REPORTE FINAL

Pedal de fuzz basado en Arduino IDE

Autores:
Castillo Acosta Josué Marcelo
Díaz Rodríguez Kaylee Michelle
Hernández Hernández Juliana
Navarro Hernández Hugo Jesús
Salas Martínez Cesar David

Fecha: 8 de junio de 2025

Universidad Veracruzana
Facultad de Instrumentación Electrónica

Índice

Introducción

El presente proyecto se realizó con la intención de realizar un circuito de pedal de fuzz funcional por medio de la programación de una placa ESP32, claro utilizando el software Arduino IDE.Todo esto con la finalidad de entender mejor el funcionamiento de la placa utilizada y su programación, investigando así las funciones requeridas en el software para la programación ante ciertas necesidades del proyecto mismo.

Objetivos

El objetivo general del presente proyecto es el diseño de un circuito de pedal de fuzz por medio de la programación de una placa ESP32 utilizando el software Arduino IDE.

Entre los objetivos generales que planteamos, tenemos:

- Comprender el funcionamiento de una placa ESP32, así como su respectiva programación.
- Observar el funcionamiento de las ondas por medio de una interfaz realizada en lenguaje Python.
- Comprobar el funcionamiento de nuestro circuito.

Antecedentes/conocimientos previos

Para la realización de este proyecto, contamos con conocimientos previos sobre la programación en Python, esto debido a que ha sido tema de trabajo desde nuestro tercer semestre de universidad por parte de la asignatura *Programación de interfaces*, hasta la fecha con la presente asignatura de *Programación avanzada*, en la cuál hemos desarrollado prácticas sobre la integración de sensores y el procesamiento de datos con Arduino y Python, la programación en micropython y la realización de gráficos con Python. Por otro lado, con apoyo de nuestro conocimiento previo en circuitos por parte de otras asignaturas, nos sentó como base para iniciar con la realización del proyecto. Independientemente al contexto escolar, también nos enfocamos en realizar investigaciones sobre la programación en Python, y el funcionamiento de las ondas, para así generar las ecuaciones requeridas para el correcto funcionamiento de nuestro trabajo. Por otro lado se realizó la investigación sobre el sonido y como funciona. Además investigamos el funcionamiento de un pedal de fuzz típico.

Planteamiento

La ídea principal de este proyecto, es la realización de un circuito distorsionador de audio que sea funcional con la programación previa de una placa ESP32. Para esta distorsión se utilizará la técnica de ”Fuzz”, la cual provoca que el sonido salga con un sonido más rasposo y saturado.

Trasfondo

Investigación respecto al sonido

Esta investigación se realizó con el objetivo de comprender mejor el comportamiento del sonido, especialmente el generado por un instrumento musical, y cómo puede modificarse para lograr un efecto deseado. A continuación, se presentan algunos conceptos clave:

Hertz

Los hertz (Hz) son la unidad de medida de la frecuencia en el Sistema Internacional. Un hertz equivale a un ciclo por segundo. Esta unidad recibe su nombre en honor a Heinrich Rudolf Hertz, quien realizó importantes descubrimientos sobre las ondas electromagnéticas.

La frecuencia indica cuántas veces se repite un evento por segundo. Por ejemplo, una onda sonora de 440 Hz se repite 440 veces en un segundo. También se usan múltiplos:

- Kilohertzio (kHz): 1,000 Hz
- Megahertzio (MHz): 1,000,000 Hz
- Gigahertzio (GHz): 1,000,000,000 Hz

Matemáticamente, la frecuencia se define como:

$$f = \frac{1}{T}$$

donde:

- f: frecuencia (Hz)
- T: periodo (segundos)

En el sonido, frecuencias bajas (20 Hz) producen sonidos graves, y frecuencias altas (20,000 Hz) generan sonidos agudos. El rango audible del oído humano va de 20 Hz a 20 kHz.

Decibeles

Los decibeles (dB) son una unidad logarítmica usada para comparar magnitudes como la presión sonora o la potencia eléctrica. Son muy útiles porque el oído humano percibe el sonido de forma logarítmica.

El nivel de presión sonora se calcula así:

$$L_p = 20 \log_{10} \left(\frac{P}{P_0} \right)$$

- *Lp*: nivel de presión sonora en decibeles.
- *P*: presión sonora medida.
- P_0: presión de referencia, típicamente 20 µPa.

Para potencia:

$$L = 10 \log_{10} \left(\frac{P_1}{P_0} \right)$$

- *L*: nivel en decibeles.
- P_1: potencia medida.
- P_0: potencia de referencia.

Armónicos

Los armónicos son componentes de una señal periódica que tienen frecuencias que son múltiplos enteros de la frecuencia fundamental. Esta frecuencia fundamental es la más baja de una onda periódica y determina el tono básico de la señal. Los armónicos aparecen debido a la naturaleza no lineal de ciertos sistemas o a la forma particular de la onda.

En el contexto del sonido y la música, se consideran los siguientes términos:

- **Frecuencia fundamental**: Es la frecuencia más baja de una onda sonora, que determina el tono principal de una nota musical.
- **Primer armónico**: También conocido como segundo armónico, tiene una frecuencia que es el doble de la frecuencia fundamental.
- **Segundo armónico**: Tiene una frecuencia que corresponde a la suma del primer armónico y la frecuencia fundamental.
- **Armónicos superiores**: Siguen esta misma lógica sucesivamente.

Por ejemplo, si una onda sonora tiene una frecuencia fundamental de 100 Hz, sus armónicos serán:

- Primer armónico (segunda frecuencia): 200 Hz
- Segundo armónico (tercera frecuencia): 300 Hz
- Tercer armónico (cuarta frecuencia): 400 Hz

Matemáticamente, si f_0 es la frecuencia fundamental, los armónicos se definen como:

$$f_n = n \cdot f_0$$

donde:

- f_n: frecuencia del n-ésimo armónico.
- n: número entero positivo (1, 2, 3, ...).

Los armónicos son importantes por varias razones:

- **Calidad del sonido**: Determinan el timbre o color de un sonido. Aunque dos instrumentos toquen la misma nota, el contenido armónico hace que se escuchen diferente.
- **Análisis de señales**: Se usan para descomponer y analizar señales periódicas en ingeniería y física.
- **Electrónica y telecomunicaciones**: En sistemas electrónicos, los armónicos pueden causar distorsión no deseada, por lo que es importante controlarlos.

En resumen, los armónicos son esenciales para comprender la estructura de las ondas periódicas y su calidad, tanto en el ámbito musical como en el técnico.

Overdrive

El efecto *overdrive*, aplicado comúnmente a guitarras eléctricas, consiste en una modificación de la onda de sonido que enfatiza ciertos armónicos, especialmente el segundo. Este efecto genera una señal con más picos, sin llegar a una distorsión completa, y ofrece un sonido cálido, ideal para géneros como el blues o el rock suave.

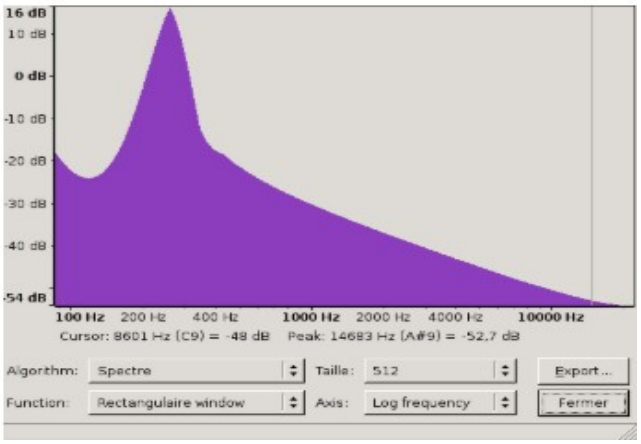

Figura 1: Onda normal.

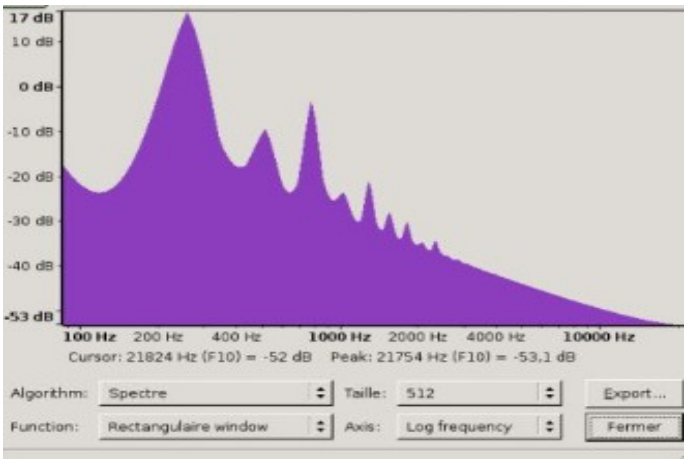

Figura 2: Onda con overdrive.

Fuzz

Es como el overdrive, solamente que con una mayor distorsión en la onda, haciendo que ahora esta tenga mas picos en los armónicos.

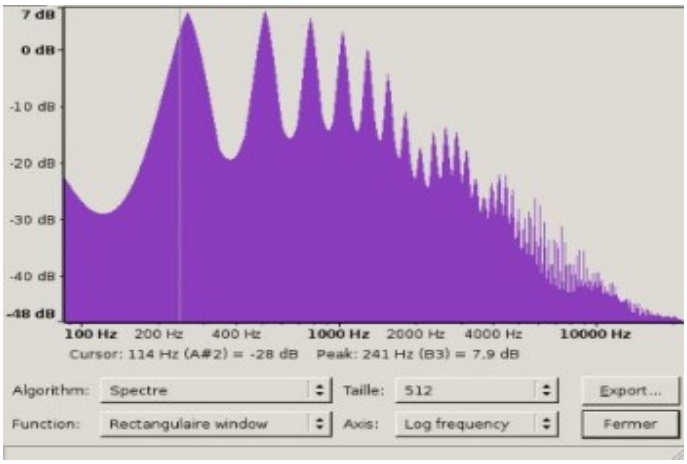

Figura 3: Onda con fuzz aplicado.

Distorsión

Este lo que tiene de diferencia es que afecta tambien al tercer armónico, y hace que tengan mas picos, pero que a su vez estos no sean tan grandes, y sean mas uniformes.

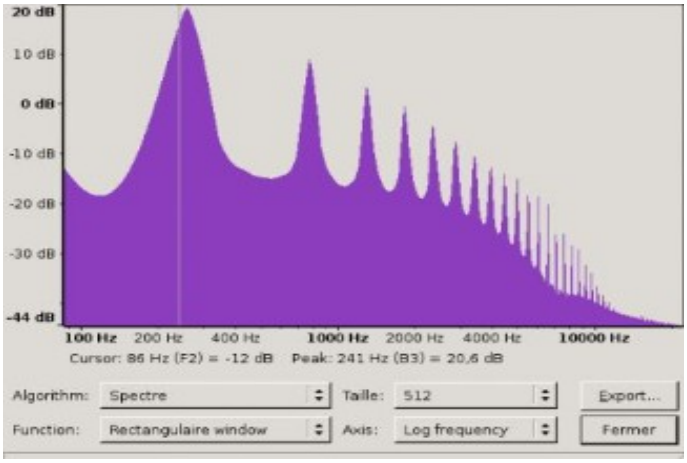

Figura 4: Onda con distorsión.

Investigación del sonido de un pedal de fuzz

Para poder entender que cambios se realizan al audio en un pedal de fuzz tradicional se realizo una simulacion.

Para esto primero se utilizó una nota ya grabada de un bajo eléctrico para luego pasarla por una simulación en LTspice, donde estaba un circuito de pedal de fuzz

genérico, para que después la salida de este circuito se guardara como archivo .WAV, donde se puede reproducir y analizar su onda con el software Audacity.

Aquí el circuito realizado para esto:

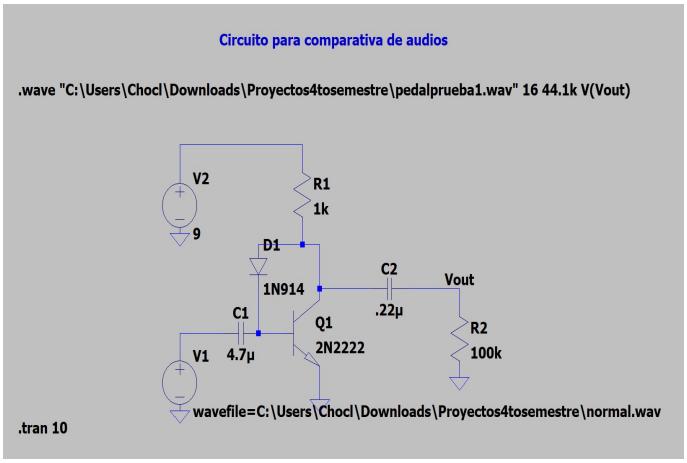

Figura 5: Simulación de procesamiento de audio.

Ya al realizar esto y poder correr la simulación, tenemos los dos archivos de onda para poder analizar, usamos un análisis transitorio de 10s para esto, aquí los resultados:

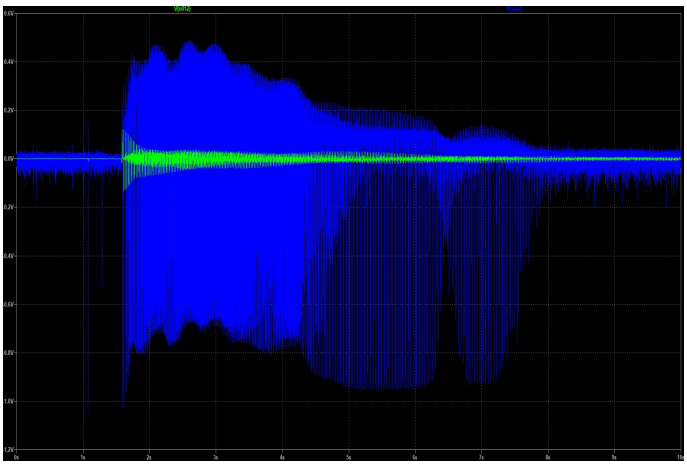

Figura 6: Simulación de onda.

Como podemos ver, el circuito de fuzz hace que se amplíe en gran medida la onda, pero no parece ser una amplitud lineal, es por esto que guardamos estas dos ondas como datos en un archivo .txt, el cual pasaremos por un código de Python, para poder graficar los datos de entrada contra los de salida, siendo el eje X la amplitud de la entrada del audio mientras Y es la amplitud de la salida del audio, obteniendo un resultado así con el código:

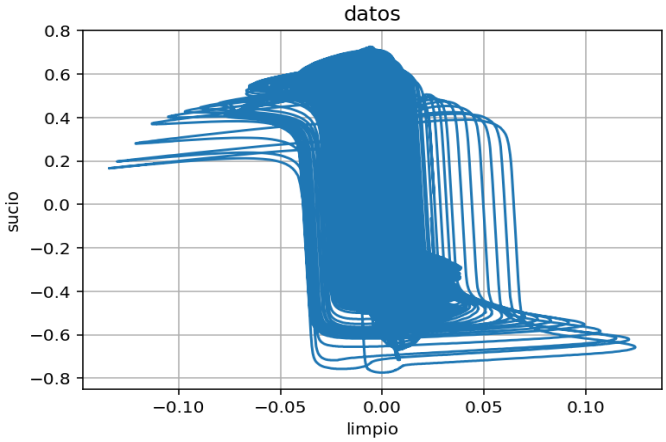

Figura 7: Graficación comparativa.

Con esta graficación tenemos la comparativa entre la entrada y salida del audio, como podemos ver, esta .ªmplificación”no es lineal, sino que parece más una función logística, conocida también como función en forma de S. Una vez obteniendo este resultado se ralizó una investigación sobre las fórmulas de estas ecuaciones.

Entrada del audio

Para poder realizar este proyecto primero se tuvo que pensar, ¿Cómo es la señal de una guitarra eléctrica?, esta normalmente es una onda de AC que suele ser de 100mv pico, el problema con esto es que el ESP32 en sus canales de entrada analógica no permite voltajes negativos, entonces para poder leer la señal de la guitarra, se tendrá que hacer un circuito de offset, el cual se alimenta con el mismo ESP32 y sube la onda para que oscile en 1.65V.

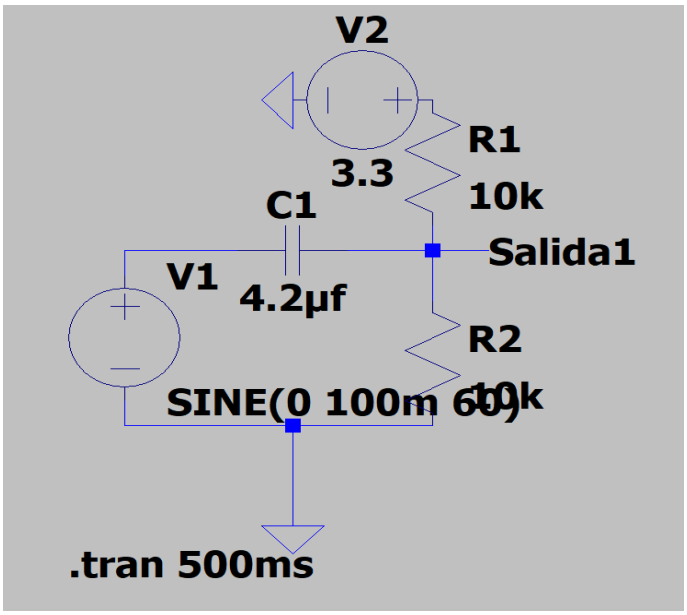

Figura 8: Circuito de offset.

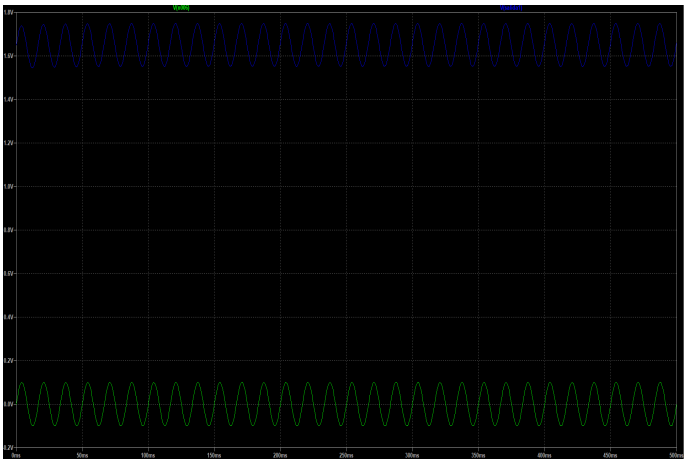

Figura 9: Simulacion offset.

Procesamiento de audio

Ya con el audio adentro del ESP32, se puede graficar usando el siguiente codigo

Código 1: Código de lectura analógica

```
void setup() {
        Serial.begin(9600);
}
```

```
void loop() {
        int val = analogRead(A0);
        Serial.println(val);
        delay(10);
}
```

El cual solamente estaría leyendo la entrada y mandándola por el canal serial, se pude usar para visualizar los datos en el Serial Plotter, o en la consola del mismo arduino IDE, Para poder usar esta información y confirmar que se esta leyendo de buena manera, se decidió hacer un código que se comunicase con Python, para que este guarde la información en un archivo .WAV.

Código 2: Código de guardado en .WAV

```
V.1.0.3
import serial #Se importan las librerias
import wave
import time

# Configura el puerto serial
puerto = serial.Serial('COM4', 115200) # Ajusta a tu puerto
time.sleep(2) # Tiempo de delay

# Parámetros de grabación
duracion = 5 # segundos
frecuencia_muestreo = 22050 # Hertz
num_muestras = duracion * frecuencia_muestreo # Se obtiene el numero de
    muestras multiplicando la duración y la frecuencia de muestreo

datos = bytearray() # Se define "datos" como la función bytearray

print("Grabando...") # Muestra la palabra mientras se hace lo siguiente

while len(datos) < num_muestras: # Cuando datos es menor a el numero de
    muestras
if puerto.in_waiting:
datos.append(puerto.read()[0]) # Lee 1 byte

print("Grabación␣terminada.")
puerto.close() # Muestra el mensaje y termina el proceso

# Guardar como .WAV
with wave.open("guitarra9.wav", "wb") as wav_file: # Abre el archivo de audio
wav_file.setnchannels(1) # Mono
```

```
wav_file.setsampwidth(1) # 8 bits
wav_file.setframerate(frecuencia_muestreo) # Se define la frecuencia
wav_file.writeframes(datos) # Se escriben los datos brindados

print("Archivo guardado como guitarra.wav") # Muestra como se llama el archivo
    y que ya quedo modificado
```

Con esto ya pudimos escuchar los datos que tenemos, escuchando un audio algo recortado, pero funcional. Ya con el poder meter los datos sin modificar, podemos empezar con el código que modificara los datos. Para esto se uso Geogebra para realizar una serie de ecuaciones que se parecieran a la grafica que se saco anteriormente:

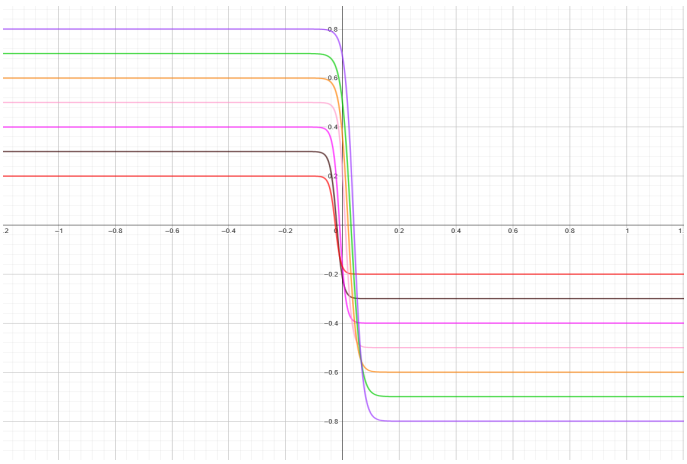

Figura 10: Ecuaciones en geogebra.

Ya con esta ecuaciones, se pueden implementar en el código, para que esto modifique los datos de entrada de una manera que simula lo que les pasa al entrar a un pedal de fuzz normal.

Código 3: Código de ecuaciones

```
//V.1.0.0
#define ENTRADA 34 // Define el pin de entrada analógica que se utilizará para
    leer datos

// Función para mapear un valor de un rango a otro rango
float mapFloat(float x, float in_min, float in_max, float out_min, float out_max) {
        // Aplica la fórmula de mapeo lineal para transformar el valor x
        return (x − in_min) ∗ (out_max − out_min) / (in_max − in_min) + out_min;
}

void setup() {
```

```
        Serial.begin(9600); // Inicializa la comunicación serie a 9600 baudios
        // Imprime los encabezados para el plotter de datos, separados por
            tabulaciones
        Serial.println("x\tR1\tR2\tR3\tR4\tR5\tR6\tR7");
}

void loop() {
        float e = 2.71828; // Valor de la constante matemática e (base de los
            logaritmos naturales)
        float lectura = analogRead(ENTRADA); // Lee el valor analógico del pin
            definido (ENTRADA)
        // Mapea la lectura del rango [0, 4096] al rango [−0.10, 0.10]
        float x = mapFloat(lectura, 0, 4096, −0.10, 0.10);

        // Se calcula los 6 resultados utilizando una función exponencial para cada
            uno
        float Resultado1 = (0.4 / (1 + pow(e, 100 * (x + 0.03)))) − 0.2;
        float Resultado2 = (0.6 / (1 + pow(e, 90 * (x + 0.02)))) − 0.3;
        float Resultado3 = (0.8 / (1 + pow(e, 90 * (x + 0.01)))) − 0.4;
        float Resultado4 = (1.0 / (1 + pow(e, 90 * (x + 0.01)))) − 0.5;
        float Resultado5 = (1.2 / (1 + pow(e, 70 * (x + 0.02)))) − 0.6;
        float Resultado6 = (1.4 / (1 + pow(e, 60 * (x − 0.03)))) − 0.7;
        float Resultado7 = (1.6 / (1 + pow(e, 65 * (x − 0.04)))) − 0.8;
        // Calcula el promedio de todos los resultados
        float ResultadoPROMEDIO = (Resultado1 + Resultado2 + Resultado3 +
            Resultado4 + Resultado5 + Resultado6 + Resultado7) / 7;

        // Se imprime x primero y luego los resultados, todo en una sola línea,
            separados por tabulaciones
        Serial.print(x, 4); Serial.print('\t'); // Imprime x con 4 decimales
        Serial.print(Resultado1, 6); Serial.print('\t'); // Se imprimen los Resultados
            del 1 al 7 con 6 decimales
        Serial.print(Resultado2, 6); Serial.print('\t');
        Serial.print(Resultado3, 6); Serial.print('\t');
        Serial.print(Resultado4, 6); Serial.print('\t');
        Serial.print(Resultado5, 6); Serial.print('\t');
        Serial.print(Resultado6, 6); Serial.print('\t');
        Serial.print(Resultado7, 6); Serial.print('\t');
        Serial.println(ResultadoPROMEDIO, 6); // Imprime el promedio y finaliza la
            línea

        delay(200); // Espera 200 milisegundos antes de la siguiente iteración del
            bucle
```

}

Despues combinamos los códigos para poder hacer la prueba en la computadora primero antes de pasarlo a la vida real.

Código 4: Código para la prueba del efecto .WAV

```
//V.1.0.0
#define ENTRADA 34 // Pin analógico conectado al circuito offset
//Definimos el pin de lectura de datos del sensor

//Mapeo del valor del rango con numeros flotantes
float mapFloat(float x, float in_min, float in_max, float out_min, float out_max) {
        return (x − in_min) ∗ (out_max − out_min) / (in_max − in_min) + out_min;
}
void setup() {
        Serial.begin(115200); // Aumenta la velocidad para evitar cuellos de botella
}

void loop() {
        float e = 2.71828; //Definir constante
        float lectura = analogRead(ENTRADA); //Leer datos del analogico
        float x = mapFloat(lectura, 0, 4096, −0.10, 0.10); //Mapear valor ya leido
            en el rango −0.10 y 0.10
        float Resultado3 = (0.8 / (1 + pow(e, 90 ∗ (x + 0.01)))) − 0.4;
        // Convertimos a 8 bits (0−255) centrado para ser más eficiente
        uint8_t muestra8bit = map(Resultado3, −10, 10, 0, 255);


        Serial.write(muestra8bit); // Enviar el valor al puerto serial no como texto
        //Serial.println(offsett);
        delayMicroseconds(45); // Aproximadamente 22.2 kHz pausa para lograr
            una frecuencia requerida
}
        }
```

Se notaba un cambio en el sonido de este archivo .WAV talvez no exactamente lo esperado, pero se empezaron a hacer mas pruebas con las demás ecuaciones.

Salida del audio

Ya con el hecho de que nuestros códigos están funcionando al poder meter el audio en un .WAV. y con la llegada de los amplificadores operacionales, ya que no se conto con estos en la parte de procesamiento de audio, ya podíamos sacar el audio a un amplificador real. Para poder sacar los datos se decidió usar el pin DAC

del ESP32, con esto podríamos hacer una onda desde 0V a 3.3V, pero como esta estaría oscilando todavía en 1.65V, se decidió hacer un circuito que quitara este offset con un amplificador operacional donde funcionaria como un sumador inversor.

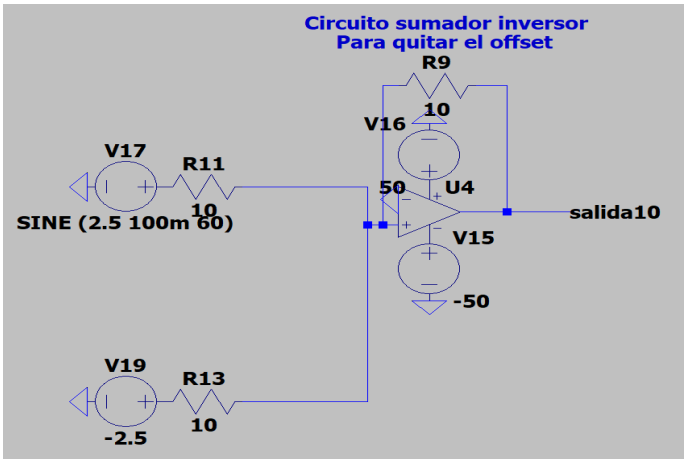

Figura 11: Circuito de sumador inversor.

Figura 12: Simulacion del sumador inversor

Ya con esto el audio oscilaría de la manera deseada en 0, con esto hecho se realizo un código para comprobar la salida del audio sin modificaciones.

Código 5: Código de salida normal

```
//V.1.0.5
// CODIGO DE SALIDA FUNCIONAL LIMPIO
#define ENTRADA 34 //Pin analógico (Entrada de audio)
#define pinDAC1 25 //Pin DAC1−GPIO25 (salida DAC)
```

```
void setup() {

}

void loop() {
        int LeerDatos = analogRead(ENTRADA); //Leer valor analógico de 0 a
            4095
        int8_t ValorDAC = map(LeerDatos, 0, 4095, 0, 255); //12 y 8 bits para el
            DAC
        dacWrite(pinDAC1, ValorDAC);//Escribir al DAC (0−255)
        delayMicroseconds(45);
}
        }
```

Donde se comprobó que los datos de entrada y salida del ESP32 se mantenían de buena manera, a pesar de meter algo de ruido. Ya con esto comprobado se realiza un código con las ecuaiones que modificaran nuestro audio.

Código 6: Código de salida modificada

```
//V.1.1.1
//CODIGO FINAL
#define ENTRADA 34 // Pin analógico conectado al circuito offset
#define pinDAC1 25 // Pin de salida del DAC integrado

// Mapeo con punto flotante
float mapFloat(float x, float in_min, float in_max, float out_min, float out_max) {
        return (x − in_min) ∗ (out_max − out_min) / (in_max − in_min) + out_min;
}

void setup() {

}

void loop() {
        float e = 2.71828;
        float lectura = analogRead(ENTRADA); // Leer señal con offset

        float x = mapFloat(lectura, 0, 4096, −1.0, 1.0);

        float Resultado = (0.4/ (1 + pow(e, 10 ∗ (x−0.03)))) − 0.2;
        float Resultado2 = (0.6 / (1 + pow(e, 90 ∗ (x + 0.02)))) − 0.3;
        float Resultado3 = (0.8 / (1 + pow(e, 90 ∗ (x + 0.01)))) − 0.4;
        float Resultado4 = (1.0 / (1 + pow(e, 90 ∗ (x + 0.01)))) − 0.5;
        float Resultado5 = (1.2 / (1 + pow(e, 70 ∗ (x + 0.02)))) − 0.6;
```

```
        float Resultado6 = (1.4 / (1 + pow(e, 60 * (x - 0.03)))) - 0.7;
        float Resultado7 = (1.6 / (1 + pow(e, 65 * (x - 0.04)))) - 0.8;

        float ResultadoAmplificado = Resultado * 1.0; // Ganancia ajustable
        ResultadoAmplificado = constrain(ResultadoAmplificado, -1.0, 1.0); //
            Seguridad

        uint8_t muestra8bit = (uint8_t) mapFloat(ResultadoAmplificado, -1.0, 1.0,
            0, 255);

        dacWrite(pinDAC1, muestra8bit); // Salida al DAC


        delayMicroseconds(45); //≈100 Hz de muestreo, podés bajarlo para mayor
            resolución de audio
}
```

Probando con diferentes valores se decanto por estos, con los cuales ya se obtiene un sonido distorsionado tipo fuzz.

Problemas y cambios

En el proyecto se presentaron varias dificultades desde el planteamiento, empezando con el hecho de querer implementar varios efectos, el tiempo para obtener solamente 1 de estos efectos fue mas largo de lo previsto, por eso se decanto por quedarnos con el que empezamos el proyecto. Mientras que el querer implementar una pantalla que grafique la onda se considero en el planteamiento, pero se tuvo que descartar por el hecho que al usar el canal serial en el ESP32, al mismo tiempo que el DAC, causaba una gran distorsión en los datos del DAC, lo cual lo volvía prácticamente inutilizable. Curiosamente se podría tomar como un código alternativo el uso del canal serial al mismo tiempo que la escritura en limpio con el DAC, ya que esto causa una distorsión simple. También se consideraron códigos alternos que se encontraron en internet, uno por ejemplo una función tangente para modificar los datos, pero nos terminamos decantando por lo que habíamos investigado ya que era parte del proyecto. El mayor cambio fue el uso del ESP32 para realizar la parte del procesamiento en lugar del Arduino UNO que se planeó en el planteamiento, esto debido a que el Arduino UNO no cuenta con un canal DAC, entonces solamente se podrían haber utilizado las salidas con PWN, para simular una onda, lo cual hubiera sido mas complicado, a su vez el Arduino UNO contaba con una menor frecuencia de datos de entrada y de salida, lo cual dificultaba el manejo del sonido, por eso se decanto por el ESP32 ya que cuenta con mayores frecuencias y un canal DAC, a su vez que ya se contaba con uno.

Resultados

En el repositorio se cuentan con los audios y videos de las pruebas/resultados, obtenidos a lo largo de este proyecto. Algunos comentarios generales serian. En la parte de los archivos .WAV, se nota el como no contar con un amplificador operacional los datos de entrada se perdían mucho, pero a pesar de esto se pudieron obtener audios con información. En los videos del circuito terminado se aprecia la gran amplificación del amplificador operacional, se podría agregar un potenciómetro en la salida para reducir en cierta medida esto, a pesar de eso se nota el sonido deseado en el pedal. El circuito se termino realizando en una placa preperforada por su versatilidad, aqui una imagen del circuito ya montado.

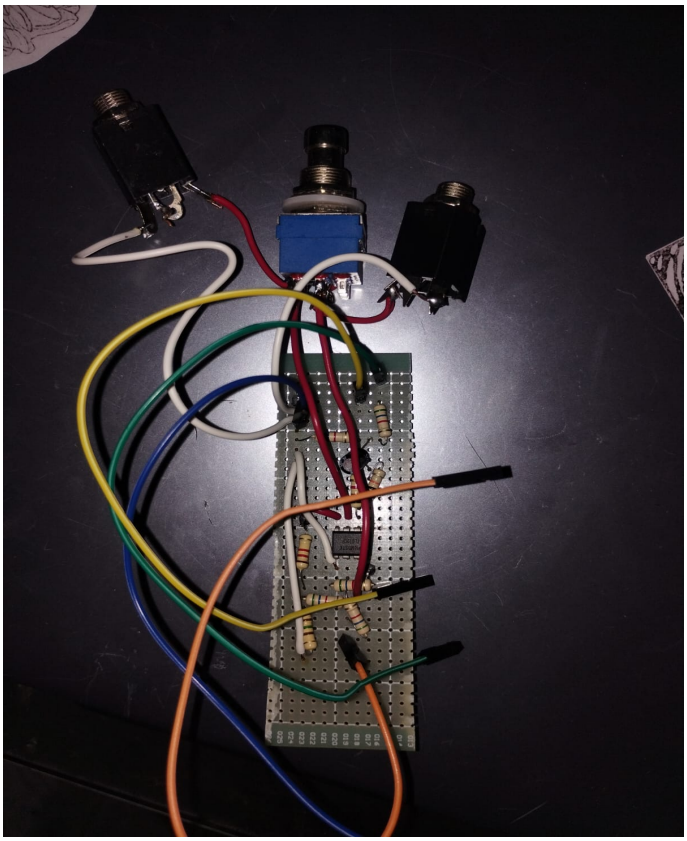

Figura 13: Circuito armado.

Se usaron los circuitos hechos simulados anteriormente para poder crear este ultimo, aqui sus conexiones que lleva.

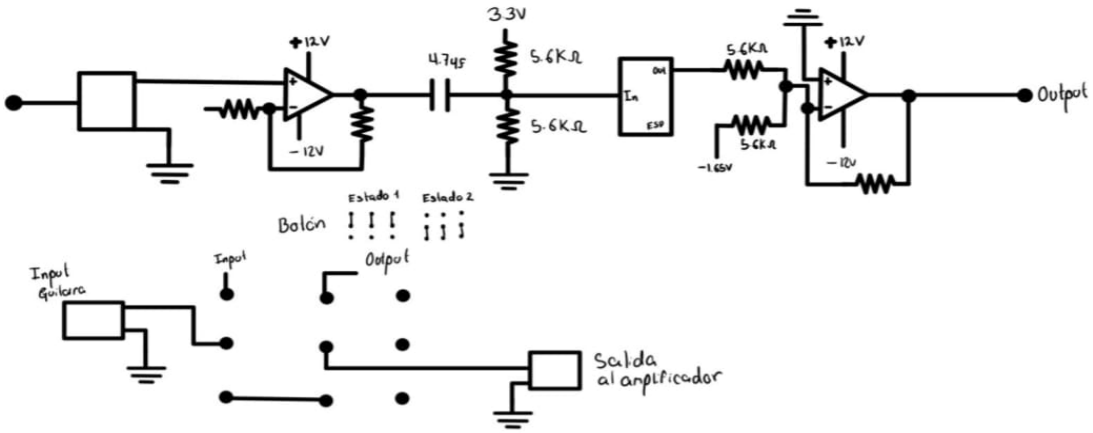

Figura 14: Diagrama del circuito.

En el siguiente link puede consultar las pruebas de audio en formato .WAV y las pruebas de audio real, junto a todos los codigos utilizados para el proyecto, se hayan mencionado en este o en otro pdf anterior a este.

`https://github.com/Choclotherock/Equipo-NANA` En los apartados *Proyecto/-PRUEBASDIGITALES*, *Proyecto/VIDEOS* y *Proyecto/PRUEBAS DE AUDIO*.

Manual de usuario

Para la realización de nuestro manual de usuario, utilizamos el software *TeXstudio*, el proceso de este manual fue dividido en secciones que realizaría cada uno de los miembros del equipo. Este manual contiene información importante sobre como el usuario debe manipular el dispositivo realizado.

El manual puede ser consultado en `https://github.com/Choclotherock/Equipo-NANA`, en el apartado *Proyecto/PDFs*.

Posibles mejoras o desarrollos futuros

Este proyecto nos enfocamos mas en la parte de codificación y en el usar un poco de todas las habilidades aprendidas en este curso, talvez se pudiera investigar mas formas de modificar el sonido para poder implementarlas al ESP32, así teniendo mas gama de selección de efectos. Para el circuito físico, se podría implementar una fuente de voltaje en el propio circuito, para que este solo sea necesario conectarlo a la corriente eléctrica de 120V, haciéndolo así mas practico. Se le podría agregar una carcasa que proteja el circuito ayudando a su transporte y buen uso. Un indicador LED para el circuito ayudaria a diferenciar entre el efectro prendido o apagado.

Conclusiones

El proyecto cumplió con los objetivos planteados. A pesar de los inconvenientes, logramos obtener los resultados esperados. Con ello comprendimos más a fondo la estructura de un código en Python y Arduino para un ESP32, además de obtener un mayor conocimiento sobre como funcionan las ondas sonoras y como estas pueden ser transformadas para obtener diferentes sonidos.

Referencias

Admin. (2023). *Decibel*. BYJUS. https://byjus.com/physics/decibel/

Baez, M. (2013). *Distorsión, overdrive y fuzz; diferencias y usos de pedales*. Guitarristas.info. https://www.guitarristas.info/tutoriales/distorsion-overdrive-fuzz-diferen
2991

El tutorial de Python. (s. f.). Python Documentation. https://docs.python.org/
es/3/tutorial/index.html

Gonzalez, L. (2022). *Introducción a la Librería Matplotlib de Python*. Aprende IA.
https://aprendeia.com/libreria-de-python-matplotlib-tutorial-practico/

Hertz. (s. f.). https://sistemas.com/hertz.php

La librería científica Scipy — documentación de Curso de Python para Astronomia - 20191128. (s. f.). https://research.iac.es/sieinvens/python-course/scipy.html

Prado, V. (2024). *Qué es un pedal de fuzz y por qué deberías usarlo*. El Blog de Stringsfield. https://www.stringsfield.com/blog/que-es-un-pedal-de-fuzz-y-por-que-deberia

Sáez, A. (2018). *Breve introducción a la librería NumPy*. Infinitos Contrastes.
https://imalexissaez.github.io/2018/08/18/breve-introduccion-a-la-libreria-numpy/

Webmaster. (2023). *Definición de Armónico: Que es, 5 Ejemplos, Tipos y Para que Sirve + Sinónimo y Significado*. SignificadosWeb.com. https://significadosweb.
com/definicion-de-armonico-que-es-ejemplos-tipos-y-para-que-sirve-sinonimo-y-signific