



UNIVERSIDAD DE BUENOS AIRES
Facultad de Ingeniería
86.65 Sistemas Embebidos

Memoria del Trabajo Final:

Shapeshifter: Pedal de Distorsión Programable

Autor:

Lautaro Javier De Lucia

Legajo: 100203

*Este trabajo fue realizado en las Ciudad Autónoma de Buenos Aires,
entre marzo y julio de 2025.*

RESUMEN

El *Shapeshifter* es un pedal de distorsión digital programable. La mayoría de los pedales de distorsión del mercado son analógicos, lo que limita las características electroacústicas del dispositivo al circuito particular de distorsión y sus componentes. En cambio, en tanto estas distintas distorsiones analógicas puedan simularse digitalmente, un mismo microcontrolador puede implementar cientos de modelos de pedal diferentes. Luego, el shapeshifter fue creado precisamente como un pedal de distorsión digital que puede emular distintos pedales del mercado (Boss DS-2, Tube Screamer, etc.), recibiendo esta configuración electroacústica de forma remota.

El proyecto está desarrollado utilizando C++ y Mbed en una placa NUCLEO de ARM, Este trabajo es realizado en el marco de la materia Sistemas Embebidos de la Facultad de Ingeniería de la Universidad de Buenos Aires y en él se aplican los temas aprendidos en la materia en particular y en la carrera de Ingeniería Electrónica en general.

En estas memorias se presenta el proyecto, su planificación y motivación, la información técnica sobre el *hardware* y *firmware*, así como la explicación de su diseño y organización.

Índice General

Registro de versiones	4
Introducción general	5
1.1 Objetivo y Beneficio de la Iniciativa	5
1.2 Análisis de Sistemas Similares	6
1.3 Diagrama de Bloques de Alto Nivel	7
Introducción específica	8
2.1 Requisitos del Proyecto	8
2.2 Casos de Uso	9
2.3 Módulos Utilizados	10
Diseño e implementación	12
3.1 Hardware - Sistema Principal	12
3.2 Hardware - Preamplificador	14
3.3 Software - Flujo Principal	15
3.4 Firmware - Integración HW/SW	16
Ensayos y resultados	21
4.1 Pruebas sobre módulos individuales	21
4.2 Pruebas de Integración	32
4.3 Evaluación del Cumplimiento de Requisitos	33
4.4 Documentación del Desarrollo Realizado	34
Conclusiones	35
5.1 Resultados obtenidos	35
5.2 Próximos pasos	36
Bibliografía	37

Registro de versiones

Revisión	Cambios realizados	Fecha
1.0	Creación del documento	13/07/2025
1.1		
1.2		

CAPÍTULO 1

Introducción general

1.1 Objetivo y Beneficio de la Iniciativa

El mercado de pedales de distorsión está dominado por dispositivos analógicos cuyas características acústicas están determinadas por sus parámetros constructivos. En este informe se presenta la planificación y desarrollo del pedal de distorsión programable *Shapeshifter*, diseñado para permitir simular digitalmente múltiples tipos de distorsión, configurables mediante comunicación remota.

Así, el *Shapeshifter* tiene la forma de un pedal de efectos convencional. Pero a diferencia de este, sus parámetros electroacústicos no se fijan por hardware ni por componentes analógicos, sino que se reciben dinámicamente a través de una conexión HTTP, en forma de mensaje JSON enviado desde un servidor remoto o aplicación.

De esta forma, permitimos al usuario acceder a una variedad de pedales de distorsión populares en un solo dispositivo. El usuario simplemente tiene que acceder a el preset específico de la distorsión que quiere simular y grabarla en el dispositivo. Si, por ejemplo, este producto fuese parte de la empresa *ArDFX SRL*, la empresa podría ofrecer una app descargable desde la Play Store que incluya los presets que pueden grabarse en el dispositivo. Luego, el usuario solo tendría que abrir un menú y seleccionar el tipo de distorsión que quiere en su dispositivo, y la app se encargaría de enviar un HTTP POST con los parámetros de esta distorsión como los campos de un JSON, los cuales pasan a integrarse al algoritmo de distorsión digital del dispositivo.

1.2 Análisis de Sistemas Similares

La mayoría de pedales de distorsión disponibles en el mercado son dispositivos compactos, analógicos y con controles físicos. En este contexto, el Shapeshifter se diferencia como un pedal digital experimental, programable, capaz de alternar entre diferentes modos de operación y efectos en tiempo real.

A continuación se presenta una tabla comparativa entre el proyecto Shapeshifter y dos pedales de distorsión populares en el mercado. Esta comparativa evidencia que, si bien los pedales tradicionales ofrecen una experiencia inmediata y probada, el Shapeshifter introduce a un precio similar capacidades de interacción, customización y grabación que no están presentes en la competencia.

Característica	BOSS DS-1	ProCo RAT 2	Shapeshifter
Tipo de procesamiento	Analógico	Analógico	Digital
Tipo de distorsión	Clipping simétrico por diodos	Clipping op-amp + diodos	Seleccionable por software
Controles físicos	Nivel, Tono, Distorsión	Filtro, Distorsión, Volumen	1 botón selector + JSON remoto
Configurable por software	No	No	Sí (HTTP)
Capacidad de grabación (output REC)	No	No	Si (MicroSD)
Salida directa a amplificador	Sí	Sí	Sí (modo DAC)
Display Informativo	No	No	Si (I2C)
Precio estimado (USD)	~60	~90	~60

Tabla 1.2: Análisis comparativo de Alternativas.

1.3 Diagrama de Bloques de Alto Nivel del Sistema

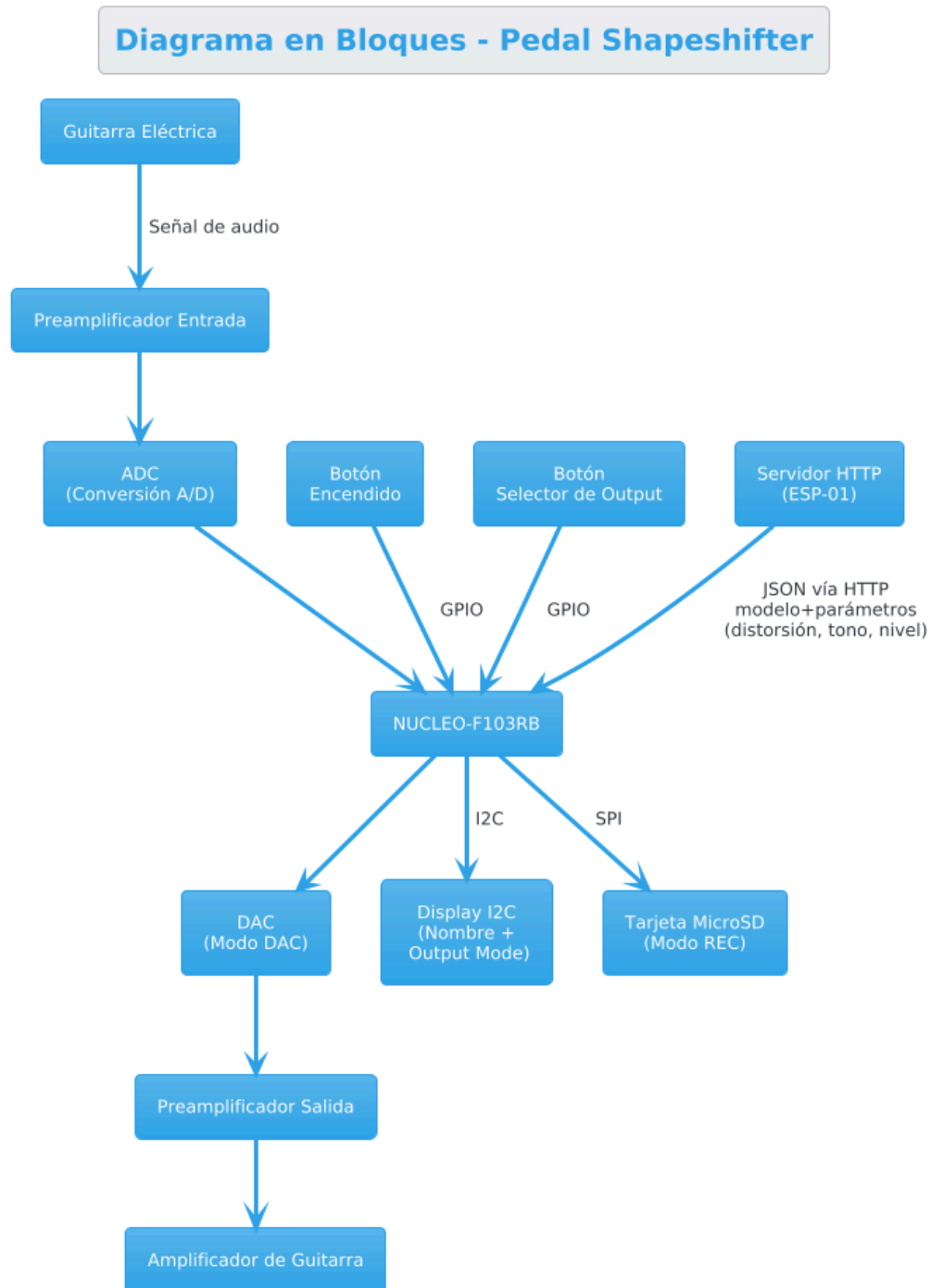


Figura 1.3: Diagrama de Bloques de Alto Nivel

CAPÍTULO 2

Introducción específica

2.1 Requisitos

Grupo	ID	Descripción
1. Firmware	1.1	Implementación de algoritmo de distorsión configurable por software
	1.2	Lógica de manejo de modos DAC/REC mediante FSM.
	1.3	Lectura del ADC, escritura del DAC.
	1.4	Procesamiento de JSON recibido por HTTP para definir el preset activo.
2. Hardware	2.1	Conexión a preamplificador para entrada de guitarra analógica.
	2.2	Conexión a DAC y preamplificador de salida para amplificador externo
	2.3	Tarjeta MicroSD conectada por SPI para grabación.
	2.4	Display I2C para mostrar estado del sistema
	2.5	Botón de encendido y botón selector de modo conectados a pines GPIO
3. Interfaz de Usuario	3.1	Mostrar nombre del preset y tipo de salida (REC/DAC) en pantalla
	3.2	Alternar entre modos mediante botón físico.
	3.3	Cambiar dinámicamente la pantalla según configuración recibida
4. Configuración	4.1	Escuchar conexiones entrantes en puerto configurable via Wi-Fi o Ethernet
	4.2	Interpretar mensajes de configuración con campos: tipo, distorsión, tono, nivel, etc.

Tabla 2.1: Requisitos del proyecto.

2.2 Casos de uso

Elemento	Definición
Disparador	El usuario presiona el botón selector de modo.
Precondición	El sistema está encendido y listo.
Flujo básico	El sistema cambia entre modo DAC y RAC, actualiza la pantalla y ajusta la salida.

Tabla 2.2.1: Selección de Modo

Elemento	Definición
Disparador	Se recibe un JSON de configuración válido por HTTP
Precondición	El sistema está conectado a la red.
Flujo básico	El sistema parsea los parámetros, aplica la configuración y actualiza el preset activo.

Tabla 2.2.2: Comunicación Remota

Elemento	Definición
Disparador	El sistema detecta señal de guitarra en modo REC
Precondición	La tarjeta MicroSD está correctamente insertada.
Flujo básico	La señal procesada se graba como archivo en la tarjeta SD

Tabla 2.2.3: Grabación en la Tarjeta MicroSD

2.3 Módulos Utilizados

2.3.1 Display I2C

El sistema incluye un módulo LCD de 16x2 caracteres con interfaz I2C integrada. Este módulo permite mostrar en pantalla el nombre del preset de distorsión activo, así como el modo de operación actual del pedal (salida directa o grabación).

Gracias a la interfaz I2C integrada, la comunicación se realiza utilizando únicamente dos líneas (SDA y SCL), lo que simplifica el cableado y libera pines del microcontrolador. El contraste del display se regula mediante un potenciómetro integrado, y se alimenta con 3.3 V.



Figura 2.3.1: Display I2C

2.3.2 Lector de Tarjeta MicroSD

Para implementar la funcionalidad de grabación (modo REC), se utiliza un lector de tarjetas MicroSD basado en el estándar SPI. Este módulo permite almacenar la señal de audio procesada directamente en formato digital, facilitando su análisis posterior.

La comunicación SPI entre el microcontrolador y la tarjeta se gestiona mediante los pines MOSI, MISO, SCK y CS, y el formato de archivo generado puede ser fácilmente recuperado desde una PC para tareas de edición o prueba.

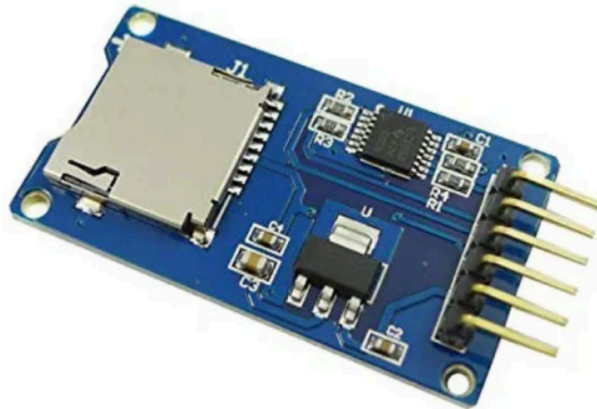


Figura 2.3.2: Lector de Tarjeta MicroSD

2.3.3 Módulo Wi-Fi ESP-01

El módulo ESP-01 basado en el chip ESP8266 se utiliza para establecer la comunicación remota con el sistema. Su función principal es recibir parámetros de configuración (modelo de distorsión, ganancia, tono, nivel) mediante requests HTTP POST con formato JSON.

Estos parámetros son luego enviados al microcontrolador a través de la interfaz UART. Gracias a este módulo, el pedal puede ser configurado dinámicamente desde una aplicación móvil o una interfaz web, sin necesidad de manipulación física.



Figura 2.3.3: Módulo Wi-Fi

CAPÍTULO 3

Diseño e implementación

3.1 Hardware - Sistema Principal

El sistema principal del pedal de distorsión programable Shapeshifter está basado en el microcontrolador STM32F103RB, montado sobre una placa NÚCLEO. Este microcontrolador actúa como centro de control del dispositivo, coordinando la adquisición de señal, el procesamiento digital, la visualización de parámetros, la grabación y la configuración remota. La entrada de audio se realiza a través de un preamplificador conectado a uno de los pines analógicos, que luego es muestreado mediante el ADC interno.

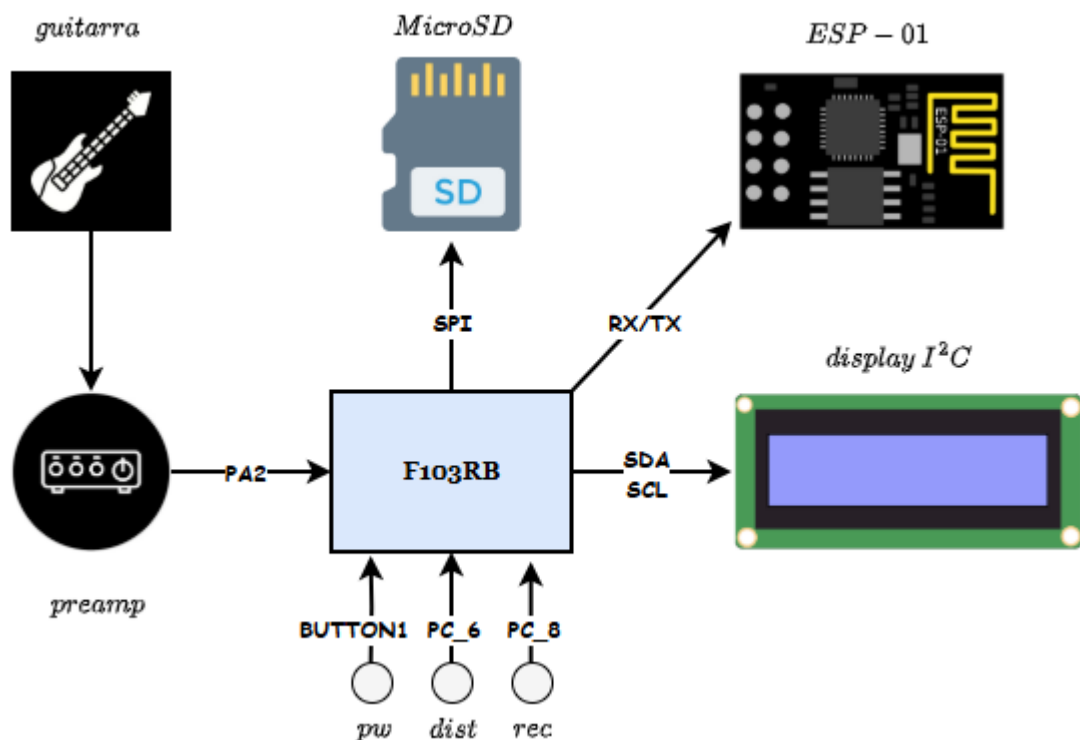


Figura 3.1.1: Diagrama de Bloques de HW de Alto Nivel

Para la comunicación con periféricos digitales, se utilizan diversas interfaces estándar. El **módulo Wi-Fi ESP-01** se conecta por UART a los pines **D2 (RX)** y **D8 (TX)** del microcontrolador, permitiendo la recepción de parámetros de configuración a través de mensajes JSON por HTTP. Por otro lado, la interfaz **I2C** permite controlar el **display LCD**, conectado a los pines **D14 (SDA)** y **D15 (SCL)**, donde se muestra en tiempo real el modo de operación y el nombre del preset activo.

El sistema también incluye un **lector de tarjetas MicroSD** para grabación de la señal procesada (modo REC). Este periférico se comunica mediante la interfaz **SPI**, empleando los pines **PB12 a PB15** para las señales de **CS, SCK, MISO y MOSI**, respectivamente. Además, tres botones físicos, **BUTTON1** del microcontrolador y dos conectados a los pines **PC_6 y PC_8** permiten al usuario encender el sistema, alternar entre modos de operación y activar la grabación.

En conjunto, la arquitectura del hardware está diseñada para ofrecer una solución flexible y expandible, capaz de integrar múltiples fuentes de entrada y salida sin comprometer la simplicidad del diseño físico. Las figuras presentadas a continuación ilustran tanto la vista funcional general del sistema como el esquema detallado de conexiones entre el microcontrolador y los módulos periféricos.

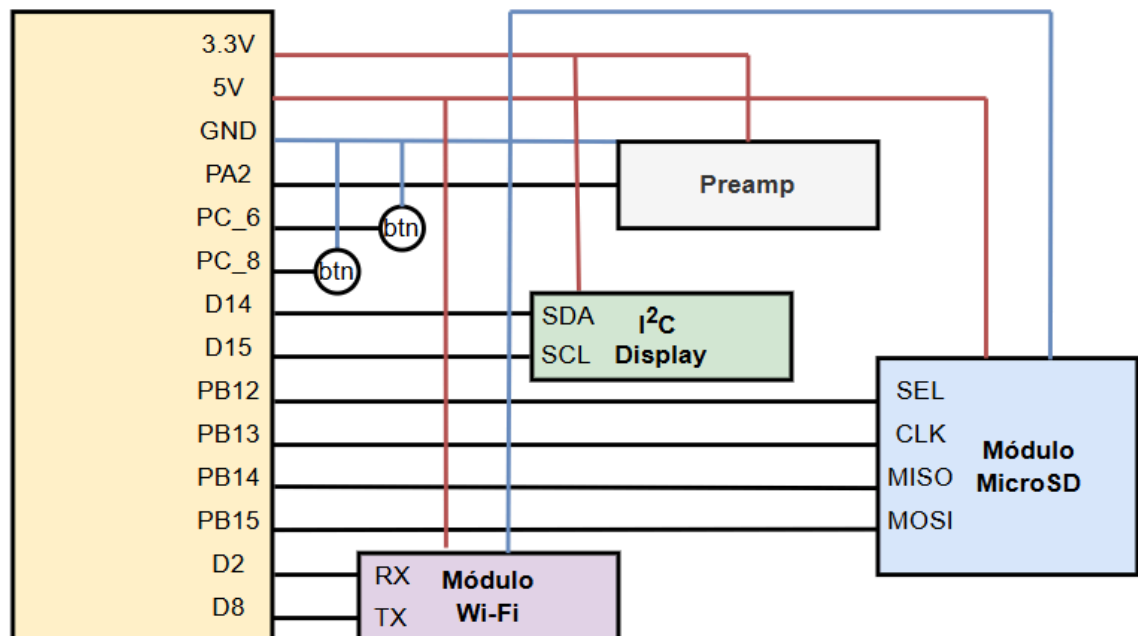


Figura 3.1.2: Diagrama de Conexiones de HW

En la siguiente figura podemos ver una foto del prototipo armado.

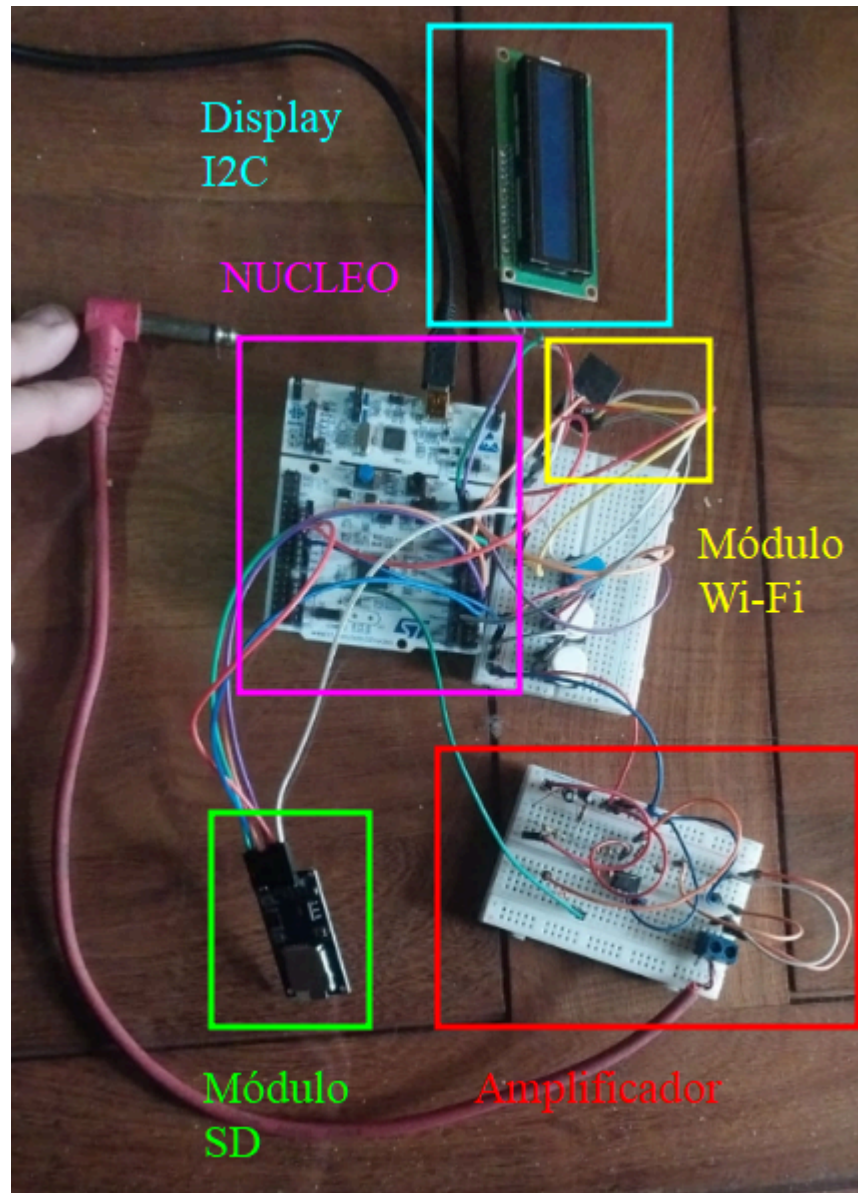


Figura 3.1.3: Fotografía del Circuito

3.2 Hardware - Preamplificador

Ordinariamente, la señal de una guitarra eléctrica no excede un V_{pp} de 1V. Cuando conectamos la guitarra directamente a un osciloscopio la señal esta centrada en GND y no excede una amplitud de 0.5V. Dado el rango del ADC de 0V a 3.3V, se volvió necesario diseñar una etapa preamplificadora que por un lado eleve la señal de la guitarra

centrandola en 1.65V y al mismo tiempo la amplifique con una ganancia conservadora entre 2 y 3.

Para esto, el amplificador operacional MCP6001 resulto ideal, dado su bajo consumo y alta fidelidad. A continuación, se muestra el esquemático y la implementación. Se observa que estamos utilizando un divisor resistivo para polarizar V_{in} a 1.65V y el amplificador en configuración no inversora con una ganancia $A_v = 1 + R_f/R_1 = 2$. El alto valor de los resistores es una elección deliberada para limitar la corriente (ya que la NUCLEO puede proporcionar un máximo de 300mA y A2 no es el único puerto consumiendo corriente).

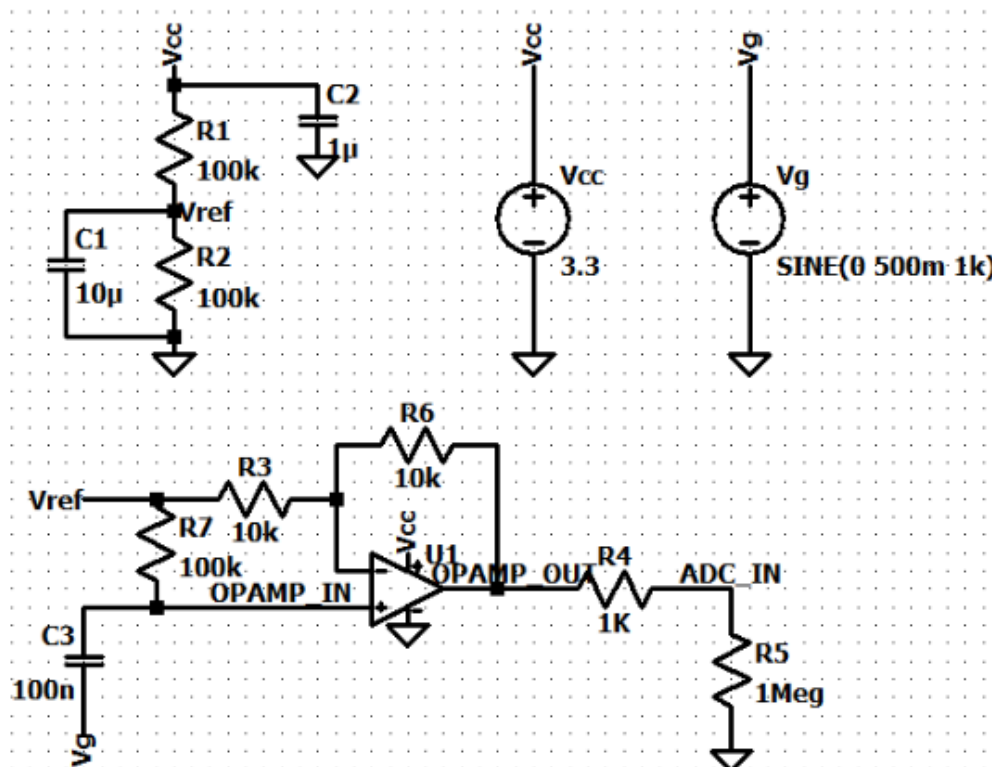


Figura 3.2: Esquemático del Circuito Amplificador

3.3 Software - Flujo Principal

El siguiente diagrama de flujo conceptual ilustra el funcionamiento general del sistema. El sistema se basa en un ciclo principal que ejecuta tareas de control y monitoreo, junto con un flujo paralelo que gestiona el procesamiento de audio en función de interrupciones periódicas.

El bucle principal inicia con la inicialización del hardware y los objetos del sistema, incluyendo la máquina de estados finitos del pedal (FSM), controladores, y temporizadores (Tickers). Luego, en cada iteración del ciclo, se realiza la lectura del

estado de los botones físicos y se actualiza la FSM en función de las entradas del usuario. A continuación, se actualizan los controladores correspondientes al estado (LEDs), a la interfaz visual (Display I2C), la comunicación remota (Módulo Wi-Fi) y al procesamiento de audio.

El flujo de procesamiento de audio, mostrado en paralelo, inicia verificando si el pedal se encuentra encendido. Si está activo, se procede a leer una muestra de la señal de entrada a través del ADC. En caso de que la distorsión esté habilitada, se aplica el algoritmo correspondiente a la muestra. Finalmente, según el modo activo (grabación o salida directa), la muestra se envía a la tarjeta MicroSD o al DAC para ser reproducida en tiempo real.

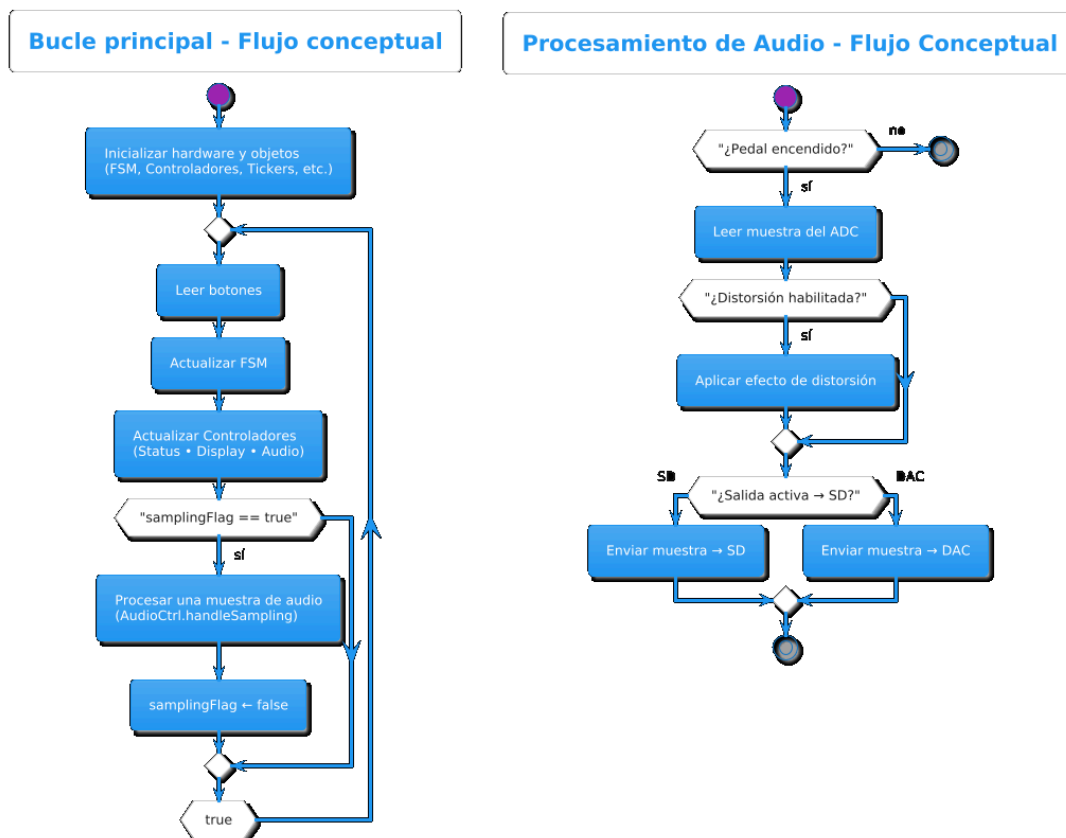


Figura 3.3: Flujo Principal del Programa

3.4 Firmware - Controladores e Integración SW/HW

3.4.1 Principio de Diseño

Para el desarrollo del firmware del sistema se optó por una arquitectura modular basada en clases, donde cada componente funcional del dispositivo (input usuario, visualización, procesamiento de audio, estado del sistema, etc.) es representado por una **clase controladora dedicada**. Este enfoque facilita la escalabilidad, la legibilidad del código y la depuración por separado de cada subsistema.

El módulo `main` actúa como orquestador del sistema, instanciando los diferentes controladores y ejecutando el bucle principal. A su vez, cada clase controladora encapsula el comportamiento y estado de un periférico o bloque lógico, promoviendo la separación de responsabilidades. Este modelo también favorece la integración fluida entre hardware y software, ya que abstrae detalles de bajo nivel (como lectura de pines o comunicación por bus) detrás de interfaces claras y reutilizables. A continuación, presentamos una breve descripción de cada módulo y su funcionalidad en el sistema.

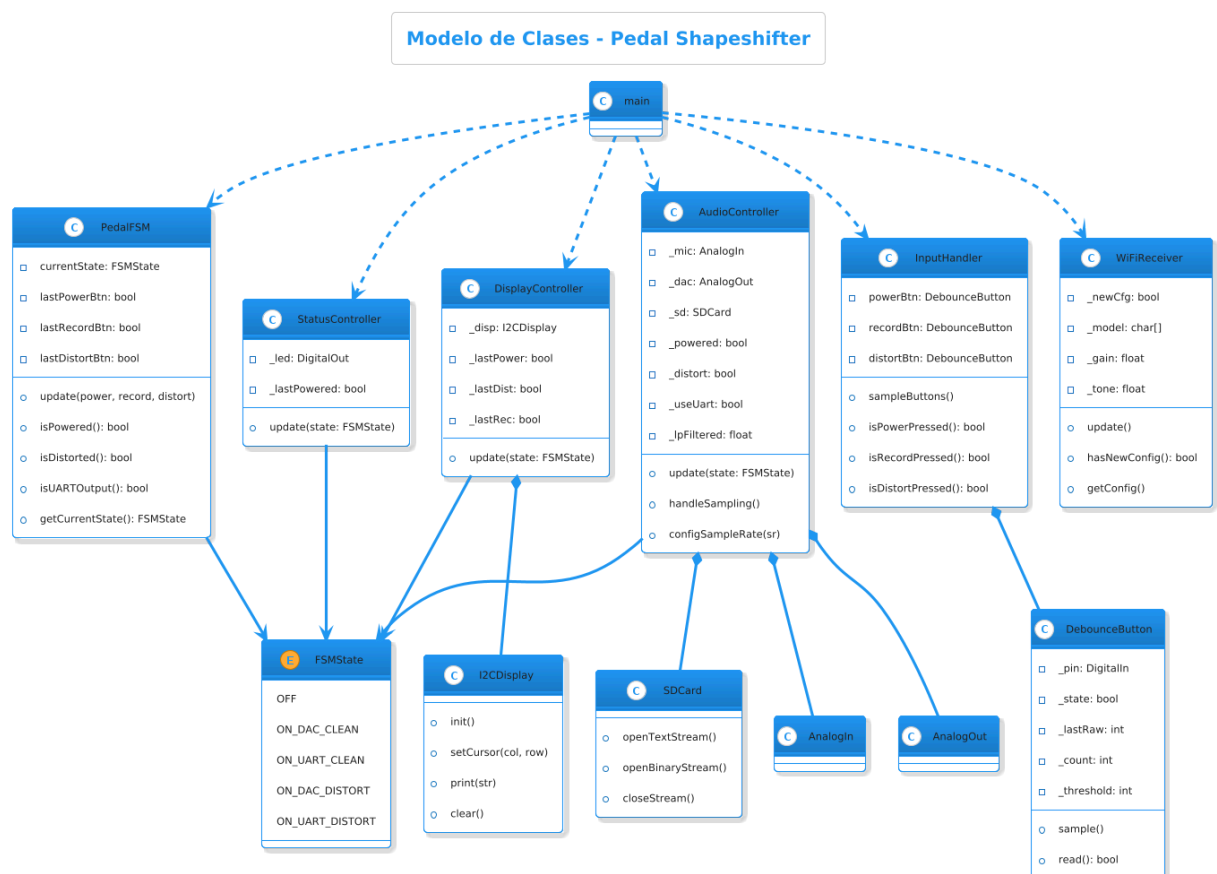


Figura 3.4.1: Diagrama de Clases del Sistema

3.4.2 PedalFSM

La clase `PedalFSM` implementa una máquina de estados finitos (FSM) encargada de modelar los modos de operación del pedal de distorsión. Utiliza un enumerador `FSMState` que define cinco estados: apagado, limpio con salida DAC, limpio con salida a tarjeta SD, distorsionado con salida DAC, y distorsionado con salida a tarjeta SD.

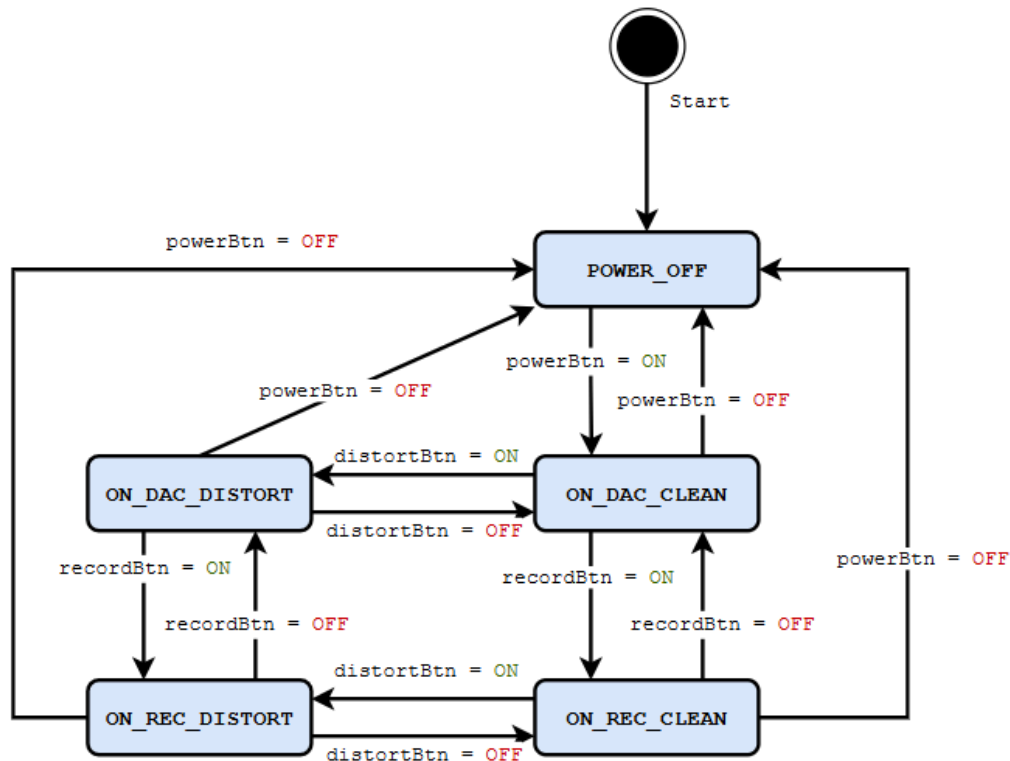


Figura 3.4.2: FSM del Pedal Shapeshifter

La FSM se actualiza a partir de los botones presionados, detectando transiciones de estado mediante variables auxiliares que almacenan el valor anterior de cada botón (`lastPowerBtn`, etc.). También expone métodos que permiten consultar el estado actual del sistema, como `isPowered()` o `isDistorted()`, lo que facilita la lógica de control en el bucle principal.

3.4.3 InputHandler

La clase `InputHandler` se encarga de la lectura y debouncing de los botones físicos del sistema: encendido, grabación y distorsión. Cada botón es instanciado como un objeto de

la clase **DebounceButton**, que se encarga de eliminar rebotes mecánicos mediante una lógica de muestreo con umbral.

El método **sampleButtons()** actualiza el estado interno de los tres botones, y métodos como **isPowerPressed()** permiten consultar si el botón fue presionado con fiabilidad, facilitando una interfaz limpia hacia el resto del sistema.

3.4.4 StatusController

La clase **StatusController** gestiona el indicador LED de estado del sistema. Su método **update(state)** enciende o apaga el LED en función del estado de la FSM, utilizando una lógica simple basada en cambios de encendido. Esta clase permite una retroalimentación visual mínima para depuración o uso final del dispositivo, sin ocupar recursos adicionales.

3.3.5 DisplayController

DisplayController es responsable de la **interfaz visual del usuario**, utilizando un display LCD controlado por la clase auxiliar **I2CDisplay**. Este controlador compara el estado actual con el último renderizado, y solo actualiza la pantalla si hay cambios en el modo de salida, distorsión o encendido, optimizando así el uso de recursos.

Este enfoque evita escritura redundante sobre el bus I2C y mejora la claridad visual del sistema.

3.3.6 AudioController

La clase **AudioController** constituye el núcleo del **procesamiento de señal de audio**. Gestiona tanto la entrada (mediante **AnalogIn**) como la salida (a través de **AnalogOut** y **SDCard**) en función del estado de la FSM.

El método **handleSampling()** se invoca periódicamente en respuesta a interrupciones y se encarga de adquirir una muestra del ADC, aplicar el efecto de distorsión (si está habilitado), y dirigir la salida hacia el DAC o hacia la tarjeta SD, según el modo activo. También incluye un filtro pasa bajos (**_lpFiltered**) para suavizado de la señal, y métodos para configurar la frecuencia de muestreo si se desea escalar el sistema.

3.3.7 SDCard

SDCard encapsula la **interfaz de almacenamiento persistente** del sistema. Brinda métodos simples para abrir flujos de texto o binarios (**openTextStream**, **openBinaryStream**) y cerrarlos una vez finalizada la grabación. Este diseño facilita la

escritura desde `AudioController` sin acoplar directamente el código de bajo nivel del sistema de archivos.

3.3.8 WiFiReceiver

La clase `WiFiReceiver` abstrae la **recepción de configuraciones externas por red**, a través del módulo ESP-01. Su método `update()` escucha mensajes entrantes y actualiza internamente el modelo y parámetros recibidos (ganancia, tono, etc.).

Provee un método `hasNewConfig()` para verificar si hay nuevos datos disponibles, y `getConfig()` para acceder a dichos valores. Esta arquitectura desacopla la recepción remota del resto del sistema, permitiendo un diseño modular y escalable.

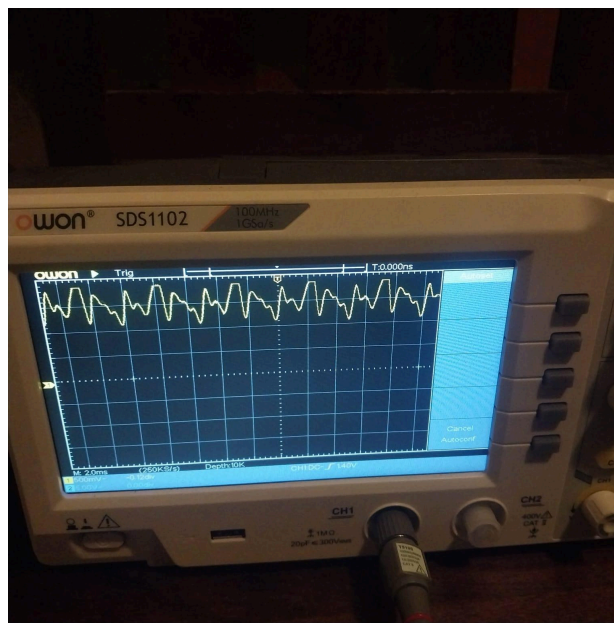
CAPÍTULO 4

Ensayos y resultados

4.1 Pruebas sobre Módulos Individuales

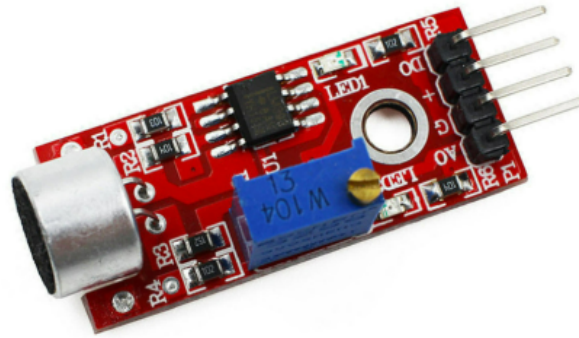
4.1.1 Pruebas sobre el Módulo Amplificador

Se probó el módulo Amplificador diseñado con un osciloscopio. Se observa que el valor medio de la señal se encuentra en los 1.65V del diseño y que se amplifica de forma conservadora asegurando que no se exceda el rango del ADC.



Previo al diseño del amplificador se había testado el sampling del ADC mediante polling simple con un micrófono KY-037. Estos ensayos se compartieron en un video en la primera entrega de este Trabajo Práctico.

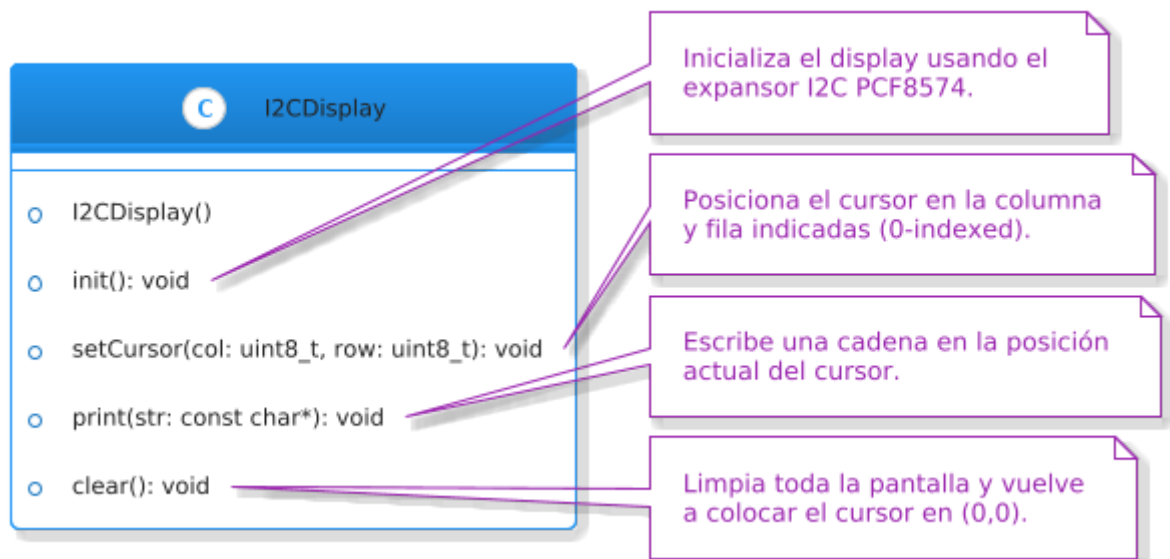
- [TP1-SE/output.wav at entrega-tp1 · Lautaro-De-Lucia/TP1-SE.](#)
- [TP1-SE/video-0.mp4 at entrega-tp1 · Lautaro-De-Lucia/TP1-SE](#)
- [TP1-SE/video-1.mp4 at entrega-tp1 · Lautaro-De-Lucia/TP1-SE](#)



En esta etapa del desarrollo se corroboró que utilizar polling con delays para muestrear audio resultaba ineficiente y que en próximas implementaciones deberíamos utilizar timers (**Ticker** en MbedOS) para un sampling preciso.

4.1.2 Pruebas sobre el Display I2C

Se diseñó una clase **I2CDisplay** que actúa como *wrapper* sobre el módulo **display** provisto por la cátedra.



Esta clase se testeó en el siguiente simple **main.cpp**

```
#include "mbed.h"

#include "I2CDisplay.h"

int main() {

    I2CDisplay lcd;

    lcd.init();

    lcd.setCursor(0, 0);

    lcd.print("Hello!");

    thread_sleep_for(1000);

    lcd.setCursor(0, 1);

    lcd.print("Goodbye!");

    thread_sleep_for(1000);

    lcd.clear();

    thread_sleep_for(1000);

    lcd.setCursor(0, 0);

    lcd.print("Hello!");

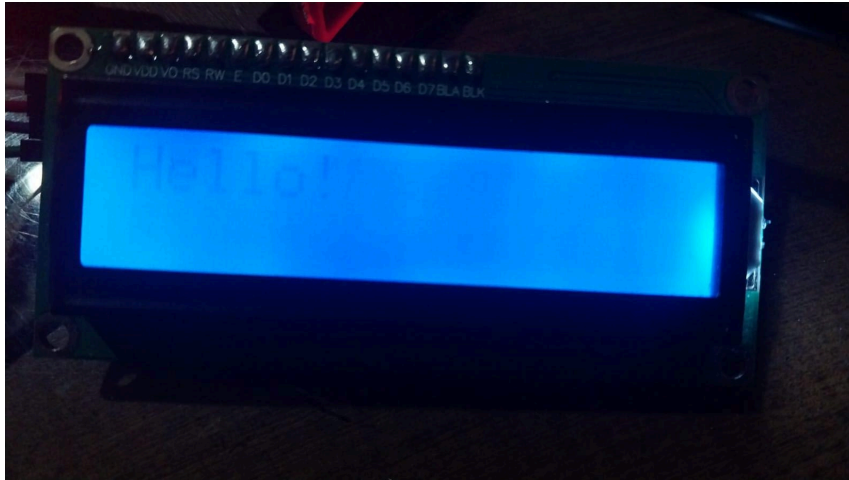
    while (true) {

        thread_sleep_for(1000);

    }

}
```

La prueba no pareció funcionar en un primer intento. Buscando las causas, se alimentó el display con 5V y se descubrió que de esa forma reproducía los caracteres fidedignamente, de modo que el problema estaba en el hardware y no en el software. Sin embargo, el uso de una alimentación de 5V fue desaconsejado por la cátedra, así que se procedió a alimentar a 3.3V y ajustar el potenciómetro de contraste tanto como fuese posible. El resultado se muestra a continuación: Los caracteres son legibles aunque es evidente que la alimentación de 3.3V no es óptima.



La clase `I2CDisplay` utilizada paso a formar parte del `DisplayController` del proyecto principal.

4.1.3 Pruebas sobre el Módulo MicroSD

De forma similar para este módulo, se definió una clase *wrapper* `SDCard` que haga uso de los métodos de las clases `SDBlockDevice` y `FATFileSystem` de MbedOS. Como se observa, su objetivo es proporcionar una interfaz sencilla para registrar datos en forma de texto o binario, ya sea de manera inmediata o por streaming, en archivos almacenados en la tarjeta.



Sobre ella, se implementó el siguiente script de prueba.

```
#include "mbed.h"
#include "SDCard.h"

int main()
{
    printf("\n--- SDCard class demo ---\r\n");

    SDCard sd; // mounts card in constructor

    // 1. Simple ASCII file
```

```
if (sd.writeText("test.txt", "Hello, world!!!\r\n"))
    printf("Text file OK\r\n");
else
    printf("Text file FAILED\r\n");

// 2. Binary example
uint16_t samples[4] = {0xAAAA, 0xBBBB, 0xCCCC, 0xDDDD};
if (sd.writeBinary("samples.bin", samples, 4))
    printf("Binary file OK\r\n");
else
    printf("Binary file FAILED\r\n");

// 3. Text stream example
printf("Opening text stream...\r\n");
if (sd.openTextStream("log.txt")) {
    for (int i = 0; i < 5; ++i) {
        char buffer[32];
        snprintf(buffer, sizeof(buffer), "Line %d\r\n", i);
        if (sd.writeTextStream(buffer)) {
            printf("Wrote: %s", buffer);
        } else {
            printf("WriteTextStream FAILED at line %d\r\n", i);
        }
        ThisThread::sleep_for(500ms);
    }
    sd.closeTextStream();
    printf("Text stream closed.\r\n");
} else {
```

```
        printf("Failed to open text stream.\r\n");
    }

    // 4. Binary stream example
    printf("Opening binary stream...\r\n");
    uint16_t more_data[3] = {0x1111, 0x2222, 0x3333};
    if (sd.openBinaryStream("streamed.bin")) {
        if (sd.writeBinaryStream(more_data, 3))
            printf("Binary stream write OK\r\n");
        else
            printf("Binary stream write FAILED\r\n");
        sd.closeBinaryStream();
        printf("Binary stream closed.\r\n");
    } else {
        printf("Failed to open binary stream.\r\n");
    }





    printf("--- demo finished ---\r\n");

    // automatic unmount in SDCard destructor
}
```

Después de una serie de ensayos fallidos se corroboró que el problema es que el dispositivo efectivamente no funciona con una alimentación de 3.3V, por lo que tuvo que utilizarse el pin de 5V de la NUCLEO-F103RB.

```
Opened with baud rate: 9600

--- SDCard class demo ---
Text file OK
Binary file OK
Opening text stream...
Wrote: Line 0
Wrote: Line 1
Wrote: Line 2
Wrote: Line 3
Wrote: Line 4
Text stream closed.
Opening binary stream...
Binary stream write OK
Binary stream closed.
--- demo finished ---
```

	log	1/1/2098 12:19 AM	Text Document	1 KB
	samples.bin	1/1/2098 12:18 AM	BIN File	1 KB
	streamed.bin	1/1/2098 12:19 AM	BIN File	1 KB
	test	1/1/2098 12:18 AM	Text Document	1 KB

log - Notepad	test - Notepad
File Edit Format View Help	File Edit Format View Help
Line 0	Hello, world!!!
Line 1	
Line 2	
Line 3	
Line 4	

Posteriormente, se hicieron una serie de pruebas de audio, combinando el script de la primera entrega (sampling de audio de micrófono KY-037) con la clase **SDCard**. El objetivo de estas pruebas era evaluar la posibilidad de escribir el archivo de audio no como un archivo binario que simplemente almacena las muestras, sino como un archivo .wav audible. Esto involucra enviar no solo la data binaria sino los headers correspondientes. Sin embargo, aunque la PC detectaba el archivo como un .wav, no se logró que el reproductor de audio de la misma pudiese reproducir la grabación. Por lo que se optó por utilizar un script de python **raw_to_wav.py** (que se encuentra en el repositorio de este proyecto) para la conversión del archivo de muestras binario a un .wav audible, dejando como objetivo a futuro que la NUCLEO-F103RB sea capaz de grabar el .wav a la SD directamente.

Nuevamente, la clase **SDCard** pasó a ser constitutiva del módulo **SDController** en el proyecto principal.

4.1.4 Pruebas sobre el Módulo Wi-Fi

Se utilizó Arduino IDE para grabar un servidor en el módulo ESP-01. El servidor escucha POST requests en un localhost que se envía por UART cuando este se inicializa (el código está disponible en el repositorio del proyecto). Cuando recibe este POST, envía su payload (un objeto JSON) por UART. Como esta es su única funcionalidad, no resultó necesario definir una clase designada para la prueba, sino que se utilizó el simple **main.cpp** que se muestra a continuación. Como se observa, el archivo simplemente recibe el payload y nos permite confirmar la recepción de los datos al imprimirlos por puerto serie.

```
#include "mbed.h"

#include <string.h>

BufferedSerial pc(USBTX, USBRX, 115200);

BufferedSerial esp_uart(D8, D2, 115200);

char line[256];

size_t idx = 0;

int main() {

    printf("NUCLEO listening for ESP-01 JSON...\r\n");

    esp_uart.set_blocking(false);

    while (true) {

        while (esp_uart.readable()) {

            char c;

            esp_uart.read(&c, 1);

            pc.write(&c, 1);

            if (c == '\r' || c == '\n') {

                if (idx == 0) continue;

                line[idx] = '\0';
```

```
if (char* p = strchr(line, '\\r')) *p = '\\0';

char model[32] = {};

float gain = 0.0f, tone = 0.0f;

if (sscanf(line,

            "\\\"model\\\":\\\"%31[^\\\"]\\\",\\\"gain\\\":%f,\\\"tone\\\":%f\\\"",

            model, &gain, &tone) == 3) {

    printf("\\r\\nParsed:\\r\\n");

    printf("  Model : %s\\r\\n", model);

    printf("  Gain : %d.%02d\\r\\n", (int)gain, (int)(gain*100)&0xFF);

    printf("  Tone : %d.%02d\\r\\n", (int)tone, (int)(tone*100)&0xFF);

} else {

    printf("\\r\\nMalformed: %s\\r\\n", line);

}

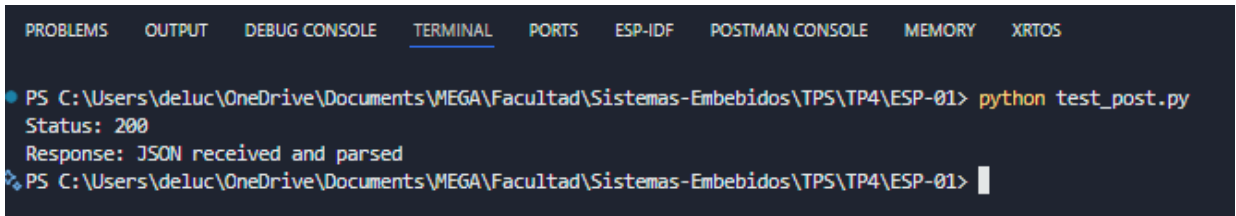
idx = 0;

} else if (idx < sizeof(line) - 1) {

    line[idx++] = c;

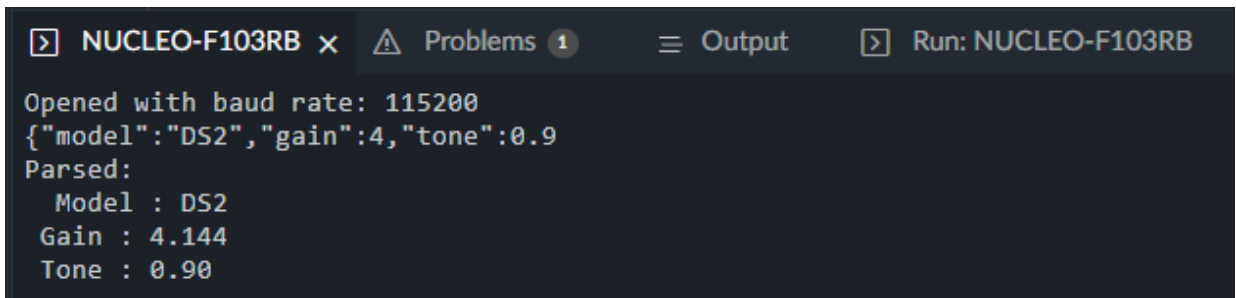
}

}
```



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  ESP-IDF  POSTMAN CONSOLE  MEMORY  XRTOS

PS C:\Users\deluc\OneDrive\Documents\MEGA\Facultad\Sistemas-Embebidos\TP5\TP4\ESP-01> python test_post.py
Status: 200
Response: JSON received and parsed
PS C:\Users\deluc\OneDrive\Documents\MEGA\Facultad\Sistemas-Embebidos\TP5\TP4\ESP-01>
```



```
NUCLEO-F103RB x  Problems 1  Output  Run: NUCLEO-F103RB

Opened with baud rate: 115200
{"model":"DS2","gain":4,"tone":0.9
Parsed:
  Model : DS2
  Gain  : 4.144
  Tone  : 0.90
```

Afortunadamente, no hubo inconvenientes y las pruebas resultaron exitosas en el primer intento. Este `main.cpp` pasó a utilizarse como base del diseño del método `getConfig` de la clase `WiFiReceiver` que se diseñó para el proyecto.

4.2 Pruebas de Integración

El funcionamiento integrado del sistema se muestra en los siguientes videos adjuntados en el repositorio del proyecto:

- [TP1-SE/media/video_1.mp4 at main · Lautaro-De-Lucia/TP1-SE](#)
- [TP1-SE/media/video_2.mp4 at main · Lautaro-De-Lucia/TP1-SE](#)
- [TP1-SE/media/test-wifi.mp4 at main · Lautaro-De-Lucia/TP1-SE](#)

4.3 Cumplimiento de Requisitos

A continuación, compartimos nuevamente la tabla de requisitos presentada al comienzo de este informe, señalando aquellos requisitos que pudieron ser cumplidos exitosamente y aquellos que no pudieron ser cumplidos. Como se observa, el único requisito que no pudo ser satisfecho involucra la escritura de las muestras de audio a través de el DAC, lo cual hubiera sido sencillo de implementar pero tristemente no pudo llevarse a cabo dado que el STM32F103RB no dispone de un DAC. De ahí que la única etapa amplificadora diseñada sea una etapa preamplificadora para adaptar el input de guitarra al ADC.

Grupo	ID	Descripción
1. Firmware	1.1	Implementación de algoritmo de distorsión configurable por software
	1.2	Lógica de manejo de modos DAC/REC mediante FSM.
	1.3	Lectura del ADC, escritura del DAC.
	1.4	Procesamiento de JSON recibido por HTTP para definir el preset activo.
2. Hardware	2.1	Conexión a preamplificador para entrada de guitarra analógica.
	2.2	Conexión a DAC y preamplificador de salida para amplificador externo
	2.3	Tarjeta MicroSD conectada por SPI para grabación.
	2.4	Display I2C para mostrar estado del sistema
	2.5	Botón de encendido y botón selector de modo conectados a pines GPIO
3. Interfaz de Usuario	3.1	Mostrar nombre del preset y tipo de salida (REC/DAC) en pantalla
	3.2	Alternar entre modos mediante botón físico.
	3.3	Cambiar dinámicamente la pantalla según configuración recibida
4. Configuración	4.1	Escuchar conexiones entrantes en puerto configurable via Wi-Fi o Ethernet
	4.2	Interpretar mensajes de configuración con campos: tipo, distorsión, tono, nivel, etc.

Tabla 4.3: Evaluación del Cumplimiento de Requisitos

4.4 Documentación del desarrollo realizado

En la Tabla 4.4 se muestran los elementos que resumen la información más importante sobre el pedal Shapeshifter y su referencia correspondiente.

Elemento	Referencia
Análisis comparativo de alternativas en el Mercado	Tabla 1.2
Diagrama de Bloques de Alto Nivel del Sistema	Figura 1.3
Requisitos del proyecto	Tabla 2.1
Diagrama de bloques de Hardware de Alto Nivel	Figura 3.1.1
Diagramas de conexiones de Hardware	Figura 3.1.2
Diagrama de Flujo del <i>software</i>	Figura 3.3
Diagrama de clases del <i>software</i>	Figura 3.4.1
Evaluación de cumplimiento de requerimientos	Tabla 4.3

Tabla 4.5: Elementos que resumen la información más importante sobre el Shapeshifter.

CAPÍTULO 5

Conclusiones

5.1 Resultados obtenidos

Se logró desarrollar un prototipo funcional del pedal de distorsión digital que cumple con los requerimientos establecidos en el planteo inicial del proyecto. Se consiguió:

- Procesar una señal de audio en tiempo real, aplicando distorsión cuando el modo lo requiere y almacenando la salida en una tarjeta MicroSD.
- Controlar el comportamiento del sistema mediante una máquina de estados activada por botones físicos, permitiendo alternar entre modos de grabación, salida directa y tipo de efecto.
- Visualizar en un display LCD el estado actual del sistema en todo momento.
- Configurar remotamente el modelo de distorsión a través de una interfaz WiFi basada en peticiones HTTP con formato JSON.

5.2 Próximos pasos

Los pasos a seguir para lograr una implementación completa del proyecto son los siguientes:

- Implementar un DAC para la salida a un amplificador. Ya sea con un PWM, DAC externo o usando una NUCLEO que posea un DAC. Esto involucraría diseñar una etapa amplificadora de salida (entre la NUCLEO y el amplificador de guitarra).
- Grabación a .wav desde la NUCLEO en lugar de tener que utilizar Python como intermediario de conversión .bin a .wav.
- Controles genéricos de nivel, tono y distorsión con potenciómetros.
- Mejora en algoritmo de distorsión incluyendo más parámetros en el preset que se envía de forma remota (en lugar de enviar distorsión, tono y nivel, estos parámetros genéricos se configuran desde sus potenciómetros, de modo que lo que se envía en el JSON son parámetros más complejos como el nivel de clipping, el modelo de filtro y sus coeficientes, etc.).

- Aplicación Móvil que provea una biblioteca de presets y nos permita conectarnos y grabar al dispositivo.
- PCB, Ensamble y Diseño de Gabinete metálico.

Estas mejoras son perfectamente realizables y nos permitirían tener un prototipo comercial que es francamente original cuando se observa lo que se ofrece en el mercado.



Figura 5.2: Prototipo comercial hipotético del pedal *shapeshifter*

Bibliografía

- [1] A Beginner's Guide to Designing Embedded System Applications on Arm Cortex-M Microcontrollers - Ariel Lutenberg, Pablo Gomez, Eric Pernia
- [2] Overview of NUCLEO-F103RB - <https://os.mbed.com/platforms/ST-Nucleo-F103RB>
- [3] Mbed OS Official Documentation - [Introduction to Mbed OS 6 | Mbed OS 6 Documentation](#)