y paso alto se implementan como subclases de *ButterworthFilter* y sobrescriben el método *calculateCoefficients()* (Figura 40 y 41 respectivamente) para calcular los coeficientes específicos de cada tipo de filtro.

- **Filtro paso bajo (LPF)**:

```
// Clase para el filtro paso bajo de Butterworth de 4º orden
class LowPassFilter : public ButterworthFilter {
public:
    LowPassFilter() {
        cutoff = 20000.0;
        calculateCoefficients();
    }

    void calculateCoefficients() override {
        float omega = 2 * M_PI * cutoff / SAMPLE_RATE;
        float sin_omega = sin(omega);
        float cos_omega = cos(omega);
        float alpha = sin_omega / (2 * resonance);

        float a0 = 1 + alpha;
        a[0] = 1 / a0;
        a[1] = -2 * cos_omega / a0;
        a[2] = (1 - alpha) / a0;
        b[0] = (1 - cos_omega) / 2 / a0;
        b[1] = (1 - cos_omega) / a0;
        b[2] = b[0];
    }
};
```

Figura 40: captura de pantalla del código de LowPassFilter y calculateCoefficients.

- **Filtro paso alto (HPF)**:

```
// Clase para el filtro paso alto de Butterworth de 4º orden
class HighPassFilter : public ButterworthFilter {
public:
    HighPassFilter() {
        cutoff = 20.0;
        calculateCoefficients();
    }

    void calculateCoefficients() override {
        float omega = 2 * M_PI * cutoff / SAMPLE_RATE;
        float sin_omega = sin(omega);
        float cos_omega = cos(omega);
        float alpha = sin_omega / (2 * resonance);

        float a0 = 1 + alpha;
        a[0] = 1 / a0;
        a[1] = -2 * cos_omega / a0;
        a[2] = (1 - alpha) / a0;
        b[0] = (1 + cos_omega) / 2 / a0;
        b[1] = -(1 + cos_omega) / a0;
        b[2] = b[0];
    }
};
```

Figura 41: captura de pantalla del código de HighPassFilter y calculateCoefficients.

3. Integración en el Loop Principal

Los filtros se aplican a la señal generada por los osciladores y modificada por la envolvente ADSR antes de aplicar los efectos.

4. Interacción con los *encoders*

Los *encoders* permiten modificar en tiempo real la frecuencia de corte de los filtros, ofreciendo una interfaz intuitiva para el usuario. Cada *encoder* está configurado para ajustar uno de los filtros:

- ***Encoder* 1**: ajusta la frecuencia de corte del filtro paso bajo.
- ***Encoder* 2**: ajusta la frecuencia de corte del filtro paso alto.

Cada vez que se detecta un cambio en el estado del *encoder*, se actualizan los parámetros del filtro correspondiente y se redibuja la curva de respuesta en frecuencia en la pantalla.

```
void displayEQSettings() {
    M5.Lcd.fillScreen(BLACK);
    M5.Lcd.setCursor(10, 10);
    M5.Lcd.print("EQ Settings");

    displayFilterSettings(lowPassFilter, 40);
    displayFilterSettings(highPassFilter, 40);

    drawEQCurve();
}
```

Figura 42: captura de pantalla del código de displayEQSettings.

Figura 43: representación de los tipos de señales: sinusoidal, diente de sierra, triangular y cuadrada.

5. Visualización y control

La visualización en pantalla de los parámetros de los filtros (Figura 42 y 43) y su curva de respuesta en frecuencia (Figura 44) es crucial para una interacción fectiva. El método *drawEQCurve* se encarga de dibujar la respuesta en frecuencia de ambos filtros en una escala logarítmica, simulando un análisis de espectro de audio.

```
void displayFilterSettings(ButterworthFilter &filter, int yOffset) {
    M5.Lcd.setCursor(10, yOffset);
    M5.Lcd.print("Cutoff:" + String(filter.cutoff, 0));

    M5.Lcd.setCursor(10, yOffset + 30);
    M5.Lcd.print("Resonance: " + String(filter.resonance, 2));
}
```

Figura 43: : captura de pantalla del código de displayFilterSettings.

Figura 44: representación de los tipos de señales: sinusoidal, diente de sierra, triangular y cuadrada.

```
void drawEQCurve() {
    int baseY = 200;
    int baseX = 10;
    int width = 300;
    int height = 100;

    M5.Lcd.fillRect(baseX, baseY - height, width, height, BLACK);

    M5.Lcd.drawLine(baseX, baseY, baseX + width, baseY, WHITE);
    M5.Lcd.drawLine(baseX, baseY - height, baseX, baseY, WHITE);

    struct FrequencyLabel {
        int freq;
        const char* label;
        int xOffset;
    };

    FrequencyLabel freqLabels[] = {
        {20, "20", 0},
        {100, "100", 0},
        {1000, "1k", 0},
        {5000, "5k", 0},
        {20000, "20k", -20}
    };

    for (int i = 0; i < 5; i++) {
        int x = baseX + log10(freqLabels[i].freq / 20.0) * (width / log10(20000 / 20.0));
        x += freqLabels[i].xOffset;
        M5.Lcd.drawLine(x, baseY, x, baseY - height, WHITE);
        M5.Lcd.setCursor(x - 10, baseY + 5);
        M5.Lcd.printf("%s", freqLabels[i].label);
    }

    for (int i = 0; i < width; i++) {
        float freq = pow(10, log10(20) + (float)i / width * log10(20000 / 20.0));
        float response = 1 / sqrt(1 + pow(freq / lowPassFilter.cutoff, 2 * 4));
        int y = baseY - height * response;
        M5.Lcd.drawPixel(baseX + i, y, RED);
    }

    for (int i = 0; i < width; i++) {
        float freq = pow(10, log10(20) + (float)i / width * log10(20000 / 20.0));
        float response = sqrt(1 / (1 + pow(highPassFilter.cutoff / freq, 2 * 4)));
        int y = baseY - height * response;
        M5.Lcd.drawPixel(baseX + i, y, BLUE);
    }
}
```

Figura 44: captura de pantalla del código de drawEQCurve.

5.6. Implementación de *setup*

La función *setup()* (Figura 45) se ejecuta una vez al inicio del programa y se utiliza para inicializar y configurar los componentes necesarios para el funcionamiento del controlador. A continuación, se detalla su implementación:

```
void setup() {
    M5.begin();

    // Configuración de pines de encoders y botones
    pinMode(ENCODER1_CLK, INPUT);
    pinMode(ENCODER1_DT, INPUT);

    pinMode(ENCODER2_CLK, INPUT);
    pinMode(ENCODER2_DT, INPUT);

    // Configuración de pines de botones pulsadores
    for (int i = 0; i < 7; ++i) {
        pinMode(buttonPins[i], INPUT_PULLUP);
    }

    // Inicializar osciladores con diferentes frecuencias
    for (int i = 0; i < 7; ++i) {
        oscillators.push_back(Oscillator(noteFrequencies[i], 1.0, SINE));
    }

    // Mostrar directamente la pantalla principal
    displayMainScreen();
}
```

Figura 45: captura de pantalla del código de setup.

5.7. Implementación de *loop*

El bucle *loop()* expuesta en Figura 46 es la función principal que se ejecuta repetidamente en el programa. Su propósito es actualizar el estado del dispositivo, manejar la interacción del usuario y aplicar efectos de audio según las entradas recibidas.

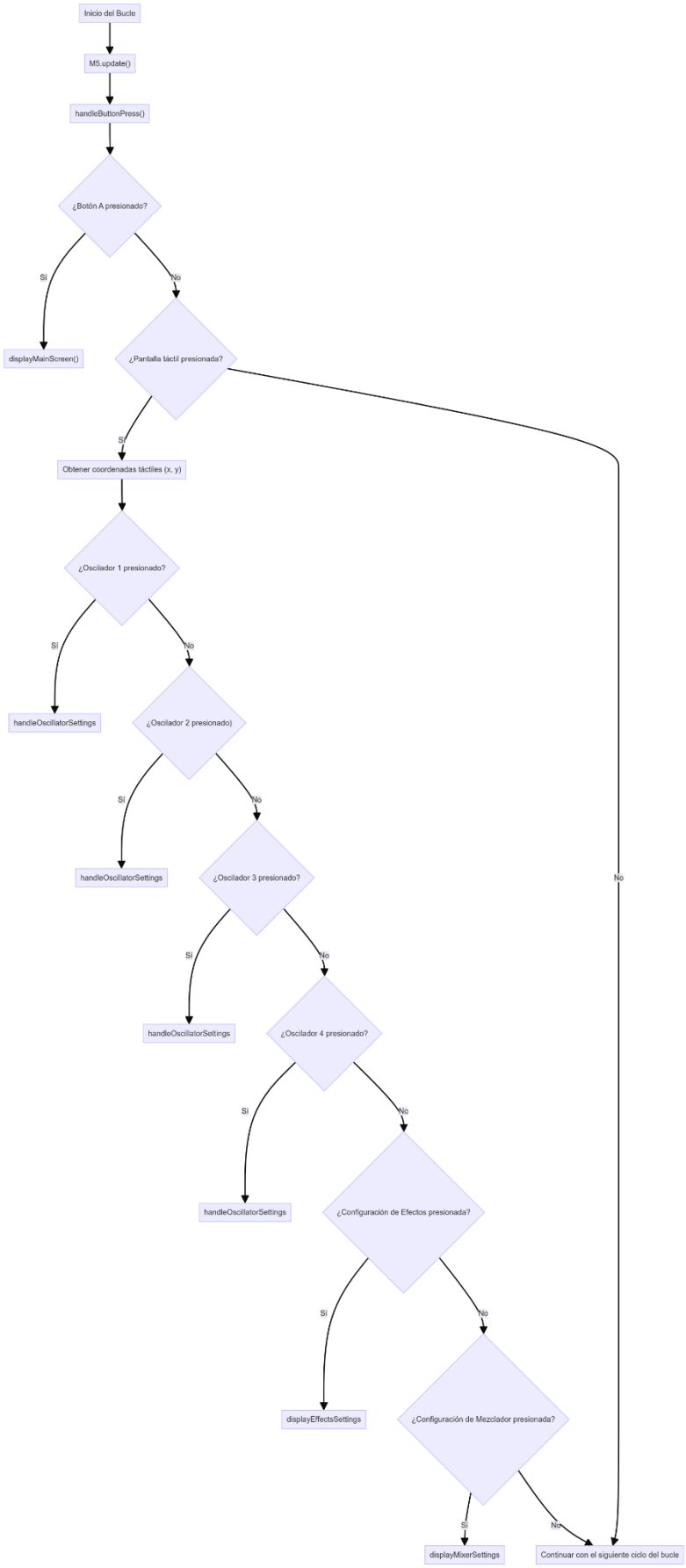

Figura 46: diagrama de flujo de la función loop()

Tabla 5: Descripción de las funciones del bucle loop.

Nombre de la función	**Propósito**	**Entradas**	**Salidas**
displayOscillatorSettings	Muestra los ajustes del oscilador (frecuencia, amplitud, tipo de onda, desplazamiento de octava) en la pantalla.	oscLabel (String): Nombre del oscilador. osc (Oscillator&): Referencia a la instancia del oscilador.	Actualiza la pantalla con los ajustes del oscilador.
displayADSRSettings	Muestra los ajustes del generador de envolvente ADSR (Attack, Decay, Sustain, Release) en la pantalla.	Ninguna	Actualiza la pantalla con los ajustes de ADSR.
drawADSR	Dibuja la envolvente ADSR en la pantalla.	Ninguna	Dibuja la curva de ADSR en la pantalla.
displayMainScreen	Muestra la pantalla principal con botones para osciladores, mezclador y efectos	Ninguna	Actualiza la pantalla principal con los botones de control
displayEffectSettings	Muestra los ajustes de un efecto específico (Reverb, Delay, Chorus) en la pantalla.	effectLabel (String): Nombre del efecto. effectType (String): Tipo de efecto.	Actualiza la pantalla con los ajustes del efecto seleccionado.
displayEffectsSettings	Muestra la pantalla de ajustes de efectos, permitiendo al usuario seleccionar entre Reverb, Delay y Chorus.	Ninguna	Actualiza la pantalla con las opciones de efectos.
displayMixerSettings	Muestra la pantalla de ajustes del mezclador.	Ninguna	Actualiza la pantalla con los controles del mezclador.

displayEQSettings	Muestra los ajustes del ecualizador en la pantalla.	Ninguna	Actualiza la pantalla con los ajustes de EQ.
drawEQCurve	Dibuja la curva de ecualización en la pantalla.	Ninguna	Dibuja la curva de respuesta de EQ en la pantalla
handleOscillatorSettings	Maneja la configuración del oscilador, ajustando parámetros como el tipo de onda y el desplazamiento de octava.	oscLabel (String): Nombre del oscilador. osc (Oscillator&): Referencia a la instancia del oscilador.	Actualiza la configuración del oscilador y la pantalla.
handleButtonPress	Maneja la detección de pulsaciones de botones, generando la señal de audio correspondiente y aplicando los efectos.	Ninguna	Genera y procesa la señal de audio en función de los botones pulsados.
displayOctaveShift	Muestra el desplazamiento de octava del oscilador en la pantalla.	osc (Oscillator&): Referencia a la instancia del oscilador.	Actualiza la pantalla con el desplazamiento de octava del oscilador.
displayFilterSettings	Muestra los ajustes de un filtro Butterworth (frecuencia de corte, resonancia) en la pantalla.	filter (ButterworthFilter&): Referencia a la instancia del filtro. yOffset (int): Desplazamiento vertical en la pantalla para los ajustes.	Actualiza la pantalla con los ajustes del filtro.
displayEffectProcessorSettings	Muestra y ajusta la configuración de un efecto específico (Reverb, Delay, Chorus) en la pantalla.	effectLabel (String): Nombre del efecto. effectType (String): Tipo de efecto.	Actualiza la pantalla con los ajustes del efecto seleccionado.

6. Pruebas y resultados

Aquí se describe la metodología de pruebas utilizada para verificar la funcionalidad y el rendimiento del sintetizador. Se detallan las pruebas realizadas y se presentan los resultados obtenidos.

6.1. Metodología de pruebas

Para asegurar el funcionamiento del controlador M5Stack Core 2 y validar su desempeño en varias tareas, se hicieron pruebas exhaustivas en diferentes componentes y funcionalidades. A continuación, se describe la metodología de las pruebas realizadas:

1. **Prueba del altavoz:**
    - **Barrido de frecuencias y formas de onda:** se realizó un barrido de frecuencias desde 110 Hz hasta 1760 Hz para evaluar la respuesta en frecuencia del altavoz. La prueba asegura que el altavoz pueda reproducir amplio rango de frecuencias, y diferentes formas de ondas.
2. **Prueba del *encoder*:**
    - **Funcionamiento general:** se verificó el funcionamiento básico del *encoder*, incluyendo la detección de giros en ambas direcciones y la correcta actualización de los valores asociados.
    - **Compatibilidad con múltiples *encoders*:** se probó la funcionalidad simultánea de varios *encoders* para asegurar que no haya interferencia entre ellos y que todos los *encoders* puedan ser leídos correctamente al mismo tiempo.
3. **Prueba de botones:**
    - **Funcionamiento general:** se evaluó la capacidad de los botones para registrar presiones y liberaciones, verificando la fiabilidad de la detección de eventos.
    - **Función de teclado de piano:** se asignaron diferentes notas a cada botón para simular un teclado de piano, comprobando que cada botón produce la nota correcta al ser presionado.
4. **Prueba de funcionalidades en el programa completo:**
    - **Integración completa:** se ejecutó el programa completo del controlador para verificar la integración y funcionamiento conjunto de todos los componentes (osciladores, filtros, efectos, ADSR, etc.).
    - **Interfaz de usuario:** se navegó por las diferentes pantallas de configuración para asegurarse de que la interfaz de usuario es intuitiva y todas las funcionalidades son accesibles y operativas.

5. **Pruebas con osciloscopio:**
    - **Captura de ondas:** Utilizando un osciloscopio, se visualizaron y guardaron las formas de onda generadas por el sintetizador. Se observaron

las ondas producidas por los diferentes osciladores y las variaciones introducidas por los filtros y efectos aplicados. Estas pruebas permiten una visualización precisa y detallada del comportamiento del sintetizador en tiempo real.

6.2. Resultados obtenidos

1. **Altavoz:**

   - **Barrido de frecuencias y formas de onda:** el altavoz mostró una respuesta adecuada en la mayoría de las frecuencias del espectro audible. Se observaron algunas caídas en la respuesta a frecuencias extremas, pero estas estaban dentro de los límites esperados. Además, se probó con el osciloscopio para poder verificar su resultado, como se puede observar en las figuras 47-51.

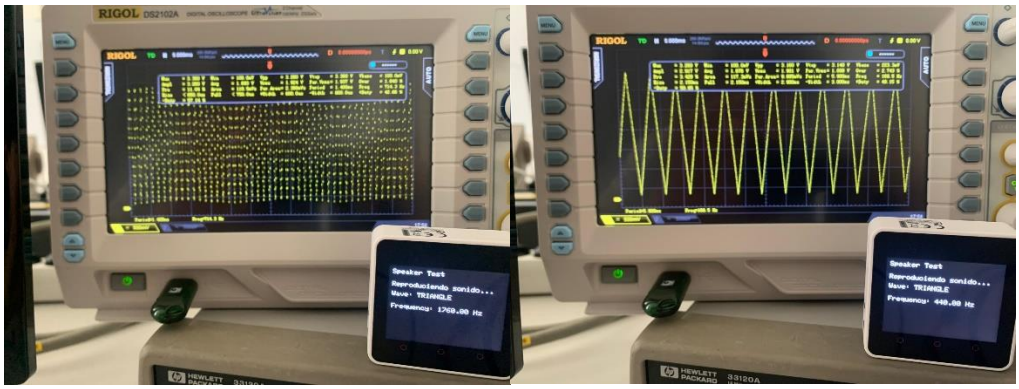

Figura 47: prueba de onda triangular a 880Hz

Figura 48: prueba de onda triangular a 440Hz

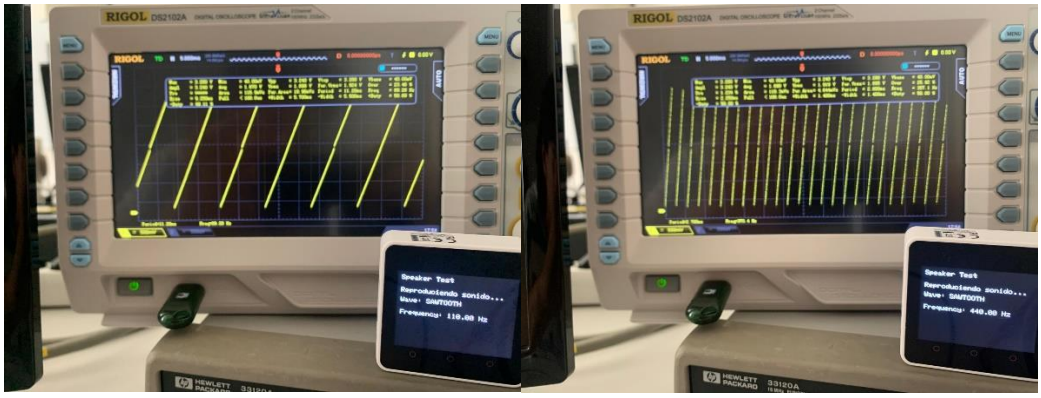

Figura 49: prueba de onda de diente de sierra a 110Hz

Figura 50: prueba de onda de diente de sierra a 440H

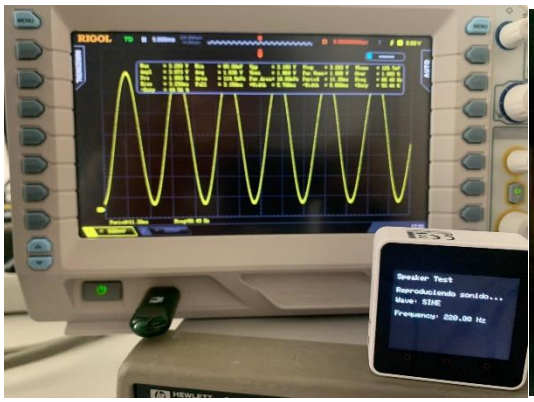

Figura 51: prueba de onda sinusoidal a 220Hz

Figura 52: prueba de onda cuandrada a 1760Hz

2. ***Encoder*:**
    - **Funcionamiento general:** los *encoders* respondieron correctamente a los giros en ambas direcciones. No se detectaron errores en la actualización de valores, y la sensibilidad de los *encoders* fue adecuada.
    - **Compatibilidad con múltiples *encoders*:** los *encoders* funcionaron simultáneamente sin interferencia, y cada *encoder* registró sus movimientos correctamente.
3. **Botones:**
    - **Funcionamiento general:** los botones registraron presiones y liberaciones de manera consistente y fiable. No se detectaron falsos positivos ni fallos en la detección de eventos.
    - **Función de teclado de piano:** cada botón produjo la nota asignada correctamente, permitiendo una funcionalidad de teclado de piano efectiva.
4. **Funcionalidades en el programa completo:**
    - **Integración completa:** el programa funcionó sin errores críticos, y todas las funcionalidades se integraron de manera coherente. Los osciladores, filtros, efectos y ADSR se controlaron adecuadamente a través de la interfaz de usuario.
    - **Interfaz de usuario:** la navegación por las pantallas de configuración fue fluida, y todas las opciones de ajuste estaban operativas. La interfaz resultó ser intuitiva y fácil de usar.
5. **Pruebas con osciloscopio:**
    - **Captura de ondas:** las formas de onda capturadas en el osciloscopio mostraron que los osciladores generan ondas precisas y bien definidas. Las pruebas demostraron que los filtros modifican la señal de manera predecible y

conforme a lo esperado. Las visualizaciones permitieron corroborar la correcta implementación y funcionamiento de estos componentes en el sintetizador como se puede observar de la Figura 53 a la 64

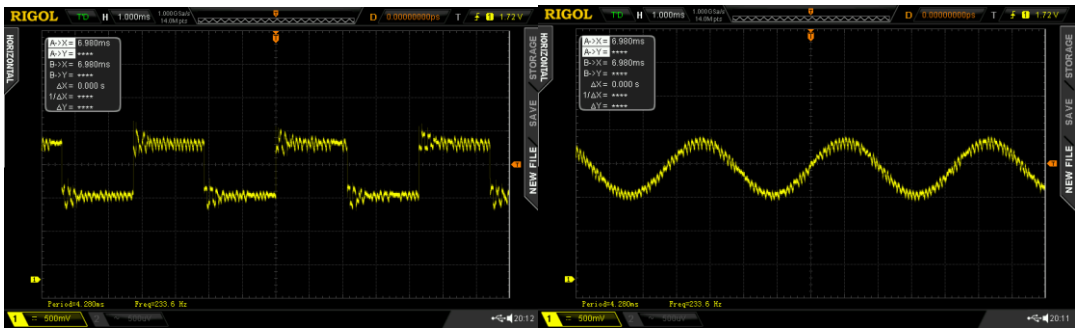

Figura 53:captura del osciloscopio de una señal cuadrada generada por el controlador.

Figura 54: captura del osciloscopio de una señal sinusoidal generada por el controlador.

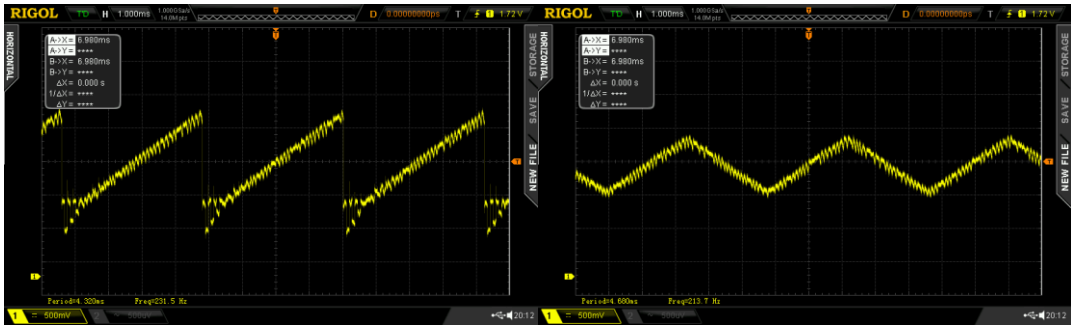

Figura 55: captura del osciloscopio de una señal de diente de sierra generada por el controlador.

Figura 56: captura del osciloscopio de una señal triangular generada por el controlador.

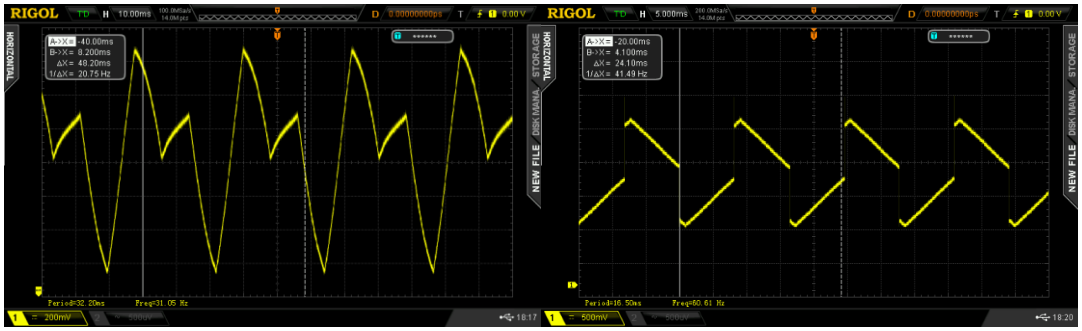

Figura 57: captura del osciloscopio de una señal sinusoidal + triangular generada por el controlador.

Figura 58: captura del osciloscopio de una señal cuadrada +diente de sierra generada por el controlador.

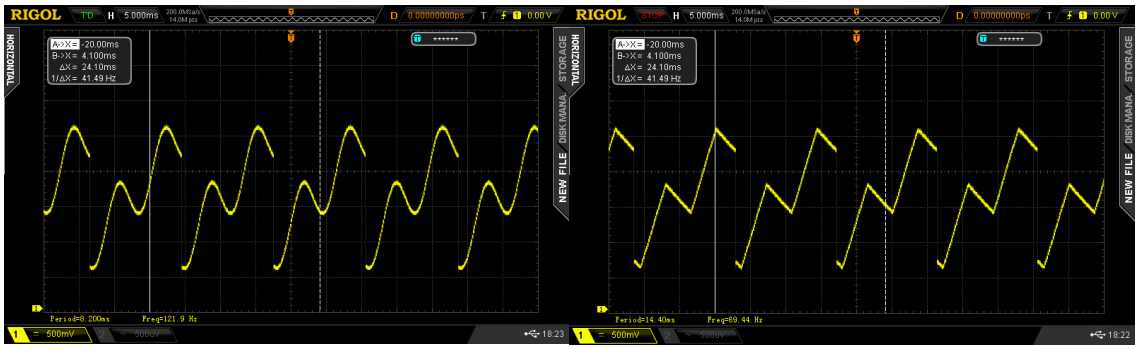

Figura 59: captura del osciloscopio de una señal sinusoidal + diente de sierra generada por el controlador.

Figura 60: captura del osciloscopio de una señal triangular + diente de sierra generada por el controlador.

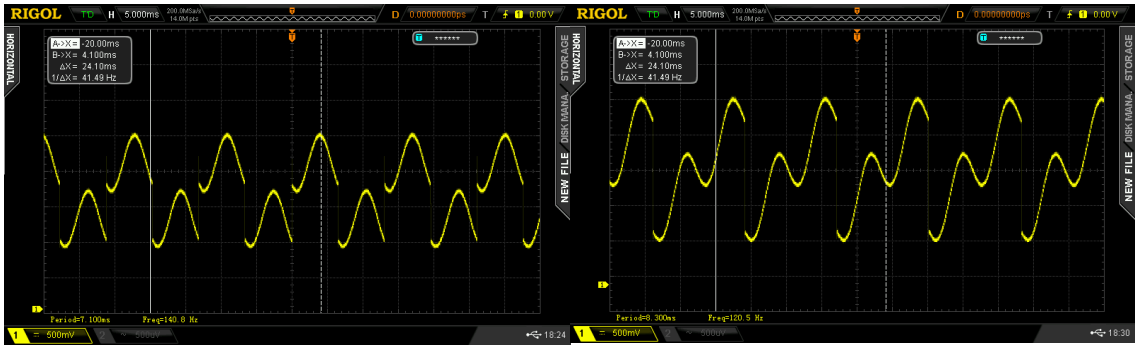

Figura 61: captura del osciloscopio de una señal sinusoidal + triangular generada por el controlador.

Figura 62: captura del osciloscopio de una señal sinusoidal + diente de sierra generada por el controlador.

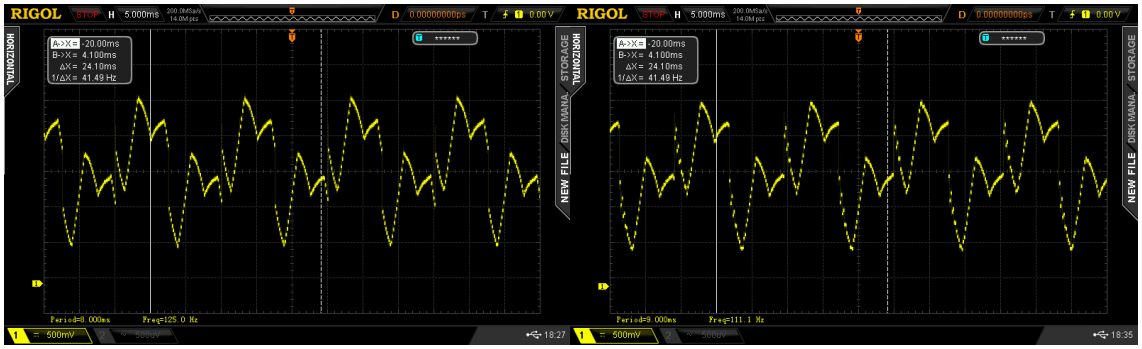

Figura 63: captura del osciloscopio de una señal sinusoidal + cuadrada + triangular con el filtro activado (400-2500 Hz).

Figura 64 : captura del osciloscopio de una señal sinusoidal + cuadrada + triangular con el filtro desactivado.

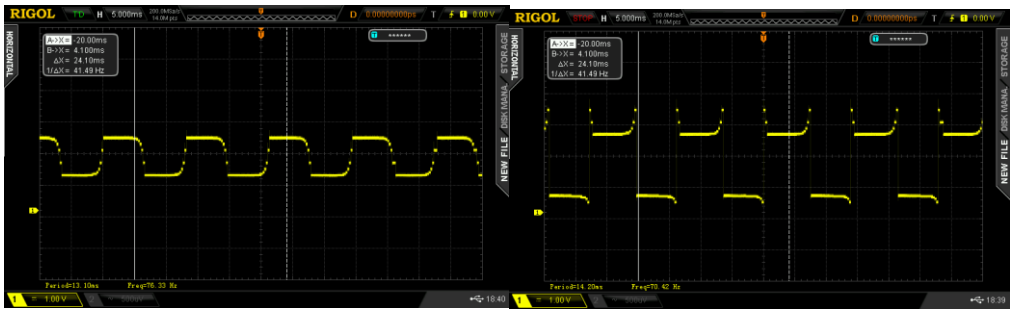

Figura 65:captura del osciloscopio de una señal cuadrada con Delay

Figura 66: captura del osciloscopio de una señal cuadrada con Reverb.

7. Conclusiones

Esta sección resume los logros alcanzados hasta ahora y las limitaciones encontradas en el proceso de implementación del proyecto. Incluye los cambios sugeridos y el trabajo futuro que mejoraría la capacidad del sintetizador.

7.1. Logros alcanzados

El desarrollo del sintetizador utilizando el controlador M5Stack Core 2 ha permitido la integración de múltiples componentes y funcionalidades que operan de manera coherente y eficiente. Entre los logros más destacados se incluyen la implementación de osciladores múltiples, generando diversas formas de onda (senoidal, cuadrada, triangular y diente de sierra) ajustables en frecuencia y amplitud, así como la incorporación de efectos de audio como *reverb*, *delay* y *chorus*, que se pueden ajustar en tiempo real, enriqueciendo la calidad del sonido generado. Además, se aplicaron filtros de paso alto y paso bajo de Butterworth de 4º orden integrados con éxito, permitiendo un control preciso del contenido frecuencial del sonido y se implementó una envolvente ADSR completa, con control de la dinámica del sonido. La pantalla táctil del M5Stack Core 2 y los *encoders* proporcionan una interfaz de usuario interactiva y fácil de usar, permitiendo ajustes precisos y en tiempo real de todos los parámetros. Las pruebas realizadas con osciloscopio confirmaron la precisión y calidad de las señales generadas, así como la eficacia de los filtros y efectos aplicados.

7.2. Desafíos enfrentados

El desarrollo del sintetizador no estuvo falto de desafíos técnicos y de integración. Algunos de los más importantes han sido las limitaciones del *hardware*, ya que el M5Stack Core 2, aunque potente, presentó limitaciones en términos de capacidad de procesamiento y memoria, lo que requirió optimizaciones cuidadosas en el código, sobre todo en la parte de los efectos. Lograr una integración fluida de osciladores, filtros, efectos y la interfaz de usuario fue un desafío, especialmente al gestionar el tiempo real y la sincronización de eventos. Asegurar que los *encoders* respondieran con precisión fue crucial para la experiencia de usuario, lo que implicó ajustes y pruebas. La implementación de efectos como *reverb* y *delay* requirió un manejo eficiente de buffers y memoria para evitar latencias y asegurar una calidad de audio consistente.

7.3. Trabajo futuro y posibles mejoras

A pesar de los logros alcanzados, existen áreas para futuras mejoras y expansión del proyecto. La optimización de la gestión de memoria permitirá una mayor polifonía y la implementación de efectos adicionales sin comprometer el rendimiento. Se podría proponer la implementación de moduladores adicionales, como moduladores de baja frecuencia (LFOs) y otros tipos, para diversificar aún más las capacidades sonoras del sintetizador. En cuanto a la interfaz de usuario, se planea desarrollar una gráfica más avanzada con imagen de la señal en tiempo real y respuestas visuales a las modificaciones de parámetros. La optimización de los algoritmos de generación y procesamiento de señales es esencial para mejorar la eficiencia y reducir la carga en el microcontrolador.

8. Bibliografía

- ADSR explained (sound design basics). (2022, November 15). Stickz. https://stickz.co/blog/adsr-explained-sound-design-basics/

- Arduino. (n.d.). https://www.arduino.cc/en/*software*

- Audio Procedural Reverb (n.d.). GitHub https://github.com/JDSherbert/Audio-Procedural-Reverb

- Chorus Effect (n.d.). GitHub https://github.com/thestk/stk/blob/master/src/Chorus.cpp

- DSP-Cpp-filters. (n.d.). GitHub. https://github.com/dimtass/DSP-Cpp-filters

- Garriga Solé, P. (2018). Diseño y construcción de los módulos de un sintetizador modular analógico de audio [Trabajo de fin de grado, Escola Tècnica Superior d'Enginyeria Industrial de Barcelona]. Universitat Politècnica de Catalunya.

- Gutiérrez-Ravé Olmos, J. (2018). Diseño e implementación de un sintetizador de audio modular basado en síntesis substractiva [Trabajo de fin de grado, Universitat Politècnica de València]. Escuela Técnica Superior de Ingeniería del Diseño.

- Joyanes Aguilar, L. (2006). Programación en C++: algoritmos, estructuras de datos y objetos (2ª ed.). McGraw-Hill.

- Reveillac, J.-M.(2018). Musical sound effects: analog and digital sound processing. ISTE.

- Redondo Padilla, R. (2023). Sintetizador de sonidos con FPGA [Trabajo de fin de grado, Universitat Politècnica de València]. Escuela Técnica Superior de Ingeniería Informática.

- Russ, M. (2009). Sound synthesis and sampling (3rd ed.). Elsevier.

- Speak.ino. (n.d.). GitHub. https://github.com/m5stack/M5Core2/blob/master/examples/Basics/speak/speak.ino

- Wilson, R. (2013). Make: analog synthesizers. Maker Media.

- ¿Qué es un oscilador analógico? (2021, June 4). Holawave. https://holawave.store/blogs/noticias/que-es-un-oscilador

9. Anexos

ANEXO I. Relación del trabajo con los Objetivos de Desarrollo Sostenible de la Agenda 2030

ANEXO II. Código del sintetizador completo.